# It all starts with the '

(SQL injection from attacker's point of view)

**Miroslav Štampar**

**(dev@sqlmap.org)**

fsec foi

# What is SQL injection?

*"SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of DBMS server for parsing and execution"*

(source: msdn.microsoft.com)

# What is SQL injection? (2)

- In plain speak, SQL injection is (mostly) about the unauthorized DBMS data access
- Just in special cases, it's about the data modification together with system takeover
- "*Hello World*" vulnerable code example (PHP/MySQL):
  - `$sql = "SELECT * FROM table_name WHERE id = " . $_GET["id"];`
  - `$result = mysql_query($sql)`

# Well known cases

- In period 2005 till 2007 Albert Gonzalez has stolen 130 million credit card numbers
- June 2007 – **Microsoft** U.K. Website defaced
- December 2009 – **RockYou** (32 million credentials stolen)
- December 2009 – **NASA**
- July 2010 – **The Pirate Bay**

# Well known cases (2)

- February 2011 – **HBGary** (Anonymous)
- March 2011 – **MySQL**
- March & May 2011 – **Comodo** (certificate reseller)
- June 2011 – **Sony**, **PBS** (@LulzSec)
- August 2011 – **Nokia**
- September 2011 – **NetNames** DNS records (Betfair, The Telegraph, The Register, The National Geographic, UPS, Vodafone...)

# Attackers' motivation

**number** of vulnerable sites collected **x** **time** spent on each site = **const.**

**time** spent on particular target / **motivation** = **const.**

(target = multiple (co)sites)

# Attackers' profiles

1) Targeting easy targets, unfocused, fast pace, predictable behavior, blabbering, don't really care about the target (“***script kiddies***”)
2) Persistent, focused, slow motion, covert, silent, higher than average technical skills, highly motivated (money, extortion, fame) (“*black hats*”/“*crackers*”/”***bad guys***”)
3) For fun and knowledge, could warn you about the problem found, don't want to cause any damage (*”gray hats”/”hackers”/”****good guys****”*)

# SQL injection techniques

- Boolean/Blind – `AND 1=1`, slow, 1 bit per request, page differentiation based, low difference ratio represents `True` response, `False` otherwise (in most common cases)
- Error-based – `CONVERT(INT,(<subquery>))`, fast, 1 (sub)query result per request, based on inclusion of subquery result(s) inside DBMS error message
- Union/Inband – `UNION ALL SELECT NULL,...,(<subquery>),NULL,NULL,...`, fastest, in FULL variant whole table dump per request, in PARTIAL variant 1 query result per request

# SQL injection techniques (2)

- Time-based – `AND 1=IF(2>1, BENCHMARK(5000000,MD5(CHAR(115,113,108,109,97,112))),0)`, slowest, 1 bit per request, delay represents `True` response, `False` otherwise
- Stacked – `;INSERT INTO users VALUES (10, 'test', 'testpass')`, affecting mainly Microsoft SQL and PostgreSQL platforms, required for data modification, system access (e.g. `xp_cmdshell`), UDF injection, there are also time and error-based variants for data retrieval
- Other (DNS exfiltration, “Lateral”, Second order attacks, etc.)

# Attacking phases

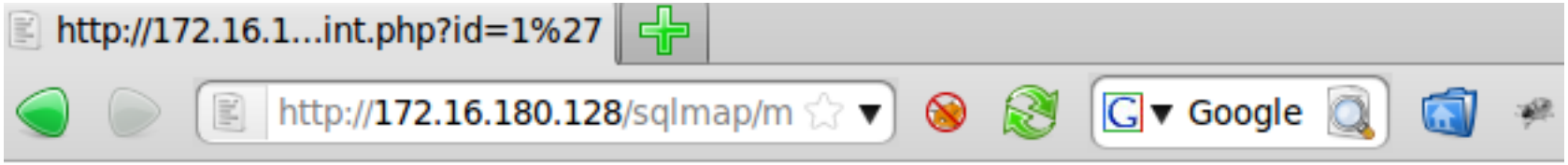
- Most SQL injection attacks can be dissected into following phases:
  1) Searching for a vulnerable parameter (aka. “Injection point”)
  2) Fingerprinting backend DBMS (for usage of proper payloads – e.g. error-based)
  3) Finding a fastest applicable technique (`U>E>B>T=S`)
  4) Enumeration (retrieving data of interest – e.g. usernames and passwords)
  5) Eventual post-exploatation (underlying OS access, data modification, web server takeover)

# Attacking workflow (1)

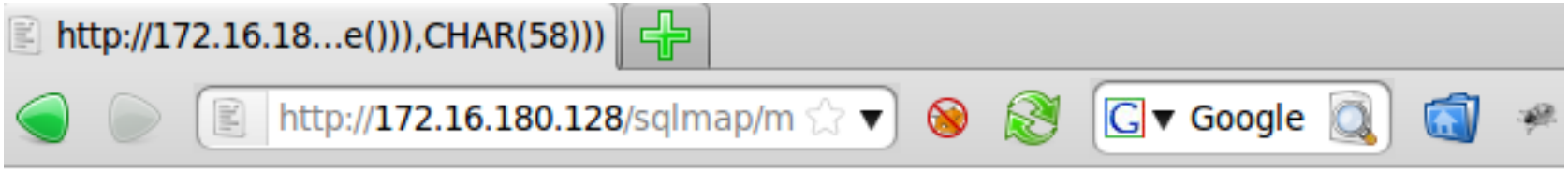
- Finding a potential target – underground forums (e.g. antichat.ru), IRC channels, Google (e.g. dorks), domain scanning (e.g. `nmap -p 80`), SHODAN (specialized Computer Search Engine), “drive-by” (e.g. proxying of all casual everyday traffic), “hit-list”, etc.
- Manual web-browser based GET/POST parameter tampering with “poisonous” SQL characters (e.g. appending character `'` to the end of the original parameter value)

# Attacking workflow (2)

- In case of a contained DBMS error attacker immediately proceeds with the “proper” (DBMS dependent) error-based payloads




**SQL error:** 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 '' LIMIT 0, 1' at line 1




**SQL error:** XPATH syntax error: '\:5.1.41-3~bpo50+1,testdb:'

# Attacking workflow (3)

- Otherwise, following step is the usage of common boolean based payloads (e.g. `AND 1=1--%20`, `OR 1=1--%20`, etc.)
- 

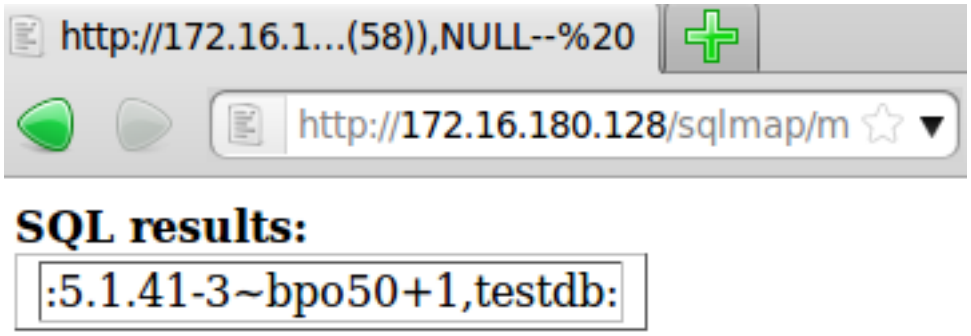
- 

# Attacking workflow (4)

- Along the attacking line, proper SQL injection prefix/suffix combination has to be “guessed”
- For example, different payloads are needed in cases: `”SELECT * WHERE id='.$_GET[“id”]` and `”SELECT * WHERE id=('.$_GET[“id”]')”`
- Adds one more dimension to the problem (more combinations)
- Most common prefixes are: <blank>, `'`, `“`, `')`, `“)`
- Problem of suffix finding can be most often circumvented by the usage of generic comment form: `--%20`

# Attacking workflow (5)

- In case that boolean/blind injection worked, there is a great possibility for union/inband
- Brute-force search for proper column number
- `ORDER BY` fast column number finding technique
- Looking if it's a partial or full inband case
- Finding usable (presented) column



# Attacking workflow (6)

- If everything fails attacker could proceed (high motivation required) with common time-based payloads and/or stacked queries
- Most commonly, this happens in `INSERT/UPDATE/DELETE` SQL injection cases (and DBMS error messages are suppressed)
- Common payloads are: `AND 1=IF(2>1,BENCHMARK(5000000,MD5(CHAR(97))),0)`, `;IF(2>1) WAITFOR DELAY '0:0:5'`, `;SELECT COUNT(*) FROM GENERATE_SERIES(1,5000000)`, etc.

# Attacking workflow (7)

- Enumeration phase consists of retrieving (”stealing”) data of interest:
  - Database and table names
  - Table dumps (e.g. `users, admin,...`)
  - System stored usernames and password hashes
  - User privileges and roles
- Highly DBMS dependent (system database and table names, common function names, etc.)
- Usage of automated tools is highly preferable in boolean/blind and time-based techniques

# Attacking workflow (8)

- Eventual post-exploitation is highly bounded by the applicable SQL injection technique(s) and current user privileges
- Best/worst case scenario is the availability of stacked technique along with the “admin” privileges
- UDF injection (e.g. functions for OS command execution), web shell (e.g. union/inband MySQL `INTO OUTFILE`), Metasploit payload upload and execution (e.g. stacked Microsoft SQL `xp_cmdshell()`), etc.

# Process automation

1) Web Application Assessment Proxy (optional) – **BURP**, WebScarab
2) Automatic SQL injection detection and exploitation tool – **sqlmap**, Havij, sqlninja
3) Second order exploitation framework – **Metasploit** (optional)

# Introducing sqlmap

- *“sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database server(s)”*
- Unofficially, sqlmap is AIO (All-In-One) SQL injection tool
- **Over 10,000** repository updates and/or checkouts **on a monthly basis**
- Part of most popular security distributions: Backtrack, Backbox, Web Security Dojo, OWASP Web Testing,...

# sqlmap capabilities

- Fully supported backend DBMSes (and growing): MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, SQLite, Firebird, Sybase and SAP MaxDB
- Fully supported SQL injection techniques: Boolean/Blind, Error-based, Union/Inband (partial & full), Timed-based, Stacked
- Enumeration of: database users, users' password hashes, users' privileges, users' roles, databases, tables and columns, etc.

# sqlmap capabilities (2)

- Automatic recognition and cracking of password hashes
- Support for IDS/WAF evasion in form of “tampering” scripts
- Web server file upload/download
- Arbitrary OS command execution and retrieval of standard console output
- Establishment of an out-of-band TCP/UDP connection between the attacker's machine and the database server (Metasploit, ICMPsh)

# Closing words (for attackers)

- Learn SQL more than "well" – “SQL injection is 100% SQL. The rest is injection”
- Setup (virtual) vulnerable environment entirely by yourself – for gaining better mindset and possessing LEGAL assessment testing machine
- Mimic with comprehension what other “attackers” do – including automated tools (e.g. sqlmap)
- Research defensive mechanisms – for advanced comprehension of the topic

# Closing words (for defenders)

- Use (only) prepared SQL statements (bullet-proof against SQL injection attacks)
- Self-filtering procedures are prone to common “programmer's” mistakes (e.g. forgetting to hard type-cast the integer parameter value)
- Turn-off error (DBMS) reports
- Use least privileged run (non-admin “query” user)
- Always expect worst scenario and setup accordingly (e.g. use `GRANT/REVOKE` mechanism accordingly)

# Questions?