RUB

RUHR-UNIVERSITÄT BOCHUM

Automated Security Analysis of Unauthorized Access in Real-World REST APIs

Christoph Heine

Exposee – July 17, 2021.
Chair for Network and Data Security.

Supervisor: Prof. Dr. Jörg Schwenk
Advisor: M. Sc. Louis Jannett
Advisor: Dr.-Ing. Christian Mainka
Advisor: Dr.-Ing. Vladislav Mladenov

hgi Lehrstuhl für Netz- und Datensicherheit

1 Introduction

Representational state transfer (REST) describes an architecture pattern for designing web service interfaces. REST promotes the usage of widespread web technologies and standards to retrieve resources and execute actions, in particular HTTP and URI. Its core paradigms are a client-server architecture, statelessness, cacheability, interface uniformity, layered systems, and code-on-demand [4].

The properties of REST APIs offer scalability and ease of use for server- and client-side applications. In addition to this, API implementations may offer machine-readable descriptions, which further emphasizes the automatability of the pattern [11]. As such, it is no surprise that RESTful API design has gained popularity, especially as interfaces for public web services.

1.1 Motivation

While general web service security has been a subject of various research topics, the concept of dedicated REST security is relatively new [8]. A major difference to browser-based web services is that REST APIs are inherently stateless. Thus, no session management is employed. Any individual request to a non-public endpoint must be authenticated. As a consequence, secure authentication and access control become even more relevant in the context of REST security.

REST is not a defined standard but an architectural style that consists of many different components, which makes the involved security risks difficult to determine. Its security relies on the safety of the involved transport security and authentication standards, namely HTTPS, OAuth, and SAML. On their own, these standards have already been extensively investigated with clearly defined attacker models. For REST itself, no definitive threat models have been established [8, p. 2]. However, the unique characteristics of REST may necessitate adjusted or new security requirements.

User resource and action management are another important factor in REST security [3, pp. 389]. A service's REST API often exposes all its functionality directly via the provided endpoints. The service must not only check if a request is authenticated, but also ensure that the authenticating client has the required permissions to retrieve

a resource or execute an action. Naturally, this stresses the need for properly defined, separated access levels.

These observations will be the starting point for our analysis. Our goal is to create a dedicated definition of REST security in the context of unauthorized access, taking into account the unique properties of RESTful architecture. To do this, we will implement an automated pentesting tool to survey real-world APIs of popular web service providers. Using the security definition, we identify and evaluate security problems in practical implementations of REST web services.

1.2 Related Work

An examination of REST security as a dedicated research topic was provided by Lo Iacono et al. [8]. The authors acknowledged the necessity for a general security framework that is specific to REST architecture. Furthermore, they identified the abstractness of REST as a concept as one of the challenges for adapting web service security paradigms for REST security. In their conclusion, the authors stressed the need for „message-oriented security mechanisms“[8, p. 29] in addition to providing authentication security.

Authentication protocols used in RESTful APIs have been well-researched. Best practices and threat models for these protocols are incorporated into their official documentation, for example, for HTTP Authentication [5], OAuth 2.0 [9], and SAML 2.0 [6]. These would have to be re-evaluated with REST security in mind.

OWASP provides a general guideline of REST security best practices [12]. The guide also gives indicators for possible attack vectors. However, it currently lacks details for security beyond authentication, especially regarding access control and resource management.

Katt and Prasher [7] proposed a quantitative model for assessing the overall security risk of REST services. In their work, they outlined that APIs and general web services require different security considerations. Furthermore, their analysis showed that overall security of REST can be assessed with an assurance score. However, this required a manual analysis of each service which could be a problem for applying the findings on a larger scale.

Nguyen et al. [10] investigated how well description languages for REST APIs are able to express security mechanisms. They showed that the analyzed description languages had only basic capabilities to represent security schemes. Furthermore, they pointed out the lack of expressiveness for security mechanisms not related to authentication or authorization.

Atlidakis et al. [1, 2] published their approach for an automated analysis of REST APIs by implementing a fuzzer. Their presented tool generates test cases from the machine-readable API descriptions. Responses are evaluated for bugs by checking for unexpected HTTP error codes. Using this approach, the tool could identify a number of bugs in the GitLab API. In a follow-up paper [3], the authors introduced security rules specific to REST API resource management and proposed an extension of their tool using rule-based checks. Their work shows that automated analysis has significant potential to support a security evaluation.

2 Tasks

2.1 Overview

The thesis will explore the current state of REST security in real-world REST APIs. For this purpose, we will first determine existing best practices in literature and documentations. This initial effort will be based on the OWASP REST Security Cheat Sheet [12] as well as the security rules provided by Atlidakis et al. [3]. We will then implement an automated analysis tool to test whether real-world REST APIs follow the established best practices.

Using the results and findings gathered by the tool during the analysis, we will evaluate whether the current REST security definitions are sufficient. In addition to this, we will expand or adjust them with our own findings. Security checks in the analysis tool will be proposed to test and verify our amendments. Based on our findings, we will conduct a thorough security evaluation of the analyzed real-world REST API implementations.

The main tasks during the thesis can be summed up as follows:

- Create a comprehensive security definition for REST APIs backed by practical testing of real-world APIs
- Implement a pentesting tool for automated testing of REST security best practices
- Select real-world REST services for the analysis
- Evaluate the real-world services selected during the analysis according to the security definition

2.2 Detailed Description

The majority of tasks for this thesis are expected to be exploratory. Thus, the extend of the work that needs to be done can only be roughly estimated. In this section, the major tasks for answering the research questions are outlined. A list of ideas for security aspects that could be investigated during practical analysis can be found in section 2.3. The tasks described here are worked on in parallel using an

iterative approach (see section 4.1) where we continuously analyze, integrate, and evaluate our findings.

Task 1 – Create a comprehensive security definition

For this task, we determine existing definitions and best practices for REST security, specifically related to authentication and access control. During the practical analysis via the created pentesting tool, we test the existing definitions and expand them with our own observations from surveying real-world services (see description of Task 3 for the service selection process). For this purpose, we also investigate a selection of the security aspects listed in section 2.3.

The resulting security definition should be service-agnostic, practice-oriented and outline the most important considerations for REST security.

Task 2 – Implement pentesting tool

The pentesting tool should execute automated security checks for the examined security aspects. Its main function is to support the creation of our security definition by evaluating security problems in the real-world services via practical analysis.

The tool will be implemented in Python3. We aim for a modular design as the tool is expected to be extended gradually during the analysis. The modules contain security checks to cover the different security aspects. The core functionalities integrated at the start of the analysis should be as follows:

- Building customized HTTP requests from parameters
- Handling authentication parameters: username, password, tokens, IDs, etc.
- Making requests to REST API service endpoints
- Basic reporting in plaintext or using a simple data exchange format such as JSON
- Proxy support for compatibility with other analysis tools such as Burp Suite

Testing an API will require a machine-readable description file as input. If no such file exists, the tool may create a self-generated description file from other available documentation such as HTML documents. Furthermore, the tool may utilize additional user-supplied metadata about the service, e.g., a list of all available authorization scopes, to refine the analysis.

Documentation of the analysis tool is integrated into the source code and the written thesis.

Task 3 – Select real-world services for evaluation

For practical testing with the pentesting tool, we will choose a selection of real-world REST API services before the start of the practical analysis. We will focus on authenticated APIs that are actively developed and used. Services with a machine-readable definition are preferred. The type of service is not deemed relevant for the selection. However, if two services of the same type are analyzed, a direct comparison of their security might be possible. Thus, the analysis may take groups of services of a specific type into account.

Interesting real-world services that may be considered are:

- Github API
- Gitlab API
- Ebay APIs
- Google Cloud API
- Youtube Data API

Another application for testing is the evaluation of practical REST interfaces generated from descriptions by REST API generators, e.g., Swagger Codegen[13]. These generators must also consider security best practices.

At least three or four real-world API implementations will be chosen before the practical analysis starts. More services will be added during the analysis. The final evaluation will examine approximately 10 real-world services.

Task 4 – Evaluation

During the analysis, results from practical testing are continuously evaluated to extend our REST security definition and the analysis tool. The evaluation both involves checking for security issues as well as looking for additional security aspects that need to be considered.

In a final evaluation at the end of the thesis, our finished security definition is used to assess the overall security of every examined real-world REST service in regards to authentication and access control. This includes a comparison of security issues in individual services regarding any discovered vulnerabilities.

We will also briefly evaluate the work done in this thesis, i.e., the effectiveness of the pentesting tool for automated analysis and the applicability of our established security definition.

Task 5 – Written thesis

The written thesis presents the results of the practical analysis. These include the final security definition, a description of the pentesting tool, and evaluation results. Structure and planned contents of the thesis are outlined in section 3.

The written thesis is created in parallel to the practical analysis. During the first half of the analysis period, a simple draft consisting of notes from preliminary evaluation results and literature references is compiled. In the second half of the analysis period, the scientific text is formulated. Additionally, the final evaluation is added.

2.3 Ideas

This section lists different security aspects that could be investigated during the practical analysis. More security aspects may be discovered when services are evaluated. Also, due to the time limit of 6 months for the thesis, some aspects may not be analyzed in full detail.

Authentication Methods

Popular and complex real-world REST APIs usually offer several authentication methods (Basic Authentication, API Keys, OAuth, OpenID Connect, SAML, etc.) to access authenticated endpoints. For each method, collections of well-known attacks that could play a role in REST security already exist. Responses and error codes when using different authentication methods for the same endpoint could be compared. Discrepancies found here could reveal security flaws in the implementation.

Offline Analysis

Public REST APIs usually provide a machine-readable description of the API to ease the integration for clients. An example for such a description form is the OpenAPI standard[11]. These documents reveal available endpoints, input parameters, HTTP response codes and accepted authentication methods by design. However, the contained information might also expose potential security issues. Vulnerabilities found by offline analysis could be especially dangerous because services cannot prevent attackers from searching their API descriptions.

The main goal of the analysis would be to find deviations between request or response parameters, e.g., by finding an endpoint that uses a different response schema

for a specific HTTP error code than other endpoints. Description files can also include information about security mechanisms that a service supports. While Nguyen et al. [10] showed that not all security measures can be expressed by description languages, existing information could still be used to identify and prepare targeted online attacks against specific security schemes.

Additional documentation in textual form made available by the services might be included in this analysis.

Undocumented Behavior

For this security aspect, the analysis tries to discover behavior that is not documented in the API description or textual documentation. Examples for this include acceptance of undocumented HTTP methods, unspecified input parameters or deviating/malformed response content. Partial documentation, i.e., something documented in the API description but not in textual form, could also be considered. Undocumented behavior could be a sign for more problems at the affected endpoint or service in general.

Deprecated Endpoints

Real-world APIs are usually not static. Their features and security standards are constantly adjusted to extend the provided service. As such, it is a common occurrence that API endpoints and the involved parameters are subject to change, deprecated, or removed. In practice, a deprecated endpoint must adhere to the same security standards as all other parts in the API. However, stopping active feature development for an endpoint may also lead to negligence of its security.

The analysis for this security aspect would focus on evaluating deprecated endpoints specifically and comparing their behavior to that of non-deprecated ones. Removed endpoints could also be taken into account if older API documentations can be easily recovered, e.g., via `git` history.

Single Data - Multiple Endpoints

A data resource of a user may be available at multiple endpoints. However, these endpoints may require different levels of access control. In a worst case scenario, this could lead to a situation where an otherwise authenticated resource of a user can be requested via a public endpoint with no access control. Thus, a service must carefully consider which resource it makes available at which endpoint.

In our analysis, we would address the issue by searching for resources that are accessible from multiple endpoints, e.g., by analyzing the API description. We can then find the minimum necessary scope for retrieving said resource. Another variation of this approach would be a search algorithm determining the endpoint with the least restrictions for a specific resource.

Scope Mapping

Every endpoint and HTTP method that requires authentication should define a list of authentication scopes defined by the service that are necessary for access. Scope mapping is a process where the associations between scopes and endpoints are determined and mapped via a 2D matching graph. For example, this would reveal which endpoints are accessible with a specific scope and vice versa. The gathered information could be used as a starting point for other parts of manual and automated analysis.

Access Control Measurements

Authorization scopes define permissions for a specific user. However, in some cases further access control measures are in place. This security aspect especially affects services where user groups with additional role-based permissions can be created, e.g., GitHub organizations. Services therefore must ensure that they not only check if the permissions granted by an access token are accepted, but also that the user issuing the token is properly validated.

Atlidakis et al. [3] have identified basic security rules related to this aspect, which can be used as a starting point for our analysis. Their work emphasizes the requirement for a consistent hierarchy of resources. For example, a user A must not have access to resources in the namespace of another user B.

The practical analysis would focus on testing whether the user is properly validated and if access control is enforced. For this purpose, we would explore methods and checks that try to confuse a service about the authenticated user.

Header Analysis

This part of the analysis investigates the usage of HTTP security headers. Services may use custom headers in their HTTP response to authenticated requests. These have potential to leak information about the service, the user, or the authentication method to an attacker. As a result, they might be used as a starting point for crafting more sophisticated attacks.

Using the pentesting tool, we could automatically identify custom headers and conduct a manual analysis to reveal further security issues. Additionally, headers could be compared across different endpoints.

Semi-Public Endpoint Security

Security best practices for REST APIs currently consider two types of endpoint regarding authentication: Public endpoints with no access control and authenticated endpoints with strict access control. However, in real-world APIs the separation between these types is not as clear. For example, an endpoint may respond to both authenticated and non-authenticated requests with a non-error response. Additionally, the response to the authenticated request may return more detailed information.

Our analysis of this security aspect will be based on the assumption that the internal authentication logic for these „semi-public“endpoints is more complex than for the other types of endpoints. Therefore, these endpoints may be subject to additional attacks and create further security requirements.

Input validation

API endpoints may receive input parameters in a request. This is usually utilized when an endpoint creates, changes or deletes a resource for the user. However, the parameters may be used for authentication purposes. Malformed or invalid user input must be properly validated by the service, otherwise injection attacks may be used to bypass authentication or access control.

An automated analysis of this aspect should make use of a fuzzer to generate a large number of possible test cases. For this purpose, existing REST fuzzing tools like RESTler [2] could be utilized.

3 Organization of this Thesis

A preliminary structure for the thesis is presented in the table of contents below. Note that large parts of the written thesis, especially Chapter 3-5, will be based on the results of the analysis. As a result, some subsections in these chapters are placeholders (marked with `To be determined...`) because the covered security aspects will be established based on findings during the analysis.

Additionally, we have to distinguish between the organization of the written thesis, which presents mainly the results of the analysis, and the approach to the practical analysis. The latter is described in section 4.1.

3.1 Structure

1. Introduction
   - 1.1 Motivation
   - 1.2 Related Work
   - 1.3 Research Questions
   - 1.4 Scientific Contributions
   - 1.5 Organization of this Thesis
2. Methodology
3. REST API Security Fundamentals
   - 3.1 Representational State Transfer
   - 3.2 Comparison to Other Exchange Methods (optional)
   - 3.3 Description Languages
   - 3.4 Authentication Methods
4. REST Security Definition
   - 4.X To be determined...
     - ...

5. Pentesting Tool
   - 5.1 Premise
   - 5.2 Test Setup
   - 5.3 Modules
     - 5.3.X To be determined...
       - ...
6. Evaluation
   - 6.1 Examined Services
     - 6.1.X To be determined...
       - ...
   - 6.2 Service Security Evaluation
     - 6.2.X To be determined...
       - ...
   - 6.3 Effectiveness of Automated Analysis
   - 6.4 Evaluation of the Security Definition
   - 6.5 Evaluation of the Pentesting Tool
7. Conclusions
   - 7.1 Future Work
   - 7.2 Final Conclusion

4 Timeline

In this section, the general approach and the timeline for the completion of tasks is described. Section 4.1 outlines the approach for analyzing the security aspects, which can be completed in any order. The individual phases of the thesis are planned in section 4.2. Sections 4.3.1 and 4.3.2 show a visual representation of the timeline.

4.1 Practical Analysis Approach

Since the topics researched for this thesis are exploratory in nature, we will use an iterative approach for the practical analysis. Work is divided into iterations (lasting three days to bi-weekly). Each iteration focuses on a specific security aspect (see section 2.3). The main reason for choosing this approach is to make the analysis process flexible, but also comprehensible and replicable.

An iteration can roughly be divided into 4 phases:

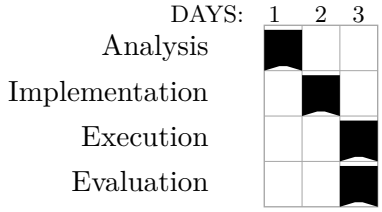
- Analysis: We examine security best practices as well as the API descriptions and documentation of real-world services in regards to the selected security aspect. Afterwards, we try to find theoretical attack vectors. Based on this research, we propose security checks to test our assumptions.
- Implementation: We integrate the proposed security checks into the analysis framework and do preliminary testing against selected real-world services. Checks may be adjusted based on feedback from the tests. The goal of this phase is to generalize the practical checks, so that they can be automated for every REST API service.
- Execution: The refined checks are run against all real-world services that should be examined. For each tested service, the analysis framework should generate a report containing the results for the checks.
- Evaluation: We evaluate the results of the checks by assessing the security impact for the tested services. The evaluation may reveal additional security issues in the tested services which can be analyzed in subsequent iterations. Furthermore, we check whether our assumptions made during the analysis phase were confirmed. If an assumption turns out to be wrong or incomplete,

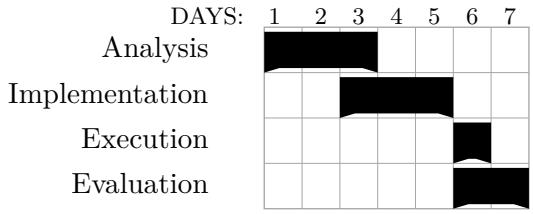
we may do a second iteration for the security aspect where we take the results of the evaluation into account.

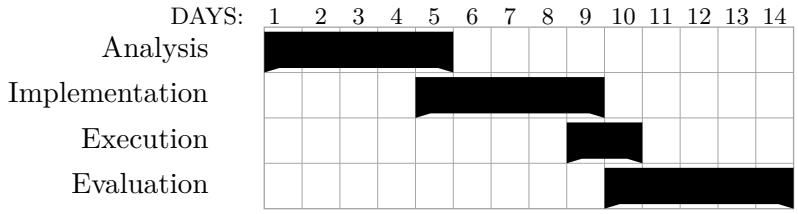
Finally, the results of the evaluation are integrated into the draft of the written thesis.

The main advantage of this approach is that the iterations provide a constant feedback loop during the practical analysis. Security assumptions are immediately tested on real-world services. Thus, we ensure that our proposed security tests are practical. In addition to this, we will get insights about the practical security of the tested APIs at an early stage of the overall analysis.

Below, a few example timelines for an iteration are shown. An iteration should take a minimum of three days but not more than two weeks. Note that the execution stage is expected to be shorter than other stages as this part would mostly be automated by the analysis tool. The exact length of each stage may depend on the difficulty of the task.







4.2 Phases

4.2.1 Phase 1: Setup

In the setup phase, the foundation for the analysis will be established. A preliminary version of the security definition will be created based on existing security best practices found in literature. The definition will be gradually expanded in Phase 2 and 3.

Additionally, the skeleton for the analysis framework will be implemented. This includes all relevant parts that support the penetration testing in subsequent phases, i.e., parsing API descriptions, making generic HTTP requests and generating basic reports from the responses.

Furthermore, a few real-world services will be selected for testing the framework. This includes generating accounts and access tokens for these services. If necessary, an isolated test environment for the analysis framework will also be created.

- Milestone 1: Basis for practical analysis has been established.

4.2.2 Phase 2: Exploration

The start of Phase 2 marks the beginning of the practical security analysis. In this phase, we will search for attack vectors and security flaws in the selected real-world services (see section 2.3). The search will be done using the iterative approach described in section 4.1. For every security aspect, a new module will be added to the analysis framework to execute the security checks. The findings are integrated into the draft of the written thesis.

The goal of this phase is to get an overview of security in real-world APIs and cover as many security aspects as possible. More in-depth testing will be conducted in Phase 3. At the end of Phase 2, a comprehensive collection of REST security aspects should have been analyzed. Checks to detect security flaws related to these aspects are integrated as modules into the analysis framework. Furthermore, the draft of the written thesis is finished (except for the final evaluation).

Selection of more real-world services for testing should preferably take place in this phase, so that there is enough time for their evaluation.

- Milestone 2: A draft of the security definition has been created and has been tested on real-world services. The pentesting tool implements practical checks to test compliance with the security definition.

4.2.3 Phase 3: In-Depth Analysis

In Phase 3, methods and tests from the previous phase will be further refined. At the start of this phase, the important REST security aspects and weaknesses of real-world REST APIs should have been identified. With this knowledge, we will take the relation between the security aspects into account. Furthermore, we will investigate whether combinations of weaknesses regarding different aspects lead to further security issues.

If practical exploits are found in the API implementations, we estimate the impact and severity of attacks. In addition to this, we will evaluate how much the selected individual services are affected by the discovered issues. Existing modules in the analysis tool are revisited and extended to make the implemented checks more robust.

Based on the evaluation of the found vulnerabilities, the security definition will be enhanced by deriving possible mitigations for the investigated attacks. The written thesis draft is finalized and formulated into a scientific text.

- Milestone 3: The security definition has been further improved and provides a solid foundation for security analysis. All real-world services have been evaluated.

4.2.4 Phase 4: Final Evaluation

During Phase 4, all previous findings will be evaluated in a larger context. Specifically, we will investigate what the findings mean for REST security in general, which aspects of REST security were especially affected, and where the security of real-world API implementations needs to be improved.

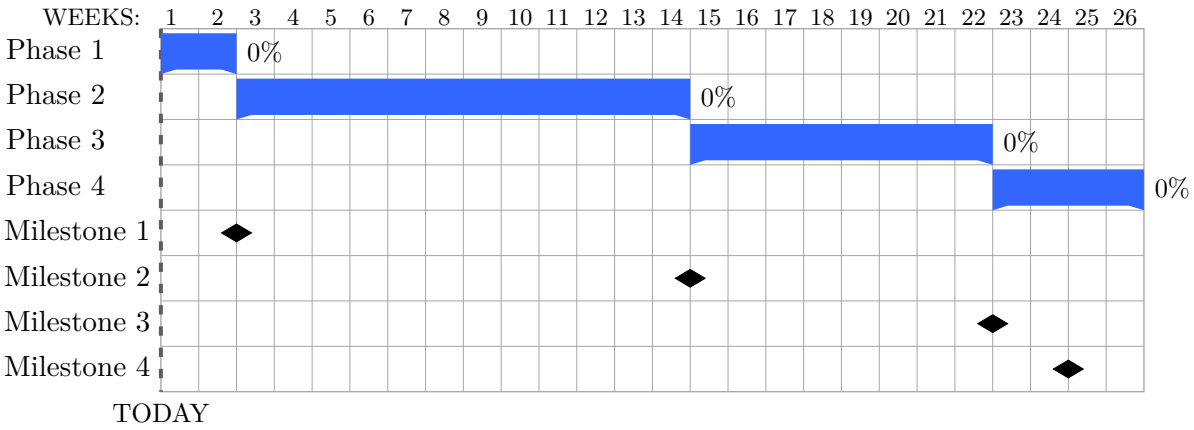
In addition to this, we will evaluate how our derived security definition addresses the security issues we found. Furthermore, we investigate how effective the analysis tool was in automating the analysis. The results are integrated into the written thesis.

- Milestone 4: Written thesis is finished.

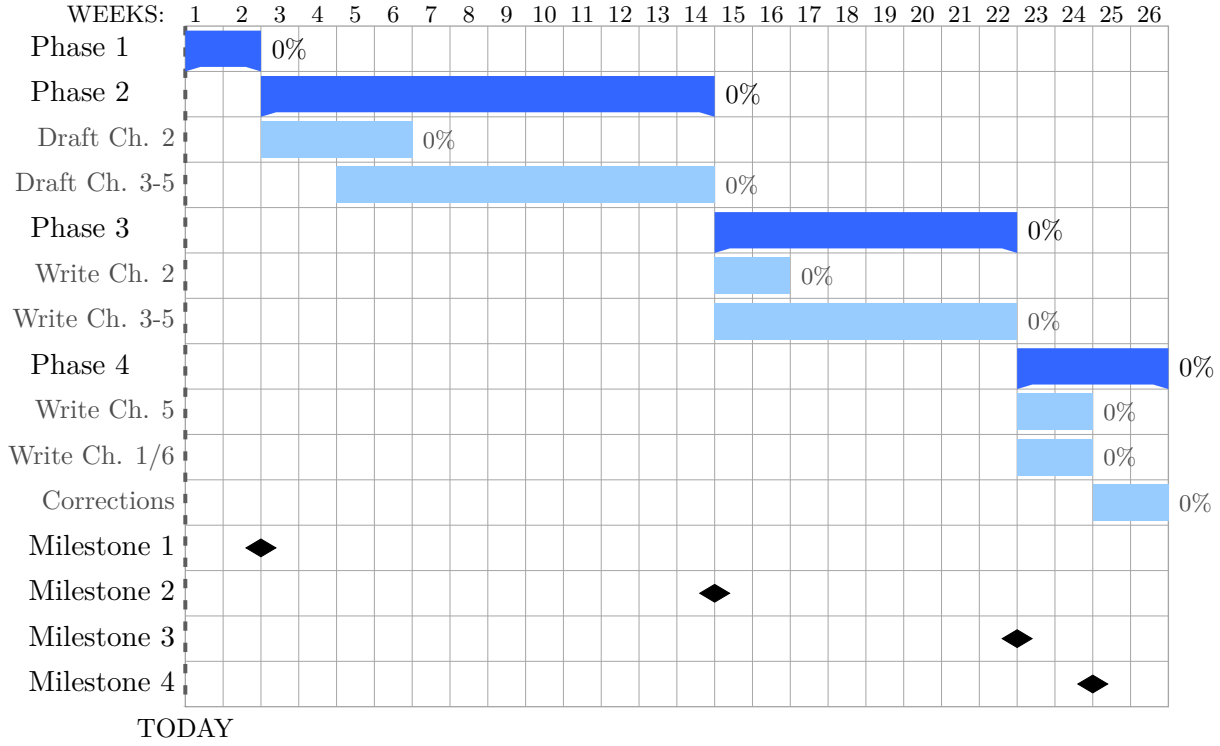
4.3 GanttChart Master Thesis

In this section, the start and end dates for the four phases are depicted. Section 4.3.1 shows a general overview, while section 4.3.2 also includes the estimated time for working on the written thesis.

4.3.1 Phases Overview



4.3.2 Phases (including written thesis)



Bibliography

[1] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. RESTler: Automatic Intelligent REST API Fuzzing. Technical Report MSR-TR-2018-11, Microsoft, April 2018. URL https://www.microsoft.com/en-us/research/publication/rest-ler-automatic-intelligent-rest-api-fuzzing/.

[2] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. RESTler: Stateful REST API Fuzzing. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pages 748–758, 2019. doi: 10.1109/ICSE.2019.00083.

[3] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. Checking Security Properties of Cloud Service REST APIs. In 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pages 387–397, 2020. doi: 10.1109/ICST46399.2020.00046.

[4] Roy Thomas Fielding and Richard N. Taylor. Architectural Styles and the Design of Network-Based Software Architectures. PhD thesis, 2000. AAI9980887.

[5] John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence C. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, June 1999. URL http://www.rfc-editor.org/rfc/rfc2617.txt. http://www.rfc-editor.org/rfc/rfc2617.txt.

[6] Frederick Hirsch, Rob Philpott, and Eve Maler. Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0. Oasis standard, March 2005. URL http://docs.oasis-open.org/security/saml/v2.0/. http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf.

[7] Basel Katt and Nishu Prasher. Quantitative Security Assurance Metrics: REST API Case Studies. In Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450364836. doi: 10.1145/3241403.3241464. URL https://doi.org/10.1145/3241403.3241464.

[8] Luigi Lo Iacono, Hoai Viet Nguyen, and Peter Leo Gorski. On the Need for a General REST-Security Framework. Future Internet, 11(3), 2019. ISSN 1999-5903. doi: 10.3390/fi11030056. URL https://www.mdpi.com/1999-5903/11/3/56.

[9] T. Lodderstedt, J. Bradley, A. Labunets, and D. Fett. OAuth 2.0 Security Best Current Practice. Draft 18, April 2021.

[10] Hoai Viet Nguyen, Jan Tolsdorf, and Luigi Lo Iacono. On the Security Expressiveness of REST-Based API Definition Languages. In Javier Lopez, Simone Fischer-Hübner, and Costas Lambrinoudakis, editors, Trust, Privacy and Security in Digital Business, pages 215–231, Cham, 2017. Springer International Publishing. ISBN 978-3-319-64483-7.

[11] OpenAPI. OpenAPI Specification v3.1.0, 2021. https://spec.openapis.org/oas/v3.1.0, as of July 17, 2021.

[12] OWASP. REST Security Cheat Sheet, 2021. https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html, as of July 17, 2021.

[13] Swagger. Swagger Codegen, 2021. https://swagger.io/tools/swagger-codegen/, as of July 17, 2021.