WebSec21 – Project Report

Eric Armbruster, Florian Freund, *Team btw we use Arch*

1 Introduction

In the following we will describe the outcome of our four weeks project phase of the Web Application Security course. During this project phase we analyzed selected components of the *Artemis* interactive learning platform [2]. We mainly put our focus on the file upload and download mechanisms, the markdown editor and the implementation of access rights on different endpoints.

In order to carry out this analysis, we mostly relied on source code inspection via techniques offered by the IntelliJ IDE (e.g. Find Usages, Debugger), but also employed the developer tools of our browsers.

The remainder of this report is structured as follows: first, an overview of the investigations carried out for different components is provided in Section 2. Afterwards, vulnerability and weakness findings are presented and discussed in Section 3 and in Section 4. Next, bugs we encountered in Artemis are shortly described in Section 5. Finally, the report is concluded in Section 6.

2 Investigations

2.1 Markdown Editor and HTML Text Input Fields

Artemis offers numerous input fields that allow for the insertion of formatted text via a Markdown editor. Internally, the application first converts the formatted text into HTML and then embeds it into the page that is served to the user. Naturally this feature demands a deep security inspection, as it could be an easy entry point for stored XSS attacks [4] by malicious students or teaching assistants. For analyzing this component we mainly relied on a close inspection of the source code, but also crafted several XSS payloads and inserted them into various text fields.

Artemis relies on the DOMPurify [6] npm package for XSS sanitizing the *output* of the Markdown to HTML converter. As this library has become an almost de facto standard for this kind of sanitization, it is considered safe by us, however, one needs to ensure its correct usage regardless. Also, Artemis uses version 2.2.9 of DomPurify, which is the most recent version, and is not known to have any security vulnerabilities [7].

The main points of interest thus lie *inside* the `htmlForMarkdown` method, where we need to ensure that the sanitizer is called correctly, and at the *call sites of this method*, as potentially unsanitized HTML or Markdown could be concatenated after the sanitization process. The method itself performs first the conversion from Markdown to HTML and afterwards calls `DOMPurify.sanitize`. As explained in the DOMPurify documentation [6], this call must be last and afterwards no modifications to the HTML must be made, which is fulfilled here.

As the purpose of the Markdown editor is to allow formatted text, the call to `DOMPurify.sanitize` is handed a list of whitelisted HTML tags and attributes. For instance, the editor allows the inclusion of links via the 'a' tag. As we are not familiar with DOMPurify, we checked manually whether this would allow the execution of Javascript inside of the 'a' tag with the href field (e.g. with the following payload: `<a href="javascript:javascript:alert(1)">test</a>` [5]), but this only prevents DOMPurify from removing all 'a' tags, and *does not exclude* href fields from sanitization.

Afterwards, we searched via the IDE for direct and indirect usages of `htmlForMarkdown`, mostly these are either calls to `safeHtmlForMarkdown` or calls to the `transform` method of the pipe `HtmlForMarkdownPipe`. We then checked on these call sites whether modifications of the HTML are done after the sanitization. Additionally, we looked for input fields that have been overlooked regarding sanitization, but could not find any.

To summarize, the output sanitization via DOMPurify seems to be implemented correctly and thoroughly. We could not find any security vulnerabilities within this functionality. Also, we want to highlight that Artemis employs the secure by default principle here by not allowing any HTML tags and attributes when the whitelist is left undefined.

This analysis was carried out by both team members

2.2 File Upload

File upload is an essential functionality in Artemis that lets students upload their solutions to exams as well as exercises. Before carrying out analysis on this component, we constructed a bullet-point list containing ideas of which kind of vulnerabilities we expect and which kind of functionality would have to be inspected closer if found:

- Are file paths secured against directory traversal?
- File overwriting/leaking/creating
- What happens if files with wrong file endings are uploaded?
- If hashing is done, is it done securely?
- Can you submit late?

In the rest of this section, we will discuss some of the code areas that initially caught our attention, but turned out to be entirely secure.

Hashing Artemis actually uses file hashing when a submitted file is saved on the server in the `save` method in the `FileUploadSubmissionService.java`. There it employs the insecure MD5 hashing function, however, but only to exclude errors during transmission and copying of the file. The hash is therefore only used to validate the integrity of the *stored* files and is not security critical.

Directory Traversal via URL Many endpoints in `FileResource.java` allow arbitrary file names directly in the URL. We expected to find these endpoints vulnerable to a path traversal attack there. However, when trying to exploit these endpoints, we get HTTP 400 responses because the Spring security features block various security relevant characters e.g. encoded slashes in URI path variables by default. Without the possibility to inject slashes into those filenames, path traversal is not possible.

Wrong File Endings Uploading files with wrong file endings is possible (e.g. when submitting a file exercise), as only the file endings are checked and not the mime type. We hoped this could be used in combination with the XSS vulnerability described in Section 3.2 such that you upload a file containing HTML but with a png file ending (which would extend the vulnerability in Section 3.2 to arbitrary files). However, browsers apparently use file endings to determine the file type, so this was not fruitful.

File Overwrite See Section 3.3

This analysis was carried out by both team members

2.3 Investigated Modules

Additionally to the modules mentioned above, the following modules were investigated (Including the resource classes, their service counterparts and the main object classes):

1. Assessment
2. Attachment
3. Complaint
4. Course
5. File*
6. Notification
7. SystemNotification

3 Vulnerabilities Found

In total, we found four security vulnerabilities and extended the scope of another vulnerability found by a different team.

3.1 Access Rights

Type Missing access rights checks

Status Unresolved, no response by Artemis team

Introduction Almost all endpoints in Artemis need to verify that the user is authenticated and has the required access rights to perform a particular action, e.g. a new course may only be created if that user is authenticated as admin. The access rights in Artemis are checked in a two-step verification process. First, each endpoint needs to check whether he or she has been assigned the role required to access this endpoint. Artemis internally employs a hierarchical Role Based Permission System (RBAC) for that purpose. The details about the defined roles and permissions are described in the documentation [1]. In code checking the role is done via the `@PreAuthorize(hasRole(<ROLE>))` Spring annotation. The `@PreAuthorize` annotation is executed before the annotated method is executed and allows the execution of the annotated method only if `hasRole(<ROLE>)` returns `true`. The second verification step checks whether an authenticated user should have access to a particular resource, e.g. a user with the role teaching assistant should only be allowed to assess exercises of lectures where he or she actually is a teaching assistant, not simply for all lectures. Artemis handles this by storing whether a user has the right to access a particular resource (lecture, exercise, ...) in its database and then retrieving these permissions upon the execution of a particular method.

Description Generally speaking, for an endpoint to correctly verify access rights, both of these checks must be performed. There are of course exceptions, for instance if an endpoint may only be accessed with admin rights, the second step may as well be omitted. From our perspective, we need to verify that these checks are done on each endpoint and that the correct checks are performed.

The lecture attachment module is missing such checks on most endpoints. The affected endpoints:

- `createAttachment` in `AttachmentResource.java`
- `updateAttachment` in `AttachmentResource.java`
- `getAttachment` in `AttachmentResource.java`
- `getAttachmentsForLecture` in `AttachmentResource.java`

For all of them, the `@PreAuthorize(hasRole(<ROLE>))` Spring annotation is present to check the user role, but an actual access right check is missing.

Impact The impact of this vulnerability is estimated to be of low severity, because the presented endpoints can only be accessed by users with the EDITOR role. Those are only course organizers and therefore (usually) trustworthy. Nevertheless, those can overwrite and add any files in others courses.

Workarounds and Fixes The `deleteAttachment` methods in the same class implements access control as expected. Access is limited to course instructors. This approach should be adopted in the other methods to fix the issue.

3.2 XSS on File Download

Disclaimer: This vulnerability was found by team "Team 9 from Outer Space", our contribution is only the extension of the vulnerability scope and the hot fix suggestion via Content-Disposition.

Type XSS

Status Fixed

Description Original vulnerability: lecture attachments are vulnerable to XSS attacks by first uploading a malicious SVG file that contains Javascript, then on download the Javascript is executed. We extended the scope of this vulnerability to the file upload exercise mechanism and to HTML. To be more precise, when a file upload exercise is set to accept SVG or HTML, the download of previously uploaded files is vulnerable to XSS. For instance, the following basic payload works as a proof of concept for HTML based XSS:

```
1 <html>
2 <script>alert("test")</script>
3 </html>
```

Listing 1: Basic HTML XSS for Proof of Concept

Impact The impact of this vulnerability is estimated to be of medium severity, as SVG or HTML must be explicitly allowed as a file type for submission. However, this is unlikely to be found anywhere in production, as Artemis targets mostly university courses for Computer Science students, which means courses typically do not use HTML and the creation of an exercise consisting of a single HTML file is found to be rather unlikely. Despite this unlikely scenario, the potential consequences are severe. When an admin or instructor presses download on such a maliciously prepared file, any actions on the Artemis platform that are within this user's permissions could be performed.

Workarounds and Fixes As an immediate workaround, file upload exercises that allow SVG or HTML should be disabled. The Artemis development team can easily fix this by setting `Content-Disposition: attachment` instead of `Content-Disposition: inline` in the HTTP response header. In code, the relevant method is `buildFileResponse` in `FileResource.java`. Setting it to `attachment` forces the user to download the file instead of viewing it inline in the browser [3], thus preventing the XSS. Since 07.07.2021: We confirm that the Content-Disposition fix resolves the problem.

3.3 Limited File Overwrite

Type file delete, file overwrite and file creation

Status Unresolved, no response by Artemis team

Description The request body of the endpoints `createAttachment` and `updateAttachment` in `AttachmentResource.java` allows setting a file path via the JSON attribute *link* of the `Attachment` object. The actual file upload and file save that needs to be done when creating or updating an attachment is handled via the `saveFile` endpoint in `FileResource.java`. Among under methods `manageFilesForUpdatedFilePath` is called to actually store the file on disk. Particularly interesting is that a file path parameter called `newFilePath` is passed that is taken from the *link* JSON API value and can be controlled by an attacker without any sanitization.

The code below is an extract of the most important part of `manageFilesForUpdatedFilePath`. The problem is that `generateTargetFile` is called with a *file path* and not as the parameter name of said method implies with a *file name*. Moreover, the `generateTargetFile` method performs no sanitization if `keepFileName` is set to true, which is fulfilled at least for the attachment endpoints listed above. Consequently, this can be used to make `generateTargetFile` create the `targetFile` with *empty contents* at an *arbitrary location* that *overwrites* the file previously saved at this location. Because of line 4 and 9 below, we can also fill the file with arbitrary contents:

```
1 public String manageFilesForUpdatedFilePath(..., String
     newFilePath, String targetFolder, ...) {
2   ...
3     // Sets 'source' to a previously by the attacker
        uploaded file
4     Path source = Paths.get(actualPathForPublicPath(
        newFilePath));
5     // Create/truncate file at arbitrary location
6     // and inject arbitrary path into 'targetFile'
7     File targetFile = generateTargetFile(newFilePath,
        targetFolder, keepFileName);
8     // Overwrite a file at an arbitrary path
```

```
 9        Files.move(source, targetFile.toPath(),
             REPLACE_EXISTING);
10       ...
11   }
```

Listing 2: Extract of `manageFilesForUpdatedFilePath` method

Impact The impact of this vulnerability is estimated to be medium-to-low, as at minimum the role *editor* is required to exploit it. Furthermore, it is limited by the requirements for file uploads set in the `handleSaveFile` method in `FileResource.java`. This method ensures that file names have one of the following file endings: png, jpg, jpeg, svg, pdf, zip. Also, the uploaded files must adhere to the file size limit set in `application.yml`, which is by default 10 MB.

Workarounds and Fixes The main problem here is that the file upload mechanism first stores the file in a temporary path and then a second user request is expected to make the server move the file into the final directory. In order for this to work, the link attribute must contain a path to the temporary file. But this path can be set utterly independent of the path of the actually uploaded file. We do not provide suggestion for a fix, as this requires a change in the design of the upload mechanism and this most likely would require some larger code changes.

In the following we discuss ways to fix the problem without major code changes.

The underlying problem is that the method `generateTargetFile(newFilePath, ...)` is called in `manageFilesForUpdatedFilePath` with a *file path* and not as the parameter of `generateTargetFile` implies with a *file name*. The best solution probably would be to ensure it is only called with the file name. This could be done with the following code that is already called in `actualPathForPublicPath`:

```
1  String filename = newFilePath.substring(
       newFilePath.lastIndexOf("/") + 1);
```

Listing 3: Retrieve File Name from File Path

It is debatable, whether this call should be done in `generateTargetFile` or in `manageFilesForUpdatedFilePath`. In case it is done in the latter one needs to keep in mind that this bug could reappear again as soon as a different method calls `generateTargetFile` with an attacker controlled path as first argument. However, if this is solved in `generateTargetFile` it would leave some code lines in `manageFilesForUpdatedFilePath` unprotected, which could lead to problems in the future as well. Another solution to consider is to call it in both methods.

A different approach that was already employed in some other places in Artemis would be to call `removeIllegalCharacters` in the `manageFilesForUpdatedFilePath` method or in all endpoints that accept a file path. The method sanitizes file paths by removing all "." and "/" characters from the argument it is handed.

3.4 Arbitrary File and Folder Deletion

Type file and folder deletion

Status Unresolved, no response by Artemis team

Affected Endpoints The following endpoints are vulnerable to the file and folder deletion attack:

- `updateAttachment` and `deleteAttachment` in `AttachmentResource.java`
- `updateCourse` and `deleteCourse` in `CourseResource.java`
- `updateQuizExercise` and `deleteQuizExercise` in `QuizExerciseResource.java`

Description The method `manageFilesForUpdatedFilePath` is also vulnerable to file deletion. The `oldFilePath` parameter in said method is attacker controlled when one of the affected endpoints listed above is called. The problem is that `manageFilesForUpdatedFilePath` calls `FileSystemUtils.deleteRecursively(oldFile)`, where `oldFile` is the corresponding `File` object to the attacker controlled `Path oldFilePath`. As there is no check whether the `oldFilePath` corresponds to a file (and not a directory), we can also utilize this call to delete arbitrary folders recursively.

On a side note, initially it looked like the method `actualPathForPublicPath`, which is called before the delete call, would sanitize this path, because it internally executes `publicPath.substring(publicPath.lastIndexOf("/") + 1);`. However, later in that function the unsanitized `publicPath` is concatenated again to the returned path if it also contains the string *files/attachments/attachment-unit* (some other strings work as well).

In order to exploit this, create a lecture attachment. Edit the attachment and set the *link* JSON attribute to something like *files/attachments/attachment-unit/../../../../../../../<path to delete>*. Then edit the attachment again and set a different *link* (exact value does not matter), this time the deletion will be executed, as the *previousLink* will be placed in *oldFilePath*. Please note, the edits *must not be done via the UI*, as it sends requests to multiple endpoints, instead ensure only the request to the `updateAttachment` endpoint is sent. Instead of a second edit, one can also delete the attachment.

Impact This vulnerability is estimated to be of low-to-medium severity, as a recursive folder deletion could be abused to destroy the system or delete uploaded files by disliked students. However, editor rights are required, thus we think it is rather unlikely to be exploited.

Workarounds and Fixes The main problem here is the same as discussed in the Workarounds and Fixes paragraph in section 3.3.

But this issue can be fixed without major code changes as well.

Firstly, this should be prevented by calling `removeIllegalCharacters` on `oldFilePath` in `manageFilesForUpdatedFilePath`. Furthermore, the intention behind the call `FileSystemUtils.deleteRecursively(oldFile)` is to delete a single file, thus `Files.delete(path)` or another method that only deletes a single file should be called.

3.5 Arbitrary Notification

Type Unrestricted notification (including spoofing another identity but excluding system notifications) reading, creation, modification and deletion

Status Unresolved, no response by Artemis team

Affected Endpoints The following endpoints are vulnerable:

- `createNotification` and `deleteAttachment` in `NotificationResource.java`
- `updateNotification` and `deleteCourse` in `NotificationResource.java`
- `deleteNotification` and `deleteQuizExercise` in `NotificationResource.java`
- `getNotification` and `deleteQuizExercise` in `NotificationResource.java`

Note: System notifications are correctly restricted to the admin user.

Description Course instructors may want to create notifications for their students for announcements. To do so, Artemis has a notification system that enables instructors to create notifications for one specific user or for a whole group of users. This notification is then shown to the target users when they visit Artemis the next time.

The recipient can check the notification author user to verify the validity of the content and there is the first vulnerability. The `createNotification` endpoint allows to create notifications with any recipient and with any author. There is no check whether the provided author is actually the current user. It is the same for the `updateNotification` endpoint. Any instructor can edit the notifications of everyone else. The next issue is in `getNotification`. Any instructor can leak the content of all notifications. To do so, the notification IDs must be leaked or just simply guessed. That is possible, because the new notification IDs are only incremented and therefore predictable. Finally, in `deleteNotification` any instructor can delete any notifications.

Impact This vulnerability is estimated to be of low-to-medium severity, as instructor rights are required for all the endpoints, thus we think it is rather unlikely to be exploited.

Workarounds and Fixes To fix this issue, only two code lines are required. First add the following code line into each of the endpoints:

```
User currentUser = userRepository.getUserWithGroupsAndAuthorities();
```

Then add a check that the current user is not spoofing the identity of another user:

```
if (currentUser != notification.author) throw ...
```

4 Weaknesses Found

In this section we describe weaknesses found during our analysis of Artemis. This differs from previous sections in that weaknesses, unlike vulnerabilities, are not immediately exploitable, but pose a threat to application security in the long term. In total, we have found one weakness.

4.1 Password Sharing with Third-Party Platforms

While other teams have already stated that passwords are stored insufficiently secure within the Artemis database, we want to emphasize that Artemis' handling of passwords towards third-party platforms is not only convenient but also negligent.

Our reasoning for this is as follows: The user management components in Artemis for GitLab and Jenkins feature methods (`getCreateUserFormHttpEntity` in `JenkinsUserManagementService.java` and `updateBasicUserInformation` in `GitLabUserManagementService.java`) that automatically create accounts on these platforms. The problem is that these methods generously share credentials that are used university wide for logins by students as well as teaching staff. We argue that this is problematic as it increases the potential attack surface of not only the Artemis platform but also the attack surface of universities or other institutions relying on Artemis. Furthermore, storing the credentials at all in Artemis counters the idea of having a central identity provider (IDP).

To counteract the risks described, we strongly recommend the Artemis team to stop storing any passwords at all for institutions that provide an IDP service. Since this feature of automatic account creation on third-party platforms is probably also needed in the future, we are suggesting to create accounts with randomly generated passwords, notify the users about these via their emails and prompt them to change their passwords immediately after login. Gitlab and Jenkins might also offer an API for creating temporary passwords that automatically prompts their users to change their passwords on first login. This would certainly be less convenient for users, but also more secure and more appropriate given the consequences that the leak of these credentials could have in extreme cases.

Another potential solution for the Artemis instance at TUM (and at other institutions) could be the use of LDAP. This is directly supported by GitLab [8] and also supported by Jenkins [9], though in the form of a plugin. Currently, this authentication method is already in use with the GitLab instance hosted by LRZ at *gitlab.lrz.de* and would offer the same convenience as the original solution.

5 Bugs

This section shows found bugs that have no impact on security.

5.1 Archive

The API endpoint method `archiveCourse` in `CourseResource.java` can be used to archive courses after they ended. The end date of a course can be unset and therefore can be NULL. The method `archiveCourse` uses the end date but is missing a NULL check.

```
1 public ResponseEntity archiveCourse(Long courseId) {
2   ...
3   if (now().isBefore(course.getEndDate())) {
4     throw new BadRequestAlertException(...);
5   }
6   ...
7 }
```

Listing 4: Potential NullPointerException

Impact This bug has no impact on security, because Spring handles the exception.

Workarounds and Fixes Adding a simple NULL check before use is sufficient.

```
1 public ResponseEntity archiveCourse(Long courseId) {
2   ...
3   if (course.getEndDate() == null || now().isBefore(
       course.getEndDate())) {
4     throw new BadRequestAlertException("End date must be
        set" ...);
5   }
6   ...
7 }
```

Listing 5: Potential NullPointerException

6 Conclusion

To summarize, we have scrutinized several Artemis components during this project phase and have found four vulnerabilities of low-to-medium security. Furthermore, we have learned to establish at least a few structures to make analyzing large projects together (and alone) easier, more efficient and more fruitful. Such as good communication about who does what and to take a moment before the analysis to think about what kind of security bugs you expect to see in a particular component, which helps priming the brain for the upcoming task.

About this course we think that it is a great opportunity to get a deep introduction into security vulnerabilities of web applications. Moreover, we have learned a lot not only about vulnerabilities, but also about some quirks in certain great — or not so great — programming languages for the web, like Javascript and PHP. We hope to see this course also being offered for future students interested in this topic!

References

[1] "Artemis access rights." (Jun. 26, 2021), [Online]. Available: https://docs.artemis.ase.in.tum.de/admin/accessRights/.

[2] "Artemis: Interactive learning with individual feedback." (Jul. 1, 2021), [Online]. Available: https://github.com/ls1intum/Artemis.

[3] "Content-disposition." (Jul. 1, 2021), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Disposition.

[4] "Cross site scripting (xss)." (Jun. 25, 2021), [Online]. Available: https://owasp.org/www-community/attacks/xss/.

[5] "Cross site scripting (xss) vulnerability payload lists." (Jun. 25, 2021), [Online]. Available: https://github.com/payloadbox/xss-payload-list.

[6] "Dompurify." (Jun. 25, 2021), [Online]. Available: https://github.com/cure53/DOMPurify.

[7] "Dompurify vulnerabilities." (Jun. 25, 2021), [Online]. Available: https://snyk.io/vuln/npm:dompurify.

[8] "Gitlab ldap setup." (Jul. 1, 2021), [Online]. Available: https://docs.gitlab.com/ee/administration/auth/ldap/.

[9] "Jenkins ldap plugin." (Jul. 1, 2021), [Online]. Available: https://plugins.jenkins.io/ldap/#documentation.