SECT-2158 - Navitabs - Tabs for Confluence (Cloud) Pentest Report

0x53c7

Atlassian's Security Testing Team

Engagement Team: @Ronel Gascon

Testing Dates: Feb 10, 2026 to Feb 16, 2026

- Executive Summary
  - Next Steps
- Scope
- Limitations or Constraints
- Testing Narrative
- Static Analysis
- Dynamic Analysis
  - Enumeration
  - Authentication
  - Access Control
  - Input validation
- Host Discovery
- Findings
- Tools Used
- Methodology
  - Scoping & Team Allocation
  - Preparation
  - Baseline Activities
  - Deep Dive Testing
  - Finding Analysis
  - Reporting
  - Closeout & Cleanup
- Appendix - Atlassian Internal

Executive Summary

Navitabs is a Connect on Forge app which lets users organise and navigate Confluence content with tabbed layouts similar to Confluence's menus. Users can create horizontal or vertical tab groups and populate them with Confluence pages, Whiteboards, Databases, Smartlinks, or external content via embeds, HTML, or iframes. With the Tab Wizard and tab macros, users structure complex information into clearer views, making long or multi-topic pages easier to read and navigate.

A blackbox testing methodology was used for Navitabs, as source code was not available for analysis.

The security assessment focused on the following areas:

- Outbound API requests made from Atlassian to the Navitabs domains were tested for authorisation and common injection vulnerabilities.
- The application-level access controls between users and licenses. The macros used by Navitabs allows users to create, update, and delete tabs within Confluence and testing looked for vulnerabilities which violated the expected access controls.
- User-controlled input fields were tested for client-side injection attacks such as XSS and HTML injection. Previous stored XSS vulnerabilities identified by the bug bounty program were also examined to determine if similar vulnerabilities have persisted since the patch.

No exploitable findings were found within the testing period.

Next Steps

Any vulnerabilities identified during this assessment have been recorded as tickets in the Atlassian Marketplace Security (AMS) project. Remediation timelines for these issues are governed by Atlassian's **Security Bug Fix Policy for Marketplace apps**, and all vulnerabilities must be resolved within the timeframes defined here: Security Bug Fix Policy for Marketplace apps

Please use the individual AMS tickets if you have questions or require clarification about specific findings. The ECOHELP ticket used to notify you of this test and deliver this report will remain the main channel for general communication about the engagement.

Once you have implemented and deployed fixes, update the relevant AMS tickets by moving the status to **Patched**, and if possible, leaving a comment with details of the remediation. The Security Testing team will review these updates and, where appropriate, perform targeted retesting to confirm that the reported issues have been successfully resolved.

For more information on how Atlassian manages Marketplace security vulnerabilities, see the FAQ: Vulnerability management for marketplace apps.

Scope

Methodology: Blackbox

Environment:

- A production instance of Confluence Cloud was set up to facilitate testing at https://rgascon2158.atlassian.net.
- A free trial license for Navitabs was utilised for testing.
- Version 2.10.0 of the app for Confluence Cloud was used - release date `Jan 21, 2026`

External infrastructure:

- **communardo.app** and its subdomains
- **bitvoodoo.app** and its subdomains
- **bitvoodoo.cloud** and its subdomains

Source Code: Blackbox testing was employed as the source code was not available for analysis.

Limitations or Constraints

N/A

Testing Narrative

Security testing involved general enumeration of the specified domains (e.g. nmap scans, subdomain enumeration, etc.) over the public internet, reading the documentation available on the vendor website for feature testing, and blackbox testing of the application itself through Confluence Cloud. The test primarily focused on dynamic testing as the application source code was not available, and automated tooling was used to scan any relevant client-side libraries which originated from the application.

Static Analysis

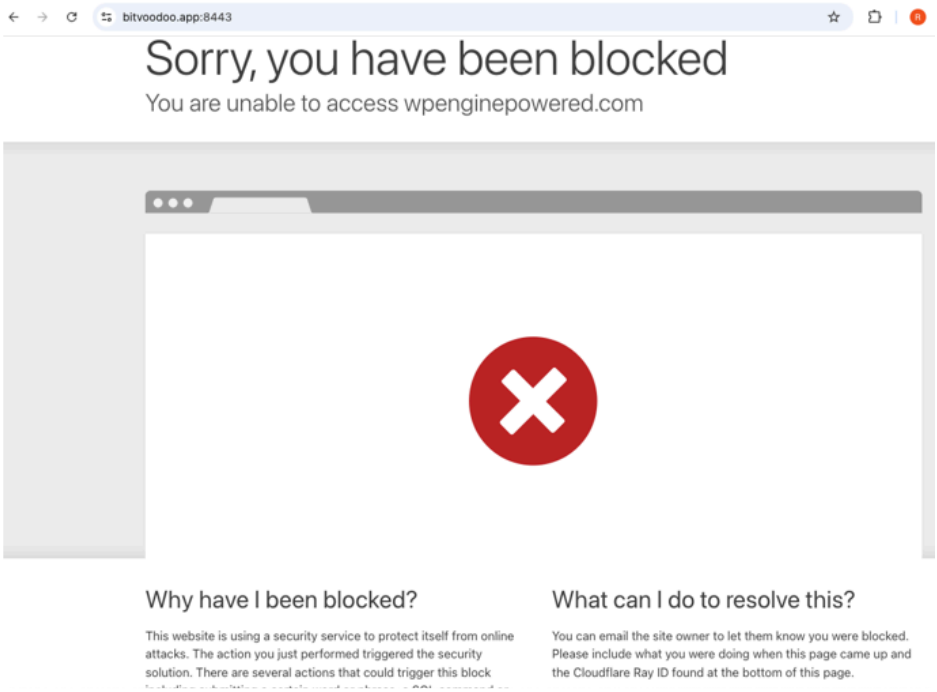
Client-side libraries and content loaded externally via iframes were scanned using automated tooling available within Burp Suite Pro. The results of these scans were manually inspected for any true-positive results, and further explored through dynamic testing. No issues were identified in this portion of the test.

Dynamic Analysis

Enumeration

The enumeration part of testing included reviewing the documentation available at `Navitabs - Navigation Macros for Confluence Cloud | Communardo App Documentation`, and using automated tooling such as **nmap**, **gobuster**, **sslscan** and various Burp Suite Pro extensions for content discovery. The aim of this phase was to flag suspicious services or information that is available for further analysis.

The **bitvoodoo.app** host exposed HTTPS services on ports 2087, 2096, and 8443. However these appeared to properly block unauthorised traffic unless the IP had been explicitly allow-listed.



Authentication

The exposed API endpoints primarily pointed to the domain owned by the previous owner of the application - **navitabs.aws.bitvoodoo.cloud**. The requests made to these endpoints returned user-specific data such as tab names and its contents, and were secured by context tokens in the form of JWTs. These authenticated endpoints were tested for common JWT confusion attacks in an attempt to bypass or circumvent authentication checks made by the backend. No exploitable issues relating to authentication was identified during the testing period.

While no issues were identified to actively exploit this behaviour, it should be noted that there were some API calls which included JWTs in GET request parameters.

Access Control

The application allowed site admins to issue application-level access controls for managing custom tab designs (e.g. restricting this feature to admins only). Testing confirmed that the controls put in place were effective and did not allow restricted users to create, modify, or delete custom designs unless they were explicitly assigned the proper permissions in the admin panel.

Input validation

As the primary use case for the application was embedding new elements within existing Confluence pages, input validation of the various user-controlled fields was heavily focused on. Previous bug bounty findings of stored XSS vulnerabilities also indicated that this would be where attacks were likely to start from.

Documentation of general use cases for the application's functionality facilitated testing each input field for potential client-side vulnerabilities. While no exploitable issues were found during the course of testing, the below observations were notable:

- It is possible to use the Tab Wizard macro to create tabs which embed iframes and HTML into the tabs themselves.
  - The iframes themselves were sandboxed, and inherent protections for modern browsers are in place to prevent active exploitation of this feature.
  - While embedding of HTML code also allows Navitab users to inject Javascript directly onto the page, the origin for the page is within a sandboxed domain **sandbox.aws.bitvoodoo.cloud** which offered little to no avenues for exploitation.
- All input that was identified and tested which gets reflected onto the page (e.g. tab titles) was HTML encoded through frontend libraries which prevented malicious input from executing in the user's browser.
- Inputs which have been severely malformed and could not be properly parsed by the backend did not render properly on the affected Confluence page, essentially rendering the page unusable.

Host Discovery

Host Domain / IP	Ext. ✅ Int. ⊘	Ports & Services Accessible
communardo.app	External	nmap output PORT STATE SERVICE 443/tcp open https 80/tcp open http
bitvoodoo.app	External	nmap output PORT STATE SERVICE 80/tcp open http 443/tcp open https 2086/tcp open gnunet

		2087/tcp open eli 2095/tcp open nbx-ser 2096/tcp open nbx-dir 8080/tcp open http-proxy 8443/tcp open https-alt 8880/tcp open cddbp-alt
bitvoodoo.cloud	External	nmap output PORT STATE SERVICE 443/tcp open https 80/tcp open http

Findings

N/A

Tools Used

The testing team employed a variety of specialised tools to support them during the assessment and ensure appropriate coverage of the scope. Some of the core security testing tools that were utilised during this engagement included:

- Burp Suite
- nmap
- sslscan
- Tenable Nessus
- Snyk
- Semgrep
- gobuster
- wpscan

Methodology

SecT's methodology is designed to achieve a consistent approach towards penetration testing which aims to deliver an exceptional level of security assurance within a defined scope and timeframe.

Our methodology has been shaped around three core pillars:

1. We have considered, leveraged, and built-upon industry recognised methods for undertaking penetration testing,
2. Our team is highly skilled, right for the job, and continuously improving to remain current with emerging threats; and,
3. Our output delivers a high-level of assurance as to the security posture of the in-scope system(s).

To that end, our processes are tailored from industry leading practices defined by such organisations as NIST and OWASP. We are big believers in white-box security testing and leverage as much information and context that we can gather, but maintain a flexible approach and are capable of performing black/grey-box testing where necessary. Our engagements begin with detailed scope preparation, documentation review, and personnel gathering that can assist our testing.

During the engagement, the testing team leverage both automated tooling and manual security testing to achieve a balanced, risk-based, approach between coverage and depth. We assess the in-scope system(s) for vulnerabilities and weaknesses including those relevant from the OWASP Top 10, CWE Top 25, and SANS Top 20. However, we also identify and define specific test cases as part of our scoping efforts to ensure a context-driven approach to testing.

Amongst our testing team we hold several highly relevant and well-regarded industry certifications including: OSCP, OSWE, OSED, CRTO, GRTE, and bachelor degrees in Computer Science or related fields.

As the core deliverable of our efforts, we take pride in our reporting and recognise that effective communication is critical to driving positive change. We take a flexible, client specific, approach to reporting to ensure we deliver our outcomes in the most meaningful way.

Each core phase of our approach is detailed below.



Scoping & Team Allocation

Each engagement is assigned an “engagement lead” who takes responsibility over the prioritisation, scoping, and and team allocation. This phase is led by SecT but can be performed

collaboratively with the client to ensure alignment with the engagement's objectives and outcomes. This phase involves several key activities, often including:

- Creating the necessary engagement artifacts such as the scoping document, report template, Jira ticket for internal tracking, Slack channel for internal communication, etc.
- Collecting scoping & engagement details such as timing considerations, prior incidents or high impact vulnerabilities, key points of contact, current risk drivers and areas of concern.
- Obtaining architecture information including the languages, frameworks, and technologies used, determining the most appropriate testing environment, and gaining access to system logs.
- Agreeing on a communication schedule and escalation avenues for the engagement.
- Reviewing existing materials pertaining to the system's security context such as design reviews, threat models, bug bounty findings, previous penetration test reports, and prior security incident write-ups.
- Hosting a scoping meeting as necessary to walkthrough responses to the scoping questionnaire, align on the rules of engagement, discuss timeframes and define roles and responsibilities.
- Finalising the scope including determining the most appropriate team members, the effort required, and locking in a schedule.



Preparation

The preparation phase involves a "pre-flight" checklist to confirm all prerequisites have been satisfied and both the testing team and client are comfortable with the engagement kicking-off. Key activities often include:

- Validating access to testing environments, source code, logs, and engineers.
- Ensuring all necessary engagement artifacts have been created (evidence folder, report template, Jira ticket, etc).
- Checking the engagement start and end dates and any timing constraints or considerations.
- Reviewing all engagement related documentation and clarifying any areas of ambiguity.
- Scheduling & hosting a kick-off call if required/requested.
- Notifying all stakeholders that the engagement is starting.



Baseline Activities

The actual penetration test begins with a comprehensive suite of tool-assisted scanning as well as manual familiarisation with the in-scope assets. These baseline activities are used to generate leads, identify potential hot spots, and capture any “low hanging fruit” within the scope, as well as ensure the testing team have a complete understanding of the system’s functionality and operation. Key baseline activities often include:

- Performing source code mapping to understand the code hierarchy, logging syntax, relevant git history, and input for static code analysis tooling.
- Exploring user stories manually, familiarising ourselves with the scope, and leveraging active scanning to help map the attack surface.
- Performing entry-point analysis and ensuring a comprehensive view of the attack surface can be obtained.
- Pragmatically breaking the scope and testing activities into logical phases relevant to the threats posed to the scope, and the time available for testing.
- Running a suite of static analysis tooling (both open-source and commercial as applicable) against relevant source code, including dependancy checking and secrets scanning.



Deep Dive Testing

The majority of the testing time is spent in this phase where manual testing and code review is focused on finding security vulnerabilities, weaknesses, and defining recommendations. Deep dive testing extends on the previous phase where preliminary dynamic analysis and static analysis has been run, to then focus on chasing down leads and undertaking priority test cases. Key testing activities & areas often include:

- Triaging of automated tooling results in a risk-based prioritised manner.
- Access control (AuthN/Z & Session management).

- Input validation & injection.
- Cryptography weaknesses.
- Auditing, logging, and error handling.
- Variant analysis.
- Infrastructure, configuration, and deployment weaknesses.
- Business logic issues.

The testing team utilise a comprehensive set of tools, techniques, and procedures to identify and exploit weaknesses within the assessed scope, intelligently including or excluding test cases based on the outputs of their automated and manual testing.



Finding Analysis

When/If gaps are identified, the testing team follow an approach to rationalise the result, determine the severity of the finding, and consider additional relevant context. Key activities often include:

- Demonstrating business impact through the development of a “Proof of Concept” (PoC) for the finding.
- Determining the severity of the finding following the industry recognised vulnerability severity framework; CVSS version 4.
- Performing root cause analysis and considering whether there are other (potentially broader or more holistic) recommendations that could be made.
- Notifying the client of Critical or High severity findings as soon as practical as to not delay remediation efforts.



Reporting

SecT recognises that the quality of our deliverables need to match the quality of our testing. As such, we work closely with our clients to ensure we are capturing what they need from us in our

reporting. This often includes a number of activities such as: report peer-reviews & draft deliverables, hosting a closeout meeting & walkthrough, and raising findings directly in the client's vulnerability management system if that's helpful. Typically, our reports contain at minimum the following information:

- Contact information for the testing team as well as information about who we worked with from the client's side.
- The dates that testing was performed on.
- Background information describing the assessed system.
- A detailed breakdown of the engagement scope, key objectives or test cases, and the approach taken.
- Core engagement tasks or phases and a breakdown of the activities performed within each.
- A "retrospective" to capture constraints or limitations that affected the engagement, or suggested priority items for next time.
- The assessment results detailing any findings with clear descriptions, severity ratings & justifications, proof-of-concept information, remediation advice, and relevant references or external material.
- Hyperlinks to any kick-off/close-out meeting recordings.
- Hyperlinks to relevant supplementary information including the Slack channel used for daily engagement communication and (if applicable) the shared drive containing additional artifacts.



Closeout & Cleanup

To complete the closeout of the engagement, the testing team will undertake a series of activities as relevant to ensure a smooth ramp down. These could include:

- Hosting a closeout meeting to walkthrough the deliverables, discuss any findings and recommendations, and gather feedback to be incorporated into the final report.
- Applying any updates that came from the closeout meeting.
- Delivering the final report(s) to relevant stakeholders.
- Cleaning up testing environments & archiving the engagement Slack channel.

- Requesting feedback from stakeholders on the process, people, and outcomes of the engagement.

Appendix - Atlassian Internal

Internal ticket for managing the penetration test. Not accessible/applicable to partners.

SECT-2158: Navitabs - Tabs for Confluence (Cloud) REPORT REVIEW