Automating SQL Injection Exploits

Mike Shema <mikeshema@yahoo.com>
IT Underground, Berlin 2006

2006-01 (v3)

Overview

- SQL injection vulnerabilities are pretty easy to detect.
- The true impact of a vulnerability is measured by the quality of information or access that can be gained with a SQL injection exploit.

Why Automate?

- An audit is only as good as the auditors.
- Verify the potential impact of a vulnerability.
- Enumeration follows a standard methodology (i.e. one that can automated).
- Enumeration can be tedious.

Types of Exploits

- Process alteration
  - Bypass a login prompt (' OR 1=1)
- Direct enumeration
  - Display the results of an arbitrary query
- Indirect enumeration
  - Indicate the success of an arbitrary query
- Command execution
  - Access some extended functionality of the database

Direct Enumeration Via UNION

- Determine number of columns
- Determine acceptable column types
- Create custom SELECT
- Parse response

Indirect Enumeration

- Determine presence of vulnerability
- Characterize positive response
  - AND 1
- Characterize negative response
  - AND 0
- Create custom SELECT
  - Retrieve a single record.
  - Must be able to iterate each bit value of the record.

Bitwise Enumeration

- Walk through the value bit by bit
- Advantages
  - String may be of arbitrary length
  - String may be of arbitrary content
- Disadvantages
  - Can take a long time
  - Subtle differences in handling different data type
    - e.g. VARBINARY may contain 0x00 characters

Bitwise Enumeration

- Convert string index to integer
  - `CONVERT(INT,SUBSTRING(`*`str`*`,`*`index`*`,1))`
- Convert NVARCHAR (Unicode) string index to integer
  - `CONVERT(INT,SUBSTRING(`*`str`*`,`*`index`*`,1))`
- Many other encodings or functions are possible
  - ASCII()
  - BYTE

Bitwise Enumeration

- Core concept demonstrated in Python:

```
>>> a = 'a'                          <-- 'a' = 0x97
>>> for i in range(0,8):
...      ord(a) & 2**i               <-- bitwise AND
...
1
0
0
0
0
32
64
0
```

Bitwise Enumeration

- Core concept applied in SQL:

```
SELECT 1 FROM 'a' & 1;
1
SELECT 2 FROM 'a' & 2;
0
SELECT 4 FROM 'a' & 4;
0
SELECT 8 FROM 'a' & 8;
0
SELECT 16 FROM 'a' & 16;
0
SELECT 32 FROM 'a' & 32;
1
SELECT 64 FROM 'a' & 64;
1
SELECT 128 FROM 'a' & 128;
0
```

Parsing the Responses

- Record responses, e.g.
  false (0)
  true (1)
  true (1)
  false (0)
  false (0)
  false (0)
  false (0)
  true (1)
- 01100001 = 0x97 = 'a'

Tips for Preparing the Query

- Use hexadecimal string representation in WHERE clauses.
  - Avoid single quotes.
  - Can also handle Unicode strings.
- For example:

Tips for Preparing the Query

- Use hexadecimal string representation in WHERE clauses.
  - Avoid single quotes.
  - Can also handle Unicode strings.
- For example:
  - 'mike' = 0x6d696b65
  - 'mike' = 0x6d0069006b006500 (Unicode)

Bitwise Enumeration

- How many requests?
  - 8 per character (strings, binary values)
    - 7 if you know the result only contains ASCII text
  - 32 per integer
- Examples
  - sa password
  - Information schema
    - Databases (catalogs), Tables, Columns
  - Multi-record results

Bitwise Query Template

```
AND #n#
IN
(SELECT
 CONVERT(INT,SUBSTRING(#COL#,#i#,1)
) & #n#
FROM #CLAUSE#
```