mnemonic

PUBLIC

Report
Security Assessment of NAIS Teams
NAV

Place	**Oslo, Norway**
Date	**05-10-2023**
Version	**1.1**
Author	**Felix Topsholm, Alexander Evans**

Executive summary

In September 2023, mnemonic performed a dynamic application security assessment of NAVs NAIS Teams web-application, an in-house portal and subsequent back-end system which handles automatic on-boarding of developers within NAV into specific groups. Additionally, the portal allows for the administration of internal NAV developer teams, with provided functionality that automatically creates peripheral team-areas such as Slack-channels and google teams.

The primary purpose of the assessment has been to provide an independent security review of NAVs software and supporting infrastructure to identify concrete vulnerabilities and advise on how these can be fixed or mitigated. By obtaining more insight into their security posture, the test enables NAV to prioritize remediation activities and better manage overall risk exposure.

The secondary goal of the project has been to increase awareness within NAV about software security and to provide documentation that an external technical security assessment has taken place.

The assessment focused on identifying common web application vulnerabilities, such as those in the OWASP list of Top 10 Application Security Risks. mnemonic has conducted this assessment from our dedicated security testing platform, using test accounts against NAV internal production environment. We experienced no significant disruptions during the test.
This technical report describes the in-depth findings and observations made during the assessment and recommendations for potential improvements.

Summary of findings

During the assessment we noted that the application responded robustly when tested using real-life attack scenarios. Areas exhibiting good security behaviour include:

Authentication & Authorization

The application relies on strong authentication utilising Auth0 implicit authentication flow provided by Google. We noted that authorization is enforced server-side at time of invocation on the function level.

Input Validation and Output Encoding

During fuzzing, which the act of providing an application with a large amount of malicious, erroneous or out-of-bounds input data, the application responded robustly. This continues to be the case during any failover scenarios, without leaking additional internal information. Additionally, we noted that all malicious in-context symbols such as ‘< ; "’ were adequately output encoded when reflected to end-users.

Error Handling

NAIS teams responded well when reaching certain error states or conditions within the app during testing. Generic error messages were displayed during erroneous operations, without divulging internal technical information of the webapp.

In total, mnemonic discovered three issues deemed as of [Low] severity. All issues revolve around deviations in defence in depth and security best practise.

The first issue embodies the leakage of GraphQL internal schema data via introspection to unauthenticated entities. This allows for the enumeration of the structure of the back-end datastore, and thereby the potential of providing would-be attackers with an advantage during certain attack scenarios.

The second issue revolves around a missing authorization check for the roles endpoint, resulting in internal user role information leaking. Lastly – we noted some configuration weaknesses within the TLS configuration for the application host, where older versions and cipher suites are supported.

Summary of recommendations

mnemonic's main recommendation is to ensure that the application implements and enforces proper access control rules for all operations. This is necessary to avoid unauthorized sensitive data access and thereby information leakage to unauthorized parties.

Additionally, we recommend disabling the introspective feature within the GraphQL endpoint. Lastly, we recommend reviewing the TLS setup for the application host, in-turn disabling support for TLS 1.0 and 1.1 versions and older, vulnerable, cipher suites.

mnemonic suggests reviewing all the findings described in Chapter 3 based on NAVs knowledge of their business context, risk appetite and other constraints, in order to re-assess the severity of that issue based on those properties and take appropriate measures.

Table of Contents

1 Introduction

1.1 Introduction to the report

mnemonic has performed a security test of NAIS Teams, a web-application and back-end solution allowing for the automation for on-boarding developers within NAV to their respective teams and groups.

The web-application portal is built using GraphQL and relies on google-front as the front-end framework.

Testing was conducted from dedicated mnemonic test hardware, against the internal production environment within NAV. Connectivity was provided via a VPN and device compliance client called 'NAIS Device'. Two accounts were created, one for each tester.

The current document describes the results from security testing. It includes mnemonic's analysis and recommendations for each of the observations and findings. mnemonic suggests using the document to identify and prioritize "quick-wins" and urgent remediation activities, as well as to support operational risk management and strategic security improvement initiatives.

A central goal of security testing is to discover and document as many security-related bugs as possible within the given scope, under constraints such as time and budget. The reader should keep in mind that security testing is a non-deterministic quality assurance activity which contains elements of both skill and luck. It is not likely that all bugs and vulnerabilities in any non-trivial software system will be discovered through testing. While the test report gives insight into the security, there may still exist additional latent vulnerabilities in the system.

1.2 Structure of the report

The report consists of four main parts, as well as appendices.

1. Chapter 1 (the current chapter) describes the overall context and structure of the report.
2. Chapter 2 provides aggregated information about the report findings and recommendations.
3. Chapter 3 provides detailed information about each finding, sorted by mnemonic's evaluation of criticality.
4. Chapter 4 provides information about how the test was carried out.

A. Appendix A provides supporting documentation regarding mnemonic's classification methodology for discovered vulnerabilities.
B. Appendix B providing additional supporting documentation.

2 Summary of findings

2.1 Summary of findings by category

The following table provides an overview of which types of vulnerabilities that were found in the test. Categories are based on the OWASP Top 10 Application Security Risks (2021).

Category	Finding(s)	Status
A01:2021 – Broken Access Control Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Access control is only effective in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.		Not found
A02:2021 – Cryptographic Failures Proper use of cryptography protects data in transit and at rest. Failures typically lead to exposure of confidential or sensitive information, and breach of privacy laws or regulatory compliance.		Not found
A03:2021 – Injection Applications are vulnerable to Injection attacks when data is used raw in dynamic code, either in the application itself or when handing over data to some other component. Failure to use safe APIs or escaping/encoding output when safe APIs are not available typically leads to access to back-end component (for server-side Injections) or modification of user interface (for HTML Injection or Cross-Site Scripting).		Not found
A04:2021 – Insecure Design Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required. Secure software requires a secure development lifecycle, some form of secure design pattern, paved road methodology, secured component library, tooling, and threat modeling.		Not found

Category	Finding(s)	Status
A05:2021 – Security Misconfiguration Security misconfiguration is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.	3.1, 3.2, 3.3	**Found**
A06:2021 – Vulnerable and Outdated Components Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.		Not found
A07:2021 – Identification and Authentication Failures Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. If application functions related to identification, authentication, and session management are implemented incorrectly, it allows attackers to compromise username-passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.		Not found
A08:2021 – Software and Data Integrity Failures Enforcing software integrity is needed to protect against introduction of attacker-controlled code or configuration during development, build, and deployment, e.g., in CI/CD pipelines and infrastructure as code. Data integrity failures includes deserialization flaws, which can lead to remote code execution.		Not found
A09:2021 – Security Logging and Monitoring Failures Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.		Not found
A10:2021 – Server-Side Request Forgery (SSRF) SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).		Not found

2.2 Summary of detailed recommendations

The following table summarizes all detailed recommendations made in the report. For additional context, please review the detailed finding descriptions in the subsequent chapter.

3 Detailed findings and observations

The security test aims to discover, demonstrate, and document technical as well as logical vulnerabilities in the system, which can be exploited to cause a loss. This section describes all such findings in detail, as well as additional observations made during the test that are relevant for understanding the security posture of the system on a technical level.

All findings have been ranked based on our evaluation of criticality and impact, which takes into account the technical impact of each finding as well as our understanding of the system context, and gives a score on a scale from [4-critical] to [0-info]. For more information about our evaluation criteria, please see Appendix A.

Findings are generally limited by the scope and level of access given during the test. In particular, we do not actively try to find security issues outside the agreed scope.

3.1 Unauthenticated GraphQL Introspection [1-low]

Location

- https://teams.nav.cloud.nais.io/query

Category

- A05:2021 – Security Misconfiguration

Summary

GraphQL introspection is enabled and can be accessed without having to authenticate with the application. This gives an unauthorized threat actor access to view what data fields exist within the backend, and potentially a means to craft queries in order to attempt to retrieve said data.

Detailed description

The GraphQL *introspection* is often used by threat actors in order to get a layout of data within the application back-end. Introspection uses built-in queries which return information on the respective GraphQL schema itself, divulging what types of data fields exist for objects stored within the backend. In this case, introspection is accessible to any user on the network since it does not require authentication.

The example below shows the execution of an introspection query, without requiring authorization (Cookie field removed):

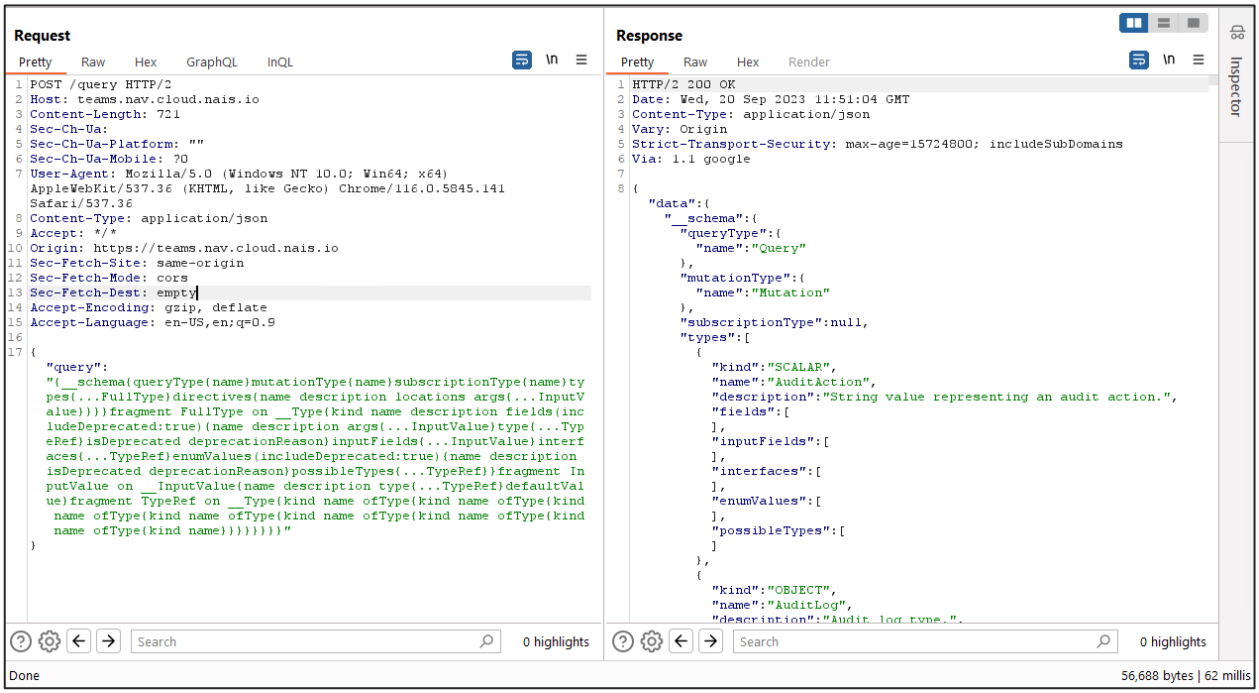

Figure 1: Snapshot illustrating raw query and returned response.

Using the introspection, it is trivial to get an overview of what types of requests are possible:

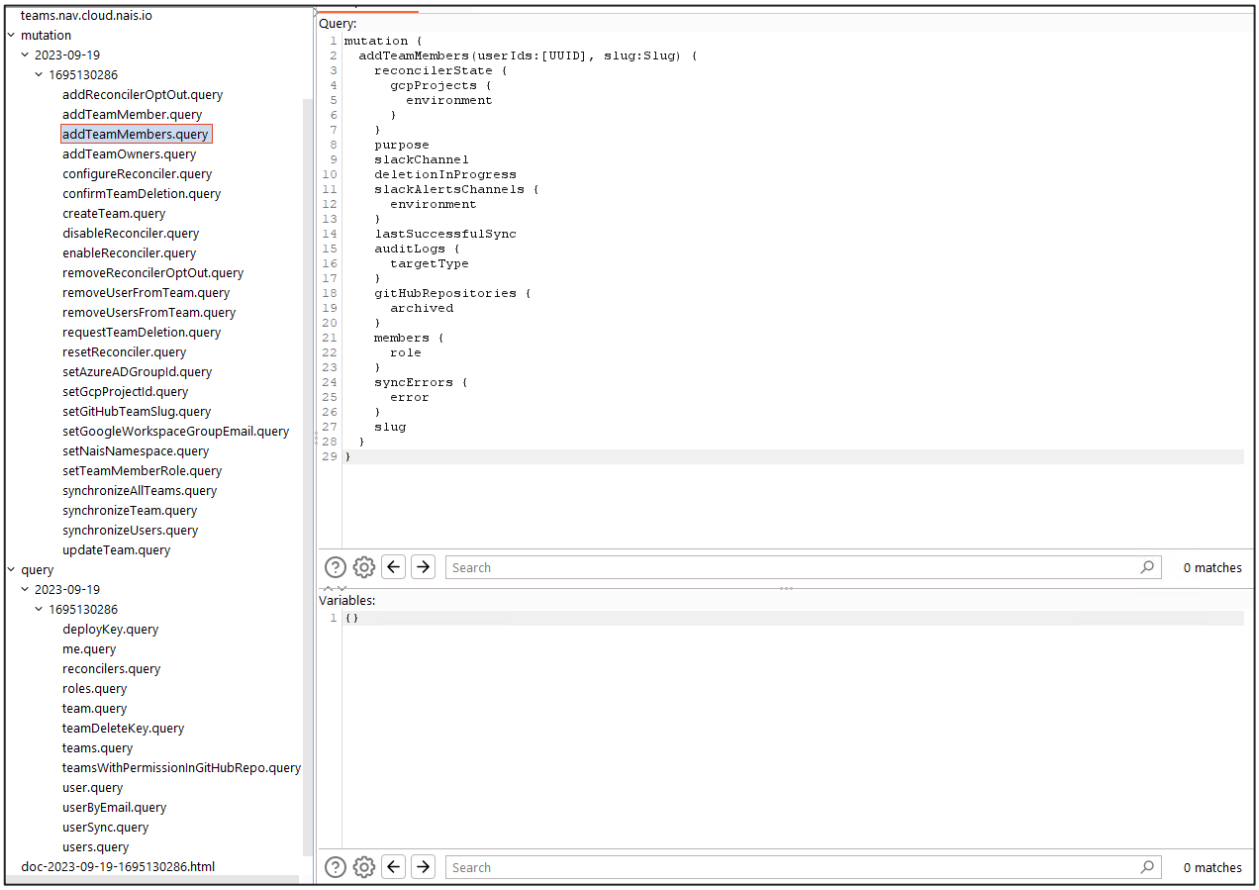

Figure 2: Snapshot illustrating raw introspective query.

Recommendations

If the introspection functionality is not used by other applications or users, we suggest disabling it. If it is required, it is recommended to restrict who can use it, enforcing strong server-side authorization checks at the function level, during the time of use or invocation.

Recommendation 1. [1-low] Disable introspection or add strict authorization requirement

References

- https://portswigger.net/kb/issues/00200512_graphql-introspection-enabled

3.2 Unauthenticated "Roles" endpoint [1-low]

Location

- https://teams.nav.cloud.nais.io/query

Category

- A05:2021 – Security Misconfiguration

Summary

The query endpoint "Roles" does not require authentication. This provides entities, present on the network, with the ability to retrieve data concerning internal roles configured within NAIS Teams.

Detailed description

During the assessment we noted that another endpoint present within the GraphQL schema does not require any authentication. Specifically, the 'Roles' endpoint does not seem to have any "request fields", meaning the only information it can give back is what types of roles are located in the NAIS Teams application.

Here is an example of the request:

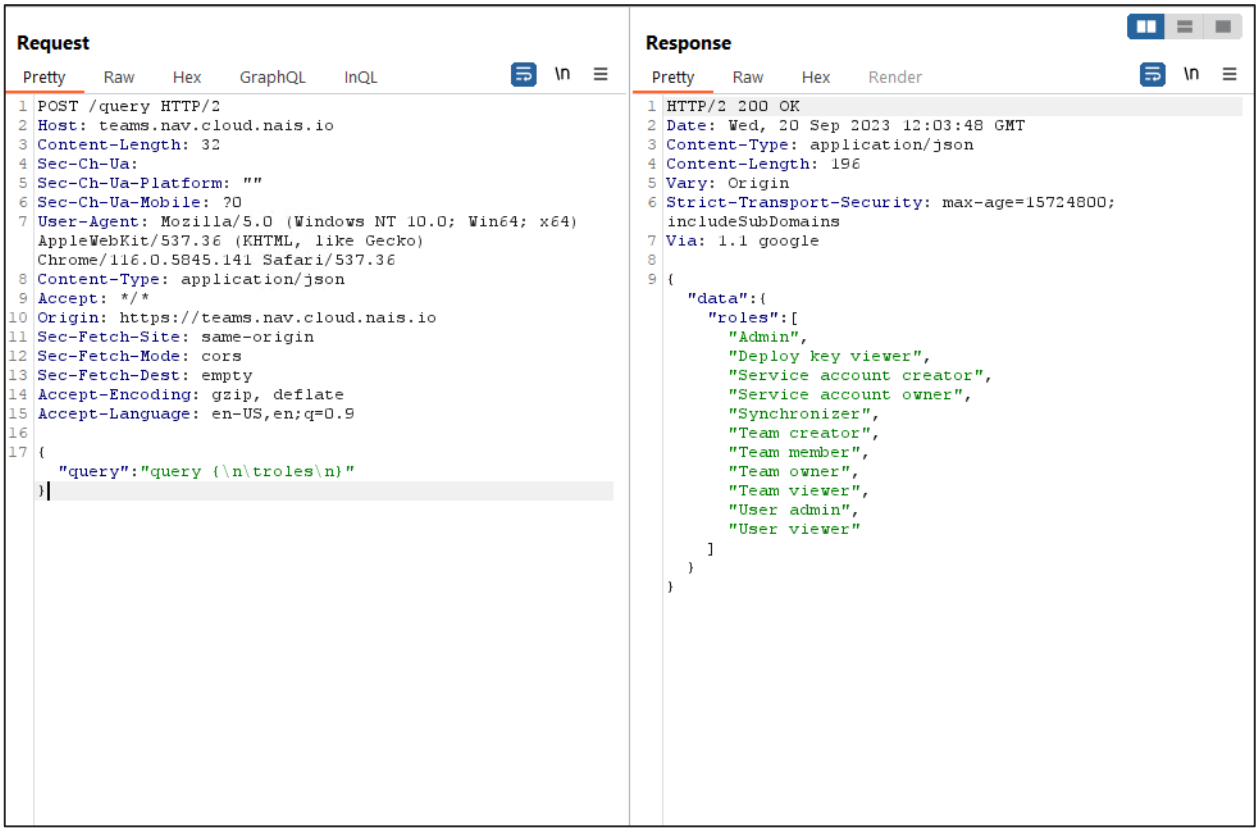

Figure 3: Snapshot showing query returning roles from the application.

This results in a minor risk, as every bit of information a threat actor can gain on a target system is useful, especially if it does not require authentication or authorization. If this endpoint were to be changed in the future by adding request fields, it could leak additional data.

It is of note here, that the NAIS Teams application is an internal NAV application, used by personnel who originate from the internal network. It is not available to the wider internet, where a VPN is required for access. In-turn the scope and range of threat actors is drastically reduced, resulting in this finding being rated as of very low severity.

Recommendations

Ensure that unauthenticated access to the endpoint is not possible. If a requirement exists for data-structure sharing in cases of integration debugging, we continue to recommend authentication or documentation in order to assist efforts.

This continues to be the case if the endpoint is used by another application. Additionally, it is recommended to restrict which user can use it and enforce authorization at the function level, at time of use or invocation by the backend where possible.

Recommendation 2. [1-low] Require authentication for everyone attempting to use the introspective query.

References

- https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

3.3 Weaknesses in TLS/SSL configuration [1-low]

Location

- https://teams.nav.cloud.nais/

Category

- A05:2021 – Security Misconfiguration

Summary

During the assessment we noted that older end-of-life TLS versions and mathematically insecure cipher suites were in-use by the application host.

This can lead to future vulnerabilities, and carries a small risk as certain configurations have been proved to be mathematically unsound.

Detailed description

Transport Layer Security (TLS) is the "S" in "HTTPS", the world's most widely used security protocol, and the mechanism to ensure that communications between the end users and the webserver (or web frontend, e.g. Cloudfront or Akamai) remains confidential and that the data integrity is maintained.

TLS has a variety of configuration options, and it is easy to configure the protocol to support insecure encryption algorithms or less secure protocol versions. While there are several documents describing TLS best practices, there is no exact "right answer" as to how organizations should configure their TLS endpoints, and guidance also evolves over time.

While there were no high-severity vulnerabilities in NAV configurations, there were some weaknesses that we do recommend remediating.
Supporting the 3DES-CBC encryption algorithm, means that an attacker in some situations will be able to break the confidentiality of the traffic, although the attacks are quite complex.

Supporting the older TLS 1.1 and TLS 1.0 protocols is generally not necessary, even for backwards compatibility reasons. Newer versions of TLS were published in 2008 (TLS 1.2) and 2018 (TLS 1.3). From a security perspective, there is no benefit, and a potential risk, associated with support for old TLS versions. Regardless, many websites still support the older versions.

There are many scanning tools available which can be used to check the security properties of a TLS server, three popular ones are the SSL Labs scanner, as well as the command line tools SSLyze (newer and more featureful), and SSLscan (older and more rudimentary).

- https://www.ssllabs.com/ssltest/
- https://github.com/nabla-c0d3/sslyze
- https://github.com/rbsec/sslscan

As the NAIS teams application is an internally used application, and not externally facing, severity ratings have been adjusted, resulting in a very low severity.

Recommendations

We recommend disabling TLS 1.0 and 1.1, in addition to moving away from old 3DES-CBC cipher suites. In order to maintain a secure configuration, we recommend standardizing the TLS configurations used for a given technology that terminates TLS. For example, defining and maintaining a common TLS configuration profile for all domains hosted in AWS.

To keep the configuration current and ensure that it is uniformly applied, it is possible to use a TLS configuration scanner such as SSLyze or SSLscan as part of the development pipeline.

We also recommend using Certificate Transparency logs actively to monitor when new certificates are issued for NAV domains, if this is not already being done.

Define secure TLS configuration profiles that can be used uniformly across all services using a given TLS stack

Recommendation 3. [1-low] Use TLS scanning tools actively as part of the development pipelines to ensure that secure TLS configurations are being used, and prevent configuration drift.

References

General guidance:

- https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices
- https://wiki.mozilla.org/Security/Server_Side_TLS

SSL/TLS test tools:

- https://www.ssllabs.com/ssltest/
- https://github.com/nabla-c0d3/sslyze
- https://github.com/rbsec/sslscan

Certificate transparency:

- https://crt.sh/
- https://transparencyreport.google.com/https/certificates
- https://developers.facebook.com/tools/ct/search/

3.4 Potentially Erroneous Functionality [0-info]

Location

https://teams.nav.cloud.nais.io/query

Category

Not Applicable

Summary

During the assessment, we noted that it is possible to add a unique user twice to the same team. As this has no security impact, it has been deemed as of informational severity.

At the time of writing, we do not believe this issue has security impact. However, there might be other circumstances or scenarios where this might become a security issue in the future. This could be exacerbated by the expansion of existing functionality.

Detailed description

During our testing, we found it possible to add the same user twice to a Team. We were unable to utilize this for an exploit, but should be looked further into.

To replicate, we first need a team with a member. In this example we will use Alex Evans:

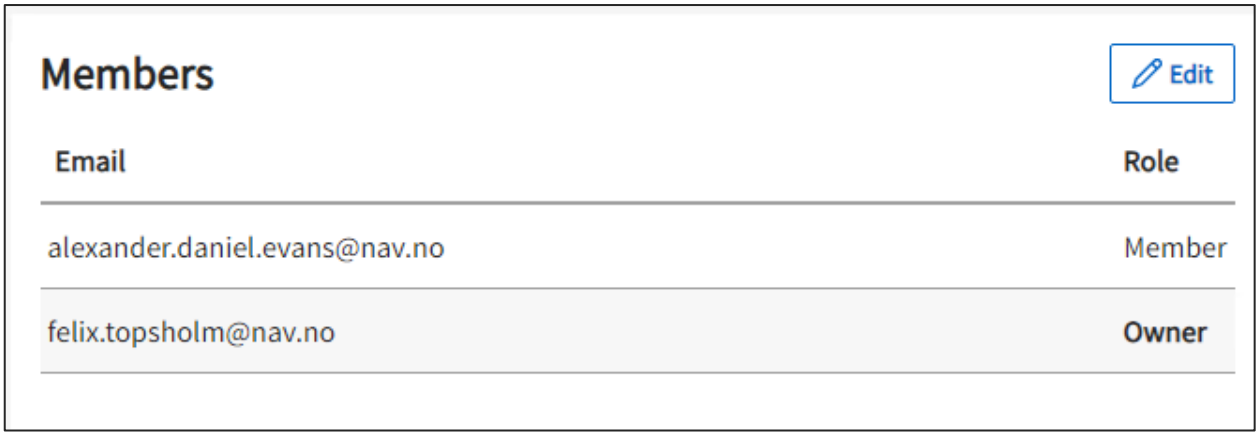

Figure 4: Snapshot showing normal user group, containing two unique users.

Then we send a request to the “addTeamOwners” endpoint with Alex Evans UID:

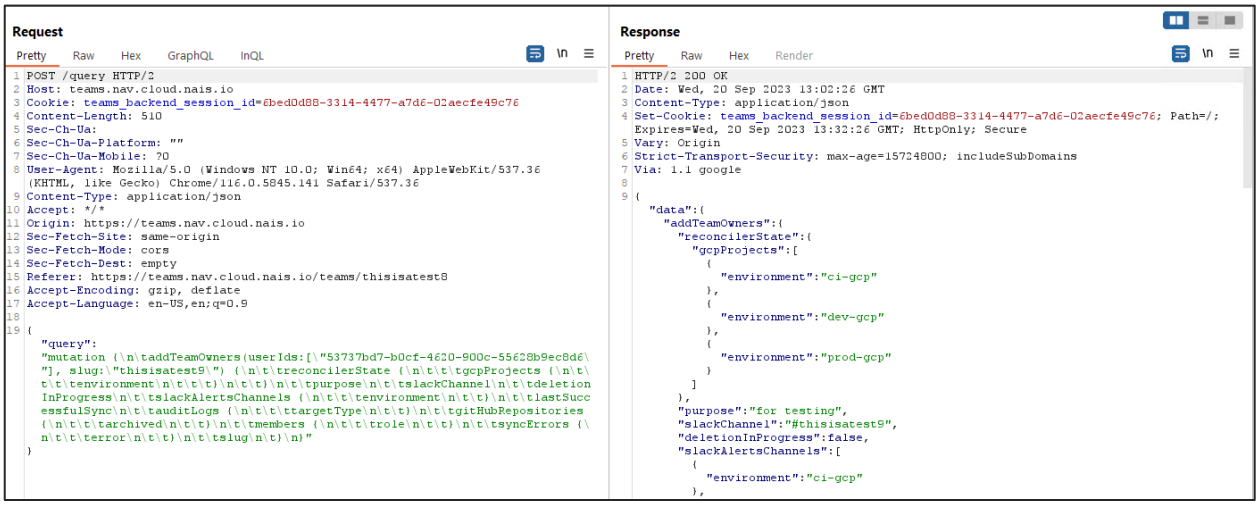

Figure 5: Snapshot showing the request responsible for adding users to teams.

Same query but structured:

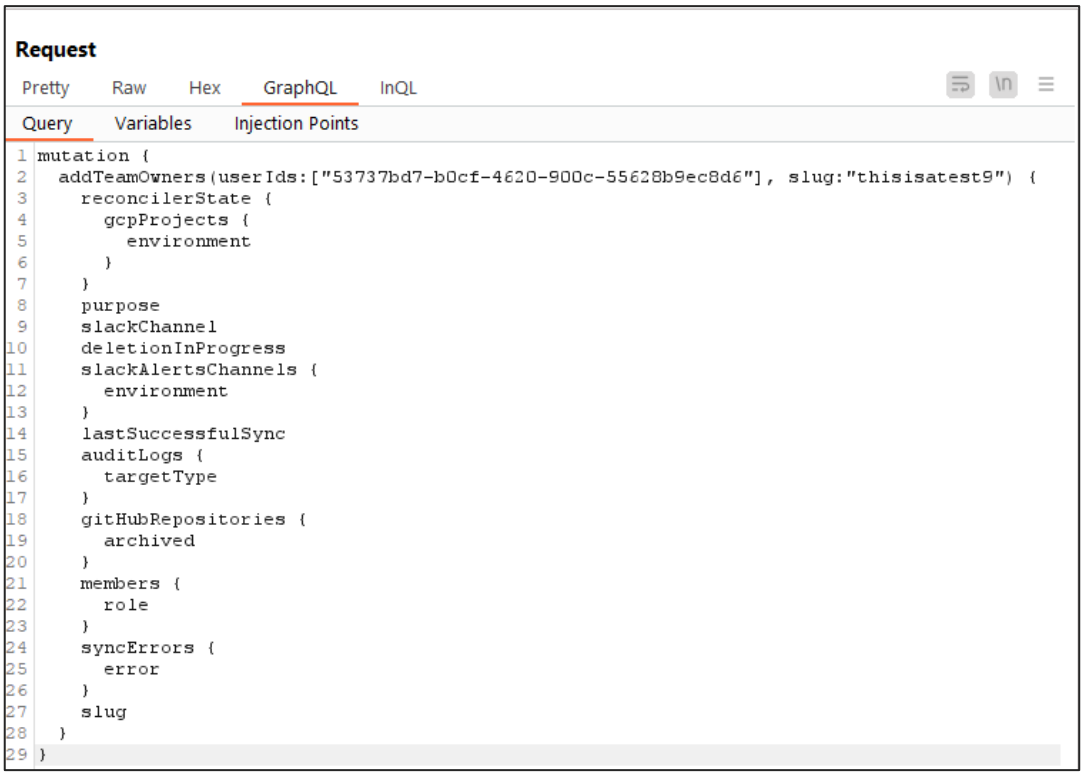

Figure 6: Snapshot showing raw query structure of the request involved.

This adds Alex as an owner of the group, but it will also keep the previous user and convert it to an owner:

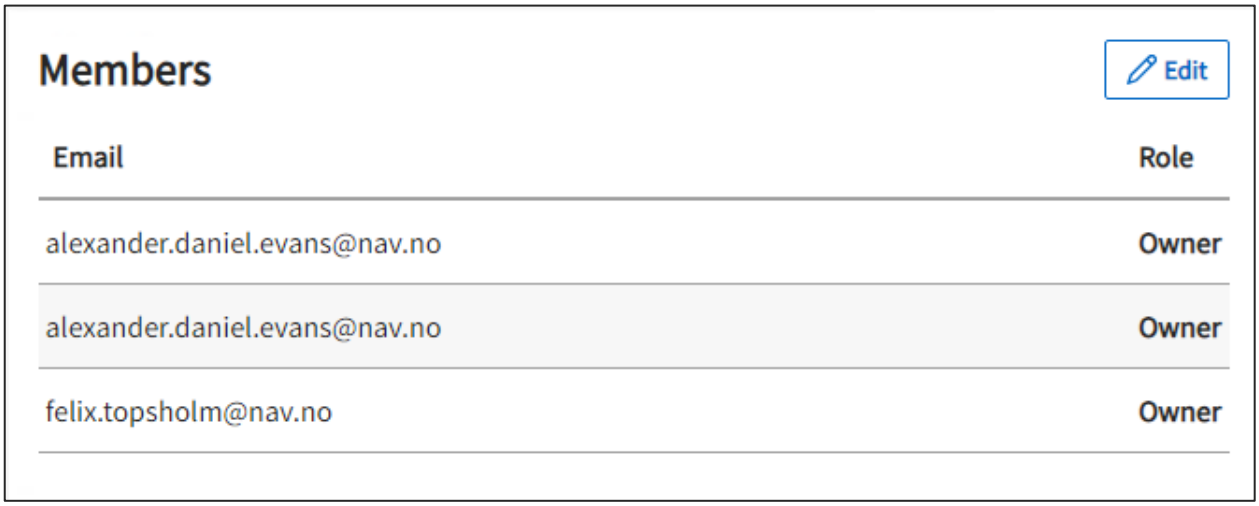

Figure 7: Snapshot showing the now modified group, containing duplicate users.

Recommendations

Investigate the cause of this issue, ensure that the authorization mechanism and ownership of groups is not impacted by the behaviour explored above.

Recommendation 4. [0-info] Investigate why it is possible to add a user twice to a team

4 About the report

4.1 Test execution

Project name	Security Assessment of NAIS Teams
Client	NAV
Conducted by	mnemonic AS
Consultants	Felix Topsholm, Alexander Evans
Internal QA	Scott Brugman
Start date	11-09-2023
End date	05-10-2023

4.2 Document version control

The current revision of this document is 1.0. All major revisions are documented in the table below.

Version	Date	Consultant(s)	Comment
1.1	05-10-2023	Alexander Evans	Publishable version drafted
1.0	05-10-2023	Felix Topsholm, Alexander Evans	Final report delivered to customer

4.3 Test scope

The NAIS Teams application resides at the following domain:

- https://teams.nav.cloud.nais.io/

All aspects of the application were considered as in-scope during the assessment. Of note is that the application is in-use as a production system, all test-cases were adjusted for production testing.

4.4 Test scenarios and mis-use cases

No specific scenarios and mis-use cases were discussed for the test. The test has simulated a knowledgeable and skilled threat actor attempting to explore the system, bypass the security controls present, or otherwise cause undefined or unexpected behavior.

4.5 Methodology and tools

mnemonic has conducted security and penetration testing, source code audits, and related services, ever since the company was founded in 2000. Our security testing methodology is based on the combination of open standards and collections of "industry best practice", together with our own experience accumulated over the last 20 years. In addition to this, testing is supported by an extensive knowledge base, as well as internally developed tools and scripts.

Our methodology is supported by the processes "P3003 Procedure for security testing" and "P3006 Use and maintenance of testing platform" in mnemonic's ISO 9001 / ISO 27001 certified quality and security management system, as well as associated templates and documentation.

Security testing conducted by mnemonic consist of the following phases:

1. **Preparation**: establish test context and scope, agree on terms of engagement and escalation routines, plan assessment activities, perform a functional review of the test objects
2. **Reconnaissance**: gather information about the test objects, based on available documentation, open source intelligence (OSINT), relevant technical standards, own research, and other sources if applicable
3. **Mapping**: scan and explore the relevant systems, using a combination of automated tools and manual or guided exploration, attempting to discover vulnerabilities or "trouble spots" along the way
4. **Verification and analysis**: evaluation and verification of initial findings, in-depth manual vulnerability assessment of selected areas, analysis of possible countermeasures and recommendations
5. **Reporting**: internal QA and presentation of findings, through a structured written report, and a joint debrief. Raw data can also be extracted and presented as an attachment to the report.
6. **(Re-test)**: if desirable, we also provide re-testing after remediation to verify that findings are closed

Central tools used in the current assessment include:

- Burp Suite Professional
- Nessus Professional
- Internally developed tools and scripts

4.6 Problems encountered

No specific problems were encountered during this assessment.

Appendix A: Vulnerability classification

The purpose of this section is to describe how mnemonic ranks vulnerabilities. Unless otherwise agreed, all findings will be ranked based on mnemonic's evaluation of criticality and impact, according to the criteria described below. In the technical report, findings will be sorted based on this ranking (and optionally grouped per test object, for tests that cover large or complex scopes).

Alternate measures, such as CVSS v3 score, may be used as supplementary criteria, or as a primary ranking on request.

It is worth pointing out that vulnerability classification and ranking is far from an exact science. Because of this, we always encourage clients to review all findings that mnemonic reports, in order to understand their potential impacts and prioritize for remediation. Preferably, findings can be debriefed together with mnemonic's team as part of delivery. This allows both sides to clear up potential misunderstandings, and ensures that any loose ends are tied up.

A.1 Classification matrix

Rating	Description	Examples
4-critical	Finding or vulnerability with potentially critical impact on the system's confidentiality, integrity and/or availability, and with few or no mitigating factors. Critical findings are "showstoppers" that mnemonic thinks should be subject to immediate / emergency remediation, based on their potential impact.	• Remote code execution or similar vulnerabilities that enable a threat agent to break in through the system and target back-end resources • Authentication or authorization bypass leading to administrative or privileged access to essential functionality • Vulnerability that may lead to a total breach of key security properties of the system, such as a database compromise
3-high	Finding or vulnerability with potentially high impact on the system's confidentiality, integrity and/or availability. High-severity findings should normally be prioritized for analysis and possible remediation within an urgent time-frame.	• Otherwise critical findings with partial mitigation in place, e.g. requiring higher effort to exploit or leading only to partial loss of security • Systematic issues that have significant impact across multiple solutions or components • Persistent cross-site scripting (XSS) that may be used to target privileged users without user interaction
2-medium	Finding or vulnerability with a medium-level impact on the system's confidentiality, integrity and/or availability.	• Vulnerabilities that require significant effort, user interaction, and/or luck to

Rating	Description	Examples
	Medium-severity findings should be subject to remediation following usual vulnerability management and triage processes.	exploit, such as reflected XSS or CSRF • Findings that provide a non-trivial benefit to a threat agent, but do not meet the criteria for high or critical impact
1-low	Finding or vulnerability with a potential low impact on the system's confidentiality, integrity and/or availability. Low-severity findings should be subject to remediation following usual vulnerability management and triage processes, but the initial evaluation is that they may be of lower priority.	• Findings that have a limited and localized security impact, such as a small information leak of non-sensitive data • Deviation from recommended security practice, or lacking defense-in-depth
0-info	In addition to the above, we may use a classification called 0-info, sometimes pointing out robust / well-designed solutions, unexpected behavior, bugs that do not have an obvious security impact, or to provide tips and hints for improvements or optimizations. Informational findings should be reviewed by relevant stakeholders.	• Findings that do not have a clear or proven security impact, but which we feel should be reviewed • Findings that do not have an impact within the test scope, but may be relevant elsewhere • Bugs without a clear security impact • Suggested future improvements • Unusual or particularly clever solutions

A.2 Rationale

While the impact of a vulnerability can often be classified purely in terms of technical impact, it is usually desirable to enrich the vulnerability with additional context. For example, the practical exploitability of a vulnerability may range from simple to unproven. A vulnerability may be directly accessible from the Internet, or located on an isolated internal network segment. Finally, the data or functionality that is being impacted may range from trivial to mission-critical.

As external penetration testers working on a limited-duration assessment, mnemonic will rarely gain a complete understanding about the business impact of a complex vulnerability, whereas its direct technical impact is easier to assess based on objective criteria. At the same time, our experience is that building on the penetration testers' experience and judgement to rank findings in context yields results that are more aligned with our customer's business need, than a purely generic framework such as CVSS.

Frameworks such as in CVSS v3 attempt to encode the impact of a vulnerability in purely technical terms. A CVSS score is an ordinal score from 0 to 10, which provides a ranking. In doing so, a lot of functional context is lost. Because it uses an ordinal scale, it is generally not useful (although quite common) to do

arithmetic on CVSS scores; concepts such as the “average CVSS” is largely meaningless, and a CVSS 10.0 vulnerability is generally not “10 times as bad” as a CVSS 1.0 vulnerability. It is our experience that CVSS scores by themselves are often given too much weight, when prioritizing remediations.

To give an example, a CVSS 10.0 vulnerability in a system which is isolated from threats by its environment may pose a lot less risk to the business than a CVSS 7.5 vulnerability in an Internet-facing webserver with a published exploit. The derived measures of CVSS temporal and CVSS environmental scores try to enrich the base scoring with this type of context, but we feel that these measures are unwieldly in practice.

In our rankings, mnemonic tries to strike a balance between these perspectives. We use the technical impact of our findings as a starting point, but take into account the knowledge we have about the system, and how vulnerabilities may be exploited, when we make our evaluation.

Nevertheless, this is not an exact science. Because of this, we always encourage clients to review every finding that is documented in the report, based on their knowledge of their own business context, internal and external requirement, technical solution, risk appetite, and other constraints.

A.3 See also

Information on CVSS from the NIST National Vulnerability Database and FIRST:

- https://nvd.nist.gov/vuln-metrics/cvss
- https://www.first.org/cvss/user-guide

For an alternate approach, see BugCrowd's Vulnerability Rating Taxonomy (VRT):

- https://bugcrowd.com/vulnerability-rating-taxonomy

Appendix B: List of attachments

The following attachments are provided as additional documentation and reference:

- SSLScan
  - teams.nav.cloud.nais.sslscan.out