CloudEvents

Security Assessment

October 26, 2022

Prepared for:

Doug Davis, Cloud Native Computing Foundation

Open Source Technology Improvement Fund

Prepared by: **Alex Useche and Hamid Kashfi**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

The Linux Foundation, via strategic partner Open Source Technology Improvement Fund, engaged Trail of Bits to review the security of its CloudEvents specification and SDKs. From September 19 to September 30, 2022, a team of two consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system. We had access to the source code and documentation. We performed dynamic automated and manual testing of the target system, using both automated and manual processes.

Summary of Findings

The audit did not uncover any significant flaws or defects that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

Some of the findings in this report have their severity marking as `Undetermined`. This is because in engagements of this nature, the code and vulnerabilities in its dependencies are highly dependent on the context in which they are used. As a result, the severity of issues cannot be determined and generalized. Moreover, due to time constraints, we did not manually triage outdated, third-party dependencies or their vulnerabilities.

EXPOSURE ANALYSIS

Severity	*Count*
Informational	1
Undetermined	6

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	1
Denial of Service	1
Patching	3

Undefined Behavior	2

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-CE-{1,4,7}**
  Reviews of multiple SDKs as well as consulting with the team indicates that SDKs are not actively and routinely maintained for security updates, leaving some of them with multiple outdated and vulnerable dependencies. Maintenance of SDK health is a subject that is already covered in the SDK Governance specification.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

Derek Zimmer, Program Manager
derek@ostif.org

Amir Montazery, Program Manager
amir@ostif.org

The following engineers were associated with this project:

Alex Useche, Consultant
alex.useche@trailofbits.com

Hamid Kashfi, Consultant
hamid.kashfi@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 7, 2022	Pre-project kickoff call
September 26, 2022	Status update meeting #1
October 3, 2022	Delivery of report draft
October 3, 2022	Report readout meeting
October 26, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the CloudEvents specification and SDKs, including a lightweight threat model. Specifically, we sought to achieve the following non-exhaustive list of goals:

- Build an understanding of the CloudEvents specification and evaluate its suitability for use in secure environments
- Threat modeling
- Individual SDK audit and conformance to the specification
- Documentation review
- Current testing evaluation and recommendations for improvement

Project Targets

The engagement involved a review and testing of the targets listed below

CloudEvents Specification

Repository	https://github.com/cloudevents/spec
Version	`2e09394c6297dad6d25edbc50717bbc71dba124a`
Type	Specification documentation
Platform	N/A

CloudEvents SDK for Go

Repository	https://github.com/cloudevents/sdk-go
Version	`a7187527ab3278128c1b2a8fe9856d49ecddf25d`
Type	Go
Platform	Linux

CloudEvents SDK for Java

Repository	https://github.com/cloudevents/sdk-java
Version	`b9eaa2fcaaf5569552e39ece4fce4a99064145e9`
Type	Java
Platform	Linux

CloudEvents SDK for PHP

Repository	https://github.com/cloudevents/sdk-php
Version	`602cd26557e5522060531b3103450b34b678be1c`
Type	PHP
Platform	Linux

CloudEvents SDK for Python

Repository	https://github.com/cloudevents/sdk-python
Version	`60f848a2043e64b37f44878f710a1c38f4d2d5f4`
Type	Python
Platform	Linux

CloudEvents SDK for Rust

Repository	https://github.com/cloudevents/sdk-rust
Version	`c380078bf45fcebe1af6299d75539cd6ba37f7d3`
Type	Rust
Platform	Linux

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches include the following:

- A review of the CloudEvents core specification
- A review of the latest release (v1.0.2) version of the following protocol specifications for CloudEvents:
    - AMQP Protocol Binding
    - AVRO Event Format
    - HTTP Protocol Binding
    - JSON Event Format
    - Kafka Protocol Binding
    - MQTT Protocol Binding
    - NATS Protocol Binding
    - WebSockets Protocol Binding
    - Protobuf Event Format
    - XML Event Format
    - Web hook
- A lightweight threat modeling exercise covering potential high-level threats that could arise when using CloudEvents and the SDKs
- Review of testing coverage for the various SDKs
- Review data validation strategies
- Review of serialization, deserialization, encoding, and decoding logic
- Review for potential cases where event data may be unnecessarily leaked
- A manual code review of the SDKs listed in the Project Targets section of this report

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Because we focused on the most used SDKs, our coverage for the following languages was limited:
    - PHP, CSharp, Python, Ruby, PowerShell
- Manual static-analysis reviews were limited to SDK implementations. Testing and example codes provided along with each SDK was skipped.
- Vulnerable third-party libraries are highlighted and included without in-depth triage of their potential impact on the given SDK.
- CloudEvents extensions and adapters were out of scope for this engagement.

Threat Model

As part of the audit, Trail of Bits conducted a lightweight threat model, drawing from the Mozilla Rapid Risk Assessment methodology and the National Institute of Standards and Technology's (NIST) guidance on data-centric threat modeling (NIST 800-154). We began our assessment of the design of CloudEvents by reviewing the documentation on the CloudEvents website and the various README files in the CloudEvents spec repository.

Data Types:

An application that produces CloudEvents logs and event data which contains various attributes. All CloudEvents contain the following:

- ID
- Source URI
- Specification version
- Type of event (often used for observability, routing, etc.)
- Data specific to the event

Additionally, CloudEvents could optionally include the following:

- Data content type
- URI identifying the data schema
- A subject string
- Time stamp

Components

The following table describes each of the components identified for our analysis.

Component	Description
Events	Include context and data about an occurrence. Events are routed from an event producer (the source) to interested event consumers.
Source	Context in which the occurrence happened. In some cases, the source might consist of multiple Producers. Typically a managed service.
Producer	Application or process producing the event (i.e. monitoring app).
Consumer	Receives the event and acts upon it. It uses the context and data to execute

Intermediary	Receives a message containing an event for the purpose of forwarding it to the next receiver, which might be another intermediary or a Consumer.
Action	Typically, custom code developed by a developer such as a lambda function or Azure function.
Message	Contains a body with context (metadata) and Event Data (the payload or actual message). Messages comprise the following: ● Message Context: Metadata about an event. Used by tools and applications to identify the relationship of Events to aspects of the system or to other Events. ● Message Event Data: Event payload

Trust Zones

Trust zones capture logical boundaries where controls should or could be enforced by the system and allow developers to implement controls and policies between components' zones.

Zone	Description	Included Components
Internet	The wider external-facing internet zone typically includes users and cloud services that use CloudEvents to send and process event data.	● All
Local Network	Any network inaccessible from the internet (i.e., private virtual network or on-prem intranet where an application sending or receiving CloudEvents resides)	● All
Local System	Server running application with CloudEvents SDK. This could be managed by a cloud provider or	● Producer ● Consumer ● Intermediary

	an on-prem system. If an attacker gets access to a local system, they would have access to the processes that make up the producer.	

In the table below, we further distinguished between two general zones. Because we are modeling the risk profile of an SDK, we consider that it can be used in multiple different ways where, for instance, both the producer and consumer components could run on the internet or an internal network. Distinguishing between the zones shown below helps us better describe sets of attacks based on communication between the consumer, producer, and location of the source.

Zone	Description	Included Components
Producer Zone	The zone where the producer runs and where CloudEvents are first created from events sent by a source. The producer can be in an internal or externally reachable network.	• Producer • Events • Message • Source (*)
Consumer Zone	The zone where the consumer runs. The consumer uses the CloudEvents SDK to read, decode, and deserialize events sent by a producer. The consumer can be in an internal or externally reachable network.	• Consumer • Message • Action
Intermediary Zone	The zone in which an optional intermediary producer may be located. An intermediary may mutate CloudEvents before sending them to the final consumer.	• Consumer • Message • Action

(*) It is possible that the source and producers are the same. For instance, an API may generate

its own CloudEvents by using the CloudEvents SDK. However, this may not always be the case, as the producer could be a separate application designed to receive log and event data.

Trust Zone Connections

We can draw from our understanding of what data flows between trust zones and why to enumerate attack scenarios.

Originating Zone	Destination Zone	Connection	Authentication & Authorization
Producer Zone	Consumer Zone	Internet → Internet	Producer or consumer dependent
		Internet → Local Network	
		Local Network → Local Network	
		Local Network → Internet	
Producer Zone	Intermediary Zone	Internet → Internet	
		Internet → Local Network	
		Local Network → Local Network	
		Local Network → Internet	
Intermediary Zone	Consumer Zone	Internet → Internet	
		Internet → Local Network	
		Local Network → Local Network	
		Local Network → Internet	
Local System	Local System	• Localhost communication with	

Local Network		consumer or producer application • Output to STDOUT	

Threat Actors

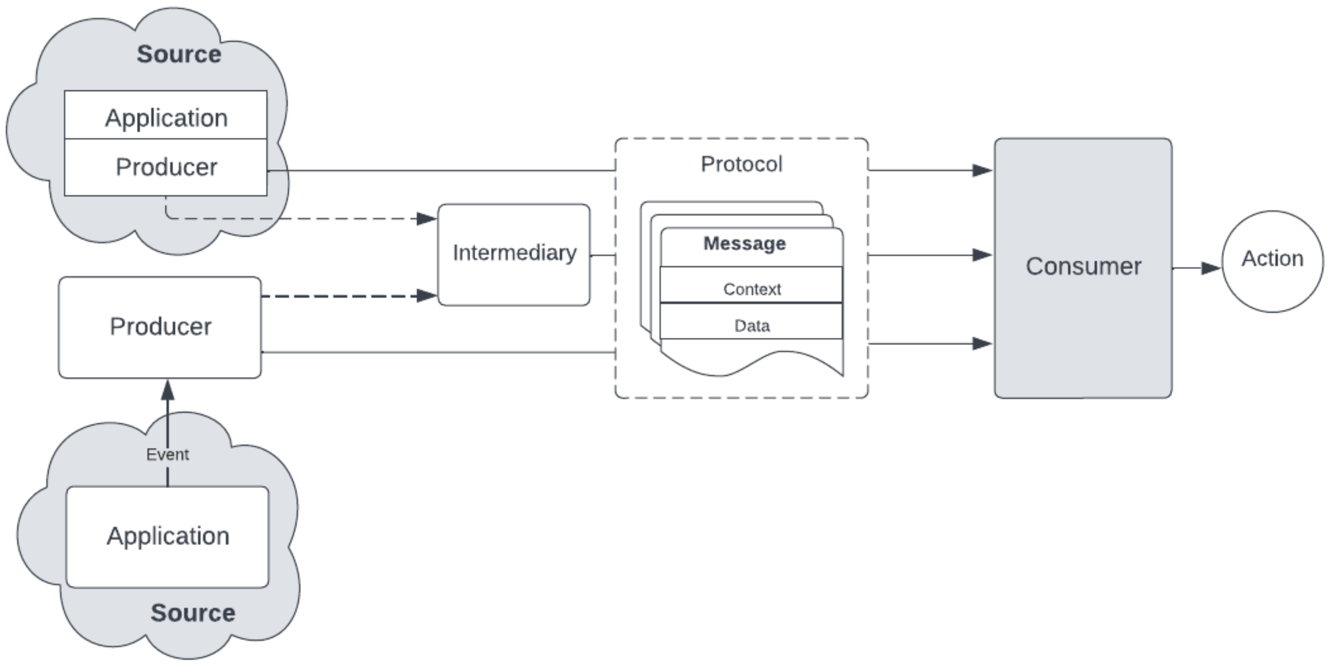
Similarly to establishing trust zones, defining malicious actors before conducting a threat model is useful in determining which protections, if any, are necessary to mitigate or remediate a vulnerability. We also define other "users" of the system who may be impacted by, or induced to undertake, an attack.

Actor	**Description**
External Attacker	An attacker on the internet. An external attacker will seek to get access to internal systems running CloudEvent sources, producers, or consumers.
Malicious Internal User	Malicious internal users often have privileged access to a wide range of resources, such as the network transporting events, systems producing or consuming events, or intermediary systems.
Internal Attacker	An internal attacker is one who has transited one or more trust boundaries, such as an attacker with direct access to the system running CloudEvents.
Administrator	A cloud, system, or network administrator with privileged access to the infrastructure and services where CloudEvents are produced or received.
Application developer	An application or service developer who uses the CloudEvents SDK.

Data Flow

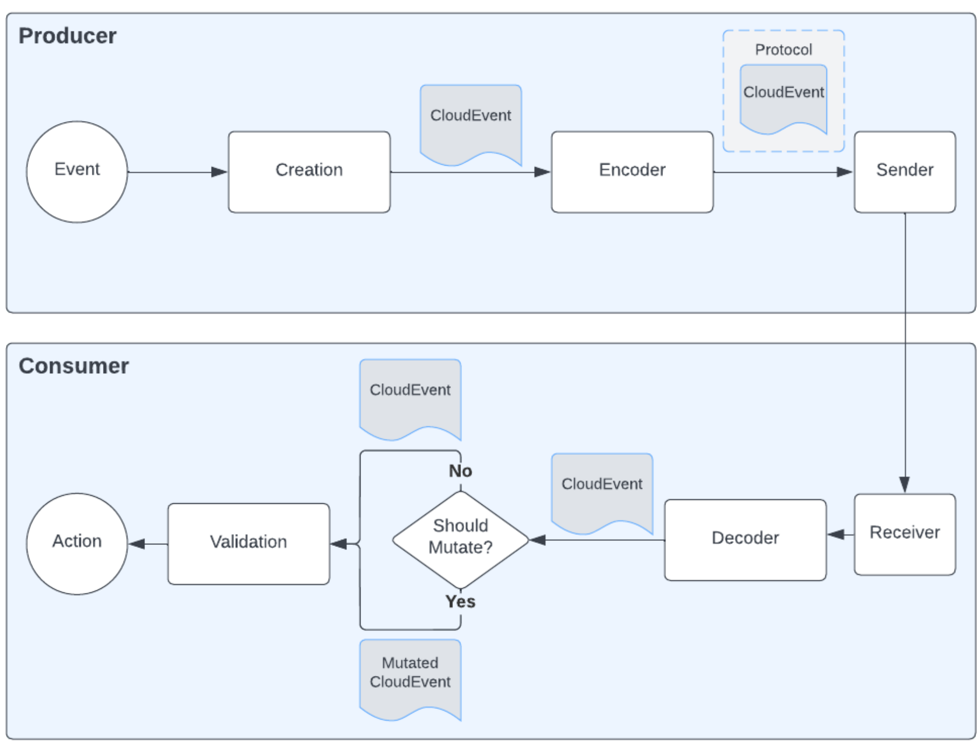
High-Level Overview

The diagram below demonstrates CloudEvents flow of events and processing and considers components that are important to contextualize possible threat scenarios.



CloudEvents SDK

The diagram below highlights the operations performed by a CloudEvents SDK:



Threat Scenarios

Threat	Description	Actor(s)	Component(s)
Producer DoS condition due to improper encoding and serialization	The producer may use a CloudEvents SDK with buggy encoding or serialization logic (e.g., nil dereferences), leading the producer to crash. An internal attacker that is aware of such a bug could generate malicious event data in order to cause the producer to crash. A lack of sufficient unit testing requirements could increase the likelihood of this threat.	• Internal attacker	• Event • Producer • CloudEvent Message • Source
Consumer DoS condition due to improper decoding an deserialization	The consumer may use a CloudEvents SDK with buggy decoding or deserialization logic (e.g., nil dereferences), leading the consumer to crash. A lack of sufficient unit testing requirements could increase the likelihood of this threat.		• Event • Consumer • Message • Source
Dropped or out of order events	The spec does not define any fields that can be used to keep track of events, as timestamps are optional. A naive developer may assume events are received in the order in which they were produced and not account for the possibility of dropped events due to disk faults or race condition bugs.	• Application developer	• Producer • Message • Source • Event

A malicious extension compromises the confidentiality or integrity of event data	A developer uses a CloudEvents extension that can introduce unexpected, malicious behavior, allowing attackers to re-route or modify CloudEvents data.	• Application developer • External attacker	• Producer • Consumer • Intermediary • Message • Event
CloudEvents are modified by a malicious intermediary, or one the developer is not aware of.	An intermediary modifies CloudEvents before rerouting them to a consumer. Because the spec does not define signing requirements, there is no way for the source to know whether the event was changed. Although maintaining the integrity of CloudEvent messages is stated as a non-goal in the CloudEvent spec, there should be a central place where developers can easily become aware of this.	• Malicious internal user • Internal attacker • External attacker	• Intermediary • Consumer • Message • Event
A malicious or vulnerable SDK dependency compromises the confidentiality or integrity of event data	A malicious SDK dependency can introduce unexpected, malicious behavior, allowing attackers to re-route or modify CloudEvents data or to introduce backdoors or RCE vulnerabilities.	• External attacker • Internal attacker	• Consumer • Producer • Intermediary • Message • Event • Message
Vulnerable use of SDK due to lack of clarity on what each SDK supports	The specs mention that SDKs SHOULD validate data. However, this is not a requirement. A developer might, for instance, assume that every SDK is required to validate CloudEvents and as a result use the SDK in an insecure manner, skipping logic such as data validation	• Developer	• Consumer • Producer • Intermediary • Event

	due to assumptions based on differences between what SDK supports.		

Recommendations

- Centralize and condense documentation regarding security considerations that developers should be aware of when using the CloudEvents SDKs. In particular, developers should be able to easily reference security concerns that they should be responsible for.

- The CloudEvents specification uses the words “MUST” to define logic or responsibilities that every SDK should implement. On the other hand, it uses the word “SHOULD” to define suggested but optional features. Developers should be able to easily reference which SHOULD features are or are not implemented by each SDK. We suggest using a table format in the spec’s repo to display this information.

- Consider adding fuzzing tests for every SDK, and adding the various SDKs to the OSS-fuzz project.

- Document minimum unit testing requirements for all SDKs. Currently, there are no formal testing requirements for pull requests sent for the various SDKs.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time
Clippy	Rust linter
JetBrains Inspectors	JetBrain build in inspectors for Go and Rust codebases

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	[Java SDK] Reliance on default encoding	Undefined Behavior	**Undetermined**
2	[Java SDK] Outdated Vulnerable Dependencies	Patching	**Undetermined**
3	[JavaScript SDK] Potential XSS in httpTransport()	Data Validation	**Undetermined**
4	[Go SDK] Outdated Vulnerable Dependencies	Patching	**Undetermined**
5	[Go SDK] Downcasting of 64-bit integer	Undefined Behavior	**Undetermined**
6	[Go SDK] ReadHeaderTimeout not configured	Denial of Service	**Informational**
7	[CSharp SDK] Outdated Vulnerable Dependencies	Patching	**Undetermined**

Detailed Findings

1. [Java SDK] Reliance on default encoding	
Severity: **Undetermined**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-CE-1
Target: Java SDK	

Description

Multiple instances were identified in which the `getByte()` standard Java API is used without specifying any encoding. Doing so causes the Java SDK to rely on the system default encoding, which can differ across platforms and systems used by event actors and cause unexpected differences in processing of event data.

The specification states that appropriate and RFC-compliant encodings MUST be followed, but the implementation in the Java SDK and documentation should be improved to highlight the importance of matching encoding across actors.

Not all observed instances are necessarily problematic, as they are handling binary data. However, this behavior should be documented and handled in the SDK implementation, documentation, and supplied examples.

````
import io.cloudevents.CloudEvent;
import io.cloudevents.core.builder.CloudEventBuilder;

import java.net.URI;

final CloudEvent event = CloudEventBuilder.v1()
    .withId("000")
    .withType("example.demo")
    .withSource(URI.create("http://example.com"))
    .withData("text/plain","Hello world!".getBytes())
    .build();
```
````

Figure 1.1: Java SDK documentation providing bad example for `getBytes()` (`docs/core.md#28–40`)
````

```java
private byte[] getBinaryData(Message<?> message) {
      Object payload = message.getPayload();
      if (payload instanceof byte[]) {
            return (byte[]) payload;
      }
      else if (payload instanceof String) {
            return ((String) payload).getBytes(Charset.defaultCharset());
      }
      return null;
}
```

Figure 1.2: Using `getBytes()` and relying on default charset can lead to unexpected behavior (`spring/src/main/java/io/cloudevents/spring/messaging/CloudEventMessageConverter.java#93–102`)

Exploit Scenario

The event producer, the intermediary (using the SDK), and the consumer use different default encodings for their systems. Without acknowledging a fixed encoding, the data is handled and processed using an unintended encoding, resulting in unexpected behavior.

Recommendations

Short term, improve the SDK documentation to highlight the importance of matching encoding acros actors.

Long term, review all similar instances across the SDK and improve test cases to cover handling of message and data encoding.

References

- The Java Tutorials; Byte Encodings and Strings
- PMD New rule: Reliance on default charset #2186

2. [Java SDK] Outdated Vulnerable Dependencies

Severity: **Undetermined**	Difficulty: **Medium**
Type: Patching	Finding ID: TOB-CE-2
Target: Java SDK	

Description

Multiple outdated dependencies with publicly known vulnerabilities, including multiple high- and medium-risk vulnerabilities, were identified in the Java SDK. The open-source snyk tool was used to automatically audit each module. Due to time constraints and ease of remediation, exploitability of these issues within the context of the SDK was not manually reviewed.

A list of Java SDK modules and their vulnerable dependencies is provided below:

Module	Dependency	Details
Io.cloudevents:cloudevents-kafka	org.apache.kafka:kafka-clients@2.5.0 introduced by org.apache.kafka:kafka-clients@2.5.0	Timing Attack [Medium Severity]
io.cloudevents:cloudevents-http-vertx	io.netty:netty-common@4.1.74.Final introduced by io.vertx:vertx-core@4.2.5 > io.netty:netty-common@4.1.74.Final	Information Exposure [Medium Severity]
io.cloudevents:cloudevents-http-vertx	io.netty:netty-handler@4.1.74.Final introduced by io.vertx:vertx-core@4.2.5 > io.netty:netty-handler@4.1.74.Final	Improper Certificate Validation [Medium Severity]
io.cloudevents:cloudevents-protobuf	com.google.protobuf:protobuf-java@3.15.0 introduced by com.google.protobuf:protobuf-java@3.15.0	Denial of Service (DoS) [High Severity]
io.cloudevents:cloudevents-protobuf	com.google.code.gson:gson@2.8.6 introduced by com.google.protobuf:protobuf-java-util@3.15.0 > com.google.code.gson:gson@2.8.6	Denial of Service (DoS) [High Severity]

Exploit Scenario

Attackers identified vulnerable dependencies by observing the public GitHub repository of the SDK. They can then craft malicious requests (HTTP, event, etc.) that will be processed by SDK APIs to exploit these issues.

Recommendations

Short term, upgrade all outdated third-party dependencies used in the SDK.

Long term, outdated and vulnerable dependencies should be automatically and continuously highlighted as part of the CI/CD pipeline. Alternatively, developers can configure GitHub actions that warns developers when new security updates are available for dependencies.

3. [JavaScript SDK] Potential XSS in httpTransport()	
Severity: **Undetermined**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-CE-3
Target: `sdk-javascript/src/transport/http/index.ts`	

Description
The `httpTransport()` method in the JavaScript SDK writes raw response messages from the endpoint when an error occurs. If user-controlled data is reflected in the error message, and the callee of this API includes the response in a web page without sanitizing the output, the application using the SDK and rendering its results will become vulnerable to XSS.

Validation and sanitization of data is not enforced by the specification, but the SDK documentation should highlight lack of sanitization of HTTP responses when this API is used in an emitter.

```
55    req.on("error", reject);
56    req.write(message.body);
57    req.end();
```

Figure 3.1: Directly writing HTTP response bypasses HTML escaping and can lead to XSS (`src/transport/http/index.ts#55–57`)

Exploit Scenario
An application consumes the API output and includes it in a web page without sanitizing. Attackers trigger XSS in the application by injecting events that trigger error responses containing their payload.

Recommendations
Short term, escape JavaScript/HTML when directly writing out responses.

Long term, improve the SDK documentation to highlight the importance of sanitization of responses from SDK APIs, as it's not mandated by the specification or the SDK.

4. [Go SDK] Outdated Vulnerable Dependencies	
Severity: **Undetermined**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-CE-4
Target: Go SDK	

Description

Multiple outdated dependencies with publicly known vulnerabilities were identified in the Go SDK. The open-source snyk tool was used to automatically audit each module. Due to time constraints and ease of remediation, exploitability of these issues within the context of the SDK was not manually reviewed.

A list of Go SDK modules and their vulnerable dependencies is provided below:

Module	Dependency	Details
protocol/ws/v2/go.mod	Introduced through: nhooyr.io/websocket@1.8.6	Denial of Service (DoS) [High Severity]
samples/ws/go.mod	Introduced through /protocol/ws/v2@2.5.0	Denial of Service (DoS) [High Severity]

Exploit Scenario

Attackers identified vulnerable dependencies by observing the public GitHub repository of the SDK. They can then craft malicious requests (HTTP, event, etc.) that will be processed by SDK APIs to exploit these issues.

Recommendations

Short term, upgrade all outdated third-party dependencies used in the SDK.

Long term, outdated and vulnerable dependencies should be automatically and continuously highlighted as part of the CI/CD pipeline. Alternatively, developers can configure GitHub actions that warns developers when new security updates are available for dependencies.

5. [Go SDK] Downcasting of 64-bit integer

Severity: **Undetermined**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-CE-5
Target: `sql/v2/parser/expression_visitor.go`, `sql/v2/utils/casting.go`	

Description

The `strconv.Atoi` function parses an `int`: a machine dependent integer type that will be int64 for 64-bit targets. There are places throughout the codebase where the result returned from `strconv.Atoi` is later converted to a smaller type: int16 or int32. This may overflow with a certain input.

```
func (v *expressionVisitor) VisitIntegerLiteral(ctx *gen.IntegerLiteralContext) interface{} {
        val, err := strconv.Atoi(ctx.GetText())
        if err != nil {
                v.parsingErrors = append(v.parsingErrors, err)
        }
        return expression.NewLiteralExpression(int32(val))
}
```

Figure 5.1: Downcasting of 64-bit integer (`sql/v2/parser/expression_visitor.go#279–285`)

```
case cesql.IntegerType:
        switch val.(type) {
        case string:
                v, err := strconv.Atoi(val.(string))
                if err != nil {
                        err = fmt.Errorf("cannot cast from String to Integer: %w", err)
                }
                return int32(v), err
        }
```

Figure 5.2: Downcasting of 64-bit integer (`sql/v2/utils/casting.go#34–42`)

Exploit Scenario

A value is parsed from a configuration file with Atoi, resulting in an integer. It is then downcasted to a lower precision value, resulting in a potential overflow or underflow that is not handled by the Golang compiler an error or panic.

Recommendations

Short term, when parsing strings into fixed-width integer types, use `strconv.ParseInt` or `strconv.ParseUint` with an appropriate `bitSize` argument instead of `strconv.Atoi`.

Long term, use open-source automated static-analysis tools such as Semgrep as part of the development process to check for common vulnerabilities in the code.

6. [Go SDK] ReadHeaderTimeout not configured	
Severity: **Informational**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-CE-6
Target: Go SDK	

Description

The `http.server` API in Go can be initialized with four different timeouts, including `ReadHeaderTimeout`. Without specifying a value for this timeout, the listener instance will become vulnerable to the Slowloris DoS attack.

```
// After listener is invok
listener, err := p.listen()
if err != nil {
	return err
}

p.server = &http.Server{
	Addr:    listener.Addr().String(),
	Handler: attachMiddleware(p.Handler, p.middleware),
}
```

Figure 6.1: ReadheaderTimeout not configured for http.server (`v2/protocol/http/protocol_lifecycle.go#34–43`)

Exploit Scenario

Attackers can exhaust server resources by opening multiple HTTP connections to the server, keeping the connections open, and slowly and continuously sending new HTTP header lines over the socket. This will eventually exhaust all open file handles.

Recommendations

Short term, specify appropriate timeout value for the `ReadHeaderTimeout` parameter.

Long term, improve the code and SDK documentation to consider other means of handling timeouts and preventing DoS attacks.

7. [CSharp SDK] Outdated Vulnerable Dependencies	
Severity: **Undetermined**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-CE-7
Target: CSharp SDK	

Description

Multiple outdated dependencies with publicly known vulnerabilities were identified in the CSharp SDK. The open-source snyk tool was used to automatically audit each module. Due to time constraints and ease of remediation, exploitability of these issues within the context of the SDK was not manually reviewed.

A list of CSharp SDK modules and their vulnerable dependencies is provided below:

Module	Dependency	Details
CloudNative.CloudEvents.AspNetCore.csproj	Introduced through: Microsoft.AspNetCore.Mvc.Core Version="2.1.16"	Remote Code Execution [High Severity]
CloudNative.CloudEvents.AspNetCore.csproj	Introduced through: Microsoft.AspNetCore.Mvc.Core 2.1.16 >Microsoft.Extensions.DependencyModel/2.1.0 > Newtonsoft.Json 9.0.1	Insecure Defaults [High Severity]
CloudNative.CloudEvents.Avro/obj/project.assets.json	Introduced through: Apache.Avro/1.11.0 > Newtonsoft.Json": "10.0.3	Insecure Defaults [High Severity]
CloudNative.CloudEvents.Avro/obj/project.assets.json	Introduced through: Apache.Avro/1.11.0 > Newtonsoft.Json": "10.0.3 > System.Xml.XmlDocument 4.3.0 > ... > System.Text.RegularExpression 4.3.0	Denial of Service [High Severity]

Exploit Scenario

Attackers identified vulnerable dependencies by observing the public GitHub repository of the SDK. They can then craft malicious requests (HTTP, event, etc.) that will be processed by SDK APIs to exploit these issues.

Recommendations

Short term, upgrade all outdated third-party dependencies used in the SDK.

Long term, outdated and vulnerable dependencies should be automatically and continuously highlighted as part of the CI/CD pipeline. Alternatively, developers can configure GitHub actions that warn developers when new security updates are available for dependencies.

Summary of Recommendations

The CloudEvent specification and SDK are works in progress with multiple SDKs implemented in ten different languages. Trail of Bits recommends that Linux Foundation address the findings detailed in this report and take the following additional steps prior to deployment:

- Introduce automated dependency auditing and vulnerability scanning into the development process for all SDKs and improve the SDK Governance guidelines to make these steps a mandatory part of contribution and maintenance.

- Use static-analysis tools such as Semgrep (or commercially available alternatives) as well as linting plugins for the IDEs to highlight and mitigate common vulnerable bug patterns and usage of deprecated APIs as soon as they are introduced into the code. Many such tools can be directly integrated into the CI/CD pipeline or used as GitHub actions.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Security Controls

The following tables describe the security controls and rating criteria used for the threat model.

Security Controls	
Category	Description
Access Controls	Authorization, session management, separation of duties, etc.
Audit and Accountability	Logging, non-repudiation, monitoring, analysis, reporting, etc.
Awareness and Training	Policy, procedures, and related capabilities
Configuration Management	Inventory, secure baselines, configuration management, & change control
Cryptography	The cryptographic controls implemented at rest, in transit, and in process
Denial of Service	The controls to defend against different types of denial-of-service attacks impacting availability
Identification and Authentication	User and system identification and authentication controls
Maintenance	Preventative and predictive maintenance, and related controls
System and Communications Protection	Network level controls to protect data
System and Information Integrity	Software integrity, malicious code protection, monitoring, information handling, and related controls
System and Services Acquisition	Development lifecycle, documentation, supply chain, etc.

C. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The CESQL parser of the Java SDK (and likely other SDKs) uses ANTLR to generate a parser (Java) code based on the supplied grammar file. The `CESQLParserParse` class file generated as the result contains a switch statement (at line 457) that lacks a default value. This can lead to unexpected behavior when parsing expressions.

  More in-depth analysis and review of the implementation of ANTLR is necessary to investigate the actual impact of this issue. Moreover, the ANTLR and expressions parsed by its implementation need to be audited to assess potential attack vectors (such as Expression Language injection) based on user-controlled data.

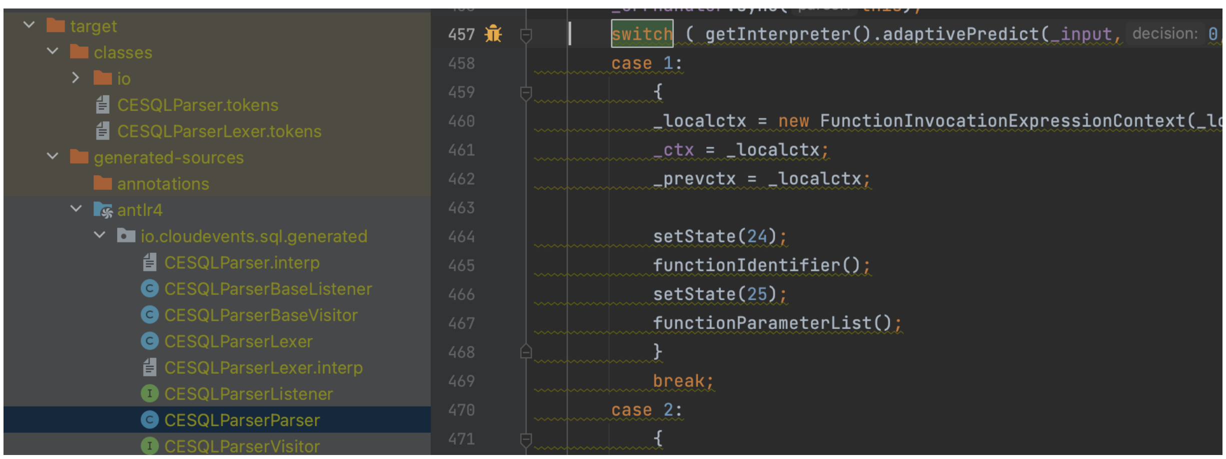


Figure C.1: Switch statement missing default value (`target/generated-sources/antlr4/io/cloudevents/sql/generated/CESQLParserParser.java#457-525`)

- The `CloudEventDeserializer` class in the Java SDK implements a switch statement for parsing the `specVersion` value. The first case ending at line 124 is not ending or breaking, which makes it fall through the next case statement.

```
111    switch (specVersion) {
112       case V03:
113           boolean isBase64 = "base64".equals(getOptionalStringNode(this.node,
this.p, "datacontentencoding"));
114           if (node.has("data")) {
115               if (isBase64) {
116                   data =
```

```
BytesCloudEventData.wrap(node.remove("data").binaryValue());
 117                 } else {
 118                     if (JsonFormat.dataIsJsonContentType(contentType)) {
 119                         // This solution is quite bad, but i see no alternatives
now.
 120                         // Hopefully in future we can improve it
 121                         data = new JsonCloudEventData(node.remove("data"));
 122                     } else {
 123                         JsonNode dataNode = node.remove("data");
 124                         assertNodeType(dataNode, JsonNodeType.STRING, "data",
"Because content type is not a json, only a string is accepted as data");
    125                          data =
       BytesCloudEventData.wrap(dataNode.asText().getBytes());
```

Figure C.2: Select statement fall through can lead to unexpected behavior (`formats/json-jackson/src/main/java/io/cloudevents/jackson/CloudEventDeserializer.java#111–125`)

- `JsonCloudEvcentData()` is documented as deprecated in the Java SDK, but was found to be used in the implementation.

```
 118                 if (JsonFormat.dataIsJsonContentType(contentType)) {
 119                     // This solution is quite bad, but i see no alternatives now.
 120                     // Hopefully in future we can improve it
 121                     data = new JsonCloudEventData(node.remove("data"));
 ...
 136             if (JsonFormat.dataIsJsonContentType(contentType)) {
 137                 // This solution is quite bad, but i see no alternatives now.
 138                 // Hopefully in future we can improve it
 139                 data = new JsonCloudEventData(node.remove("data"));
```

Figure C.3: Using deprecated API methods (`formats/json-jackson/src/main/java/io/cloudevents/jackson/CloudEventDeserializer.java#118–139`)

- The Go SDK is using deprecated Golang APIs in multiple places across the codebase. The staticcheck tool from the Golang toolchain was used to identify and highlight the following cases:

```
    114    ioutil.ReadAll(reader)
  client_protocol.go:13:2: "io/ioutil" has been deprecated since Go 1.16: As of Go
     1.16, the same functionality is now provided by package io or package os, and
     those implementations should be preferred in new code. See the specific
     function documentation for details.  (SA1019)
  internal/connection_test.go:43:38: grpc.WithInsecure is deprecated: use
     WithTransportCredentials and insecure.NewCredentials() instead. Will be
     supported throughout 1.x.  (SA1019)
  internal/connection_test.go:45:38: grpc.WithInsecure is deprecated: use
     WithTransportCredentials and insecure.NewCredentials() instead. Will be
     supported throughout 1.x.  (SA1019)
```

```
protocol_test.go:25:38: grpc.WithInsecure is deprecated: use
   WithTransportCredentials and insecure.NewCredentials() instead. Will be
   supported throughout 1.x.  (SA1019)
write_message.go:11:2: "io/ioutil" has been deprecated since Go 1.16: As of Go
   1.16, the same functionality is now provided by package io or package os, and
   those implementations should be preferred in new code. See the specific
   function documentation for details.  (SA1019)
parser/expression_visitor.go:35:9: assigning the result of this type assertion to
   a variable (switch tree := tree.(type)) could eliminate type assertions in
   switch cases (S1034)
test/tck_test.go:9:2: "io/ioutil" has been deprecated since Go 1.16: As of Go
   1.16, the same functionality is now provided by package io or package os, and
   those implementations should be preferred in new code. See the specific
   function documentation for details.  (SA1019)
client/client_test.go:12:2: "io/ioutil" has been deprecated since Go 1.16: As of
   Go 1.16, the same functionality is now provided by package io or package os,
   and those implementations should be preferred in new code. See the specific
   function documentation for details.  (SA1019)
client/observability_service.go:28:2: this value of ctx is never used (SA4006)
binding/test/mock_binary_message.go:12:2: "io/ioutil" has been deprecated since
   Go 1.16: As of Go 1.16, the same functionality is now provided by package io
   or package os, and those implementations should be preferred in new code. See
   the specific function documentation for details.  (SA1019)
binding/test/mock_structured_message.go:12:2: "io/ioutil" has been deprecated
   since Go 1.16: As of Go 1.16, the same functionality is now provided by
   package io or package os, and those implementations should be preferred in
   new code. See the specific function documentation for details.  (SA1019)
binding/utils/structured_message_test.go:11:2: "io/ioutil" has been deprecated
   since Go 1.16: As of Go 1.16, the same functionality is now provided by
   package io or package os, and those implementations should be preferred in
   new code. See the specific function documentation for details.  (SA1019)
client/client_test.go:13:2: "io/ioutil" has been deprecated since Go 1.16: As of
   Go 1.16, the same functionality is now provided by package io or package os,
   and those implementations should be preferred in new code. See the specific
   function documentation for details.  (SA1019)
protocol/http/message_test.go:12:2: "io/ioutil" has been deprecated since Go
   1.16: As of Go 1.16, the same functionality is now provided by package io or
   package os, and those implementations should be preferred in new code. See
   the specific function documentation for details.  (SA1019)
protocol/http/protocol.go:303:36: should use constant http.StatusTooManyRequests
   instead of numeric literal 429 (ST1013)
protocol/http/protocol_retry.go:13:2: "io/ioutil" has been deprecated since Go
   1.16: As of Go 1.16, the same functionality is now provided by package io or
   package os, and those implementations should be preferred in new code. See
   the specific function documentation for details.  (SA1019)
protocol/http/result_test.go:95:5: should use t.Errorf(...) instead of
   t.Error(fmt.Sprintf(...)) (S1038)
protocol/http/write_request.go:12:2: "io/ioutil" has been deprecated since Go
   1.16: As of Go 1.16, the same functionality is now provided by package io or
   package os, and those implementations should be preferred in new code. See
   the specific function documentation for details.  (SA1019)
```

Figure C.4: Usage of Golang deprecated methods in the SDK (`protocol/ws/v2/client_protocol.go#114`)

- Unhandled errors were identified. Below is an example. The same pattern was observed multiple times across the codebase:

```
func consumeStream(reader io.Reader) {
	//TODO is there a less expensive way to consume the stream?
	ioutil.ReadAll(reader)
}
```

Figure C.5: Unhandled error (`protocol/ws/v2/client_protocol.go#112–115`)

D. Automated Analysis Tool Configuration

As part of this assessment, we performed automated testing on the Skiff codebase using five tools: Semgrep, CodeQL, snyk-cli, yarn audit, and `composer outdated` tools and commands.. Details about testing are provided below.

D.1. Semgrep

We performed static analysis on multiple SDK source code repositories using Semgrep to identify low-complexity weaknesses. We used several rule sets (some examples are shown in figure D.1.1), including our own set of public rules, which resulted in the identification of some code quality issues and areas that may require further investigation. Note that these rule sets will output repeated results, which should be ignored.

```
semgrep --metrics=off --sarif --config="p/r2c"
semgrep --metrics=off --sarif --config="p/r2c-ci"
semgrep --metrics=off --sarif --config="p/r2c-security-audit"
semgrep --metrics=off --sarif --config="p/r2c-best-practices"
semgrep --metrics=off --sarif --config="p/eslint-plugin-security"
semgrep --metrics=off --sarif --config="p/javascript"
semgrep --metrics=off --sarif --config="p/typescript"
semgrep --metrics=off --sarif --config="p/clientside-js"
semgrep --metrics=off --sarif --config="p/react"
semgrep --metrics=off --sarif --config="p/nodejs"
semgrep --metrics=off --sarif --config="p/nodejsscan"
semgrep --metrics=off --sarif --config="p/owasp-top-ten"
semgrep --metrics=off --sarif --config="p/jwt"
semgrep --metrics=off --sarif --config="p/xss"
semgrep --metrics=off --sarif --config="p/supply-chain"
semgrep --metrics=off --sarif --config="p/security-audit"
semgrep --metrics=off --sarif --config="p/golang"
semgrep --metrics=off --sarif --config="r/dgryski.semgrep-go"
```

Figure D.1.1: Commands used to run Semgrep

Alternatively, Semgrep can be configured to automatically detect and use relevant rulesets based on an identified programming language or filename. Note that the auto mode requires submitting metrics online, which means some metadata about the package and repository will be disclosed to the tool developers. This is not an issue with open-source projects but should be considered if Semgrep is used against private or internal repositories.

```
semgrep --config=auto
```

Figure D.1.2: Commands used to run Semgrep in auto mode

D.2. CodeQL

We intended to use CodeQL to analyze multiple SDK codebases. Due to time constraints and the requirements of developing custom queries in order to properly process SDK APIs, this step was skipped in the engagement. However, developers can benefit from this tool in the future, especially when the SDK is integrated with larger codebases.

```
# Create the JavaScript database
codeql database create codeql.db --language=javascript

# Run all JavaScript queries
codeql database analyze codeql.db --format=sarif-latest --output=codeql_res.sarif --
tob-javascript-all.qls

# Create the Golang database
codeql database create codeql.db --language=golang

# Run all Golang queries
codeql database analyze codeql.db --format=sarif-latest --output=codeql_res.sarif --
tob-golang-all.qls
```

Figure D.2.1: Commands used to run CodeQL

D.3. TruffleHog

We ran `composer outdated` on the PHP SDK source repository to highlight outdated packages. Two outdated packages were identified, but the vulnerabilities they contain would not affect the SDK.

```
composer outdated
```

Figure D.3.1: Command used to run TruffleHog

D.4. `snyk-cli`

We ran `snyk-cli` on multiple SDK source repositories to identify outdated and vulnerable third-party packages. Snyk automatically performs recursive checks on sub-modules and different programming languages as dependency configuration files for every used language are found. `snyk-to-html` is a third-party tool and should be installed as a separate Node package if needed. It is worth noting that using snyk cli often depends on the existence of a functional tool-chain for the language. For instance, in order for snyk to be able to produce complete results for Java, the package should be buildable by `maven`.

```
snyk test
snyk test --all-projects --json |snyk-to-html --output ../snyk.html
```

Figure D.4.1: Command used to run `snyk-cli`

D.5. `yarn audit`

We ran the `yarn audit` tool on the JavaScript SDK source directory to identify outdated and vulnerable third-party packages. It is recommended to use `yarn audit` alongside `snyk-cli` for better coverage. The `yarn-audit-html` is a third-party tool and should be installed as a separate Node package if needed.

```
yarn audit
yarn audit —group dependencies
yarn audit --high |grep -E 'high|critical' |sort|uniq
yarn audit --json |yarn-audit-html --output ../yarn.html
```

Figure D.5.1: Command used to run `yarn audit` against JavaScript SDK

D.6. Intellij IDE Plugins

We benefited from the following Intellij IDE plugins during our manual code review process to quickly highlight common vulnerable code patterns.

- FindBugs (with FindSecurityBugs plugin)
- Snyk Security (Identify vulnerabilities in dependencies)
- CheckMarx AST (Identify vulnerabilities in dependencies)
- SonarLint (Identify common vulnerable code patterns)
- PVS-Studio (Identify common vulnerable code patterns)
- Built in inspectors