Website tester documentation

Description of the program:

Usage: requirements are dotnet environment 8.0

Project can be built and started with the included run.bat file for Windows based machines or optionally manually compile the solution file ”**SecureProgrammingProject.sln**”.

There is also included services I used to test the application myself. They require docker to run and they can be found within ”Test services” folder they can be started with the ”StartTestServices.bat” file. Please note that the first service ”Webdev 2” may crash when it is built for the first time. It should just work if it is restarted.
They will start servers listening on localhost:8008 and localhost:8080

The tester itself has 3 main tests that can be run:

Weak password test, injection test and swagger document tester.

When selecting test you can type in for example ”**123**” to start the corresponding tests as instructed in the executable. It will be parsed to run all of the available tests, or just run them partially with mixed combinations of ”**23**” or whatever..

The program then promts other needed parameters for the tests that should be inputed as instructed

for example the server address: ”**http://localhost:8008**” works for tests 1 and 2 while test 3 works on ”**http://localhost:8080/v1**”

Test description and parameters:

Password test:

Test attempts to input combinations of usernames and passwords into the website. Test is deemed as failed if the tester jumps out of the login page. In this case it records the combination it used and resets itself back to the login page.

Parameters will promt to enter a list of passwords. If using the ”run.bat” file the password file is copied to the cwd and can be used as is by leaving the field blank, optinally input a given file.
Path of login portal is the address of the actual login page. For the grueye website this is just ”**login**”
Please note that the server may occasionally freeze for a moment. This is means the server is currently not responding due to high amount of requests it gets. It will reattempt to send the requests in about 10 seconds.

Injection test:
Attempt to insert the alert script into various fields in the website. It is configured to travel trough links it finds to other pages to test the whole website tree. There is however a security measure preventing it from jumping accross domains. Test is deemed as failing if the alert window ever pops up. The parameters are recorded and the alert window is closed.
No extra parameters required

Swagger test:

Reads a given swagger file for example tests and tests if the server is returning expected status codes. Test is deemed as failing if the returned status code is not defined in the swagger file. The tester also tries to add some random path parameters to see if there is undefined behavior with them.

Parameters: path to the swagger file. For my testing I used the file in ” .**Test services\webdev2\backend\server-a\api\swagger.yaml**”

If using the provided test service the address should be ”**http://localhost:8080/v1**”

All of the outputs made by the program are stored as text files in the folder the program was started in. Each file corresponds to the test ran with a timestamp. The records can be found from there.

Structure of the program:

- Main function start in the Program.cs file which ask for which tests to select and the url of the server.

- TestOrchestrator contains functionality for initializing the actual tests and asking more detailed parameters that are required.

- The tests implement the TestInterface which are used to run the tests.

- The base tester is partially abstract class which contains implementains for writing the test reports. The concrete tests are inheriting this class and adding their own actual testing functionality to it.

- The BaseSeleniumDriver is a base for all selenium based tests and contains implementations for initializing the driver.

- Concrete tests: Include CommonPasswordtest, InjectionTest and SwaggerTest. They contain actual logic for the testing tasks. Each concrete test start their functionality from the RunTest method.

- SwaggerParser is used to load a swagger file. currently only supports local files but could be expanded to read a file from the internet for example.

Security concerns faced by the program:

The program has a very limited user interactability but even those inputs made by user are checked in a whitelist basis to be correct. Only prefined values are accepted for further processing.

The validity of URL’s are check by either selenium webdriver or C# default http client implementation depending on the test, In a case where the server can’t be reached the test exits without further without doing anything.

Files are handled with the same permissions as the currently logged in user. So the program can’t be used to steal files from restricted memory spaces. Files are stored with unique identifiers including the name of the test and a timestamp so that they would not overwrite other files.

The file generated by weak password tester may contain sensitive information as it records the weak username and password. Other tests record the inputs that were used during a failing test, which may be used to attack the tested server. Distribution of these files should be handled with caution. As this

is a testing tool made to specifically find this type of sensitive information it is considered intended behavior.

As the part of the program has browser based functionalities it shares same vulnerabilities as Google Chrome. Meaning if the tool is used to connect to a malicious website it could have bad outcomes.

SANS top 25 was used as a checklist to audit the vulnerability of the tester itself. The checklist table can be found at the bottom of this document.

Security Testing:

Program has been mainly tested with the help of the SANS 25 checklist and comparing the code in a whitebox manner. The issues found have been corrected.
Found issues include: path traversal and all possible null pointer refrences have been corrected. There may still be null pointer refrences due to the fact that swagger files are generic and the exact structure is not possible to verify in compile time.

As the program is a web browser it shares vulnerabilities with chrome. These vulnerabilites were found that can't be fixed in the program:

OS-Command injection trough chrome url

- It is technically possible to run programs trough the browser url. But chrome promts the user if they wish to actually run that command. Meaning the program itself can't accept the prompt.
- Possible to recreate by passing in for example "calculator:" as the website address

This same vulnerability is also found with the following SANS 25 vulnerabilities:

- Command injection

There may be possibility to fabricate a swagger file to include malicious requests in the examples to send to the target server. So make sure to verify the swagger files you are testing.

There also was manual blackbox testing inorder to find unexpected behavior all found such cases have been fixed.

Present vulnerabilites:

- Null pointer refrences may be present in SwaggerTest
- Same vulnerabilities as Google Chrome

Suggestions for improvement & further implementations:

- Rewriting the swagger test. The generic type system used in swagger documents suchs as the number type being ambiguous with integer and double is not very flexible when it comes to statically typed language. It resulted in somewhat messy code that could be improved.
- Also tests could be added by simply creating more implementations of the TestInterface

1	CWE-787	Out-of-bounds write	x	Memory is managed by C#, no pointers are used
2	CWE-79	Improper Neutralization of input during Web Page Generation ("Cross-site scripting")	x	Not a web server
3	CWE-89	Improper Neutralization of special Elements used in an SQL command ("SQL Injection")	x	No SQL used
4	CWE-416	Use after free	x	Garbage collection is handled by C#
5	CWE-78	Improper Neutralization of special Elements used in an OS command ("OS command Injection")	/	Possible vulnerability to pass in an OS command as an URL, Chrome prompts the user if they actually want to run it, but the program can't accept itself.
6	CWE-20	Improper Input Validation	x	Inputs are validated
7	CWE-125	Out-of-bounds read	x	Memory is managed by C#, no pointers are used
8	CWE-22	Improper limitation of pathname to restricted directory ('Path traversal')	x	Handled by disallowing parent path traveling. Child folders are allowed
9	CWE-352	Cross-site request forgery (CSRF)	x	Not a web server
10	CWE-434	Unrestricted upload of file with dangerous type	x	Use of standardized file loaders should not allow for code execution from file
11	CWE-862	missing authorization	x	No authorization used
12	CWE-476	Null pointer dereference	/	Most variables are null safe according to visual studio code analysis. However due to generic structure of swagger files this can't always be guaranteed.
13	CWE-287	Improper authentication	x	No authentication used
14	CWE-190	Interger overflow or wraparound	x	There is 1 counter in each test that increments per test run. This counter wrapping only results in the final test result output being messed up.
15	CWE-502	Deserialization of untrusted data	x	Not applicable
16	CWE-77	Improper neutralization of special elements used in a command ("Command injection")	/	Possible vulnerability to pass in an command as an URL, Chrome prompts the user if they actually want to run it, but the program can't accept itself.
17	CWE-119	Improper restrictions of operations within the bound s of a memory buffer	x	Memory is managed by C#
18	CWE-798	Use of hard-coded credentials	x	Not applicable
19	CWE-918	Server-Side request forgery (SSRF)	x	Not a web server
20	CWE-306	Missing authentication for critical function	x	No authentication used
21	CWE-362	Concurrent execution using shared resource with improper synchronization ("Race condition")	x	Concurrency is handled by independed static methods, so no race conditions
22	CWE-269	Improper priviledge management	x	No authorization used
23	CWE-94	Improper control of generation of code ("Code injection")	/	There maybe a way to pass in code in the swagger file examples that would target the server. If they target local machine I assume chrome again prompts the user
24	CWE-863	Incorrect authorization	x	No authorization used
25	CWE-276	Incorrect default permissions	x	Program follows same permissions as the current user