ADALOGICS

Fastify Security Audit 2023-2024

Security Audit Report

(Arthur) Sheung Chi Chan, Adam Korczynski, David Korczynski

2024-05-15

Contents

About Ada Logics

Ada Logics is a software security company founded in Oxford, UK, 2018 and is now based in London. We are a team of dedicated, pragmatic security engineers and security researchers that work hands-on with code auditing, security automation and security tooling.

We are committed open source contributors and we routinely contribute to state of the art security tooling in the fuzzing domain such as advanced fuzzing tools like Fuzz Introspector and continuous fuzzing with OSS-Fuzz. For example, we have contributed to fuzzing of hundreds of open source projects by way of OSS-Fuzz. We regularly perform security audits of open source software and make our reports publicly available with findings and fixes, and we have audited many of the most widely used cloud native applications.

Ada Logics contributes to solving the challenge of securing the software supply-chain. To this end, we develop the tooling and infrastructure needed for ensuring a secure software development lifecycle, and we deploy these tools to critical software packages. On the tooling and infrastructure side, we contribute to projects such as the OpenSSF Scorecard project as well as the Sigstore projects like SLSA and Cosign.

Ada Logics helps some of the most exposed organisations secure their software, analyse their code and increase security automation and assurance, and if you would like to consider working with us please reach out to us via our website. We write about our work on our blog. You can also follow Ada Logics on Linkedin, Twitter and Youtube.

Ada Logics ltd 71-75 Shelton Street, WC2H 9JQ London, United Kingdom

Project dashboard

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
(Arthur) Sheung Chi Chan	Auditor	Ada Logics Ltd	arthur.chan@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Matteo Collina	Fastify Maintainer	Fastify	matteo.collina@gmail.com
Amir Montazery	Facilitator	OSTIF	amir@ostif.org
Derek Zimmer	Facilitator	OSTIF	derek@ostif.org
Helen Woeste	Facilitator	OSTIF	helen@ostif.org

Executive summary

Ada Logics conducted a security audit of Fastify at the end of September 2023 to January 2024. The goal of the audit was to perform a holistic security assessment of the Fastify framework and its plugins with a particular focus on manual audit and its continuous fuzzing by way of OSS-Fuzz. The audit was facilitated by the Open Source Technology Improvement Fund (OSTIF) and funded by the Sovereign Tech Fund.

The audit was focused on the following modules:

- fastify
- @fastify/auth
- @fastify/basic-auth
- @fastify/bearer-auth
- @fastify/busboy
- @fastify/caching
- @fastify/circuit-breaker
- @fastify/compress
- @fastify/cookie
- @fastify/cors
- @fastify/csrf
- @fastify/csrf-protection
- @fastify/elasticsearch
- @fastify/fast-json-stringify
- @fastify/fast-json-stringify-compiler
- @fastify/fast-uri

- @fastify/formbody
- @fastify/http-proxy
- @fastify/jwt
- @fastify/middie
- @fastify/multipart
- @fastify/oauth2
- @fastify/reply-from
- @fastify/response-validation
- @fastify/secure-json-parse
- @fastify/secure-session
- @fastify/session
- @fastify/soap-client
- @fastify/static
- @fastify/under-pressure
- @fastify/websocket

We performed the following tasks during the audit:

- Developed a threat model
- Performed a manual audit of the code
- Developed and extended the continuous fuzzing set-up

In summary, during the engagement we:

- Developed threat models for the Fastify framework
- Performed manual auditing of each of the codebases
- Wrote a fuzz test suite and integrated Fastify into the OSS-Fuzz project

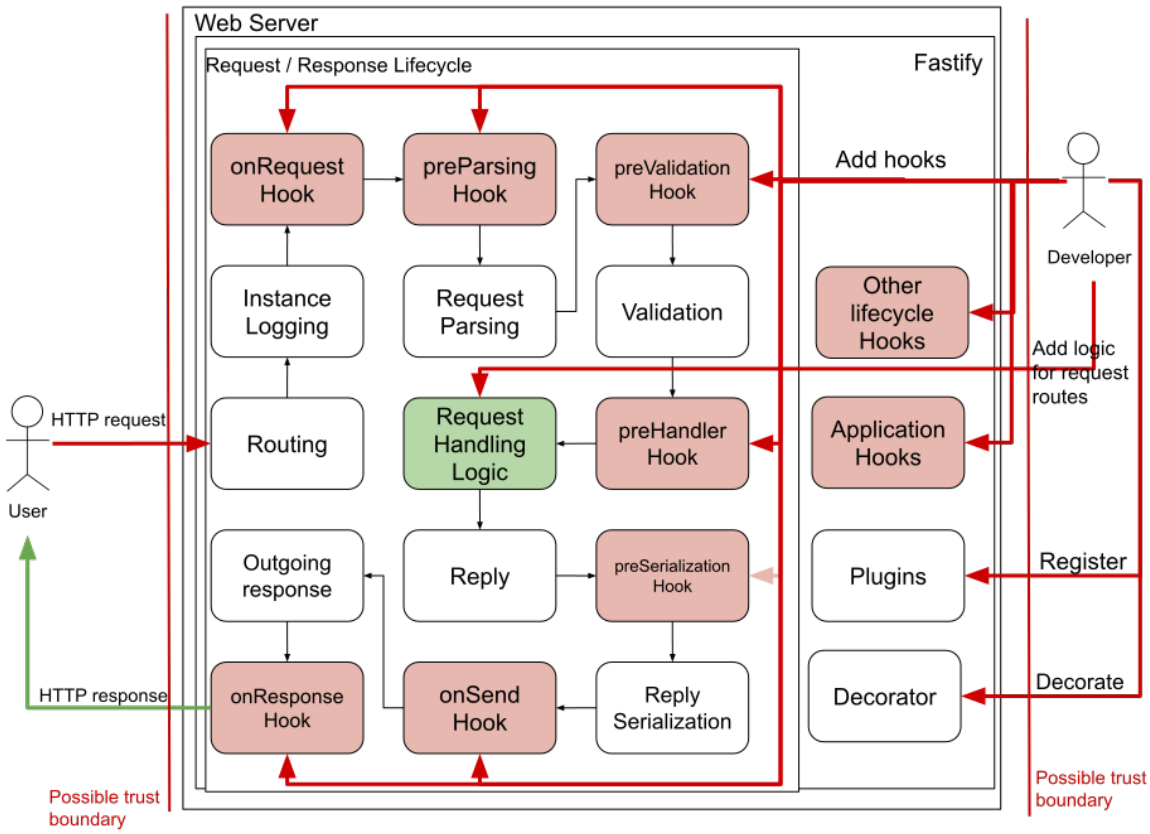
Threat model

Introduction

Fastify is a web application framework with a core product and a series of plugins to add-on to the core depending based on the adopters needs.This audit targets the core fastify framework and a list of core plugins which are maintained by the Fasitfy team.

Data flow of Fastify

In this section we present how data flows through the Fastify ecosystem. At a high level, we consider two categories: The first category is the configuration and development of the server-side web application based on the Fastify web framework. The second category is the lifecycle for handling untrusted HTTP requests and responses by the Fastify-based web applications and developed functions, hooks and plugins. The following diagram displays the request / response handling lifecycle and how the developer's configurations and developments could affect the data handling in different stages of the lifecycle.



Major components of Fastify

Content type parser

Content type parser is one of the major components of Fastify that handles parsing of raw HTTP request when the web application receives it. The core Fastify package only handles request payload types of json or plaintext. Additional types needed to be handled by other plugins or third party packages by registering them as new content-type parser in Fastify servers.

Decorators

Decorators are functions or parameters that could be added to a Fastify web instance for additional functionalities. Fastify provides decorators API for that purpose. After registering new functions or parameters through the decorators API, they can be used by either the Fastify general life cycle, or the in Hooks function or request handlers. Decorators are controlled by scope which is defined directly in a Fastify instance. Fastify opens the decorated objects to all components of the current Fastify instance of web application.

Hooks

When a web request is passed from the web server, the web server relays it to the fastify entry point and goes through 15 different life cycle steps. Some steps (red in the graph) are hooks that allow the developer to add specific handling logic at a given stage. For example, developers could add in custom parsing logic or plugins towards the preParsing hook in order to change the behaviour of how the raw http request parsing is done. This hook setting allows detailed customisation of each of the life cycle stages of the web application on top of the default process done by Fastify. Besides the general life cycle hooks, there are also additional hooks like `onError` that would trigger if case of errors or http response are aimed to be sent and that could also be customised by adding additional hooks with custom lambda function or logics. Besides, Fastify also provides a list of application hooks which are triggered when the status of the web application has been changed, like restarting or terminating of the whole web application.

Logging

Fastify core makes use of Pino Json logging for the default instances when handling request and response, or any state changes of the web application services. Logging can be configured by the server factory during web application starts up or provide additional functionalities via custom logic or

plugins. Custom logging can be added by the developer through hooks or request handlers. Logging level and other settings could be configured during web application impelemntation.

Plugins

Plugins are collections of predefined and predeveloped components, including route handling, decorators, functions or any JS objects that provide predefined web application functionality. The main idea of the plugin is to provide additional actions, features or functionality to Fastify instances and encapsulate all of the underlying actions. Fastify maintains a set of core plugins which provide common web application functionality. There is only one unified scope for a Fastify web instance. Fastify provides scope registration in order to create new scopes and encapsulate decorators and routing into different scopes. This could be used to separate the scope of components in the same Fastify web instances. Plugins can be registered in a web instance and executed in their own scope, providing extra functionality to the Fastify instance life cycle on handling web requests and responses. They can also change the default engine used in different request/response life cycles to a custom or plugin's plugin-defined engine.

Routing

Routing of HTTP requests is a core feature for every web framework. Fastify's routing follows the developer's configuration to determine which route or HTTP method is supported in this web application and which custom logic or handler functions are responsible for handling requests and generating responses. When the web server receives an HTTP web request and relays it to the Fastify web instance, the request is routed to the corresponding registered handler functions or error handlers. Developers could register custom handling logic or hook functions in order to control how the request and response are going through the web applications.

Fastify server configuration

Fastify core package exports the main server factory class in order for developers to customise some basic web application settings, including the port and host listened to by the server or additional logic to execute when the web service is started or terminated. These settings affect how the Fastify web instance executes on the servers.

Errors handling

Fastify core does not handle any of the errors, it just relays all the underlying errors towards the surface. Developers are meant to implement their own error handling functions and register them as callback functions in the Fastify web instance. When no callback functions is registered, a Promise object are returned when an error occurs. Both error handling callback functions or Promise object can be captured or handled by try catch blocks.

Scope of Fastify

Fastify takes up the responsibility when it receives a HTTP raw web request from the underlying web servers and ends when the final hook has been invoked after a web response (either a success or fail with error status code) is created and passed to the web server. The general procedure of the Fastify framework abstracts the need to handle each step of the web request and response processing. Developers only need to define the logic for the lifecycle steps by way of hooks, functions and decorators. Plugins are a combination of any of the above that has bundled together for easy implementation. They are treated like bundled services for some common applications of Fastify configurations.

Audit scope of Fastify

The audit scope of Fastify includes the core lifecycle to handle the http request / response, and a list of plugins managed by the Fastify team which provides general functionality for a web application.

Threat actors

A threat actor is an individual or group that intentionally attempts to exploit vulnerabilities or the infrastructure of the Fastify web application framework, its users, its source code platform, build processes, release cycles and its releases. Threat actors can target users of the web applications developed on top of Fastify framework.

Actor	Description	Have already escalated privileges
Web application users	Users of the web application developed by way of the Fastify framework	No

Actor	Description	Have already escalated privileges
Web application developers	The developers configured and developed the web applications by Fastify framework and plugins.	Yes
Other users of the servers	Other privileged users of the server hosting the web applications developed by Fastify framework.	Partly
Contributors to Fastify plugins	Contributors to Fastify plugins used by the web application developers.	No
Contributors to 3rd-party dependencies	Contributors to dependencies used by Fastify core framework and plugins.	No
Third-party maintainers	A third-party maintainer that turns malicious is a an actor that can attempt to compromise Fastify and its user base. This is a possible attack vector because Fastify trusts the maintainers of their third party libraries.	No

Threat actors' objectives

As Fastify web framework is meant to be the backbone of server-side web applications, the threat actors' objectives remain the same for general web applications.

Stealing information

Web applications developed by the Fastify framework open up ports to receive HTTP requests and process them. If the general handling logic of Fastify or the custom logic from developers contains vulnerabilities, threat actors could exploit these to steal information from the web applications or other services running on the web server, which may include databases, credentials or other local resources.

Gain access to servers

Threat actors may make use of vulnerable web applications to gain access to the underlying web servers.

Perform unauthorized or unexpected activities

Some services in the web applications and the underlying servers may require different levels of privileges. Threat actors may target the vulnerability in the Fastify web instance to perform privilege escalation or generate malicious requests, pretending to be initiated from legitimate users (Cross-Site Request Forgery).

Denial of services

Web applications are generally vulnerable to Denial of Services attack because it is expected to receive and process HTTP request from any source. Threat actors target vulnerable web applications without proper handling of invalid or large raw requests to consume a high amount of server resources, making the web applications and servers fail to handle requests from legitimate users.

Relay attacks to other users

Web applications generally store some information for further processing. It is no different for Fastify-based web applications. Threat actors may try to add or inject malicious code into the existing authorized web applications to relay the attacks or malicious activities to other users using the web applications. These Cross-site scripting attacks make the vulnerable web applications the media for relaying attacks.

Attack surface

An attack surface is a components that could be manipulated by a threat actor to perform unexpected or malicious activities on legit services. The attack surface of Fastify is similar to a general server-side web application. Including the following. 1. Fastify request / response life cycle
Fastify request / response life cycle handles web requests like general sever-side web applications from any users. It opens an entry point for untrusted users to perform several attacks on the web applications or underlying web servers. 2. Decorators / Plugins
Decorators or plugins provide additional functionality to the web request handling. If developers use a vulnerable plugin, it could open up new attack surfaces for attacking the web applications. In addition, wrong implementation of plugins or decorators could bypass some of the validation mechanisms of Fastify and open up more holes in the web applications. 3. Request URL routing The Fastify routing engine relays the raw web request to the designated handlers. If the routing is being polluted, it could redirect to a wrong handler and perform unexpected actions. 4. Resource storage Vulnerable handling of resource storage could make Fastify based web applications become a storage of redirect attacks

like Cross site scripting. 5. Dependency libraries Fastify implements some fast dependencies, like Pino Json logging, find-my-way Http routing, AJV json schema validator or fast json stringify serializer. These algorithm libraries provide the fast processing of the Fastify framework and could also be part of the attack surface target as they are integrated as a core part of the Fastify life cycle.

Manual Audit

The manual audit targets the core Fastify framework and a list of core plugins that are maintained by the Fastify team. Fastify plugins developed and maintained by third parties and that are not hosted by the Fastify organization (https://github.com/fastify) have not been in scope during the audit. The manual audit targets the core lifecycle in the core Fastify package for general HTTP request/response handling and also a list of Fastify plugins managed by the Fastify team. In general, the manual audit aims to look for common coding and security sinks and vulnerabilities to see if they are handled correctly and securely. As plugin registration is done by the developer and user of the Fastify framework, vulnerabilities from the incorrect configuration of plugins are not in scope. This is only limited to problems or issues that are triggered by the use of the plugins solely. Vulnerabilities that could be triggered by HTTP request/response handling because of the wrong configuration of the plugins are however in the scope of the audit. For example, if the developer registers a Fastify plugin with the wrong type of data and crashes the applications, that is not in scope. However, if a plugin introduces a new default request content type parser during plugin registration and it could crash the program if a malformed HTTP request is sent in, then it will be reported as an issue. The Fastify plugins is audited separately as a standalone library module, the scope does not include vulnerabilities that are introduced when the plugin is configured incorrectly or maliciously when registering or adopting in the core Fastify web instance or other applications.

Besides separate auditing of the Fastify plugin, there are some audits on the core Fastify web instance. Although most of the plugins do not have strict protection and checking of input types and values, some of them are still immune to attacks and vulnerabilities from processing untrusted data. The main reason for that is that most of them are used as a registered plugin for the core Fastify framework. The core Fastify framework and some security designated plugin does provide a strict set of validation against designated schema and deny unknown input name, type and values. That make most of those Fastify plugin does not requires additional strict protection of input as the input reaching the code of the plugin is already been checked and sanitized by the core Fastify web instance and are considered as trusted data. In the scope of the manual audit, although we only consider the plugins as a standalone applications, we do checck how it suppose to receive data and if the data is passed in from the core Fastify web instance or directly from the untrusted users. Thus if it is believed that the data is passed from the core Fastify web instance, we are considered that as trusted data. There may have certain false negative cases if the developer does not register correct plguins or configurations, but that is out of scope of the audit since it is up to the developer to ensure the correct configurations and plugins applications. In general, the core Fastify web instance already provides enough data protection, including protection against DOM-based injection or prototype pollution attacks.

In addition to the Fastify plugins that are meant to register to a core Fastify framework and to be used together with the web instance to handle HTTP requests, there are also some Fastify plugin, like

`fast-json-stringify`, that are only meant to provide additional or replacement utility functions for certain common actions. For example, the `fast-json-stringify` plugin provides a faster replacement of the `JSON.stringify()` function and the `secure-json-parse` plugins provides a secure replacement of the `JSON.parse()` function. Our manual audit also includes such plugins that are meant to handle untrusted data securely and safely.

We found that some of the Fastify plugins are working with the same set of source data from the HTTP request/response. They are called "sibling plugins". Registering multiple of these colliding sibling plugins to the same Fastify web instance could create an unexpected effect since the data may be consumed by one of the plugins and make the other plugin fail to retrieve the data and cause unexpected errors. Such action could also cause a race condition if the collision is not warned and some sensitive operations could be affected. For example, the Fastify-reply-from and Fastify-multipart plugins are a pair of sibling plugins. Fastify-reply-from wraps an HTTP request and forwards it to another web instance for handling while Fastify-multipart consumes and parses the data from the multipart form in the HTTP request. Both plugins will handle the multipart form while the Fastify-multipart plugin will consume the data in the HTTP request. Thus the Fastify-reply-from plugin has no guarantee that the forwarded HTTP request still has any part of the unconsumed multipart form data left in it. This could create a race condition and result in unexpected performance. As the core Fastify framework does allow multiple web instances to run at once without disturbing each other, the above problem only happens if those sibling Fastify plugins are registered together in the same web instance.

List of audited modules

Here we list the Fastify modules that we audited. The first one is the core Fastify framework module and following that are the Fastify plugins. Each of the modules in the following list uses its npm module name as an identifier.

Modules	Description
fastify	The core Fastify framework module
@fastify/auth	Plugin for managing different authentication modules
@fastify/basic-auth	Plugin that provides basic username:password authentication functions
@fastify/bearer-auth	Plugin that provides bearer authentication functions
@fastify/busboy	Plugin for parsing incoming HTML form data

Modules	Description
@fastify/caching	Plugin that provides server-side cache and Cache-header cookie control
@fastify/circuit-breaker	Plugin that breaks circuits in HTTP request routing
@fastify/compress	Plugin that provides compression utility functions
@fastify/cookie	Plugin that handles cookies
@fastify/cors	Plugin that enables CORS functionality
@fastify/csrf @fastify/csrf-protection	Plugin that enables automatic CSRF protection
@fastify/elasticsearch	Plugin that acts as a wrapper for Elastic Stack searching
@fastify/fast-json-stringify @fastify/fast-json-stringify-compiler	Plugin that provides a faster version of the native `JSON.stringify()` function
@fastify/fast-uri	Plugin that provides URI handling utility functions
@fastify/formbody	Plugin for adding a content type parser for x-www-form-urlencoded data
@fastify/http-proxy	Plugin for adding a proxy for request forwarding
@fastify/jwt	Plugin that provides JWT utility functions
@fastify/middie	Plugin for managing the middleware of Fastify
@fastify/multipart	Plugin for supporting the multipart type of data
@fastify/oauth2	Plugin that enables Oauth2 authentication
@fastify/reply-from	Plugin for request forwarding
@fastify/response-validation	Plugin that enables HTTP response validation
@fastify/secure-json-parse	Plugin that provides replacement of `JSON.parse()` with prototype poisoning protection
@fastify/secure-session	Plugin that enables secure stateless cookie session
@fastify/session	Plugin that enables general session
@fastify/soap-client	Plugin that provides functionalities to manage SOAP clients
@fastify/static	Plugin that manages static resource requests

Modules	Description
@fastify/under-pressure	Plugin that manages pressure load and “Server unavailable” status
@fastify/websocket	Plugin that enables basic web socket support

General List of items to look for

The below list represents a high level of the vulnerability classes that we audited for during this security audit.

1. Uncaught exceptions cause Denial-of-Service
2. Cross-site request forgery
3. Session/cookie fixation
4. URL redirection attack
5. Cross-site scripting
6. Command injection
7. Path/directory traversal
8. Regular expression Denial-of-Service
9. Zip-bomb/large file handling and resource exhaustion
10. Input escaping/sanitization
11. Timing/resource side-channel attack
12. Insecure serialization/deserialization
13. Remote code execution
14. Broken authentication/authorization/access control
15. Memory exhaustion
16. XML external entity attack
17. Dependencies vulnerabilities
18. Sensitive data exposure
19. Illegal static object reference
20. Weak cryptography
21. Prototype Poisoning
22. JSON Injection
23. WebSocket URL Poisoning
24. XPath injection
25. DOM Injection
26. DOM clobbering

Fastify plugin audit

The above vulnerability classes are both general-type vulnerabilities as well as classes relevant for only certain

@fastify/auth

Mmanages and applies registered authentication functions for the core Fastify web instance.

Common vulnerable components

This plugin only manages and applies registered authentication functions from other Fastify plugins or custom validation functions. The real authentication and validation functionalities are provided by those registered functions and plugins. Thus the only possible attack surface is from the registered authentication functions and plugins. As it does not process any DOM and JSON object from untrusted source, it is not vulnerable to client-side DOM based attacks or prototype poisoning.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓							✓		✓	✓			✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓		✓	✓				✓	✓

@fastify/basic-auth

The Fastify-basic-auth plugin is one of the authenticators for the Fastify-auth plugin. It mainly provides validation and authentication functions for authenticating users with username and password provided from an HTTP request authentication header.

Common vulnerable components

The main verification and authentication logic is provided as parameters during plugin registration. The possible attack surface comes from the parsing of the authentication headers and how the data is being passed to the registered processing functions. As it does not process any DOM and JSON object from untrusted source, this plugin is not vulnerable to client-side DOM based attacks or prototype poisoning.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓					✓		✓		✓	✓			✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓		✓	✓				✓	✓

@fastify/bearer-auth

The Fastify-bearer-auth plugin is one of the authenticators for the Fastify-auth plugin. It mainly provides a hook for custom bearer authentication functions.

Common vulnerable components

The main verification and authentication logic are provided as parameters during plugin registration. The attack surface exists in the parsing authentication request data. After code checking, it is believed that the handling does not have any viable vulnerabilities.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓					✓		✓		✓	✓			✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓		✓	✓				✓	✓

@fastify/busboy

Fastify-busboy parses incoming HTML form data from HTTP requests. It provides a handler and parser for accessing those data from the HTTP request for the core Fastify web instance.

Common vulnerable components

Multipart files path traversal

A common use case for HTML forms is to upload files which the server will handle through multipart file processing which requires reading of the source files with paths and URLs, this could make the process vulnerable to path traversal attacks that read files illegally or cover wrong files in the web server. The plugin code retrieves fields from the HTTP request directly and passes them to the request object for further processing. Although it does not have include any checking of the retrieved file path information, it just retrieves and passes the information to the developer without additional process. Thus if the developer does not check or use the file name incorrectly, it could create a possible path traversal vulnerability. It is the developer's responsibility to check or sanitise that information. Therefore, when we audit this plugin, we assume that this is not the vulnerabilities in the plugin.

DoS on form data

The data processed through HTML form generally comes from application users which we consider untrusted. As a result, it is possible to receive HTTP requests with large file sizes or malformed field parameters. Invalid or malicious data can crash the web instance or exhaust the memory of the web server and create a Denial-of-Service attack. Enforcing limitation for the maximum size of multipart fields and the maximum file size allowed is necessary. Otherwise, an attacker can send very long and large multipart requests with a high number of parts or multiple large files to attack the server. From the manual audit, we found that the default configuration does have a maximum limitation set and could be configurable by the developers.

Memory leaks

Sometimes during the HTTP request processing, the process could be exited, returned or halted before the end of the process because of different server errors, data validation or maximum limitation enforcement. For multipart data handling, the uploaded data file is stored in memory until all the parts have been received. Halting because of different reasons could result in memory leaking if those uploaded data are not cleaned when request halting happens. Memory leaking could be critical if an attacker launches a distributed Denial-of-Service which sends loads of requests with large multipart files that could trigger the early halting of the web request. We found that the plugin cleans the data when an HTTP request is ended, no matter if it is ended normally or by another halting approach.

Client-side injection

The Busboy plugin reads and parses form data and is exposed to different kinds of injection attacks, like DOM, JSON or command injection, and prototype pollutions if the data is redirect to the HTTP response without further sanitization. From the manual audit, we believe that this plugin only reads and parses the data and stores it without further processing. The core Fastify web instance is responsible for handling and validating the retrieved data with configured schemas before further processing. If considering the request handling only for this plugin, it is not vulnerable to client-side injection attacks. Also, as the core Fastify web instance only takes in prototyping during plugin registration of this plugin, the input data is trusted.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓			✓			✓		✓	✓				✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓			✓	✓	✓			✓	✓

@fastify/caching

The Fastify-caching plugin creates and manages the cache headers in HTTP responses. These cache headers are then returned by the core Fastify web instance in the HTTP response to control the client-side cache of specific pages or resources.

Common vulnerable components

This plugin only handles the creation and setting of the cache headers following the default or developer's configurations. The cache header is never parsed on the server side and is only affected on the client side when it is configured in the HTTP response. Thus there is no viable attack surface since the developer configuration is considered trusted and no other untrusted data is going through this plugin towards the core Fastify web instance.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓														

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓				✓	✓

@fastify/circuit-breaker

The Fastify-circuit-breaker provides circuit-breaking features for the core Fastify web instance. It wraps around each of the designated routes and monitors its route and failures, if a certain configurable threshold is reached, the plugin will halt the route and return server error response.

Common vulnerable components

Denial of service

The circuit breaker aims to halt problematic HTTP requests that could trigger infinite route redirection or a long time waiting for a response. If an attacker can find a vulnerable process in the web instance that is suffering from that, it can manipulate the circuit breaker to halt certain services in the web instance and create a Denial-of-Service situation. From the manual code audit, we found that the basic operations and threshold are configurable by developers and the real processing logic of each HTTP route and also controlled by the developer. Thus, the developer should correctly configure the plugin to ensure the only viable attack target is configured correctly to avoid unexpected breaking of request handling. Other than that, we do not consider any other attack surface.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓			✓					✓						✓

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/compress

Fastify-compress implements compression utilities and hooks for the core Fastify web instance for handling reply objects compression and the decompression of request payload. These utility functions allow the core Fastify web instance to handle object compression on request and reply objects during the HTTP request process.

Common vulnerable components

BREACH attack

BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) is a kind of CRIME (Compression Ratio Info-leak Made Easy) type security vulnerability that uses some of the timing side-channel observation to guess secret info that is used during the compression process or stored in the compressed objects. In general cases, the attackers target those compressed payloads with secrets or tokens, attempting to use different trials to retrieve those secrets or tokens for other attacks. This plugin is only limited to performing automatic compression of reply payloads and provides a set

of hook functions to allow decompression of data during different stages of the web processing life cycles. The real data in the HTTP reply objects to compress and what to decompress during the request handling is completely up to the developers. It is up to the developer to configure which information to include in the compression and in most cases, these compressions are not recommended for use on secrets or confidential objects.

Zip bomb or large payload

Since the plugin provides decompress functionality for HTTP request payload, it may be vulnerable to zip bomb attacks. In the manual audit, the plugin detects the types and payload size before doing payload decompression which mitigates the possible zip bomb from the request payload that causes resource exhaustion.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓							✓	✓		✓				✓

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/cookie

Fastify-cookie provides cookie management functionality for Fastify web instances. It supports two modes of cookie handling, one with plain cookies and the other with signed cookies to protect the integrity of cookies.

The plugin provides three configurable parameters, namely secret, hook and parsing options. If the secret is defined, cookie signing is used instead of plain cookies. The hook is used to define the trigger point for cookie parsing before the handler handles its value. The developer could pass in custom parsing functions or options for default parsing functions for the parsing of cookies.

Common vulnerable components

Node-cookie library dependency

The Node-cookie library is used for cookie parsing and serialisation. The plugin has been required by `plugin.js`. Only the `parse` and `serialize` functions are used by the plugin.

The parameters passed to `cookie.serialize` come from the developer's logic when they call either `serializeCookie` or `setCookie`decorated functions. Thus it is not controllable by the untrusted input from user request. The `cookieHeader` parameter passed to `cookie.parse` functions comes from raw user requests, thus it could contain untrusted input. The code from this plugin does not specifically validate or interpret the `cookieHeader`, but no known vulnerability report has been found for the underlying Node-cookie library, thus it is assumed to be safe for handling untrusted cookie parsing.

Cookie Creation

The only code that creates cookies in the reply is located in `plugin.js`.

```
1 reply.header('Set-Cookie', cookie.serialize(c.name, c.value, c.opts))
```

All cookies stored in `kReplySetCookies` are processed and create a `Set-Cookie header` in the HTTP response. Since the `cookie.serialize` results with parameters from developers are assumed to be trusted, this simple cookie creation logic should not be affected by cookie poisoning.

Cookie signing and unsigning

If a string secret has been provided during plugin registration, a default Signer object is created for handling the signing and unsigning of cookies. The code for deciding whether a Signer object is needed is defined in `plugin.js` as the following.

```
1 const isSigner = !secret || (typeof secret.sign === 'function' &&
     typeof secret.unsign === 'function')
2 const signer = isSigner ? secret : new Signer(secret, options.algorithm
      || 'sha256')
```

Here, the plugin checks if both the `sign` and `unsign` functions are provided in the secret, or if it is a **`false`** object to ensure `sign` and `unsign` functions are correctly defined. But there is an error in the checking logic that could cause sign and unsign functions undefined and cause errors. The `sign` and `unsign` method provided by the default `Signer` class uses the `node-crypto` package to do the signing with the default algorithm of SHA256 which does not have any known vulnerability report for the version adopted by the fastify-cookie plugin. Although the developer could choose the algorithm to use, it would be the developer's responsibility if they use a weak or broken algorithm. The default SHA256 is safe. The logic of the `sign` function from the default `Signer` class is provided by the `_sign` function. It takes only the cookie value, secret and algorithm as parameters, all of which come from the plugin registration configuration or developer's logic. Thus it handles trusted data.

The logic of the unsign function from the default `Signer` class is provided by the `_unsign` function. It takes in the signed cookie, secret and algorithm as parameters, which the secret and algorithm are coming from plugin registration. Although the `signedValue` comes from an untrusted HTTP user request header, the validation of the signature is done to ensure the integrity of the cookie. Thus the

logic is still able to detect possible problems and can deny malicious changes to cookies.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓		✓		✓	✓		✓	✓	✓		✓			

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓			✓	✓			✓		

Issues found

#	ID	Title	Severity	Fixed
1	ADA-FASTIFY-2023-1	Weak signing key in default signer	Informational	Yes

@fastify/cors

Fastify-cors provides CORS header management for the fastify web instance. The plugins allow turning on and off of the cors features and specific configuration for the methods and origins that are allowed to use CORS in the web instance domain. Some default CORS headers to be returned in some conditions could also be configured.

Common vulnerable components

Origin parsing

Origin parsing is needed to determine if an origin header in the raw HTTP request from the user is a CORS-allowed origin that is valid for `Access-Control-Allow-Origin` headers to be added in the HTTP reply. As the origin header is located in a raw HTTP request which is untrusted, it could cause problems if the origin parsing is not done correctly. The matching of origin is done in the `isRequestOriginAllowed` function.

The `allowedOrigin` is determined by the origin parameter provided by the developer during plugin registration. Only if the provided origin parameter is a regular expression pattern or custom function will the logic handle the untrusted input of reqOrigin. If the regular expression pattern is bad or if the

provided function handles the `reqOrigin` incorrectly would cause security problems like REDOS. Since the origin parameter is provided by developers, the logic of this plugin is assumed to be safe.

CORS headers creation and pre-authorization CORS attack

Subsequence request handlers on the server side and the web browser on the client side consult the CORS headers to determine if a cross-origin request could be initiated. Thus the CORS-related header must be set as expected by the developers. The CORS headers are set by two functions which are triggered by the hook. The function `addCorsHeaders` adds CORS-related headers that are directly affecting the permissions of the cross-origin service.

In general, the untrusted data that is controllable by a malicious user is the origin header in the raw HTTP request and all other information comes from the developer's configuration.

Websocket Poisoning

WebSocket connections are generally long-lived connections. HTTP requests/responses can be sent at any time during the live time of the web socket connections. As it is a long-lived connection, if an attacker could steal (or legitimately retrieve a web socket cookie of the super-domain/sub-domain of the targets), they could control the whole web socket connection. Checking for origins and cross-origin requests is necessary to deny this kind of attack. From the manual audit, we understand that this plugin does allow the developers to enforce strict cross-origins request control and thus it is believed that this plugin could help defend against WebSocket attacks with cross-domain cookies or requesting a web socket with poisoned URL pointing to sensitive sub-domains/super-domains.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓			✓				✓		✓					

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓		✓	✓		

Issues found

#	ID	Title	Severity	Fixed
3	ADA-FASTIFY-2023-3	Possible regular expression DOS in Fastify-Cors	Informational	Yes

@fastify/csrf @fastify/csrf-protection

Fastify csrf and csrf-protection provide a cryptographic way to create CSRF tokens to protect users and the fastify instance from Cross-Site Request Forgery attacks.

Common vulnerable components

Cookie Tossing

This is an old issue discovered by previous auditing and is fixed by adding custom user information in the token. Currently, additional user info is configurable by developers to ensure cookie tossing does not exist.

CSRF token freshness

A CSRF token is a key part of CSRF protection. When a web application receives a request from a usere, the existence of a legit CSRF token can prove that the request is indeed initialized by the legitimate users. This helps prevent the possible CSRF attack in which an attacker sends a request on behalf of a legit user which is not noticed by the user or pretends it to be a legit request from that user. Since this token is an important factor in the CSRF protection plugin, its freshness is also a concern. If an attacker could steal a CSRF token and replay it with the previous legitimate request, the CSRF protection is broken. Thus the CSRF token freshness is important to have good CSRF protection. After auditing the code for token handling, it is believed that if this plugin is integrated with a fastify session or fastify secure session, it will be a session-based token. It does not provide any time validity options nor allows per-request tokens. Thus if the Fastify web instance is hosted on HTTP but not HTTPS, the CSRF token could be leaked and the user could be vulnerable to a token replay attack since the token is not renewed per request.

Predictable CSRF token

The CSRF token should be able to identify if the request is indeed initialized by the user. If an attacker could craft a CSRF token offline, then the CSRF protection is broken. A possible problem found in integrating the csrf-protection plugin with the fastify-cookie plugin could mean both the secret and token sent to the user which makes the CSRF token from secret predictable and craftable. This is possible because validation of secrets and tokens solely depends on client-provided information when integrating with fastify-cookie and don’t seem to have configurable settings.

CSRF token identity linkage

A CSRF token should link securely to a users identity to maintain CSRF protection. We believe that the CSRF token integrated with the Fastify-session plugin does connect to identity and does not use a general CSRF token pool for validation. On the other hand, the CSRF token integrated with the Fastify-cookie plugin is not linked to user identity and could be vulnerable if an attacker could control the secret cookie value for altered CSRF token generation.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓	✓					✓		✓				✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓			✓	✓	✓			✓	✓

@fastify/elasticsearch

The Fastify-elasticsearch plugin provides functionality for every part of the core Fastify web instance to access and manage the same Elastic Search client with the support of the upstream Elastic Search module.

Common vulnerable components

The plugin is found only to initialise the Elastic Search client through the upstream Elastic Search module by passing on default or developer-configured options. It does not have many custom operations and all untrusted data are passed directly to the upstream module. Thus the only viable attack surface is targeting its dependency through the upstream Elastic Search module. The Elastic Search module dependency version used by this plugin has one known vulnerability that could leak data through sensitive logging if the configuration is not correctly set. This could affect this plugin.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓	✓					✓		✓				✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓			✓	✓	✓			✓	✓

@fastify/fast-json-stringify @fastify/fast-json-stringify-compiler

The Fastify-fast-json-stringify plugin provides a faster version of native `JSON.stringify()` function for coverting JSON object to string.

Common vulnerable components

Regular expression Denial-of-Service

This plugin does not need registration to the core Fastify web instance. It only provides an exported module for transforming JSON objects to string following a preset schema. It also provides additional functionality to validate if the provided JSON object match the configured schema. Since it is not necessary to be used in a core Fastify web instance, the major focus of the audit is the vulnerability in the JSON object conversion and schema validation process. To parse or process string patterns, a certain type of regular expression matching is generally used. Tthe default regular expression used in the process is not vulnerable to Regular Expression Denial-of-Service. The input is mostly validated against the developer-provided schema, and the matching is strict and does not rely too much on regular expression.

String/command injection

In general, if the JSON object is untrusted, an attacker can attempt to provide a malformed JSON object to perform an injection attack. From the manual audit, we have observed that this plugin does not accept random JSON object. The JSON object is validated against the configured schema and confirm the fields and types are matched with the schema before parsing it to String. This strict control is believed to deny malformed input if the developer provides a strict and correct schema. Since this plugin is only meant to provide the functionality when the developer register the plugin with correct schema, it is considerede safe from injection attack on its own.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓			✓	✓		✓		✓		✓			

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓	✓				

@fastify/fast-uri

The Fastify-fast-uri plugin provides a set of utility functions for processing URI.

Common vulnerable components

Regular expression Denial-of-Service

This plugin does not need registration to the core Fastify web instance. It only provides some exported module functions for processing the URI of a set of supported schema. For example, it implements functions to parse or serialise URI into JSON objects. Since it is not necessary to be used in a core Fastify web instance, the major focus of the audit is the vulnerability in those URI handling functions. To parse or process URI, a certain type of regular expression matching is generally used. From the manual audit, we found that some of the patterns are vulnerable to Regular Expression Denial-of-Service if an evil string or input is passed to the functions and matches it with those vulnerable patterns. Since this plugin can be used by any node.js module, the possibility of Regular Expression Denial-of-Service could affect the project adopted by this library and call the function that uses the vulnerable Regular Expression pattern for input matching.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓							✓		✓					

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify-formbody

The Fastify-formbody plugin adds a third-party content type parser for x-www-form-urlencoded data to enable parsing of x-www-form-urlencoded data in the Fastify instance.

Common vulnerable components

This plugin registers a third-party content parser for form body query string without doing any validation or parsing itself. It does not have any viable possible attack surface. The only possible attack surface is from the parser used. For an attack vector to be possible, either the developer has customised the plugin with a vulnerable parser, or the default parser used is vulnerable. From our auditing, we found that the default parser used (`fast-querystring@1.0.0`) does not have any known vulnerabilities.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓														

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/http-proxy

The Fastify-http-proxy plugin automatically forwards all HTTP requests received by the core Fastify web instance with a configured prefix to an upstream instance while preserving the lifecycle hooks.

Common vulnerable components

Parsing of untrusted request parameters

This plugin relies on the developer's configurable prefixes and upstream destination to determine which HTTP requests need to be forwarded and where they need to be forwarded to. This plugin only relies on trusted data for automatic HTTP request forwarding, and the only processing of untrusted data is when matching the given prefix with the request URI in the HTTP request. Thus the only viable attack target for an attacker is to provide a malformed URI, attempting to create a Regular Expression Denial-of-Service. From the our audit, we found that the prefix matching for the HTTP requests redirection is not vulnerable to Regular Expression Denial-of-service.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓														

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/jwt

The Fastify-jwt plugin provides JWT (JSON Web Token) decode, sign and verify features with support from the default `fast-jwt` library. These functionalities provide an additional layer of security in terms of confidentiality and integrity to the core Fastify web instance.

Common vulnerable components

Fast-JWT library dependency

The main cryptographic operations of the Fastify-jwt plugin rely on the upstream `fast-jwt` library. If the upstream library contains vulnerability or weak cryptographic algorithms, it could be used by an attacker as an entry point to attack the security features provided by this plugin.

Force no verification mode

In some settings, the JWT engine does allow specifying the type of keys within the JWT headers. This opens a sink for an attacker to use a `none` type (which does not match the provided random key) to force the JWT to work in no verification mode. This allows the attacker to bypass the JWT header verification. After checking the code for parsing the JWT header (handled by `fast-jwt`), we found that this plugin (and its upstream) auto-detected algorithm from the provided key and failed the verification if the provided algorithm is not matching with the auto-detected algorithm, thus it is not possible to force “none” algorithm and bypass the verification.

JWT header injection

In some settings, the JWT engine does allow specifying custom parameters with the headers. This opens a sink for an attacker to attach its key and point the verification key to that uploaded key. In this situation, the attacker can bypass the verification with header injection and a self-key signing attack. After auditing the plugin, we have found that the plugin ignores all parameters except for those which are necessary. This avoids header injection attacks.

JWT algorithm confusing attack

In some settings, the JWT engine does allow specifying a specific key location for the verification. This opens a sink for an attacker can claim that the request is signed with a symmetric key (but for real it is signed with an asymmetric key). In this situation, the attacker can sign anything because the false assumption that the JWT header is signed with a symmetric key makes the web application use the same public key for encryption and decryption. Thus the attacker can bypass the verification. After

auditing the plugin, we have found that the plugin detects the algorithm from keys and throws an error if the key algorithm does not match the provided algorithm from the JWT header. This setting avoids algorithm confusion attacks.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓		✓					✓		✓	✓			✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓									

@fastify/middie

Fasitfy-middie register additional middleware to different lifecycle hooks on the core Fastify instance and provide a decorated function in the core Fastify instance to access and use the registered middleware.

Possible Regular Expression Denial-of-Service from dependencies

After plugin registration, the plugin relies on a decorated function to access and use the registered middleware. From the manual audit, we have found that the Fastify-middie plugin relies on the `pathToRegexp` function from the `path-to-regexp` dependency module to access a target middleware. After some searching, we believe that the `pathToRegexp` will return a regular expression pattern that is vulnerable to a Regular Expression Denial-of-Service attack if the path used for the matching is malformed. Since the path is possible to come from untrusted input, this plugin is vulnerable to Regular Expression Denial-of-Service attacks.

Path traversal

To access, use or manage a target registered middleware from the core Fastify web instance, a path prefix pointing to the target middleware is needed. The path prefix is provided to the decorated function registered by the Fastify-middie plugin. Since the path is possible to come from untrusted input, it could contain special characters that traverse into illegal directories. From the manual audit, we believe that the path matching is only done by the regular expression from the `pathToRegexp` which correctly rules out paths with unexpected path traversal characters like ~ or `...`

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓						✓	✓		✓					

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓								

Issues found

#	ID	Title	Severity	Fixed
7	ADA-FASTIFY-2023-7	Possible regular expression DOS in fastify-middie dependency	Informational	Yes

@fastify/multipart

Fastify-multipart handles multiple request bodies from HTTP requests. It provides a handler and parser for accessing the data from the HTTP request for the core Fastify web instance.

Common vulnerable components

Multipart files path traversal

Multipart file processing may require reading of the source files with paths and URLs, this could make the process vulnerable to path traversal attacks that read files illegally or cover wrong files in the web server. The plugin retrieves those fields from the HTTP request directly and passes them to the request object for further processing. Although it does not have any checking of the retrieved file path information, it just retrieves and passes the information to the developer without additional process. Thus if the developer does not check or if they use the file name incorrectly, that could result in a possible path traversal vulnerability. It is the developers responsibility to check or sanitise the input.

Possible DOS on fields and file size

Multipart form body handling is meant to handle file data from untrusted sources, thus it is possible to receive HTTP requests with large file sizes or incorrect field parameters. Attackers could attempt to crash the web instance or exhaust the memory of the web server by sending invalid or malicious input. Enforcing limitation for the maximum size of multipart fields and the maximum file size allowed is

necessary. If these values are not limited, an attacker can send very long and large multipart requests with a high number of parts or multiple large files. From the manual audit, we found that the default configuration does have a maximum limitation set and could be configurable by the developers.

Possible memory leak

Sometimes during HTTP request processing, the process could be exited, returned or halted before the end of the process because of different server errors, data validation or maximum limitation enforcement. For multipart data handling, the uploaded data file is stored in memory until all the parts have been received. Halting because of different reasons could result in memory leaks if the uploaded data is not cleaned when request halting happens. Memory leaking could be an issue if an attacker launches a Distributed Denial-of-Service (DDoS) which sends loads of requests with large multipart files that could trigger the early halting of the web request. We found that the data is cleaned when an HTTP request has ended, no matter whether it is ended normally or by another halting approach.

Client side injection

The multipart plugin handles multipart form data. It is susceptible to different kinds of client-side injection attacks, like DOM, JSON or command injection, and prototype pollutions if the data is not sanitized and used directly or used for response generation. From the manual audit, we understand that this plugin only reads and parses the data and stores it without further processing. The core Fastify web instance is responsible for handling and validating the retrieved data with configured schemas before further processing. Thus if considering the request handling only for this plugin, it is not vulnerable to client-side injection attacks. Also, as the core Fastify web instance only takes in prototyping during plugin registration of this plugin, those data are assumed to be trusted.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓			✓			✓		✓	✓				✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓	✓		✓	✓	✓

@fastify/oauth2

Fastify-oauth2 is an OAuth2 wrapper on the NodeJS simple OAuth2 library for the fastify framework.

Common vulnerable components

Token retrieval and authorization

The major authentication process for the OAuth2 protocol is done by the third-party OAuth2 service. To gain user information and authorize a user for certain services, an OAuth2 token is used for communicating with the third-party OAuth2 service and accessing certain security properties and parameters. From the manual audit, it is understood that the plugin passes on the configuration, including callback URL, secret or Oauth2 service URL from developers to the underlying NodeJS simple OAuth2 library without further interpretations. This plugin works like a wrapper for the node.js OAuth2 library and relies its security property on the plugin itself. Since the route for the callback URL and how to store the retrieved token is provided by the developer and the plugin does not have logic to handle the callback from the Oauth2 services, it is believed that the plugin does not take on interpretation and responsibility for handling the storage and security of the OAuth2 authentication process and the tokens.

Token revocation

Token revocation is another important key security feature of the OAuth2 service. When a user chooses to logout from the service, or believes that their credentials or tokens are leaked, then the token needs to be revoked from the third-party OAuth2 service to avoid further access to services with the identity linked with the token and the authorized status should also be revoked. From the manual audit, we understand that the plugin just relays requests, and does not store or record any of the OAuth2 process and tokens. The security and correctness of the token revocation process rely on the developers configuration and the third-party OAuth2 service providers.

Credential leakage

Credentials are used for authentication on the OAuth2 service. Since this plugin only relays the users to the developer-configured authentication URL from the third-party OAuth2 service providers, it does not read, process or store any of those credentials. Thus those data are not handled from this plugin. The plugin does provide a callback URL route for the third-party OAuth2 service provider to handle the redirection of the user from the OAuth2 service back to the web instance. Since those settings are provided by the developers and can be configurable. thus it is believed that those sensitive settings do not come from untrusted user input nor allow re-registration. Thus there are no viable targets through this plugin for illegal access of services or credentials.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓	✓	✓				✓		✓	✓			✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓		✓	✓					

@fastify/reply-from

The Fastify-reply-from plugin automatically forwards all HTTP requests received by the core Fastify web instance on a specific route to another core Fastify web instance. The request is then processed by another Fastify web instance and the web instance that registers the plugin will wait for the reply and relay it to the user. It relays the request and response for the configured route between the user and another Fastify web instance.

Common vulnerable components

Parsing of request and response parameters

When an HTTP request is received in the Fastify web instance that has registered the Fastify-reply-from plugin, the request parameters and other request details are parsed and are then used to create another virtual HTTP request for sending to the other Fastify web instance. The same logic has been used to repackage the HTTP response received from the other Fastify web instances before sending them back to the user.

Client side injection

The Fastify-reply-from plugin does parse and repackages received HTTP requests. It also parses and repackages the HTTP response received from the other Fastify web instance before sending it back to the users. If the input is untrusted and is returned directly to the user without validation or sanitization, it could open up for client-side injections. From the manual audit, it is understood that the parsing process does not have any known vulnerabilities as mentioned above. The data is not processed by the web instance that has registered this plugin; instead, it is the other Fastify web instance that handles requests to ensure no untrusted data is included directly in the HTTP response without sanitization. With this assumption, it is assumed that the HTTP response from the other Fastify web instance is trusted. Therefore, this plugin is not directly vulnerable to those attacks.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓					✓				✓					

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓	✓		✓	✓	✓

@fastify/response-validation

The Fastify-response-validation plugin enables automatic response validation by a configurable schema. This plugin ensures the response fulfills the necessary schema structure.

Common vulnerable components

This plugin only accepts the schema from the configuration and adds hooks to validate the HTTP response before it is serialized to ensure it follows the configured schema. The validation is done by upstream ajv-related modules. The plugin itself does not interpret either the configured schema or the HTTP response. Also, the data passing through this plugin is all from the processing of the core Fastify web instance thus it is considered trusted.

Items audited

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓														

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/secure-json-parse

The Fastify-secure-json-parse plugin provides a replacement for native `JSON.parse()` function for covering string to JSON object with prototype pollution protection. It is presented as an opposite operation with the Fastify-fast-json-stringify plugin.

Common vulnerable components

Regular expression Denial-of-Service

This plugin does not need registration to the core Fastify web instance. It only provides an exported module for transforming string to JSON object following a preset schema. Since it is not necessary

to be used in a core Fastify web instance, the major focus of the audit is the vulnerability in the JSON object conversion. To parse or process string patterns, a certain type of regular expression matching is generally used. From the manual audit, it is believed that the default regular expressions used in the process are not vulnerable to Regular Expression Denial-of-Service.

Prototype pollution

This plugin helps to transform a string into a JSON object. In general, if the string is malformed, it could contain a special character that makes the parsed JSON object vulnerable to prototype pollution. The plugin contains mechanisms to ensure only the expected JSON field is parsed and ignores other unknown fields, which makes the parsing process not vulnerable to prototype pollution on the resulting JSON object. The plugin also provides a safe parsing option to wrap around possible exceptions and to ensure no unexpected exceptions are thrown to the user. This setting could avoid crashing the web instance with unexpected exceptions.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓			✓	✓		✓		✓		✓			

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/secure-session

The Fastify-secure-session plugin adds session-handling functionalities to the core Fastify web instance. These functionalities include session generation, session data storage, session data reading and the destruction of sessions. When registered, this plugin provides a set of hooks and decorated functions for the developer to manage the session in the core Fastify web instance. It is similar to the Fastify-session plugin except that the session is stateless and stored in an encrypted cookie on the client side.

Common vulnerable components

Key generation and cryptographic algorithm

This plugin is different from the Fastify-session plugin: Instead of storing all session data on the server side and setting a session ID cookie in the HTTP response to make a stateful communication, this plugin provides a stateless secure session implementation which stores the whole session in an encrypted

cookie and sends it to the client side. Since the whole session is encrypted and sent to the client side, all the session data will reach an untrusted zone out of the control of the core Fastify web instance. To protect the confidentiality of the data, the encryption key (and decryption key) or the cryptographic algorithm used is required to be strong. From the manual audit, we understood that the key generation is provided by utility functions of the sodium-native library. The default cryptographic algorithm is strong and the key is generated and provided by the developer during plugin registration. It also supports key rotation with multiple keys. The key generation and cryptographic algorithm used are strong.

Data storage

When a Fastify instance with this plugin added receives an HTTP request with the encrypted session cookie, the Fastify instace decrypts and validates that cookie. Fastify then creates a new server-side session for the core Fastify web instance to process using the decrypted data. At the end of the HTTP request processing, the data is encrypted and stored in a session cookie before sending the HTTP response. The temporary server-side session storing the data is then destroyed. All the session data is encrypted and stored in the secure cookie after the request is completed and a new session is created with the decrypted data from the secure cookie passing in within a request. In addition, a nonce exists as part of the session data which Fastify uses for session liveness validation.

Session validity

Since this plugin manages a stateless session, it is important to have limited validity for the session cookie and to distinguish if the session is being destroyed or not. From the manual audit, the session destroy logic only applies to the temporary server-side session that is used during the HTTP request processing period. An attacker could potentially reuse a session cookie which may be destroyed (within the nonce period) because the server does not track the state of the session.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓	✓									✓		✓	

#16	#17	#18	#19	#20	#21	#22	#23	+#24	#25	#26
	✓	✓		✓	✓					

Issues found

#	ID	Title	Severity	Fixed
9	ADA-FASTIFY-2023-9	Possible reuse of destroyed secure session cookie	High	Yes

@fastify/session

The Fastify-session plugin implements additional session-handling functionalities to the core Fastify web instance. These functionalities include session generation, session data storage, session data reading and the destruction of sessions. When this plugin is registered, it provides a set of hooks and decorated functions for the developer to manage the session in the core Fastify web instance.

Common vulnerable components

Session fixation

In general HTTP sessions, the data is stored server-side and is separated by users with a separate session ID. The session ID is used to identify different users using the web instance. In general, the session ID is set via a cookie in the HTTP response after session creation. If an attacker could guess or reuse the session ID cookie, it creates a Session fixation situation in which the attacker could access or control data on the server side which is linked to the original user of the session. Fastify-session uses the node.js default random byte generation with `base64url encode`, thus it is assumed to be secure. As the plugin only provides hooks and decorated functions for developers to handle the lifecycle of the session and the data storage and retrieval in the session, the developers are responsible for controlling the life cycle of the session and the data stored within. This plugin is only responsible for the real actions handling the session when the decorated functions are called. An example of session fixation happens on Fastify-passport which does not regenerate the session ID before and after authentication. This allows an attacker to reuse the session ID cookie. However, since the session lifecycle is controlled by the developer, it is believed that this plugin is not vulnerable to session fixation.

Session cookie

One of the important factors for maintaining identity linked to the session is the cookie that stores the session ID. When a session is created, a session ID cookie is automatically set in the HTTP response object with the Fastify-cookie plugin. From the manual audit, the session ID cookie is maintained by the Fastify-cookie plugin, it does provide secure cookie settings like cookie signing, validating, serialising and deserialising. Although a session ID needs to be generated every time, it is still safe because it is the developer using this plugin determines the lifecycle of the session and when to regenerate or destroy a session.

Session persistence hash

Sometimes, if the data in the session has a long expiry period, it may be more vulnerable to session fixation because the attacker has a much longer time to guess for a valid session ID and change those data on the server side. There is an additional protection for session data integrity by adding a session persistence storage hash. The hash is calculated on the whole session and could detect unexpected changes to the session and ensure the session is different when regeneration of session ID is needed. From the manual audit, we found that the session persistence hash is generated with a weak algorithm. Although it is not easy to access the session without the correct session ID, it still poses a threat by not using a more secure cryptographic message digest algorithm.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓	✓	✓									✓		✓	

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓		✓	✓					

Issues found

#	ID	Title	Severity	Fixed
10	ADA-FASTIFY-2023-10	Possible use of weak SHA-1 algorithm for session persistent hash	Informational	Yes

@fastify/soap-client

The Fastify-soap-client plugin provides decorated functions to access a soap client instance through the core Fastify web instance with the support of the upstream soap client module.

Common vulnerable components

The plugin only decorates functions to allow the core Fastify web instance to access functions and modules from the upstream `soap` client library module. Most of the provided decorated functions are only wrappers which relay the request to the upstream functions. As all untrusted data is passed directly to the upstream `soap` client module, the only viable attack surface is targeting the upstream soap

client module. We found that one of the exported functions from the soap client module is vulnerable to regular expression Denial-of-Service when handling malformed input relayed to it. This could exhaust the memory and resources which then crashes the web instance and causes Denial-of-Service.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓							✓				✓			

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓									

@fastify/static

Fastify-static provides access to and sending of local static files in the web server. For example to return a web page template or resources when handling web requests.

Common vulnerable components

Path traversal of static files

In general, HTTP request for accessing static files in the web server, like graphics or documents, requires specific paths. If the path is not validated or sanitised correctly, it could point to the illegal location by path traversal constant and leaking files from inaccessible locations. From the manual audit, we found that the decorated functions of the static plugin retrieve a path prefix, root directory and constraint (like file whitelist wildcard string or allowed prefix) from configuration and use them to manage what static files directory could be accessed. Besides, the target files' path/name is passed in from the developer logic to the decorated function of the plugin. It is the developers responsibility to check the validity of the path. In addition, the plugin throws an error if the file name ends up traversed outside of the root directory from the configuration or violates the constraints option set. Thus path traversal outside of the designated directory is not possible. It is possible to retrieve static files with a path or file name directly from HTTP requests. That could cause path traversal problems but it is the developer's configuration to either whitelist which files or directories could be accessed or sanitise user input before using them as static files retrieving path.

Illegal directory listing

Similar to path traversal, sometimes it is sensitive to allow the user to list the remote directory from a web request by providing a static request to a directory instead of a file. This could aid an attacker in

discovering possible static files that can be accessed and could leak out information about the directory structure of the web server. From manual audit, it is found that the directory listing is also limited by the same set of configurations for the plugin. This means that if the developer specifically whitelists a set of directories, the attacker cannot access the other location for directory listing. The file path for directory listing is provided by the developers logic, and the developer should make sure it is safe.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓			✓			✓			✓		✓			

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓	✓		✓					

@fastify/under-pressure

Fastify-under-pressure provides support for memory usage periodic checking and handling when the configured threshold is met.

Common vulnerable components

Node.js Perf Hook dependencies

This plugin requests the memory usage information by the node.js pref hook library. This plugin only retrieves memory usage information from the pref hook library without doing further processing on the information. The only custom process done is to use a larger than comparator to consider if the memory usage passed the threshold, or returns the memory usage figure from the retrieved histogram directly. For this reason, there is no viable attack surface because this plugin does not process any untrusted data and there are no known vulnerability reports for the pref hook library.

Denial-of-Service from frequent memory usage retrieval

As the plugin relies on web server requests through the pref hook library for retrieving information related to memory usage, if the application makes an excessive ammount of such request, the server could be overwhelmed and a Denial-of-Service situation could arise. From the manual audit, the logic for determination of the time interval for the memory usage retrieval is checked. It is confirmed that the time interval is either coming from the developer configuration or using the default value, it does not come from any untrusted user input or HTTP request. Thus the possible Denial-of-Service could only happen as a misconfiguration from the developers.

Possible race condition for the memory checking

If the periodic time interval for the memory usage retrieval is too long, or if the attacker would be able to create a large amount of traffic during the configured interval, the plugin fails to detect the passing of the memory threshold and handle it. From the manual audit, we verified all the configurable values or error handling that affect the processes are only coming from misconfigurations of the developers. Thus, these input values are trusted.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓														

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓				✓					

@fastify/websocket

Fastify-websocket handles incoming requests and add hooks to the routing process in order to distribute requests to the correct handlers or deny requests if the specific route is not configured.

Common vulnerable components

Denial-of-Service on the websocket service

An older version of the websocket plugin is vulnerable to DoS because of a missing error handler and the error throws to the web instance and crashes the web instance. During the the manual audit, we reviewed the fix for that vulnerability.

No-authentication or authorization

Authentication and authorization is a one of the key for the security of the web socket applications. This plugin does not have default authentication features provided. It does have documentation mentioning that a `preValidation` hook event to handle authentication may be necessary. The general and default code does not provide authentication options. Thus it is the developers responsibility to add in custom authentication or authorization to provide that layer of security for the web socket applications and are not in scope of this plugin.

Possible different return header for HTTP Head and Get method

By specification, HTTP HEAD and GET methods should return the same headers, however, the Fastify-websocket plugin may return different headers. This could create a posibility of leaking out information from wrong HTTP method specification from malformed input.

Websocket URL Poisoning

WebSocket connections are generally long-lived connections. HTTP requests/responses can be sent at any time during the lifetime of the wesocket connections. If an attacker could control the web socket connection URL, they could redirect the users to malicious locations or sensitive locations if the web server does not verify the web socket connection URL. This plugin does not take the web socket URL from untrusted data input from the users. Thus it is believed that this plugin is not vulnerable to URL Poisoning attacks.

Items audited

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
✓					✓				✓					✓

#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26
	✓	✓			✓		✓	✓		

Fuzzing

The Fastify framework provides an extensive list of plugins for supporting different customized features for the core Fastify web framework functionality. The list of fuzzers targets different processing methods that take random input across different Fastify plugins.

Fuzzers

Each of the fuzzers targets one or more of the Fastify plugins by creating objects of the target Fastify plugins or registering them to the core Fastify framework to retrieve its exposed functions. Then each of those exposed functions in the target Fastify plugins is fed with random data to fuzz its ability to handle random data. Known Errors and Exceptions are caught to avoid fuzzing blockage. The fuzzers can be found in https://github.com/AdaLogics/ada-fuzzers/tree/cf6b2dcf7c0931f0da556b6998ba56a2b9c9054c/projects/fastify.

There is a very long list of Fastify plugins in the wild, many of them are maintained by the Fastify team, but also some that are maintained by separate developers. In this audit report, we only target the Fastify plugins that the core Fastify maintains. To further narrow down the fuzzers to target more fuzz-worthy functions, we rank the plugins in their sensitivity and importance and aim to create fuzzers for those high in rank.

As the core Fasitfy framework is a simple and fast web application, it relies on the inclusion and registration of different Fastify plugins to provide certain designated features. Some of those features are related to the security perspective of a web application, like `@fastify/secure-session` or `@fastify/csrf-protection` which handle HTTP sessions or provides CSRF protection for the web application sessions and cookies. Besides, some of them are more vulnerable to web applications framework attacks, like `@fastify/cors` or `@fastify/fuzz_auth` which handle cross-origin-request control or basic user authentication. These plugins are usually more sensitive to malicious or invalid input. Malformed or malicious input passes through these plugins are more likely to bypass security mechanisms of the core Fastify web framework or crash the framework and cause Denial-of-Service. For that reason, we target these more sensitive plugins (which are high in our ranking) in our limited list of fuzzers.

For each of the plugins, we analyse the main entry points and their `lib` directory as the main logic of the plugin is located in these locations. The fuzzers are in separate `.js` files which export the `Fuzz` module for the `Jazzer` engine (https://github.com/CodeIntelligenceTesting/jazzer.js) to pick up and call with different sets of random input. As it is a separate .js file, it can only access the functions or components that have been explicitly exported by other .js files. We discovered these exported functions from the plugins and their `lib` directories. We also capture the expected `Error` that is

thrown explicitly from the code to avoid crashing the fuzzer easily and allow exploring more branches in the code. Each of the fuzzers targets one of the Fastify plugins on our list.

Besides exported functions, some of the Fastify plugins will decorate or register some internal functions to the Fastify core instance during the plugin registration process. Decorated functions and functions registered on different life cycle hooks can be accessed through the Fastify instance. For this kind of function, we first register to target the Fastify plugin to a newly created Fastify core instance, and then we access and fuzz those decorated or hooked functions through the Fastify instance with random data. This could extend the fuzzers coverage and possibly discover potential problems in the code.

At the entry point of the `Fuzz` module in each of our fuzzers, the fuzzer takes in a random set of byte arrays from the Jazzer fuzzing engine and uses that as the random data to fuzz those target methods. Although Javascript does not have strong typing, some functions may still require a certain format of data and could throw `TypeError` if the data does not pass some of the checks. In most cases, string, byte buffer, byte array and JSON. is the most common type of data expected by those target functions. To avoid these kinds of false-positive crashes in the fuzzers, we specifically create a random JSON generator. The generator takes in the byte array provided by the Jazzer engine and generates a random JSON object with maximum depth. These settings provide additional data types on top of the base string, byte buffer and byte array and could decrease those false-positive crashes because of the wrong data type used during the fuzzing process.

In conclusion, the fuzzers target the more sensitive Fastify plugins with random data that is slightly controlled to less false-positive crashing and optimizing for better target function fuzzing.

List of fuzzers

Fuzzers	Description
fuzz_auth.js	This fuzzer registers both @Fastify/basic-auth and @Fastify/bearer-auth plugins to the Fastify instance and fuzz those decorated functions from the Fastify core and those exposed functions from Bearer-Auth plugin with random data.
fuzz_cookie.js	This fuzzer fuzzes those exposed cookie-handling functions from the @Fastify/cookie plugin with random data.
fuzz_cors.js	This fuzzer fuzzes those exposed Cors and Vary handling functions from the @Fastify/cors plugin with random data.
fuzz_ct_parse.js	This fuzzer fuzzes those exposed parsing functions from the @Fastify/content-type-parse plugin with random data.

Fuzzers	Description
fuzz_json.js	This fuzzer fuzzes those exposed JSON and string handling functions from the @Fastify/fast-json-stringify, @Fastify/fast-json-stringify-compiler and @Fastify/secure-json-parse plugins with random data.
fuzz_jwt.js	This fuzzer registers the @Fastify/jwt plugin to the Fastify instance and fuzzes those decorated JWT-processing functions from the Fastify core with random data.
fuzz_response_validation.js	This fuzzer registers the @Fastify/response-validation plugin to the Fastify instance and fuzzes those decorated validating functions from the Fastify core with random data and schema.
fuzz_secure_session.js	This fuzzer registers the @Fastify/secure-session plugin to the Fastify instance and fuzzes those decorated secure session handling functions from the Fastify core with random data.

Here we present the issues that we identified during the audit.

Findings

#	ID	Title	Severity	Fixed
1	ADA-FASTIFY-2023-1	Weak signing key in default signer	Informational	Yes
2	ADA-FASTIFY-2023-3	Possible regular expression DOS in Fastify-Cors	Informational	Yes
3	ADA-FASTIFY-2023-7	Possible regular expression DOS in fastify-middie dependency	Informational	Yes
4	ADA-FASTIFY-2023-9	Possible reuse of destroyed secure session cookie	High	Yes
5	ADA-FASTIFY-2023-10	Possible use of weak SHA-1 algorithm for session persistent hash	Informational	Yes

[Fastify-Cookie] Weak signing key in default signer

Severity	Informational
Status	Fixed
id	ADA-FASTIFY-2023-1
Component	Fastify-cookie/signer.js

Description

The default signer of the cookie plugin directly uses the secret key and algorithm provided by the developer when the cookie plugin is being registered to the Fastify instance.

Source direct link

https://github.com/fastify/fastify-cookie/blob/9f883217af8751437e82f91feff183621a668fd1/signer.js#L13-L24

```
13 function Signer (secrets, algorithm = 'sha256') {
14   if (!(this instanceof Signer)) {
15     return new Signer(secrets, algorithm)
16   }
17
18   this.secrets = Array.isArray(secrets) ? secrets : [secrets]
19   this.signingKey = this.secrets[0]
20   this.algorithm = algorithm
21
22   validateSecrets(this.secrets)
23   validateAlgorithm(this.algorithm)
24 }
```

fastify-cookie implements checking for both the algorithm and secret to ensure the corresponding algorithm provider does exist in the web server and all the provided secrets are string or buffer. However, the current checks are insufficient to maintain sufficient security protection towards the cookie signing and unsigning. The signer uses the node js crypto framework to create hmac as the signature of cookies. According to Hmac definition in RFC2104 (https://www.rfc-editor.org/rfc/rfc2104.html), the cryptographic strength of the HMAC depends upon the size of the secret key that is used and the security of the underlying hash function used. In the current configuration of the fastify cookie plugin, it is possible to use an insecure cryptographic algorithm like MD5 and SHA1. Also, it is possible to have a very short secret. These two insecure settings will affect the security properties of the signed cookie

which a man-in-the-middle attacker could easily use a collision attack to create a malicious cookie which could still match the signature if the developer fails to provide a secure key and algorithm for the plugin.

We recommend adding a paragraph to the documentation detailing that for production use cases, the user should use a secure hash function and a long secret.

[Fastify-Cors] Possible regular expression DOS in Fastify-Cors

Severity	Informational
Status	Fixed
id	ADA-FASTIFY-2023-3
Component	Fastify-cors/plugin.js

Description

According to the documentation https://github.com/fastify/fastify-cors/tree/master#options, the developer could configure the fastify-cors plugins with an origin option which specify the allowed origin for cors processing. This option is used to match if the requested origin is allowed to use cross origin resources sharing. One of the accepted types of the origin option is JS RegExp object which is vulnerable to ReDos. Although the pattern is crafted by the developer and not using any untrusted input, a badly crafted regular expression pattern could allow an attacker to craft a specific origin in the request and create a ReDos situation. For example, if the pattern aims to match any subdomain of trusted.com and the developer created the pattern as `/([0-9.]+)*trusted.com$/`, an attacker could create an input to trigger the ReDos, like `0000000000000000000000000000000000000000t`.

Source direct link:

https://github.com/fastify/fastify-cors/blob/042b14b7d6bbd5ddf1ca73482c6c9dcfa57304f9/index.js#L290-L292

```
290     } else if (allowedOrigin instanceof RegExp) {
291       allowedOrigin.lastIndex = 0
292       return allowedOrigin.test(reqOrigin)
```

It shows that both the allowedOrigin pattern and the origin from http request are not checked and used directly. It is possible that if the developer is not aware of those dangerous regular expression patterns and creates one of them and causes the problem. Without a proper checking and validation of regex pattern (like using safe-regex package), an unaware developer could imply a dangerous regex pattern for the cors origin matching.

We recommend adding a section to Fastifys documentation about the risks of usign Regex in this scenario. Specifically, users should audit their regular expressions before putting them to use in production.

[Fastify-Middie] Possible regular expression DOS in fastify-middie dependency

Severity	Informational
Status	Fixed
id	ADA-FASTIFY-2023-7
Component	Fastify-middie/plugin.js

Description

The Fastify-middie provides a function to register additional middleware to different lifecycle hooks on the core Fastify instance. After certain middleware registration, Fastify provides a decorated function to use that registered middleware from the core Fasitfy instance. The use of that decorated function calls the `pathToRegexp` function from the `path-to-regexp` module to retrieve a regular expression pattern for retrieving the path of the target middleware to manage. `pathToRegexp` could return a vulnerable regular expression pattern which could be abused to cause Regular Expression Denial-of-Service.

```
1 var groupsRegex = /\((?:\?<(.*?)>)?(?!\?)/g;
```

Fastify developers using the method with unexpected or strange paths for the method could result in Redos. Similar to issue 1, it could be detected during web instance initialisation.

Users should audit the paths of their middlewares before deploying to production. The risk of being affected is low, however users with the risk of generating vulnerable paths - such as by way of automated services - could be affected by this.

[Fastify-Secure-Session] Possible reuse of destroyed secure session cookie

Severity	High
Status	Fixed
id	ADA-FASTIFY-2023-9
Component	Fastify-secure-session/index.js

Description

The Fastify-secure-session plugin is not really using a “traditional session” which stores everything in the server side and just returns a cookie storing the session ID to keep track of the user. It uses stateless session which stores everything in an encrypted cookie on the client side and depends solely on that encrypted cookie.

At the end of the request handling, it will encrypt all data in the session with a secret key and attach the ciphertext as a cookie value with the defined cookie name. After that, the session on the server side is destroyed. When an encrypted cookie with matching session name is provided with subsequent requests, it will decrypt the ciphertext to get the data. The plugin then creates a new session with the data in the ciphertext. Thus theoretically the web instance is still accessing the data from a server-side session, but technically that session is generated solely from a user provided cookie (which is assumed to be non-craftable because it is encrypted with a secret key not known to the user).

The issue happens in the session removal process. In general, when a “traditional session” is destroyed, that data no longer exists on the server side, even if you provided the old session ID cookie, it still cannot be used because that data no longer exists on the server side. But it is a different story for this stateless cookie based session. In the delete function of the code, when the session is deleted, it is marked for deletion.

Source direct link:

https://github.com/fastify/fastify-secure-session/blob/881694b2364229d61aefff688d4c4ad81a5cb7a6/index.js#L289-L292

```
289    delete () {
290      this.changed = true
291      this.deleted = true
```

```
292     }
```

When the request has ended, during the onSend hook, it will execute the session deletion logic if it is flagged as deleted.

Source direct link:

https://github.com/fastify/fastify-secure-session/blob/881694b2364229d61aefff688d4c4ad81a5cb7a6/index.js#L237-L255

```
237           } else if (session.deleted) {
238             request.log.debug('@fastify/secure-session: deleting session'
                  )
239             const tmpCookieOptions = Object.assign(
240               {},
241               cookieOptions,
242               session[kCookieOptions],
243               { expires: new Date(0), maxAge: 0 }
244             )
245             reply.setCookie(cookieName, '', tmpCookieOptions)
246             continue
247           }
```

We can see that the only session removal logic is to clear the encrypted session cookie. That is no more encrypted session cookie is returned in the reply and assuming the subsequent request won't send in that cookie anymore. That could create a problem. If an attacker could control what cookie to be sent, it could still attach the encrypted session cookie and still gain access to some route which requires the data in the session cookie because the session does not keep track of any information in the server-side and thus it does not have the ability to verify if a provided encrypted session cookie is "destroyed" or not if the validity of the cookie is still valid. It has been tested that if the session cookie is still attached to subsequent requests after "destroying" the session, it can stilsl be identified as a "legit" cookie.

This issue was assigned CVE-2024-31999

[Fastify-Session] Possible use of weak SHA-1 algorithm for session persistent hash

Severity	Informational
Status	Fixed
id	ADA-FASTIFY-2023-10
Component	Fastify-session/lib/session.js

Description

The fastify-Session plugin provides session support to the base Fastify framework. It uses a custom `Session` object in the `lib/` directory to manage the configuration and data for the creation and management of the session. Hash values are very often used as checksum to verify the integrity of data. When an attacker finds another set of information which gives the same hash value after hashing, then the illegal change of the data may not be detected and result in collision attacks.

The `Session` class created in `/lib/session.js` used by the Fastify-session plugin adopts `SHA-1` as the hashing algorithm for generating the hash value for checking the session persistency. There is an instance method `[hash]()&#123;&#125;` in the `Session` class which stringifies the whole session object and uses the result string to generate a hash value. From Line#84 of the `Session` class constructor, Fastify-Session uses the result of the instance method `[hash]()&#123;&#125;` as the persistent checksum. Since `SHA1` is considered weak and broken, an attacker can potentially find a collision with altered data in session.

Source direct link:

https://github.com/fastify/session/blob/8ea7e46a5542a1093a8f66cfbb2efd18024e49a5/lib/session.js#L206-L221

```
206   [hash] () {
207     const sess = this
208     const str = stringify(sess, function (key, val) {
209       // ignore sess.cookie property
210       if (this === sess && key === 'cookie') {
211         return
212       }
213
214       return val
```

```
215       })
216
217       return crypto
218         .createHash('sha1')
219         .update(str, 'utf8')
220         .digest('hex')
221     }
```

Source direct link:

https://github.com/fastify/session/blob/8ea7e46a5542a1093a8f66cfbb2efd18024e49a5/lib/session.js#L22-L57

```
22    constructor (
23      sessionStore,
24      request,
25      idGenerator,
26      cookieOpts,
27      cookieSigner,
28      prevSession,
29      sessionId = idGenerator(request)
30    ) {
31      this[sessionStoreKey] = sessionStore
32      this[generateId] = idGenerator
33      this[cookieOptsKey] = cookieOpts
34      this[cookieSignerKey] = cookieSigner
35      this[requestKey] = request
36      this[sessionIdKey] = sessionId
37      this[encryptedSessionIdKey] = (
38        prevSession &&
39        prevSession[sessionIdKey] === sessionId &&
40        prevSession[encryptedSessionIdKey]
41      ) || cookieSigner.sign(this.sessionId)
42      this[savedKey] = false
43      this.cookie = new Cookie(prevSession?.cookie || cookieOpts, request
          )
44
45      if (prevSession) {
46        // Copy over values from the previous session
47        for (const key in prevSession) {
48          (
49            key !== 'cookie' &&
50            key !== 'sessionId' &&
51            key !== 'encryptedSessionId'
52          ) && (this[key] = prevSession[key])
53        }
54      }
55
56      this[persistedHash] = this[hash]()
57    }
```

Although the persistent hash is only used on the server side for validating the integrity of the stored session data, it never assures that value will or will not pass on to the client side. Also, with a combination of session prediction, session fixation or other server-side attacks in which an attacker could modify session data on the server side or point the users to different sessions, it could break the integrity check of the persisted hash with altered data which still result in the same hash. This could happen since SHA1 is considered as weak and could result in collision easily. But in other cases, the abuse of that hash is not too possible thus this is considered an informational report on the possible use of a weak hashing algorithm.