# Automating Blind SQL Injection for SQLite Database Based Application

29 October 2021

Alaa (aka b1tByte)

# 1 مُقدّمة

هذه الورقة تناقش أتمتة إستغلال ثغرات الـ Blind SQLi وعلى وجه التحديد قواعد البيانات من نوع SQLite. الورقة مُقسّمة الى 13 فصل من ضمنها هذه المقدمة. الفصل رقم 2 يوضح بشكل مختصر جدًا ثغرات الـ Blind SQLi. الفصل رقم 3 يشرح المعمل الذي تم إختبار عملية أتمتة الثغرة عليه. الفصول من 4 حتى 12 تشرح أولًا خطوات إستغلال ثغرة Blind SQLi ومن ثم تقترح Syntax/Query ثابت بالإمكان إستخدامه لأتمتة إستغلال الثغرة بإستخدام أي لغة برمجة، كما يجدر بنا الذكر بأن هذه الورقة مُعتمِده على لغة Python في أتمتة إستغلال الثغرة. آخرًا، الفصل 13 يستعرض المُخرج النهائي ( AutoBlindSQLite.py ) الخاص بأتمتة هذه الثغرة ويلخّص الدوال الخاصة به وعملها.

# 2 Blind SQL Injection

في ثغرة الـ Blind SQLi المخترق يقوم بحقن Query المُخرَج النهائي منها ليس طباعة أسماء الجداول أو تحديث جدول معين وغيرها من العمليات المباشرة، إنما المُخرَج من الـ Query هو إستجابة تطبيق الويب بنعم ( True ) أو لا ( False ) ، وعلى هذا الأساس يبدأ المخترق بإرسال Queries هي في الأساس عبارة عن أسئلة لتطبيق الويب من نوع Boolean Questions، مثل :

- هل قاعدة البيانات تحتوي على 5 جداول ؟
- هل اسم الجدول رقم 1 يبدأ بالحرف a ؟
- هل عدد الصفوف في الجدول رقم 1 هو 5 صفوف ؟

وغيرها من الأسئلة التي تكشف قاعدة البيانات ومحتوياتها، وكلما أراد المخترق إستخراج معلومة من قاعدة البيانات فعليه أن يبني **السؤال (Boolean SQL Query )** الذي يتيح له إستخراج المعلومة من إستجابة التطبيق . أيضًا يجب على المخترق فهم **إستجابة التطبيق ( Response )** في حال كانت الإجابة نعم و في حال كانت الإجابة لا.

بالنسبة للجزء الأول **بناء السؤال** نعتقد بأن هذا الجزء يعتمد على نوع التطبيق الذي يدير قاعدة البيانات ( SQL Software )، في هذه الورقة نستهدف قواعد البيانات SQLite ، لكن بشكل عام في مختلف قواعد البيانات يكون الـ Syntax ثابت طالما أن قاعدة البيانات التي يتم التعامل معها هي من نوع SQL، وقد توجد الإختلافات في نداء الدوال ( Functions ) التي قد تختلف مسمياتها بين التطبيقات التي توفر ادارة قواعد البيانات

وبالنسبة للجزء الثاني **تحديد نوع الإجابة** ( نعم أم لا ) نعتقد بأن هذا الجزء يعتمد بشكل كلي على تطبيق الويب، ويجب على المخترق تحليل التطبيق وتحديد الحالات التي يستجيب فيها

التطبيق بنعم ( True ) والحالات التي يستجيب فيها التطبيق بلا ( False )، بالتالي يستطيع المخترق إستخراج البيانات بناءً على الأسئلة التي يقوم بتمريرها لتطبيق الويب

في هذه الورقة نحاول استخراج البيانات التالية من قاعدة البيانات الخاصة بالتطبيق:

1. **عدد الجداول**
2. **أسماء الجداول**
3. **عدد الأعمدة لكل جدول**
4. **أسماء الأعمدة لكل جدول**
5. **عدد الصفوف في كل جدول**
6. **استخراج كامل البيانات الخاصة بكل جدول**

ولتحقيق ذلك قمنا بسؤال تطبيق الويب العديد من الأسئلة لاستخراج هذه البيانات، كما سيتم شرحه في الأقسام القادمة لاحقًا.

نود الإشارة أيضًا بأنه توجد أدوات مختلفة تحاكي عملية إستخراج البيانات مثل أداة SQLMap وهذه الورقة ليست مبنية على هذه الأداة وليست تطوير لها، إنما الغرض الأساسي منها هو أولًا الخوض بشكل يدوي في استغلال ثغرات الـ Blind SQLi وثانيًا محاولة أتمتة الإستغلال لتأكيد مفهومنا لهذه الثغرة وطُرق إستغلالها.

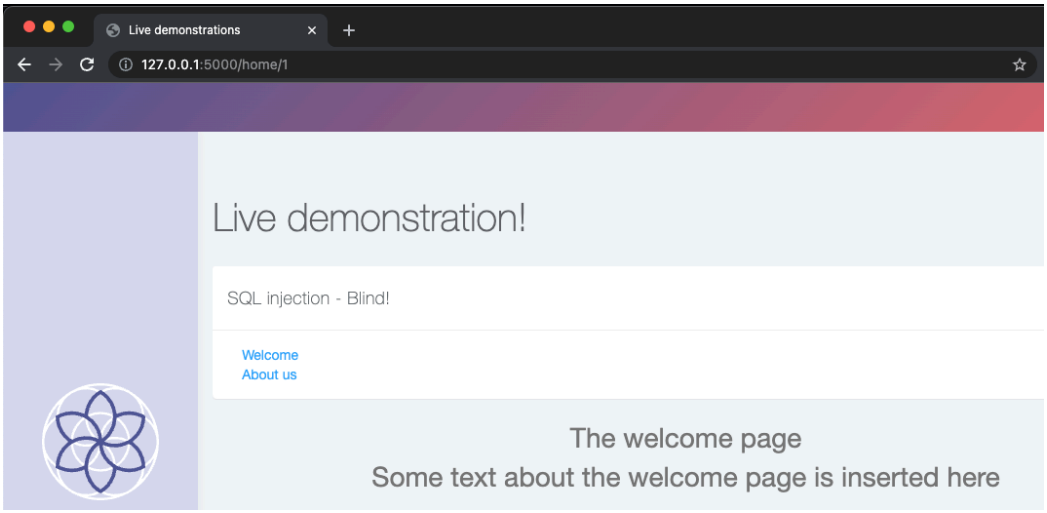
# 3 Lab Environment

هذه الورقة معتمده على التطبيق التالي:

- **OWASP SKF** : https://owasp.org/www-project-security-knowledge-framework/
- **الثغرة :** KBID 156 - SQLI (Blind)

## 3.1 OWASP SKF Blind SQLi Cases

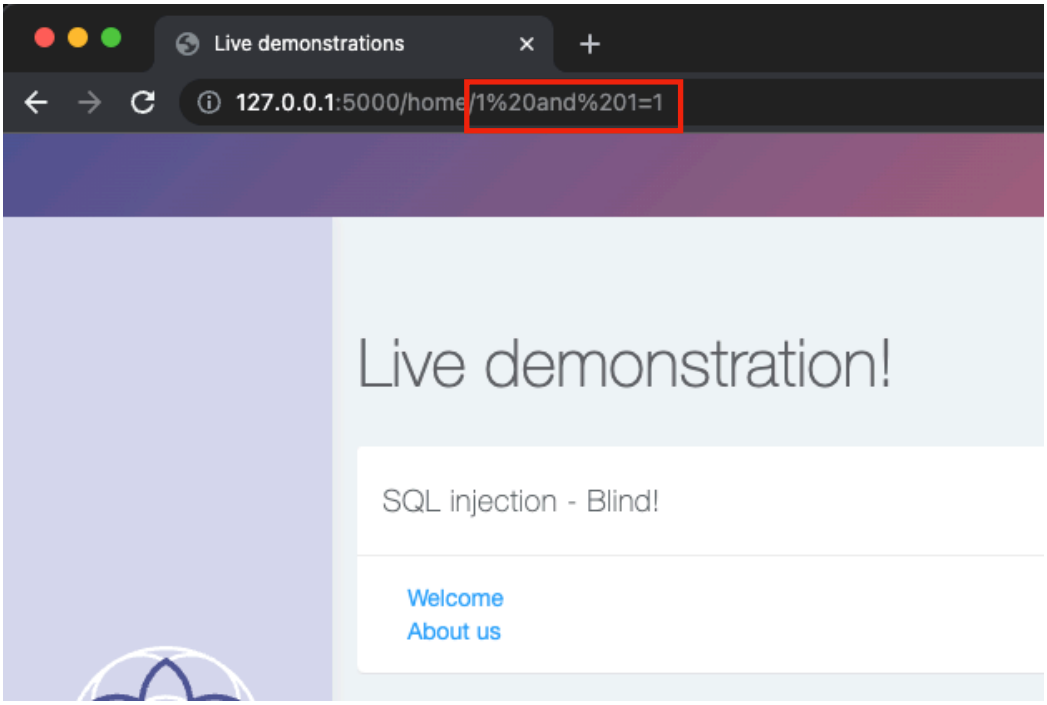
بعد تحليل التطبيق وجدنا أن الدالة المصابة بالحقن هي الدالة الخاصة بعرض الصفحات الفرعية من التطبيق، على سبيل المثال لو قمنا بإستعراض الصفحة رقم 1 في التطبيق سنجد أن التطبيق يستجيب بشكل سليم عبر طباعة محتوى صفحة الـ Welcome Page كالآتي :




ولو قمنا بحقن الـ Parameter الخاص برقم الصفحة كالآتي :

```
1 and 1=1
```

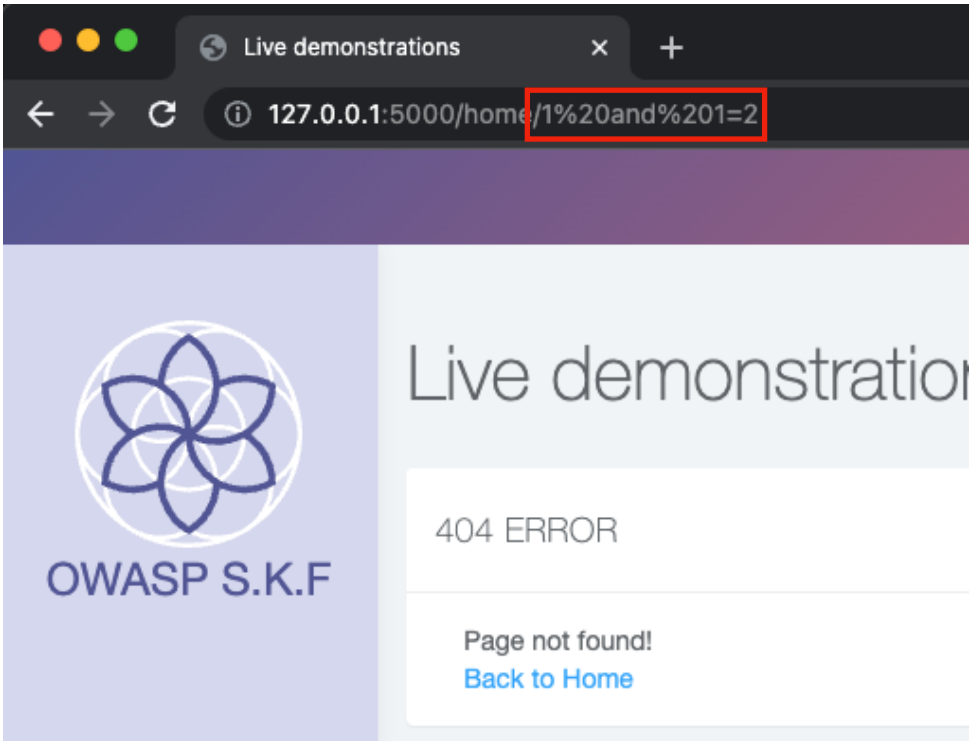
سنجد أن التطبيق يستجيب بشكل سليم أيضًا ويعرض لنا الصفحة




لكن لو قمنا بجعل ناتج الشرط and خاطئ عبر حقن الآتي

```
1 and 1=2
```

سنجد أن التطبيق لا يستجيب ويعرض لنا صفحة 404 مما يؤكد أن التطبيق مُصاب بثغرة Blind SQLi




بناءً على هذا قمنا بتحديد الحالة التي يستجيب فيها التطبيق ( True case ) والحالة التي لايستجيب فيها التطبيق (False case ) كما يظهر في الجدول الآتي:

| Case | Injection Payload | Application Response |
|---|---|---|
| **TRUE** | 1 and (true condition) | في هذه الحالة سيقوم التطبيق بالاستجابة بصفحة Welcome Page |
| **FALSE** | 1 and (false condition) | في هذه الحالة سيقوم التطبيق بالاستجابة بصفحة 404 |

# 4 Extracting Number of Tables

لإستخراج عدد الجداول في قاعدة البيانات قمنا بإستخدام الـ Payload التالية :

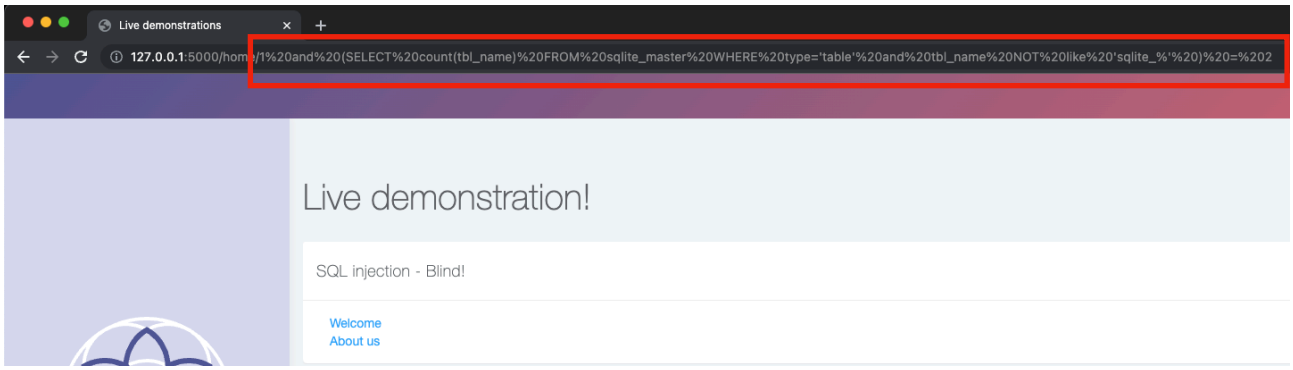
```
1 and (SELECT count(tbl_name) FROM sqlite_master WHERE
type='table' and tbl_name NOT like 'sqlite_%' ) = FuzzingNumber
```

الجزء بين الأقواس () يقوم بـ عَد (counting) أسماء الجداول والتي هي في الأساس عبارة عن قيم مخزنة في العامود tbl_name من الجدول sqlite_master . جدول الـ sqlit_master عبارة عن جدول يخزن البيانات من نوع الـ Metadata . سيتم عد القيم حسب الشرط المذكور بعد الـ WHERE condition والذي يقوم بفلترة النتائج وإستخراج أسماء الجداول فقط ( الجداول التي قام المستخدم بتعريفها وليست الجداول الخاصة بالـ Metadata )
بعد ذلك القيمة النهائية العائدة من التعليمة بداخل الأقواس () هي عبارة عن رقم، سنقوم بمقارنة هذا الرقم مع الـ FuzzingNumber

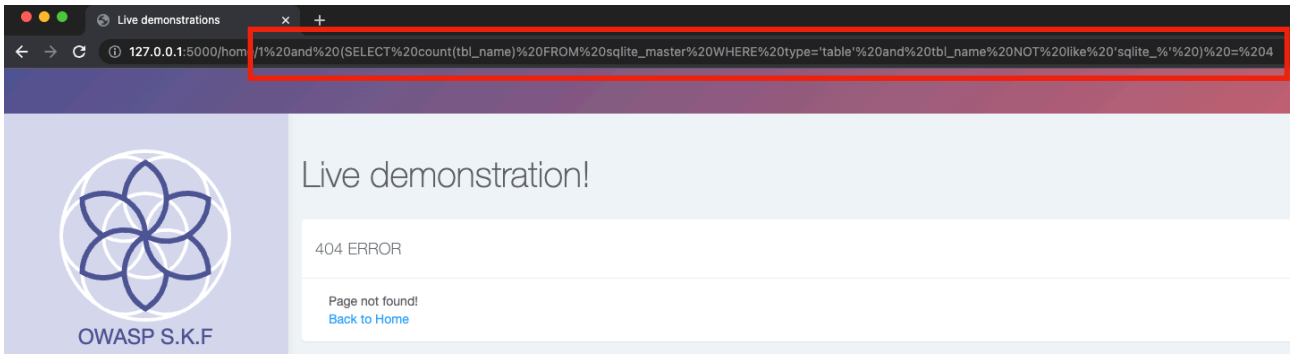
الـ FuzzingNumber هو رقم سنقوم بتمريره لمقارنته مع عدد الجداول الذي تم جلبه عن طريق التعليمة بداخل الأقواس ()
في حالة كان الـ FuzzingNumber الذي قمنا بتمريره مساوٍ للرقم العائد سيصبح شرط الـ and الثاني صحيح (true)، على سبيل التبسيط سيكون كالآتي :

```
1 and (returnedValue) = FuzzingNumber
1 and (2) = 2
1 and true
true
```

وللتلخيص، في حالة قمنا بتمرير رقم مساوٍ لعدد الجداول المعرّفة في قاعدة البيانات فسيقوم التطبيق بطباعة الصفحة بشكل سليم كالآتي :




ولو قمنا بتمرير عدد خاطئ سيستجيب التطبيق بصفحة 404 كما تم شرحه سابقًا




# 5 Extracting Table Name Length for Each Table

قبل إستخراج أسماء الجداول نحن بحاجة لاستخراج **طول اسم الجدول** (عدد الحروف)، ولتطبيق ذلك قمنا بإستخدام الـ Payload التالية :

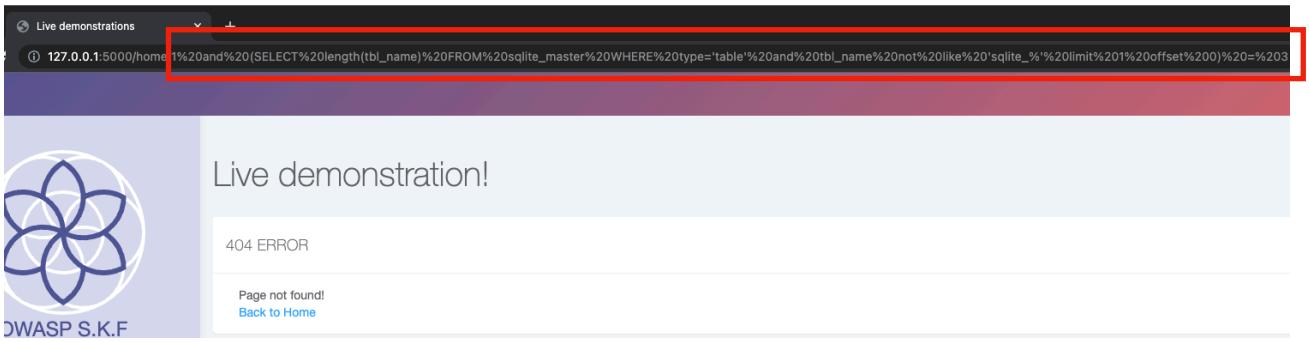
```
1 and (SELECT length(tbl_name) FROM sqlite_master WHERE
type='table' and tbl_name not like 'sqlite_%' limit 1 offset rowOffset) =
FuzzingNumber
```

الـ Payload هذه مشابهه في عملها الـ Payload السابقة لكن الفرق أننا قمنا بإستخدام الدالة length() ومررنا لها اسم الجدول tbl_name ، حيث ستقوم الدالة length بـ عَد الحروف في اسم الجدول، قمنا أيضًا بالإعتماد على الـ limit والـ offset . الـ limit ستقوم بإرجاع صف واحد ، والـ offset سنقوم بإستخدامها للتحرك الى الصف التالي ( الصف التالي في حالتنا هو طول اسم الجدول التالي الذي نريد استخراجه ) على سبيل المثال لو أردنا إستخراج طول اسم الجدول الأول سنمرر القيمة 0 الى الـ offset ، ولو أردنا طول اسم الجدول الثاني سنمرّر القيمة 1 للـ offset ، وهكذا حتى ننتهي من استخراج جميع القيم لجميع الجداول

الـ FuzzingNumber يمثل الرقم الذي سنقوم بمقارنته مع النتيجة العائدة من التعليمة داخل الأقواس ()

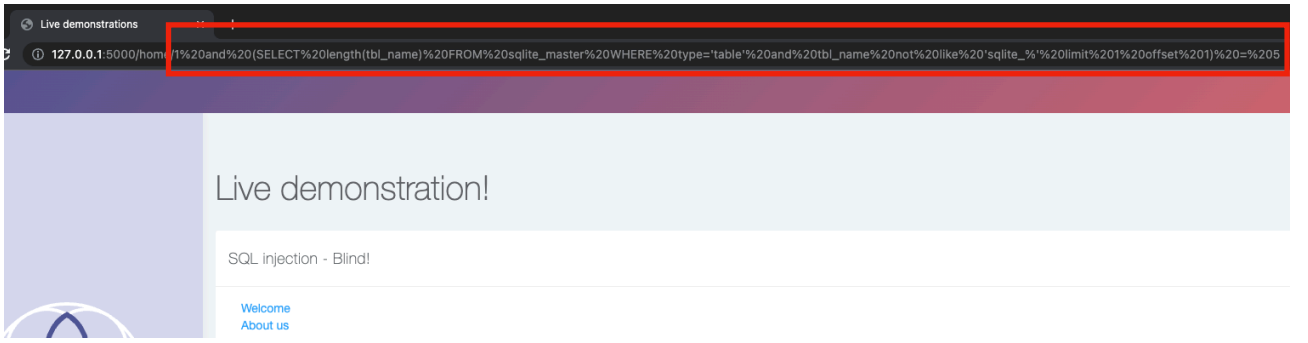
في الصورة التالية قمنا بسؤال التطبيق عن عدد الحروف لأسم **الجدول الأول** ( تخمين خاطئ )

```
http://127.0.0.1:5000/home/
1%20and%20(SELECT%20length(tbl_name)
%20FROM%20sqlite_master%20WHERE%20type='table'%20and%20
tbl_name%20not%20like%20'sqlite_%'%20limit%201%20offset%200)
%20=%203
```




وفي هذه الحالة قمنا بسؤال التطبيق عن عدد الحروف لأسم **الجدول الثاني** ( تخمين صحيح )

```
http://127.0.0.1:5000/home/
1%20and%20(SELECT%20length(tbl_name)
%20FROM%20sqlite_master%20WHERE%20type='table'%20and%20
tbl_name%20not%20like%20'sqlite_%'%20limit%201%20offset%201)
%20=%205
```



# 6 Extracting Tables Names

بعد إستخراج طول اسم الجدول في القسم السابق الآن نستطيع استخراج اسم الجدول ، حيث أننا نعلم اسم كل جدول يتكوّن من كم حرف، مع العلم بأنه نعتقد أنه بالإمكان استخراج الاسم مباشرة بدون حساب عدد الحروف، لكننا قمنا بإتباع الطريقة المفصلة في استخراج البيانات من قاعدة البيانات حتى نستطيع أتمتة العملية لاحقًا

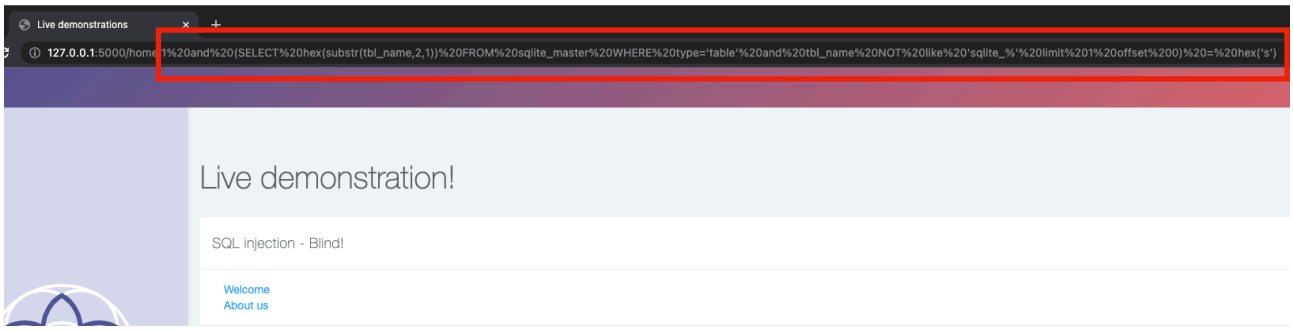
الـ Payload التي قمنا بإستخدامها لإستخراج أسماء الجداول هي كالآتي :

```
1 and (SELECT hex(substr(tbl_name,charIndex,1)) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset rowOffset) = hex('fuzzingChar')
```

الاختلاف في هذه الـ Payload أننا قمنا باستخدام دالتين، دالة hex() ودالة substr() . دالة hex() سنستخدمها لتحويل قيمة الحرف الى Hexadecimal بالتالي نستطيع إجراء المقارنة ، ودالة substr() سنستخدمها لاسترجاع حرف معين من الأسم ، بمعنى آخر تمكننا من التحرك (Iterating) حول حروف اسم الجدول، نستخدم الـ charIndex لتمرير رقم الحرف الذي نريد استخراجه من الاسم، فلو أردنا الحرف الأول سنمرر القيمة 1 ولو أردنا الحرف الثاني سنمرر القيمة 2 ، وبما أننا نعلم عدد الحروف لكل اسم جدول، فسنمر على عدد محدود من المحاولات الـ rowOffset يمثل هنا اسم الجدول الذي نريد استخراج حروفه والقيمة العائدة من التعليمة بين الأقواس ستكون قيمة الحرف لكن ممثلة بـ Hexadecimal ، سيتم مقارنة هذه القيمة مع الـ fuzzingChar والذي يمثل قيم الحروف الأبجدية ( في كل مرة سنقوم بتمرير حرف واحد للتخمين )

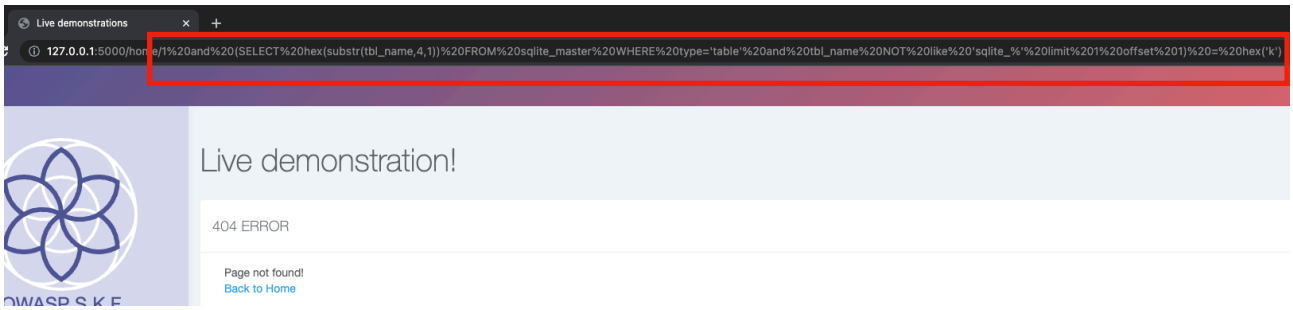
على سبيل المثال، في هذه الـ Payload قمنا بسؤال التطبيق : هل الحرف الثاني من اسم الجدول الأول هو s ( تخمين صحيح )

```
1 and (SELECT hex(substr(tbl_name,2,1)) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) = hex('s')
```



وفي هذا المثال قمنا بسؤال التطبيق : هل الحرف الرابع من اسم الجدول الثاني هو k
( تخمين خاطئ )

```
1 and (SELECT hex(substr(tbl_name,4,1)) FROM sqlite_master WHERE
type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 1) = hex('k')
```




# 7 Extracting Number of Columns for Each Table

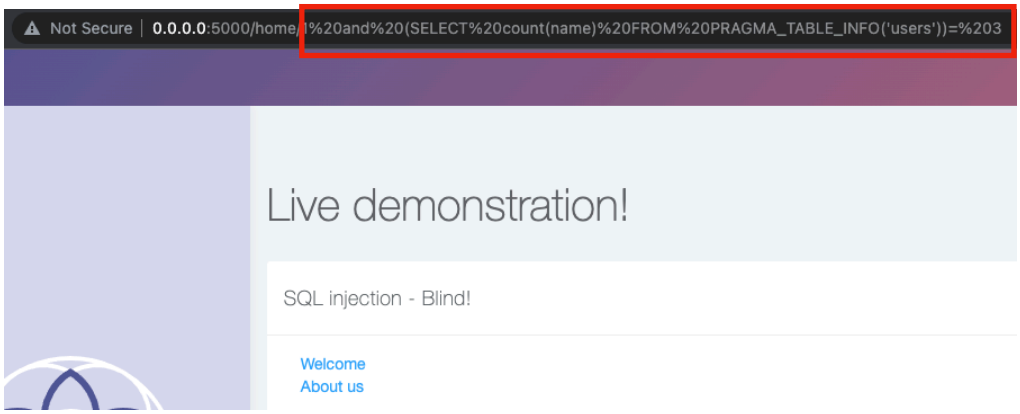
لاستخراج عدد الأعمدة لكل جدول قمنا باستخدام هذه التعليمة

```
and (SELECT count(name) FROM
PRAGMA_TABLE_INFO('tableName'))= FuzzingNumber
```

التعليمة مشابهه في عملها تعليمة إستخراج عدد الجداول ( راجع القسم 4 ) لكن تختلف في اننا قمنا باستخدام PRAGMA_TABLE_INFO() ومررنا اسم الجدول الذي نريد الأستعلام عن عدد الأعمدة الخاصة به، والعائد من التعليمة بين الأقواس () عبارة عن رقم سيتم مقارنته بالـ FuzzingNumber

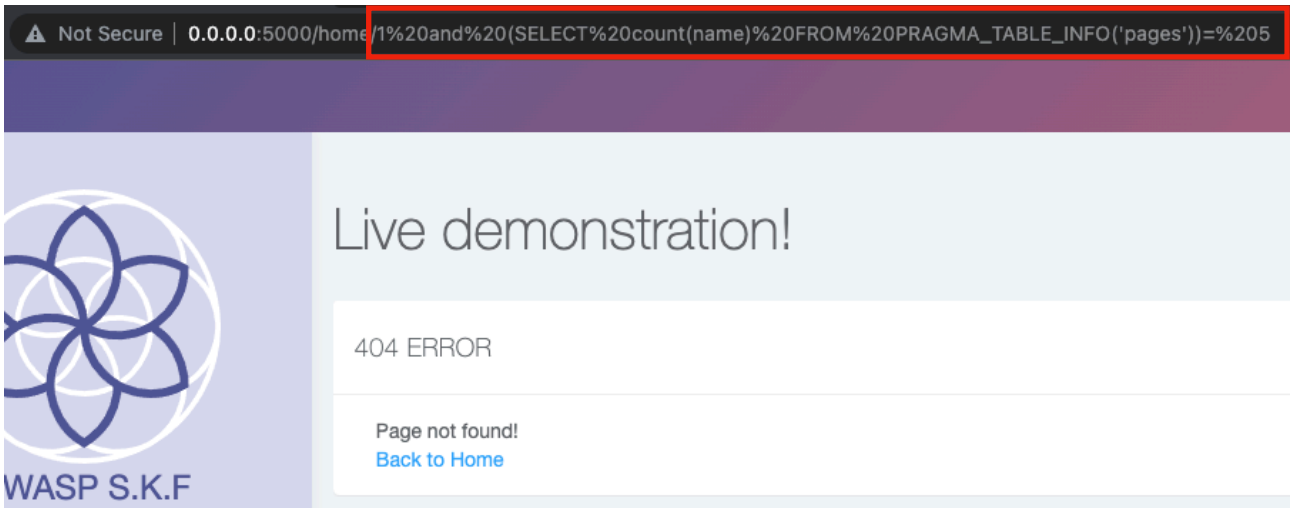
في المثال التالي قمنا بالاستعلام عن عدد الأعمدة للجدول users ( تخمين صحيح )

```
http://0.0.0.0:5000/home/1%20and%20(SELECT%20count(name)
%20FROM%20PRAGMA_TABLE_INFO('users'))=%203
```



وفي هذا المثال قمنا بالاستعلام عن **عدد الأعمدة** للجدول **pages** ( تخمين خاطئ )

```
http://0.0.0.0:5000/home/1%20and%20(SELECT%20count(name)
%20FROM%20PRAGMA_TABLE_INFO('pages'))=%205
```




## 8 Extracting Column Names Length ( غير مكتمل )

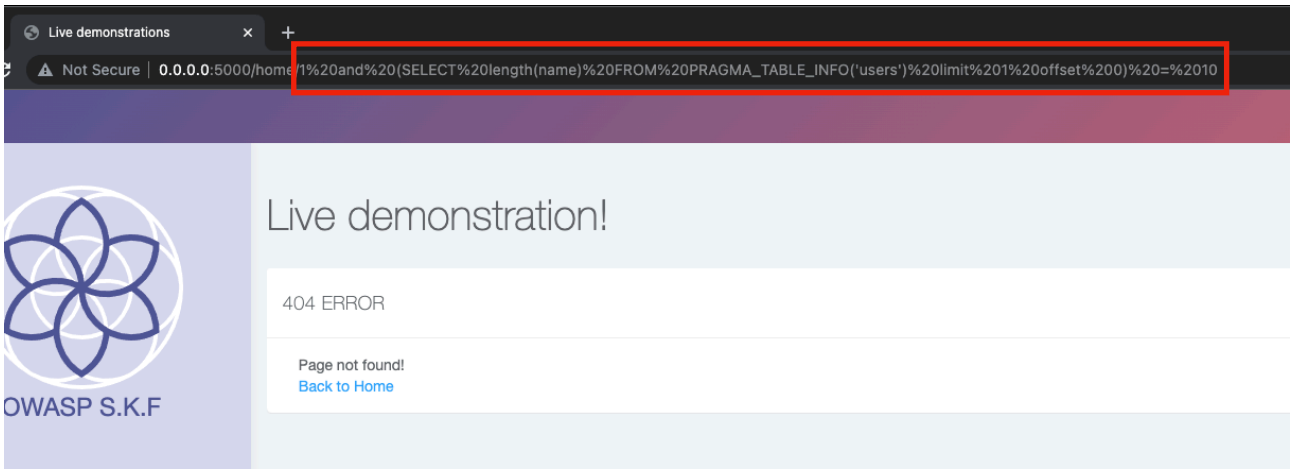
لاستخراج طول اسم العامود استخدمنا التعليمة التالية

```
and (SELECT length(name) FROM
PRAGMA_TABLE_INFO('tableName') limit 1 offset rowOffset) =
FuzzingNumber
```

قمنا بالاعتماد على الدالة length() لعد عدد الحروف لاسم العامود و
PRAGMA_TABLE_INFO() لاستخراج معلومات العمود عبر تمرير **اسم الجدول** الذي ينتمي له العامود، ومن ثم قمنا باستخدام الـ limit للحد من عدد النتائج العائدة ( في حالتنا هنا نريد نتيجة/صف واحد) والـ **offset** للتحرك الى الصف التالي (القيمة التي نمررها هنا للـ offset تمثّل رقم العمود الذي نريد استخراج بياناته)، ومن ثم النتيجة العائدة من التعليمة بين الأقواس () ستكون طول اسم العمود.

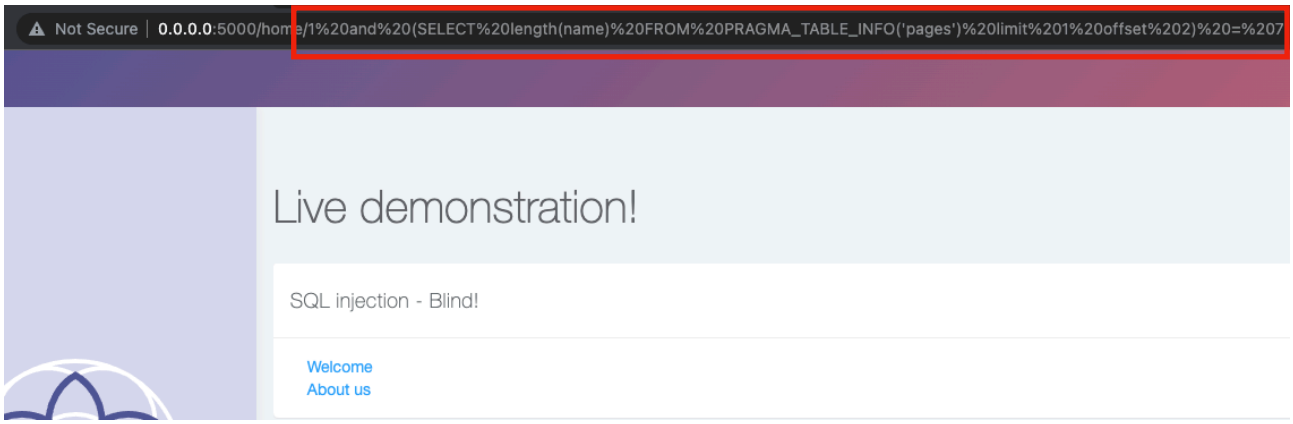
في المثال التالي قمنا بالاستعلام عن **طول** اسم **العامود الأول** في الجدول **users** ( تخمين خاطئ)

```
http://0.0.0.0:5000/home/1%20and%20(SELECT%20length(name)
%20FROM%20PRAGMA_TABLE_INFO('users')
%20limit%201%20offset%200)%20=%2010
```




في المثال التالي قمنا بالاستعلام عن **طول** اسم **العامود الثالث** في الجدول **pages** ( تخمين صحيح )

```
http://0.0.0.0:5000/home/1%20and%20(SELECT%20length(name)
%20FROM%20PRAGMA_TABLE_INFO('pages')
%20limit%201%20offset%202)%20=%207
```




## 9 Extracting Columns Names ( غير مكتمل )

## 10 Extracting Number of Rows ( غير مكتمل )

## 11 Extracting The Length for Each Row Value ( غير مكتمل )

# 12 Extracting Rows Data ( غير مكتمل )

# 13 AutoBlindSQLite.py ( غير مكتمل )

| Function Name | Description |
|---|---|
| **numberOfTablesQuery** | This function builds the SQL query of retrieving the number of tables in the database |
| **numberOfTables** | This function retrieves the number of tables |
| **tableNameLengthQuery** | This function builds the SQL query of retrieving Table Name length |
| **tableNameLength** | This function builds a dictionary (tablesNamesLength) which contains the table number (index) and name length value of that table |
| **tableNameQuery** | This function builds the SQL query of retrieving Table Name |
| **tableName** | This function builds a dictionary (tablesNames) contains the table number (index) and name of that table |
| **numberOfColumnsQuery** | This function builds the SQL query of retrieving the number of columns in the table |
| **numberOfColumns** | This function builds a dictionary (numOfColumns) which contains the table name and the number of columns for that table |
| **columnNameLengthQuery** | This function builds the SQL query of retrieving Column Name length |
| **columnNameLength** | This function builds a dictionary (tableInfo) which contains the table names and number of columns and their name length for each column<br>tableInfo :<br>- key : the key format is : Table:tableName,Column:columnIndex<br>- value : the value is a dictionary which holds the column index (key) and column name length (value) |
| **columnNameQuery** | This function builds the SQL query of retrieving Column Name |
| **columnName** | This function builds a dictionary (columnsNames) which contains the key (table name & column index) and the value (column name) |
| **numberOfRowsQuery** | This function builds the SQL query of counting the number of rows in the table |
| **numberOfRows** | This function retrieves the number of rows for each column in the table |
| **rowDataLengthQuery** | This function builds the SQL query of retrieving the row data length |
| **rowDataLength** | This function retrieves the row data length |
| **rowDataQuery** | This function builds the SQL query of retrieving the row data |

| Function Name | Description |
| --- | --- |
| **BlindSQLiCases** | This function test Blind SQLi cases<br>BlindSQLi Cases in OWASP SKF :<br>1 - "404 ERROR" case: is the false case for BlindSQLi<br>2 - In else case: the application will print the content of welcome page |
| **getTableNameFromKeyValue** | This function extracts the table name from the keyValue |
| **getColumnNameFromKeyValue** | This function extracts the column name from the keyValue |
| **getRowOffsetFromKeyValue** | This function extracts the row offset from the keyValue |
| **getRowData** | This function returns the row Data. |
| **getNumberOfRows** | This function returns the number of rows for the table |
| **printTablesData** | This function prints tables data |

# 14 المراجع ( غير مكتمل )