UrbanCode Deploy Appscan Enterprise Plugin

One aspect of testing an application after deployment is to test the application for security issues.

UCD provides a plugin that executes an Appscan Enterprise scan and retrieves the report.

Topology

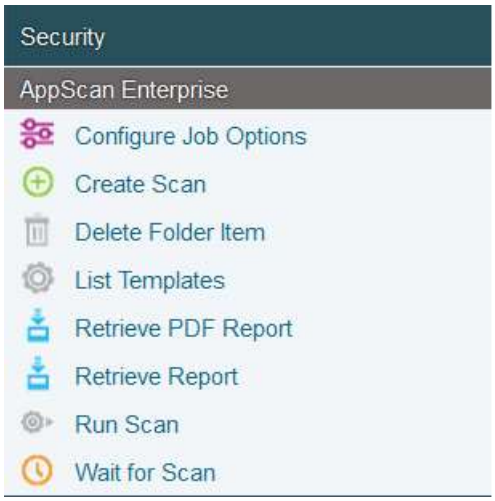
In this demonstration, Appscan Enterprise and UrbanCode Deploy are both installed on an Amazon Web Services Win 2016 server.

UCD Appscan Enterprise plugin:

The UCD Appscan Enterprise plugin must be downloaded and installed in UCD before proceeding. Download the plugin from:

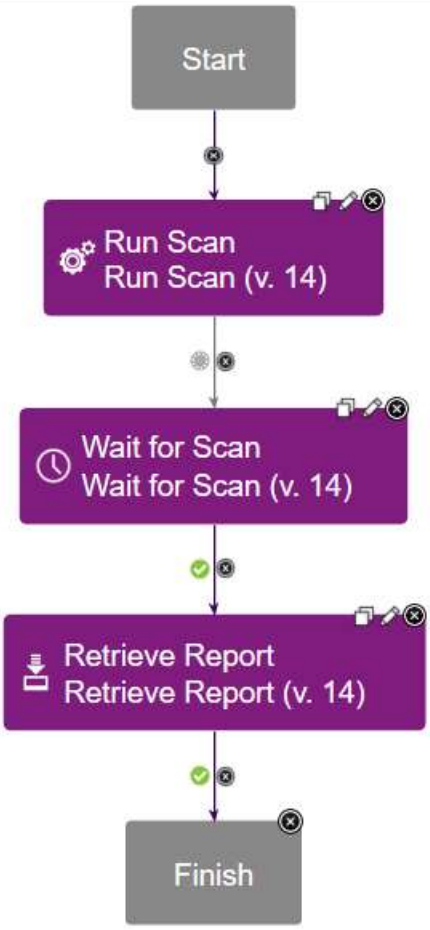
https://developer.ibm.com/urbancode/plugin/ibm-appscan-enterprise/

The security options that are available are:

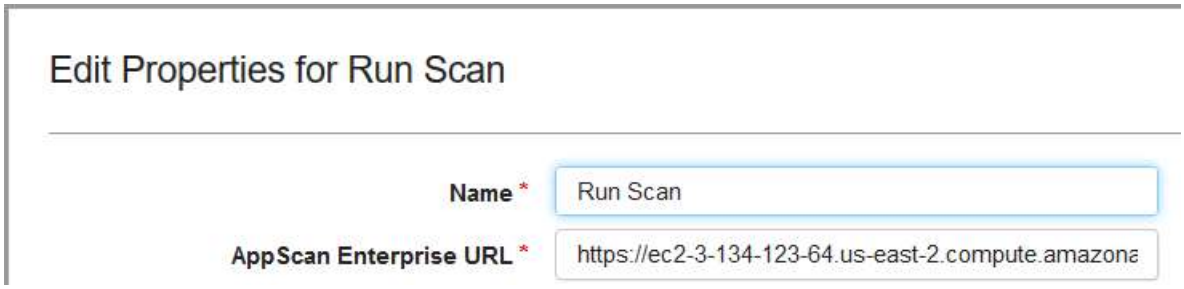


UCD Configuration

The process flow for the Appscan process consists of 3 steps and a post-processing script.



Configuration of the parameters for each step requires a bit of a caveat.



The AppScan Enterprise URL should not include the port or the path. This was a mistake that I made and it lead to unexplained errors.

Presently, all the parameters are hard coded, however when you first define the steps, parameters are suggested. The only parameter used in this example is the report name, which is set to Security Issues.

Post Processing Script for Retrieve Report step:

To determine if the deploy scan was successful or not, a post-processing script scans the log files, looking for Critical and High errors. The script is based on the following from the knowledge center: https://www.ibm.com/support/knowledgecenter/SS4GSP_7.0.3/com.ibm.udeploy.doc/topics/comp_postprocess_examples.html

The actual script is:

```
var exit = properties.get('exitCode');


// scan the log looking for High Severity issues.
scanner.register("High Severity Issues:", function(lineNumber, line) {
    properties.put('Status', 'Failure');
    var NumHighIssues = line.replace("High Severity Issues: ", "");
    commandOut.print( "High Severity Errors found");
});
scanner.scan();


// scan the log looking for Critical Severity issues.
scanner.register("Critical Severity Issues:", function(lineNumber, line) {
    //var thing = 'do stuff';
    properties.put('Status', 'Failure');
    var NumCriticalIssues = line.replace("High Severity Issues: ", "");
    commandOut.print("Critical Severity Errors found");
});
scanner.scan();
```

The results can be observed in the log file, which highlights the matched lines in the log file:

```
[OK] Authentication was successful.
Logged in.
Retrieving Report...
[Fri Jan 10 20:39:05 UTC 2020] Status: Running
[Fri Jan 10 20:39:35 UTC 2020] Status: Ready
Scan or Report is in READY state.
Reports URL: https://ec2-3-134-123-64.us-east-2.compute.amazonaws.com/ase/services/folderitems/60/reports
Report Pack Last Run: 2020-01-10T20:38:56.153Z
Retrieving report summary information...
```

[Ok] Generated Specific Report XML: C:\Program Files\ibm-ucd\agent\var\work\JPetStore-APP\60-Summary.xml
Security Issues Report URL: https://ec2-3-134-123-64.us-east-2.compute.amazonaws.com/ase/services/reports/356/data
Acquired the count of all Security Issues issues.
Critical Severity Issues: 0
High Severity Issues: 7
Medium Severity Issues: 0
Low Severity Issues: 1
Retrieving complete Security Issues information...
[Ok] Generated Specific Report XML: C:\Program Files\ibm-ucd\agent\var\work\JPetStore-APP\60-Security-Issues.xml
Retrieved Report successfully.

==========
Post Processing Script Execution Console Output:

High Severity Errors found

Reporting results to JIRA

The next step will be to capture the post-processing script results and report the issues on a JIRA ticket. This will be similar to work done before with an RFT script that checked behaviour of a deployed application.

The flow of the deployment will be modified to branch based on the results of the Retrieve Report step. In case of success, the flow will pass to the end step. In the case of a failure, the flow will retrieve the report as a PDF file, which then can be attached to the JIRA ticket.

JIRA Integration

Creating Token for Jira Integration

The steps to create a token for accessing JIRA in the cloud are documented in this document:

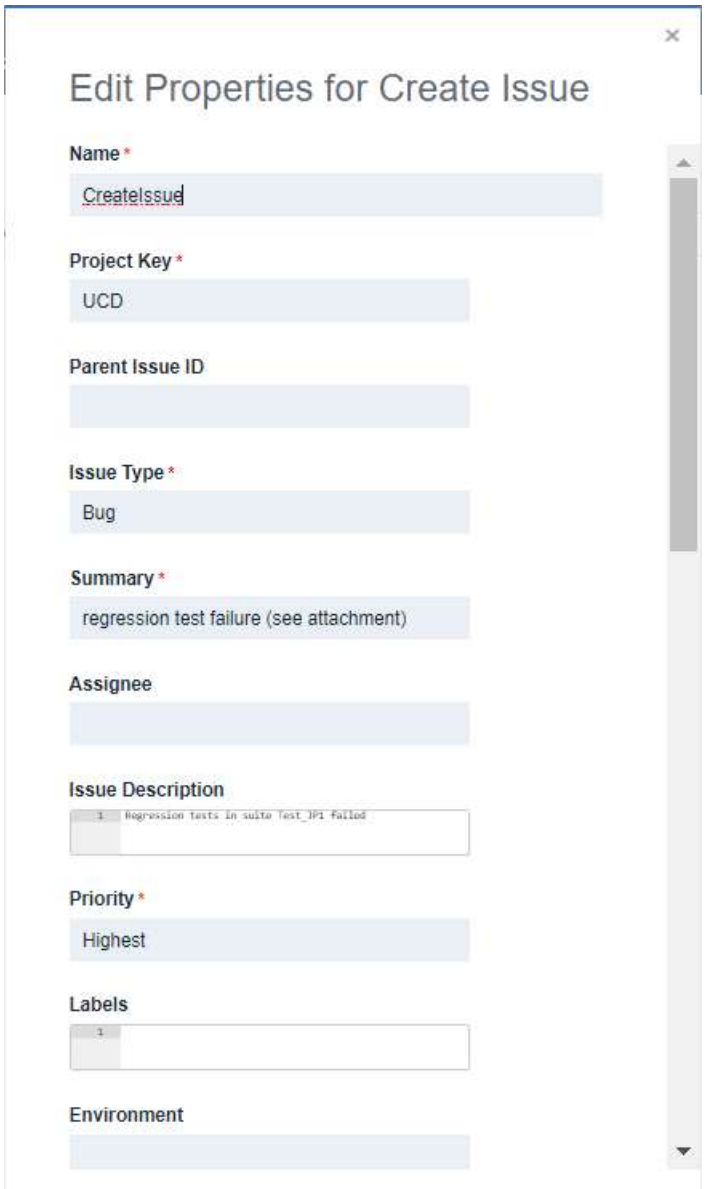
1) Login to JIRA https://hclparagon.atlassian.net
2) In your account settings, look at the Security settings
3) Find the section titled Create and Manage API tokens
4) You will see a list of tokens and a button Create New Token
5) When you create a token, you have a single opportunity to copy the token, the only other option is to revoke the token and start over.
6) Use the token in lieu of a password when accessing JIRA via the plugin.

To setup the integration, JIRA needs to create a couple of tokens. I did this for RPT six months ago, but I do not recall the values of those tokens. So here are new tokens for ASE, which will serve in this integration:

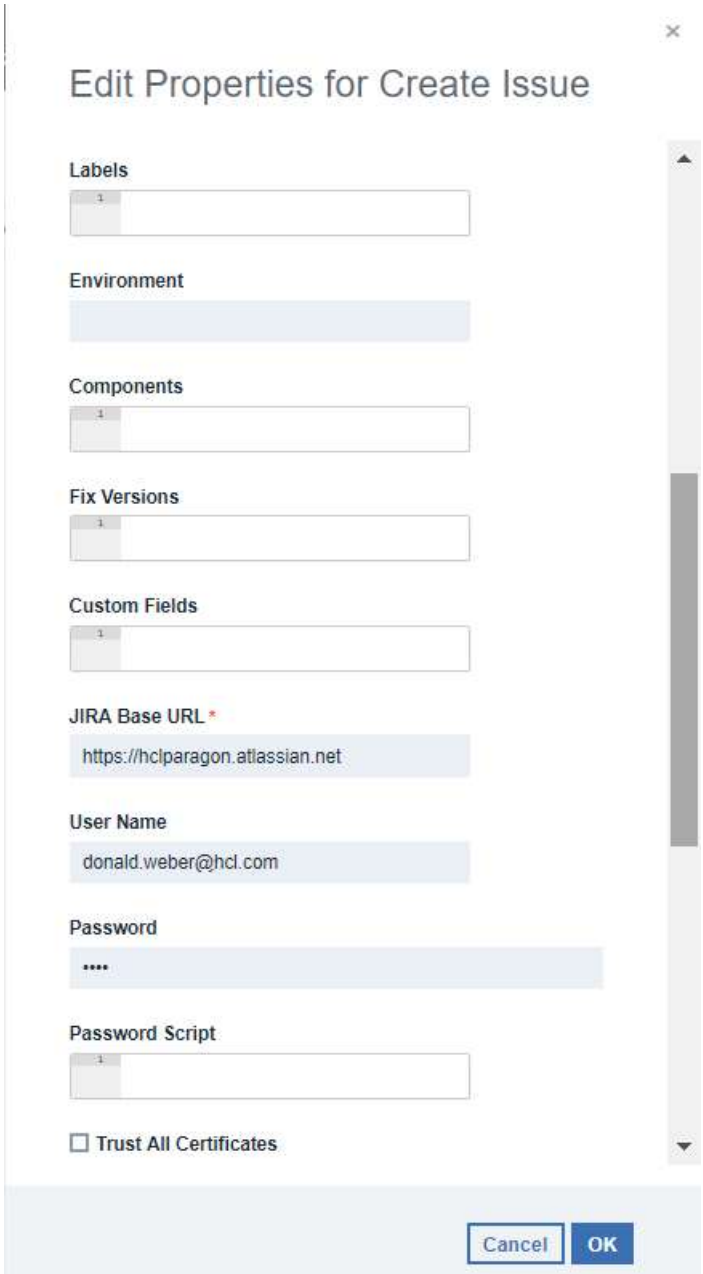
ASE_CLI_token = UcuoRQJeSswr4u1y3Y8E9C4C

Create Issue

The first step is to create a JIRA ticket, which uses the JIRA plugin. The JIRA plugin input properties are:

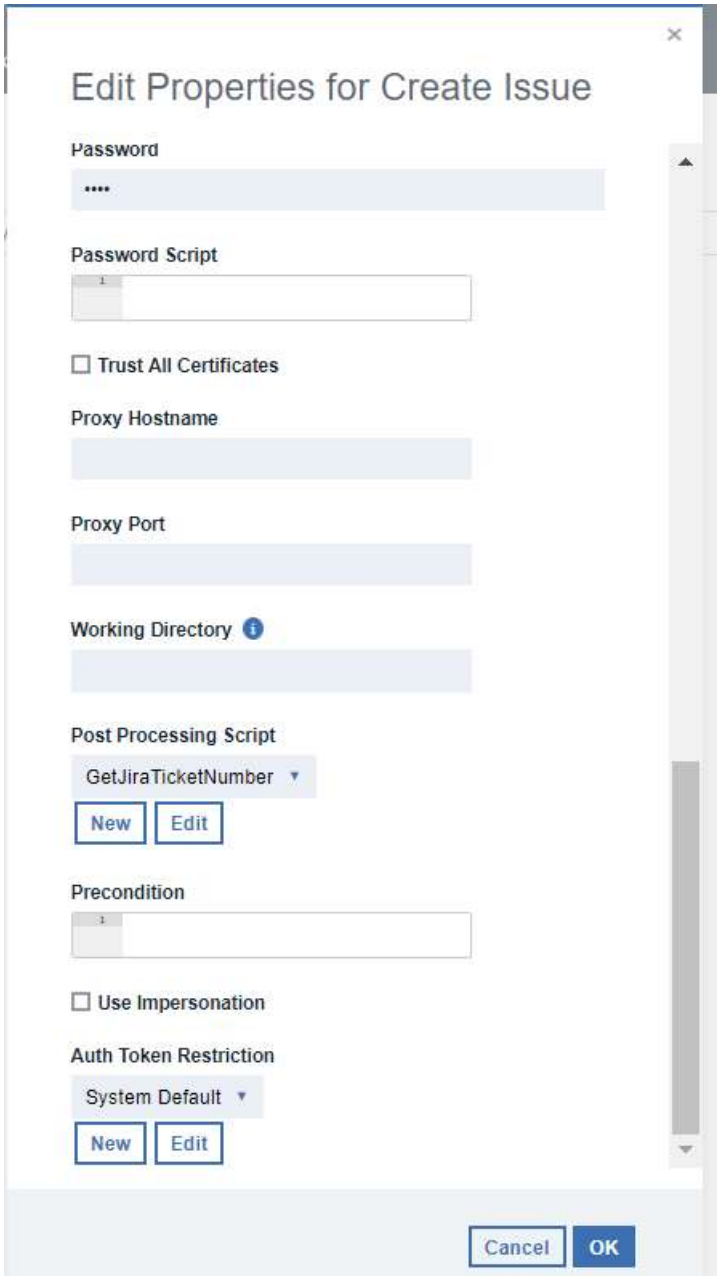


The name of the step is Create Issue, the key is UCD which was established by David Greggs for this project integrating UCD/RFT. The issue type corresponds to the JIRA type Bug, the summary is a rather boring text string, leaving details to the attachments. The issue description is another rather boring text string. The priority is set to Highest, which is the desired status based on a consensus of the project Paragon team. Farther down the page:



The JIRA Base URL is a required field. Note that because JIRA is located in the cloud (atlassian.net) that affects the credentials. The username is my e-mail address, but the password is the Token that I assigned in JIRA for the integration.

Farther down the page is one last important detail:



The post processing script will scan the log for this step, capturing the Jira Ticket Number, which is required for the next steps of attaching files. As with the post processing script for the RFT Plugin, the script can be edited right at this point, or in the administrative tab.

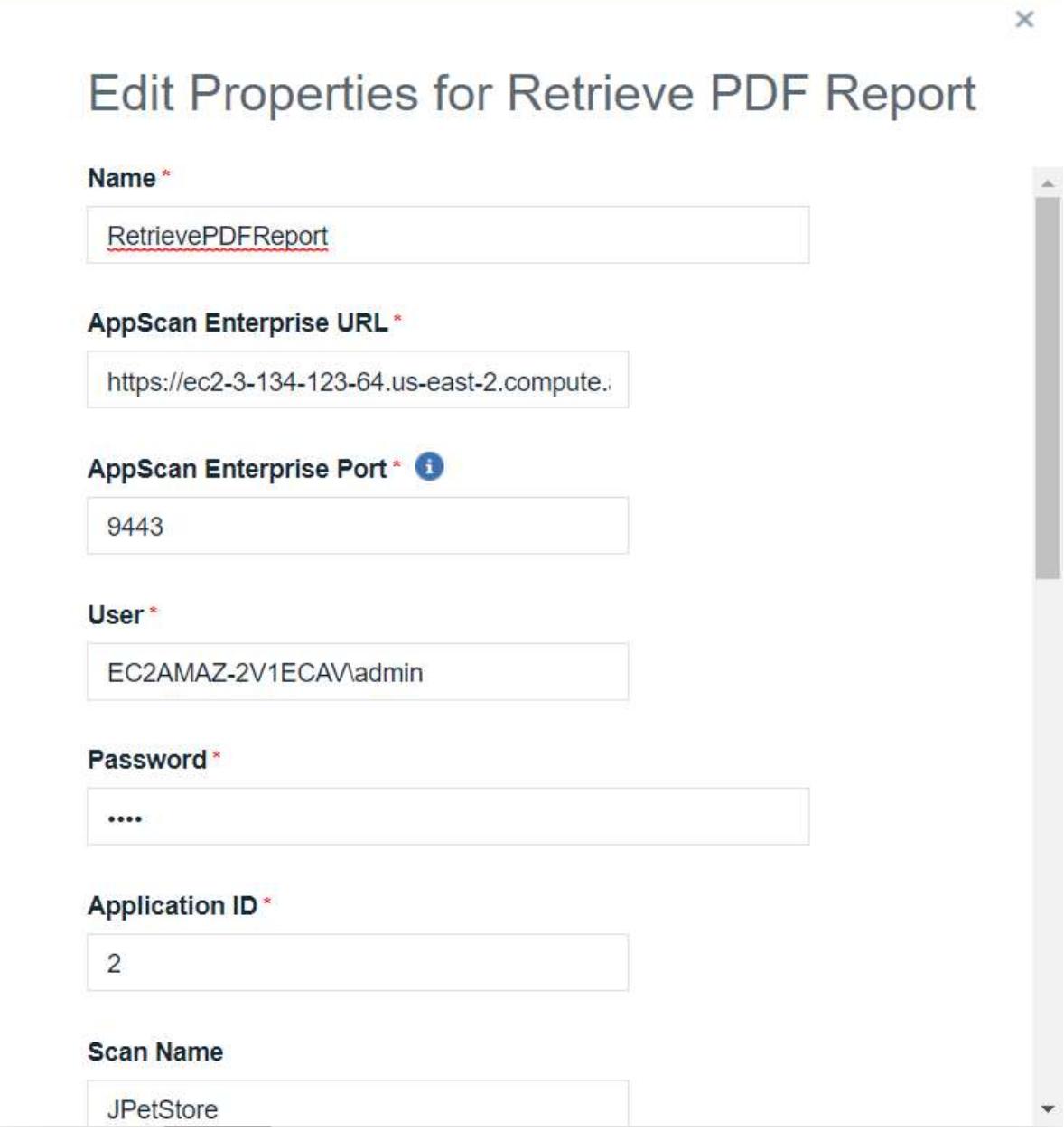
One key detail of the post-processing script is that any property you capture, such as JiraTicket, is qualified by the step name. So, the step name should be CreateTicket and the property name JiraTicket.

Create Issue - Post-Processing Script

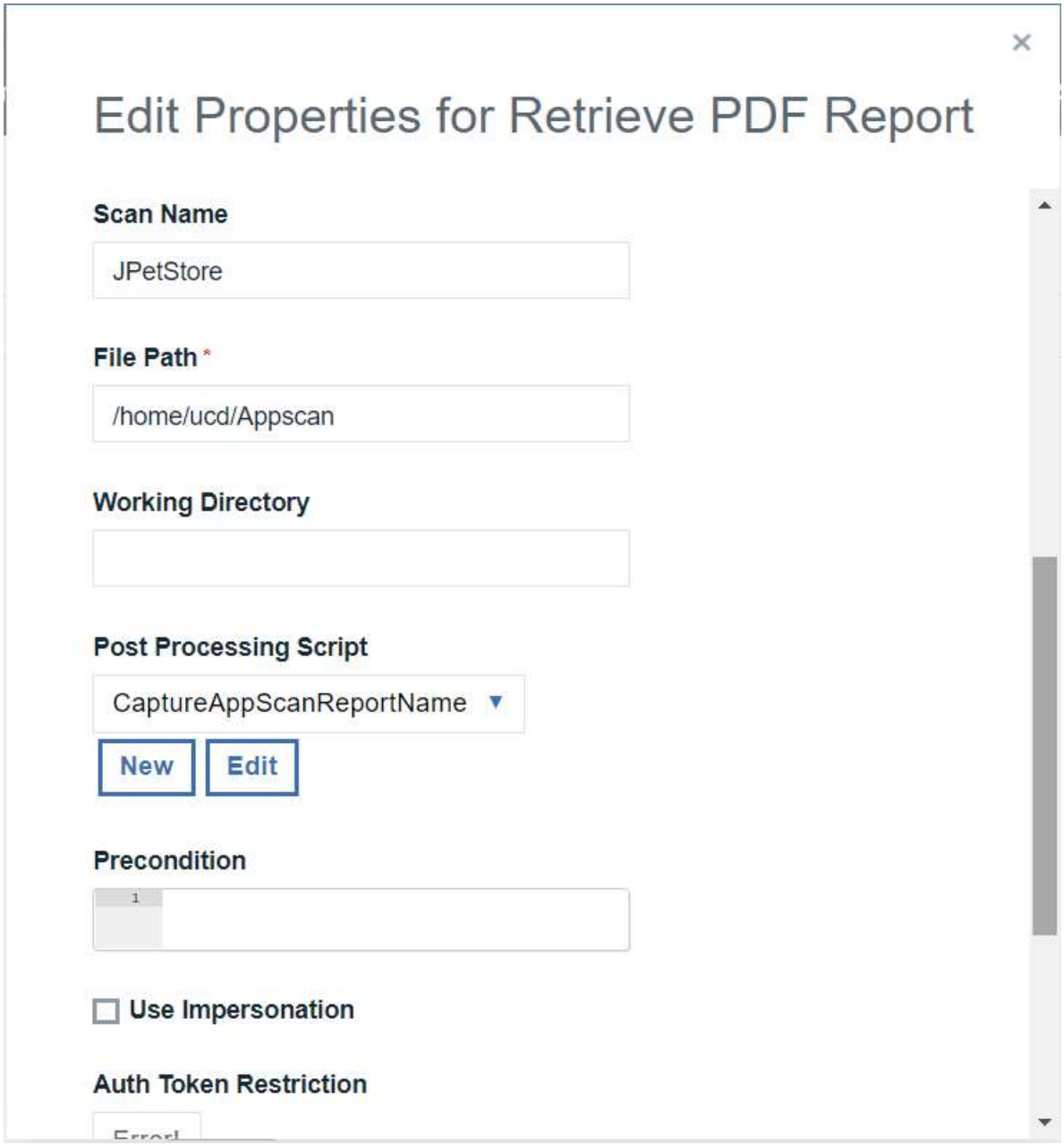
```
var exit = properties.get('exitCode');

scanner.register("Creation of new Issue", function(lineNumber, line) {
     line=line.replace("Creation of new Issue","");
     line=line.replace("was successful.","");
     line=line.trim();
     properties.put("JiraTicket",line);
});
scanner.scan();

// See if we can print the results:
var ticket = properties.get("JiraTicket");

commandOut.print("JiraTicket : ");
commandOut.println(ticket);
commandOut.print("\nJiraTicket : " + properties.get("JiraTicket"));

if (exit == 0) {
    properties.put('Status', 'Success');
}
else {
     properties.put('Status', 'Failure');
}
```

Retrieve PDF Report

After creating the JIRA ticket, details regarding the issue should be attached. The ASE plugin provides a Retrieve PDF Report step, which is used to get the attachment. The input parameters are:



And farther down the page:



A key point was that the file path must point to the directory where you intend to write the attachment (which will be a zip file containing the PDF file). At present the PDF file isn't very detailed, an issue to discuss with Shawn. At first I tried to put the files into /tmp/Appscan, but that was causing errors, writing to /home/ucd/Appscan was successful.

As with Create Issue, an important caveat, the step name (RetrievePDFReport) will be used with the property AppscanPDF to identify the file created in /home/ucd/Appscan. So name the step accordingly and set the property in the post-processing script.

RetrievePDFReport – Post Prosessing Script

The post processing script will locate the name of the AppscanPDF property and set the property value for use in the Attach Security Report step to follow.

The post processing script is:

```
var exit = properties.get('exitCode');


// scan the log looking for High Severity issues.
scanner.register("^.*Generated Report Zip:.*", function(lineNumber, line) {
     properties.put('Status', 'Success');
     var ReportZip = line.replace("[Ok] Generated Report Zip: ", "");
     commandOut.print( "Found line " + line );
     properties.put('AppscanPDF',ReportZip);
     commandOut.print("\nAppscanPDF = " + properties.get('AppscanPDF'));
});
scanner.scan();



if (exit == 0) {
    properties.put('Status', 'Success');
}
else {
     properties.put('Status', 'Failure');
}
```

Note that the report name is stored in AppscanPDF

Attach Security Report

Attaching the security report depends on 2 previously captured values, the JiraTicket from the Create Issue step and the AppscanPDF from the Retrieve PDF Report step. They are just shell scripts that execute a curl command.

The actual curl command is:

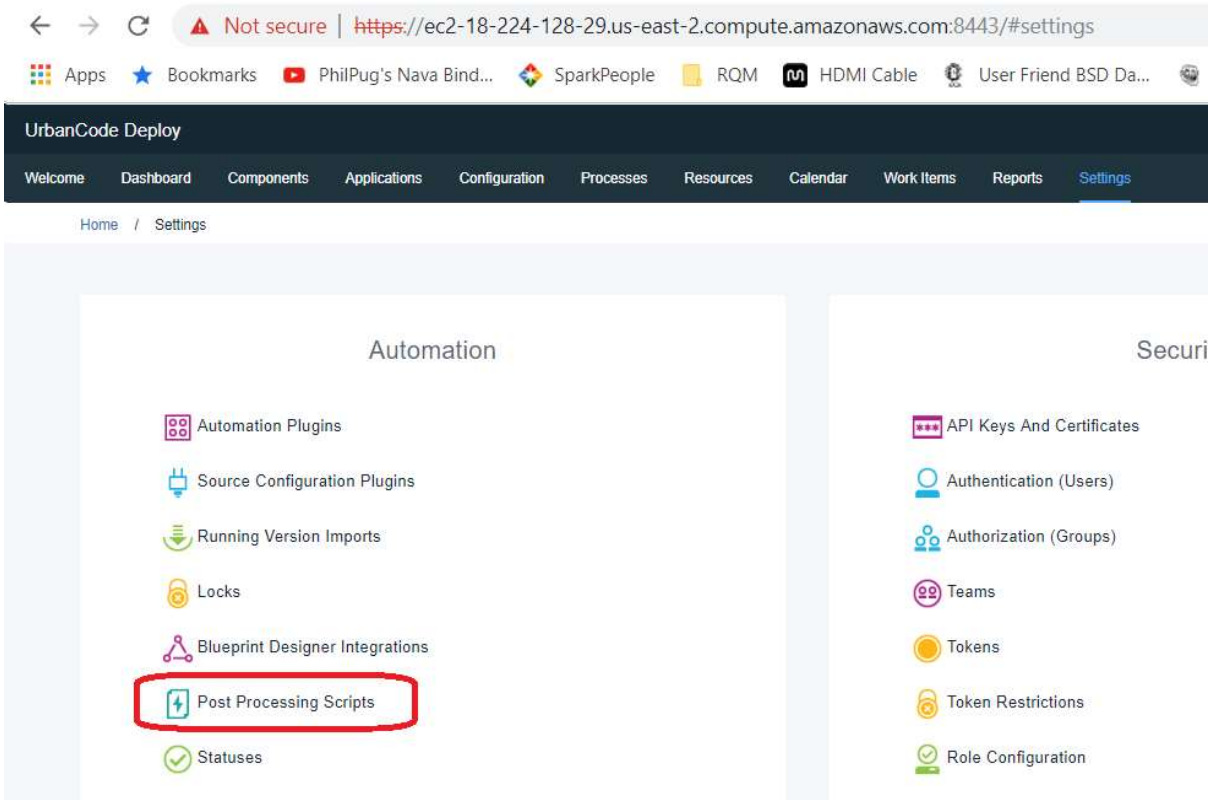
```
/usr/bin/curl -D- -u donald.weber@hcl.com:UcuoRQJeSswr4u1y3Y8E9C4C -
X POST -H "X-Atlassian-Token: no-check" -F
"file=@${RetrievePDFReport/AppscanPDF}"
https://hclparagon.atlassian.net/rest/api/2/issue/${CreateIssue/Jira
Ticket}/attachments
```

Note that the variable extracted in the post process step ${CreateIssue/JiraTicket} is used to specify where the attachment should be made. The variable extracted in the post process step ${RetrievePDFReport/AppscanPDF} is used to specify the file to be attached.

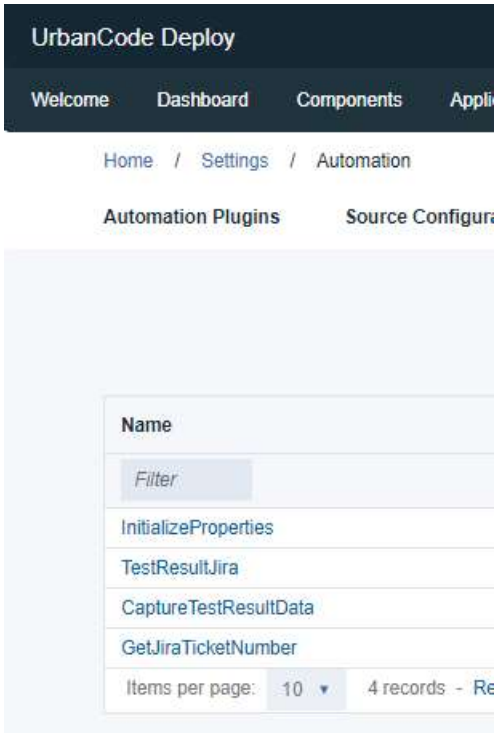
Post Processing Scripts

The post processing scripts used in this demonstration retrieve data from the log file for each test step and the JIRA create ticket steps. The purpose of the scripts is to extract string data that is necessary for following steps.

Post processing scripts are assigned in the plugins, where they can be edited, however you can also view all post-processing scripts from the Settings view of UCD:



The post processing scripts that have been created are:



Basic Post Processing Script structure

The UCD knowledge center contains helpful information on the usage of post processing scripts.

https://www.ibm.com/support/knowledgecenter/en/SS4GSP_7.0.1/com.ibm.udeploy.doc/topics/comp_postProcess.html

There are even examples, which were helpful to creating the first iteration of the script.

https://www.ibm.com/support/knowledgecenter/en/SS4GSP_7.0.1/com.ibm.udeploy.doc/topics/comp_postprocess_examples.html

Get Jira Ticket Number

After the Jira ticket is created, the ticket number is required to attach files. This post processing script scans the Jira Ticket log to capture the ticket number, which is then re-used in the attach file steps.

```
var exit = properties.get('exitCode');


scanner.register("Creation of new Issue", function(lineNumber, line)
{
     line=line.replace("Creation of new Issue","");
     line=line.replace("was successful.","");
     line=line.trim();
     properties.put("JiraTicket",line);
```

```
});
scanner.scan();

// See if we can print the results:
var ticket = properties.get("JiraTicket");

commandOut.print("JiraTicket : ");
commandOut.println(ticket);

if (exit == 0) {
    properties.put('Status', 'Success');
}
else {
     properties.put('Status', 'Failure');
}
```

CaptureAppscanReportName – Post Prosessing Script

The post processing script will locate the name of the AppscanPDF property and set the property value for use in the Attach Security Report step to follow.

The post processing script is:

```
var exit = properties.get('exitCode');


// scan the log looking for High Severity issues.
scanner.register("^.*Generated Report Zip:.*", function(lineNumber, line) {
     properties.put('Status', 'Success');
     var ReportZip = line.replace("[Ok] Generated Report Zip: ", "");
     commandOut.print( "Found line " + line );
     properties.put('AppscanPDF',ReportZip);
     commandOut.print("\nAppscanPDF = " + properties.get('AppscanPDF'));
});
scanner.scan();



if (exit == 0) {
    properties.put('Status', 'Success');
}
else {
     properties.put('Status', 'Failure');
}
```

Note that the report name is stored in AppscanPDF