TDT4237 2023, Exercise 2:
Finding vulnerabilities

General information

This exercise is the first group exercise in the course TDT4237 – Software Security and Data Privacy concerning the SecureHelp application. Students must achieve a score of 70% on the exercises to take the exam. This exercise counts for 30% of the exercise workload and must be delivered by the deadline on 28th February 23:59.

The overall task in the exercise is to identify vulnerabilities in the application by performing black-box and white-box testing as specified by The Open Web Application Security Project's (OWASP's) Web Security Testing Guide (WSTG 4.2) in sections 4.2.X -> 4.11.X[1]. You must deliver a list of vulnerabilities, show vulnerable code, and demonstrate exploits.

Any questions regarding the exercise should preferably be asked in the course forum on Blackboard. Alternatively, emails can be sent to TDT4237@idi.ntnu.no.

The target application: Secure help

The application is a web application created to simulate functionality for refugees to find volunteer help services. Certified volunteers provide help services. Administrators handle these certifications to verify the volunteers. A detailed description is provided in the appendix. Additionally, a video will be uploaded to blackboard where one of the TA's describe the application and the code.

The source code is available on Gitlab[2]. Access to the Gitlab repository require being connected to VPN or the campus network and authentication with NTNU credentials. This exercise does not require to run the application locally, but it is possible. Pay attention to the instructions in README to run the application. Be aware that you will be assigned to individual repositories for Exercise 3. Meaning that you are not required to establish a group repository. Neither does this exercise require any changes to the code.

Each group will have their own target instance running on http://molde.idi.ntnu.no:21XXX/, where XXX is your assigned group. For instance, group 9 would use http://molde.idi.ntnu.no:21009/, while group 120 would use http://molde.idi.ntnu.no:21120/. If you haven't enrolled into a group already, then you can do that in Blackboard. Since the targets are running on an internal server it is required to be connected to the campus network or use VPN[3]. Notice that the applications run on a single port, so port scanning will not be helpful.

The task

Students should look for vulnerabilities among those listed in WSTG. For each vulnerability found you should localise the vulnerable code (white-box) and demonstrate an exploit (black-box) to yield

[1] https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/
[2] https://gitlab.stud.idi.ntnu.no/tdt4237/2023/securehelp
[3] https://i.ntnu.no/wiki/-/wiki/English/Install+VPN

the full score of a vulnerability. Each vulnerability in the report should have a corresponding WSTG as far as it is applicable, alternatively, a CVE[4] code or an OWASP top 10[5] category could be applied.

A complete list of 15 vulnerabilities, including white-box and black-box descriptions, would yield a score of 30 points for this exercise (two points for each vulnerability). However, there are more vulnerabilities in the application than 15. If you, for instance, are not able to demonstrate the black-box exploit for all the vulnerabilities you find, then you can compensate by identifying more than 15 vulnerabilities.

The list below indicates the distribution of vulnerabilities among the different test categories in WSTG. It shows that the vulnerabilities are distributed among all categories and where one might want to spend the most time. Be aware that some vulnerabilities can entail more than a single tests, and that the sum of vulnerabilities in categories is larger than the total sum of vulnerabilities. The most straightforward approach to this exercise is to go through every single test in WSTG. However, do not get stuck with as single tests if it doesn't yield any result.

Distribution of vulnerabilities in the WSTG categories:

- WSTG-IDNT: 3
- WSTG-ATHN: 6
- WSTG-ATHZ: 2
- WSTG-SESS: 4
- WSTG-INPV: 3
- WSTG-ERRH: 2
- WSTG-CRYP: 2
- WSTG-CLNT: 4
- WSTG-BUSL: 2
- WSTG-CONF: 2

White-box

Students should review the source code of the application and report the vulnerabilities they find. You should explain which file(s) and line(s) of code that are vulnerable and why. *Any unique line(s) of vulnerable code will be counted as a single vulnerability.* For instance, an input field vulnerable to HTML injection will usually be vulnerable to CSS injection as well. However, the vulnerable code is still the same and would only be counted as a single vulnerability. On the other hand, if you find two separate input fields vulnerable to HTML injection, those will be counted individually.

In other cases, a vulnerability might not be related to specific line(s) of code, and the vulnerability exists merely because of the lack of implemented security features. For instance, if the application does not encrypt sensitive data in general, then that would be one single vulnerability. Pointing out that every single endpoint sends or receives unencrypted data does not mean that you have found more than one vulnerability. In the cases where a vulnerability exists because of the lack of implemented security features, then you should report which file(s) where this should have been included.

Black-box

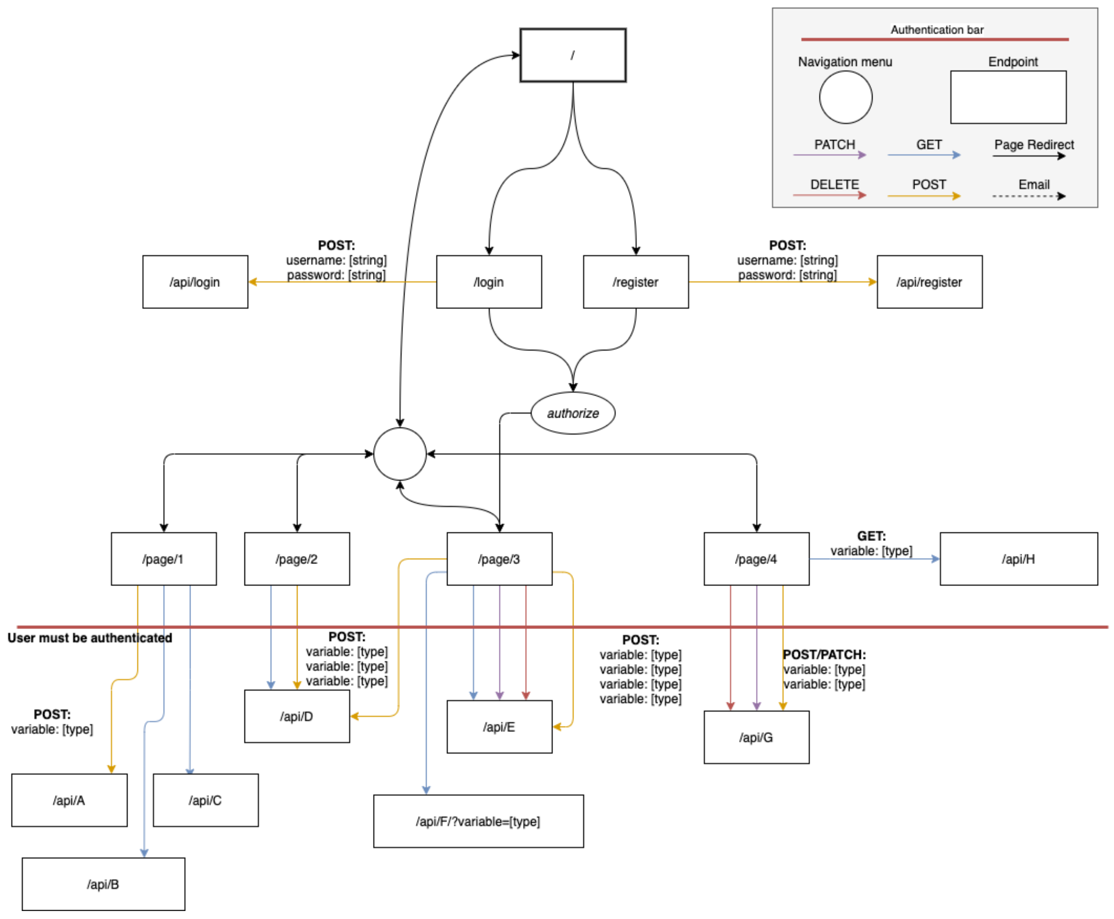
For each vulnerability you are expected to demonstrate an exploit. You should explain step-by-step how the exploit can be conducted and provide screenshots to prove that you were able to carry out

[4] https://www.cvedetails.com/
[5] https://owasp.org/Top10/

the exploit. Any tools used in the process must be documented. If an exploit would require substantial resources, social engineering, or other things not practical in the exercise, then you should explain how the exploit could be conducted in theory. However, in most cases, you are expected to perform an exploit.

Recommendations

To support your search for vulnerabilities it is recommended to perform information gathering activities[6] before and during the search. It is valuable to map the behaviour of the application: Which frameworks are used in the app, which apps are running, what are the execution paths, what are the parameters, and which actions require authentication? A page map can be drawn to support your search and could help identify endpoints with bad authentication (category: WSTG-ATHN). The figure below shows how a page map can be drawn using www.draw.io. The file is provided in Blackboard as a template for those that wish to make a page map.



Getting an overview of the application before going in depth in testing can be helpful. It is recommended to perform information gathering, such as mapping execution paths[7] before and during testing. A simple page map could reveal whether the actual functionality of the application matches the description. For instance, are roles and privileges implemented as they are described?

Furthermore, the procedure to find all the application vulnerabilities is to go through every single test case in WSTG. It is recommended that you acquire an overview of the different tests before

[6] https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/README

[7] https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/07-Map_Execution_Paths_Through_Application

going into depth on every one of them. At first, you might find some of the more trivial tests and look for low-hanging fruits in terms of vulnerabilities. Additionally, do not get stuck performing a single test without making progress for too long. Chances are that there is no vulnerability present corresponding to that particular test.

Report and evaluation

You should create a report based on the **template** provided in blackboard. The template can be imported to Overleaf[8] where group members can edit the same document simultaneously. The maximum achievable points are 30, and the content of the report and point distribution is as follows:

1. Vulnerabilities (2 points for each vulnerability):
   a. White-box: Locate the vulnerability (1 points)
   b. Black-box: Demonstrate an exploit (1 points)

Presentation

The report will be evaluated by its presentation. Failure to meet the following requirements might lead to a reduction of the score, up to 1 point. Points are rarely deducted on this post, but we need some minimal formal requirements on the report's content. Those are:

- Language in a proper and academic style
- Reference list formatted according to the IEEE citation style
- The precision of facts
- A logical structure of content

Tools

You are allowed to use almost any tool when interacting with the hosted target, *except* for aggressive brute force scanners such as OWASP ZAP's active scanner (AKA automated scan). However, you can do whatever you want when running the target locally. Recommended tools are:

- Firefox browser - https://www.mozilla.org/en-US/firefox/
- OWASP ZAP - https://www.zaproxy.org/
- Postman - https://www.postman.com/

Rules of engagement

- Mainly use your student email when authenticating for the test application. NTNU's postmasters will not be happy if they see a lot of "junk" mail spam to external email servers.
- Hack non-destructively, don't try to shut down the application or server.
- Don't interact with the targets belonging to other groups.
- Denial of service style brute force attacks should not be conducted against the server.
- Plagiarism will not be tolerated

Failure to obey these rules may result in a penalty.

[8] https://www.overleaf.com

TDT4237-2023 Vulnerable Application

SecureHelp

Abstract description

SecureHelp offers a secure solution for administering volunteers and refugees to manage

- Medical Services
- Transport Services
- Food Services
- Shelter Services

Refugees can register that they need one or more of these services, and then volunteers with adequate competence can use SecureHelp to find refugees they can help.

To ensure that people with dishonest intentions do not abuse the system to contact the refugees, an administrator must approve of a registered volunteer's competence. Volunteers can apply for competence verification in the application.

Technology

- Back-end: Django 3 with Django Rest Framework
- Front-end: HTML5/CSS/JS/React/NGINX

Roles

A user can be one of the roles given below:

- **Refugee**
  - Refugees need help with one or more services and can apply for this help in the application.
- **Volunteer**
  - Volunteers can apply for competence verification for the 4 different services the refugee camp offers. When the competence is verified by an administrator, volunteers can see the help requests from the refugees, and can assign a help request to themselves. When the help is done, the volunteer marks the request is as finished.
- **Administrator**
  - Administrators can approve competence verification requests from volunteers. One cannot register as an admin in the application. To become an admin, you must get an existing admin to create a user for you or ask a developer to create one.

Functional requirements

ID	Title	Role(s)	Description
FR1	Register user	Volunteer, Refugee	A person without an account can register as a new user. You must register as either a refugee or a volunteer. After registering, the user must confirm their email before they can log in and start using the application. The following attributes are required when registering: - Email - Username - Password - Account type (refugee or volunteer)
FR2	Log in	All	A registered user can log in to the application.
FR3	Apply for competence certification	Volunteer	Volunteers can apply for verification for competence of one or more of the 4 services (medical, transport, food, shelter)
FR5	Publish a request for help	Refugee	Refugees can publish requests for help with one or more of the 4 services (medical, transport, food, shelter)
FR6	View help requests	Volunteer	Volunteers can see the help requests from the refugees but are only allowed to see requests for services they have an approval for.
FR6	View own help request	Refugee	Refugees can view their own, and only their own help requests.
FR7	Accept help request	Volunteer	Volunteers can accept the help requests from the refugees. A help request can only have one assigned volunteer.
FR8	Mark help request as finished	Volunteer	Volunteers can mark an active accepted help request as finished after they have carried out the work.
FR9	View their own competence verification request	Volunteer	Volunteers can view their own, and only their own competence verification requests.

FR10	View all pending competence verification requests	Administrator	Admins can view all competence approval requests.
FR11	Approve/Deny competence requests	Administrator	Admins can approve or deny requests for competence verification.
FR12	Delete own help request	Refugee	Refugees can delete their own, and only their own help requests.
FR13	Delete their own competence verification request	Volunteer	Volunteers can delete their own, and only their own competence verification requests.
FR14	Upload documents about themselves	Refugee / Volunteer	Refugees and Volunteers can upload PDF documents with info about themselves like passports and certifications.
FR15	View their own documents	Refugee / Volunteer	Refugees and Volunteers can view/download their own PDF documents.
FR16	View documents for a refugee they are helping	Volunteer	Volunteers can view/download PDF documents which are owned by a refugee they are currently helping or have helped before.
FR17	Delete own documents	Refugee / Volunteer	Refugees and Volunteers users can delete their own uploaded documents
FR18	Reset password	All	If a user has forgotten the password, a reset link can be sent by email.