An Unauthenticated Journey to Root :

Pwning Your Company's Enterprise Software Servers

Yvan Genuer
Onapsis
ygenuer@onapsis.com

Pablo Artuso
Onapsis
partuso@onapsis.com

1. Abstract

Companies consist of a plethora of software, hardware, vendors, and solutions working together to keep the business running and alive. When it comes to Enterprise Software, because of its intrinsic criticality and complexity, companies rely on expert vendors in the field. SAP, one of the largest Enterprise Software vendors, is definitively part of this list. With huge presence around the globe, SAP systems could be found in almost every mid-to-large company providing one of the most critical actions: Administrating the company's business.

This paper will illustrate an in-depth analysis performed against one of the most important assets of every company: its SAP implementation. All phases of the research will be depicted, starting from the introduction of each analysed component, continuing with the techniques that helped us find the vulnerabilities, up to discussing all security measures that could be followed in order to be protected against them. Furthermore, post exploitation scenarios will be discussed which will help to better illustrate the impact that the found flaws could have, not only in technical but also in business terms.

2. Keywords

SAP, Enterprise Software, RCE, root, Solution Manager, SolMan, Host Agent, CVE-2020-6207, CVE-2020-6234, CVE-2020-6236

3. Acknowledgements

To our leader, Nahuel D. Sánchez, who helped us throughout the research, providing support, sharing discussions and pushing us in order to get the best out of ourselves.

4. Introduction

Enterprise software is one of the most important topics when discussing about company's assets. They usually manage sensitive and critical information. It is because of this reason that companies opt for experts in the field to trust one of their most critical assets. SAP is one of the largest vendors of Enterprise Software. They have been successfully developing business applications for almost 50 years. With more than 450k customers and presence in more than 180 countries, is possible to believe that almost every mid to large company today is using SAP systems for keeping its business up and running.

The list of products that SAP offers is very extensive. Customers may choose which product to use based on their particular needs. However, there is one particular solution that will be present in every customer network: SAP Solution Manager (SolMan).

This paper is the outcome of an extensive security research performed against the SAP Solution Manager: the core component of every SAP implementation. A completely practical attack which leverages not only the power of SolMan but also the relation with its agents will be presented.

4.1. Solution Manager

In SAP landscapes, the SAP Solution Manager (SolMan) could be compared to a domain controller system in the Microsoft world. It is a technical system that is highly connected with powerful privileges to all other SAP systems. Once an SAP system is connected to the solution manager it receives the name of "managed" or "satellite" system. As an administration solution, SolMan aims to centralize the management of all systems within the landscape by performing actions such as implementing, supporting, monitoring and maintaining the enterprise solutions.

4.2. SMD Agent

If an SAP customer wants to fully utilize the capabilities of the Solution Manager, they must install an application called Solution Manager Diagnostic Agent (SMDAgent) on each host where an SAP system is running. This Agent manages communications, instance monitoring and diagnostic feedback to the Solution Manager. From the operating system perspective, the unique user involved in all SMDAgent activities is daaadm.

4.3. Host Agent

The SAP Host Agent[1] is a component installed automatically during the installation of a new SAP system. It is OS and database independent, and it can accomplish several life-cycle tasks such as:

- Monitoring
- Start/Stop instances
- Preparing for upgrade

5. Architecture

From an architecture point of view the Solution Manager, as explained before, is connected to every single system within the landscape. As usually every managed/satellite system use the full capabilities of SolMan, the SMDAgent could be found in every server where they are running. Finally, the SAP Host Agent is a solution that could be used for both SAP and non-SAP applications. Nevertheless, every SAP application will have this agent running within its server.

To summarize it and visualize it in a more graphical way, the following image can be used as an example:

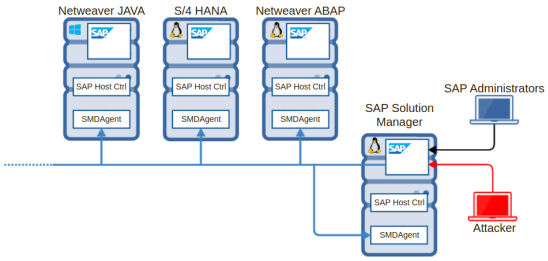

Figure 1: *Example architecture including Solution Manager, SMD Agent and Host Agent.*

This picture shows an SAP landscape composed by 4 systems (one Solution Manager, 3 managed systems) running different operating systems and different SAP NetWeaver stacks. As can be seen in the image, the SolMan is connected to every other system. SMD and Host agents are present in all servers (including SolMan itself).

6. Solution Manager Analysis

6.1. Motivation

There were several reasons why the Solution Manager was a interesting target for an in-depth analysis of its security. Among all them, it is possible to highlight the following two:

- It is an hyper-connected system: It is connected to all other systems in the same landscape.
- Every SAP implementation must have one.

Finding any issue in such critical asset would potentially imply an impact on some other system of the landscape. The main objective was:

[1] https://help.sap.com/doc/saphelp_nw73ehp1/7.31.19/en-US/48/c6f9627a004da5e10000000a421937/content.htm

First to demonstrate if there was a way for an attacker to take control on the system and second, understand how deep they could go.

6.2. Initial Phase

One of SolMan's components is the web server part of the Java stack. As every other SAP solution running on top the NetWeaver Java technology, several applications, services and functionalities are provided through it. For instance, to perform configurations on the SMD Agents, a particular application running on this web server must be used.

The initial phase of the Solution Manager analysis involved reviewing the exposed applications and web services with the goal of identifying those which did not require authentication. Even though authenticated ones may also be vulnerable, the impact of an unauthenticated exposed application would be naturally higher.

Looking for unauthenticated applications

This search focused mainly on application and web services exposed by default. In order to find them, the following approaches were followed

- Log analysis, with the goal of finding previous communications.
- Configurations analysis, with the goal of finding apps and their exposure based on configurations written either in files or in the database.
- Documentation analysis.

After an initial phase of discovery a list of candidates applications was identified and ready to be analysed. Among this list, there was one that was worth to be analyzed: End-user Experience Monitoring (EEM).

6.3. Unauthenticated application discovery: EEM

Once this potential vulnerable endpoint was identified an intensive analysis on it began. Based on online documentation, self testing, and more, it was possible to understand the goal and nature of this application.

The End user Experience Monitoring application allows to mimic any activity that could be performed by an end-user using proprietary SAP protocols such as: RFC, DIAG or HTTP protocols. From a technical point of view, in order to achieve such actions, scripts of the aforementioned protocols can be developed, uploaded and executed against other systems. Despite the fact that EEM runs in the Solution Manager, the actual execution of the uploaded/deployed scripts is carried out by an EEM Robot; A system that is running a tiny Java application which knows how to interpret and execute these scripts. By default, every SMD Agent connected to the Solution Manager could act as an EEM Robot. From now on, the concept of SMD Agent or EEM robot will be used indistinguishably.

In summary, these are the steps that are carried out:

1. SAP administrator gets a script (either by developing it or asking somebody else to do it).
2. The admin chooses the EEM Robot where the script will be deployed.
3. Once the script is deployed, the EEM robot executes.

As it was mentioned before, this application was not requiring authentication. That was the reason why the following questions came up:

- Is this application able to be used without neither authentication nor authorization?
- Is there any critical action that could be performed through scripts?

6.4. EEM Technical Analysis

Technically speaking, EEM was a SOAP endpoint. Once its WSDL was retrieved and parsed, a set of 19 methods were available to be called. Among these methods it was possible to find:

- getAllAgentInfo
- runScript
- setAgeletProperties
- uploadResource

A quick test to ensure that action could be performed without authentication was performed. Using SOAP UI[2] the method **runScript** was executed. This method required two parameters: the **EEM robot host** and **script name**. As explained before, all SMD Agents are EEM robots by default. At that point, a valid **script name** was unknown, that's why the following values were used:

- **EEM robot host:** A valid SMD Agent host
- **Script name:** "Foo"

The answer received by the server was not the one expected:

```
<errorMessage>com.sap.smd.eem.admin.Eem-
Exception: EEM is not enabled on this
agent. Operation only supported when EEM
is enabled.</errorMessage>
```

The returned message was not giving any kind of information related to the question of "Is this application able to be used without neither authentication nor authorization?". Moreover, it added a new one:
If this application comes deactivated by default, should an administrator explicitly enable it?

Remotely enabling EEM without authentication

The result obtained from the previous test, forced to continue with the analysis of exposed methods. "Getter" methods are usually easier to execute when the application is a bit unknown as sometimes they do not require parameters and return back information that may be valuable. This case was not the exception. Among all the listed methods, there was one called **getAllAgentInfo**. This method did not require any parameter, therefore was easy to execute.

[2]https://www.soapui.org/

Just because of having an answer from the execution of **getAllAgentInfo**, confirmed the idea that neither authorization nor authentication was needed to use the application. And therefore, the first vulnerability could be confirmed.

Once executed the information returned was really interesting:

- OS version
- JDK version
- Environmental information (variables, configs)
- EEM properties

EEM properties held configuration directly related with the application in a key-value fashion. Among all those properties there was a really interesting one:

- ...
- eem.enable = False
- ...

The name of this property led to a new search over the methods in order to try to find a way to change this value. After a quick search the method **setAgeletProperties** appeared to be a candidate. The idea behind it was to try to activate the EEM through changing its value to True. **setAgeletProperties** required three parameters:

- EEM robot host.
- Key.
- Value.

The test used a valid SMD Agent host as **EEM Robot host**, "eem.enable" as **Key** and "True" as **Value**. Once this execution was accomplished, **getAllAgentInfo** was launched again. This time the value of **eem.enable** was "True".

In order to definitively confirm that the change worked, the first test was re-executed. The method runScript was called with the same parameters as before but this time the message was different:

```
<errorMessage>com.sap.smd.eem.admin.Eem-
Exception: Script foo_script not found.
</errorMessage>
```

This message confirmed that EEM was enabled and ready to be used by anyone without ever providing neither authentication nor authorization. However, what was the impact of having such an application accessible to anyone? In order to answer this question it was necessary to make a more in depth analysis of the features provided by it.

Uploading and running custom scripts

One of the main goals was to find the way of creating custom scripts, upload them and get them executed. Until this point, thanks to the previous analysis, it was known how to run a script that was already present in some SMD Agent. It was necessary to find a way to upload any arbitrary script.

Among all functions exported by the endpoint, there was one called **uploadResource**. This function required several parameters. The most important ones were: **SMD Agent host** and **Content**. The latter had to be base64 encoded. The first test was to encode some random string and upload it in order to see if the returned message was giving some information. The test was kind of successful as the returned message was:

```
<errorMessage>FatalError validating
XML document: Content is not allowed
in prolog</errorMessage>
```

Analysing the error message it was possible to realise that the content of the scripts should be XML.

With the help of documentation found on the Internet[1], along with resources provided by the application itself, a more in-depth understanding of the application was achieved. Recalling what was mentioned in section 6.3, several protocols were able to be scripted: RFC, DIAG, HTTP, SOAP, etc. In order to build and create your own scripts in a more friendly way, SAP provides a tool called EEM Editor. Unluckily, the power of this tool was leveraged after the analysis was finished.

Among all files provided by the application itself, there was an example of an HTTP script. This example script was really valuable for the research as it was helpful to understand more about the protocol and how scripts were actually written.

Whenever an invalid script (incorrect tag, syntax error, etc) was uploaded, a really detailed error message was returned. For example:

```
Error validating XML document: Invalid
content was found starting with element
'blahblah'. One of '{Annotation, Headers,
Param, Check, Search, Part}' is expected
```

Modifying the aforementioned example script plus leveraging the error messages returned, it was possible to create a custom script that was able to perform arbitrary HTTP requests. As EEM scripts are executed by EEM Robots, this meant that a Server Side Request Forgery could be achieved.

The possibility of executing arbitrary HTTP requests was enough to illustrate the impact of this unauthenticated application. This vulnerability was identified with CVE-2020-6207.

However, we wanted to understand how far an attacker could go in terms of exploitability. Until this point, it was possible to execute any type of script, but ... what actions can be scripted? Is it possible to execute OS commands?. In order to find the answer to these questions it was necessary to better understand the scripting language and its capabilities.

Understanding and exploiting the scripting language

Besides having all the available documentation online for the aforementioned scripting language, it was needed to go deeper in technical terms. Until this point, every part of the research was done in a Black-box style. Leveraging access to Solution Manager's OS files, the analysis shifted to a White-box approach.

Among all files related to the EEM application, there was one called **Config1.1.xsd**. This schema file, defined the structure the scripting language had to follow: Fields, Type of Fields, Tags, etc. This discovery allowed to deeply un-

derstand how this language was designed and how to get the most of it.

It was a message-type language based on XML. Each script could have one or several transaction steps. Each step could be made of one or multiple messages. According to the schema file, each message must had one of the following types:

- Think
- Reset
- ServerRequest
- Command

The third one was the type used by the SSRF script explained in the previous section. The last one, **command**, seemed to be interesting to analyze. Based on online documentation and some local analysis of files, it was possible to list all available commands:

- Assign
- AssignJS
- AssignFromList
- AssignFromFile
- WriteVariableToFile
- ReadVariableFromFile

The reason why this scripting language had commands available to be executed, was basically to provide support for some actions that may be out of scope of the language itself. It was possible to execute any HTTP script, but unless having extra features (provided by commands in this case) it would be impossible to store data persistently or share data between several scripts.

An in-depth analysis started in each of these commands looking for any flaw that may end up in a potential vulnerability. After several tests creating new scripts that made use of each of them, it was discovered that **AssignJS** was vulnerable to code injection.

Seemed that **AssignJS**, was evaluating any arbitrary piece of JS code that was sent inside a parameter of the script, without executing any prior sanitization. Given that the application in charge of executing the script (part of the SMDAgent) was written in java and that this command was evaluating Javascript code, it is possible that the flaw was related to the ScriptEngine API[2]. Below is an example of a vulnerable function to illustrate this flaw:

```
private String ExecuteCommand(final
 String expression){
    final ScriptEngineManager manager =
      new ScriptEngineManager();
     final ScriptEngine js_engine =
       manager.getEngineByName("js");
     final String res =
       engine.eval(expression);
     return res
}
```

With the appropriate payload this could lead an attacker to have a Remote Code Execution. Due to this payload is directly obtained from the script itself it meant that any unauthenticated attacker could be able to exploit it. Additionally, as the scripts were executed by the Java application running in the SMD Agent, the commands run with the privileges of the daaadm user. As explained in section 4.2, daaadm was the OS layer user of the SMDAgent component, which means the attacker would be able to full compromise it.

6.5. Conclusions

As a summary, two vulnerabilities were found:

1. Authentication bypass of EEM application.
2. Remote Code Execution abusing specific commands.

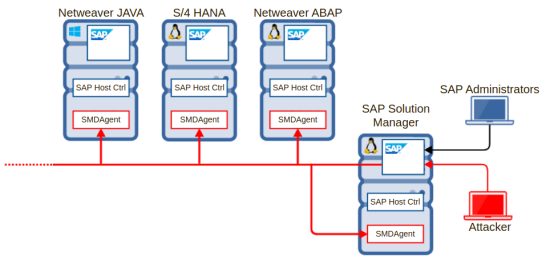

Figure 2: *Unauthenticated attacker compromises every SMDAgent connected to the Solution Manager.*

Chaining both vulnerabilities, as shown in 2, would allow an unauthenticated attacker to gain full control over all SMD agents connected to the Solution Manager.

6.6. Staying protected

The following section will illustrate some recommendations and actions that should be followed in order to detect attacks and protect systems from unauthorized attackers.

Applying patches

Due to the fact that both attacks presented in section 6 are based on vulnerabilities, the best way to be protected against them is by patching. In March 2020, SAP released a patch that will protect the Solution Manager against both flaws (authentication and injection):

Note	Title	CVSS
2890213 [3]	Missing Authentication Check in SAP Solution Manager (User-Experience Monitoring)	10

Table 1: *SAP Security notes related to SolMan vulnerabilties*

SAP security note 2890213[3], provides new versions of the affected component which will guarantee protection against the discussed flaws:

Component	Sup package	Patch Level
	SP004	000012
	SP005	000013
SOLMAN	SP006	000014
DIAG 720	SP007	000020
	SP008	000016
	SP009	000008
	SP0010	000002

Table 2: *Solution Manager patched versions*

It is strongly recommended to install the appropriate patched version as soon as possible. Nevertheless, if for some reason it is not possible to install it, SAP provided a step by step guide to manually add authentication to the application. To find this guide please refer to the SAP security note 2890213[3]. This "partial" fix should be treated as a temporary solution until it is possible to install the full patch. The reason to believe this is because it will only force authentication but will not provide any protection against the injection flaw. In other words, any authenticated attacker will still be able to launch the attack.

Networking protection measures

Installing the patches should be the first step towards being protected against the presented attack. However, there are other types of measures at different levels that can help to mitigate and reduce the attack surface.

The SAP Solution Manager it is a technical component that should only be accessible by SAP Administrators. There is no need for final users to have access to it. Therefore, limit who can reach the SolMan from a networking point of view will cause:

- An extremely reduced attack surface, as only Administrators will have access.
- A protection to potential future vulnerabilities. For instance, If in the near future a new flaw in another application is found, attackers will not be able to exploit as they will not have access to it.

The Solution Manager, despite not having business data, it is a critical system due to its highly connected architecture with other SAP systems. Therefore, it is strongly recommended to keep it as secure as possible, not only through installing patches but also through any other method that could help to keep it protected.

Detection of incidents

In case there is a need to investigate if something already happened or to monitor actions

in order to detect an attempt of attack, SAP provides a way to help with it. It is possible to activate a particular log which will start logging all actions performed by the EEM application. To manage the server's log configuration, SAP provides a particular application called log-config[4], part of the SAP Netweaver Administrator (NWA).

In order to activate the log for the EEM application, the tracing location **com.sap.smd.eem.admin.EemAdminService** should be searched. Once located, the severity level should be selected (info level is recommended). The target file where the logs will be written, is defined under the **System Configuration view**.

After executing the aforementioned steps and saving changes, a new entry will be written to the target defined file (**defaultTrace"** by default) each time an action is performed by the EEM.

7. Host Agent Analysis

7.1. Motivation

There were several reasons why this component was interesting from a security point of view. Some of them were:

1. There were two services associated with this component, running with very high privileges: Root for unix-like systems / NT AUTHORITY/System for Windows systems:

```
$> ps -ef | grep hostctrl
root 92067 hostctrl/exe/saphostexec[...]
sapadm 92072 hostctrl/exe/sapstartsrv[...]
root  92338  hostctrl/exe/saposcol[...]
```

2. It seemed possible to communicate with this agent through port 1128 (exposed to all interfaces):

```
$>  ss -larntp | grep 92072
LISTEN 0 20 *:1128 *:*
  users:(("sapstartsrv",pid=92072,fd=18))
```

3. The user "daaadm" was mentioned in the SAP Host Agent configuration file, as value of a very promising parameter **"service/admin_users"**.

```
$> grep daaadm hostctrl/exe/host_profile
service/admin_users = daaadm
```

At this point, a vulnerability that may allow an unauthenticated attacker to get daaadm privileges on every SAP host was already found. The idea behind analyzing this component was to try to find a way to escalate privileges from daaadm user to root/system user.

7.2. Powerful agent.. with restricted access !

During a legit use of SAP Host Agent, local administrators or root users would communicate with the agent using the binary **saphostctrl**. This binary, part of Host Agent binaries, works as a wrapper allowing to execute all Host Agent functions. Below are some names of those functions:

StartInstance	StopInstance
StartDatabase	StopDatabase
ExecuteOperation	ACOSPrepare
ExecuteInstallationProcedure	

Table 3: *Some functions exposed by SAP Host Agent*

Inside the Host Agent OS directory, there was a file called **host_profile**. This profile stores several configurations (in a key-value fashion) of the Agent. Among all these configurations, there was a parameter called **service/admin_users**, whose objective is to whitelist all additional local OS users authorized to communicate with the Host Agent.

As explained in 7.1 **daaadm** user was part of this list, which meant that this user would be able to communicate with the sapstartsrv process. However, after trying to execute some of its functions, it was discovered that being logged as a whitelisted user was not enough.

Even for this list of users, like **daaadm**, they must provide their password when calling the **saphostctrl** binary. Therefore even in the scenario where an attacker uses the SolMan to execute commands as **daaadm**, it will not be possible to communicate with the Host Agent as they would not know the password.

7.3. SOAP Friendly Agent

Another configuration parameter present inside **host_profile** was **service/porttypes**. This configuration basically states all web services that are exposed by the **sapstartsrv** (port 1128). By default, three are remotely accessible: SAPHostControl, SAPOscol and SAPCCMS.

After some analysis it was determined that SAPHostControl was the equivalent of the binary **saphostctrl**. In other words, by only using HTTP SOAP requests, it was possible to call all functions provided by the Host Agent. Again, even locally, all requests required authentication.

An extensive research over the **saphostctrl** binary was performed to finally realized that every action done with this wrapper was producing HTTP requests to localhost on port 1128. This meant that the binary itself was using the web service running on port 1128. With the objective of understanding how everything was working, an analysis of the traffic was performed through sniffing the local network interface of the SAP system.

The first analyzed HTTP request already gave some interesting outcomes:

```
POST /SAPHostControl.cgi HTTP/1.1
Content-type : text/xml;charset="utf-8"
Authorization: Basic ezJENEE2RkI4LTM3RjEtNDN
kNy04OEJFLUFEMjc5Qzg5RENEN306MjcwMjI4MjQ0MzE
zNzIzNDYzNDUyMjg4MTI2NDIzMDQ3NDY3MTUwMg==
Soapaction: ""
User-Agent: JAX-WS RI 2.1.6 in JDK 6
Host: target:118
[...]
```

The authorization header was not empty. Its decoded value was:

```
{2D4A6FB8-37F1-43d7-88BE-AD279C89DCD7}:
270228244313723463452288126423047467150 2
```

After some more tests it was found that although the user remains always the same, the password (that "list" of numbers), changed at every single HTTP request. It seemed to be important to further understand what was going on, and therefore it was decided to look deeper into the Host Agent kernel to learn from where these passwords came from.

7.4. Internal Trusted Connection

Using the hardcoded username as an entry point for the analysis, it was discovered that a special feature exists inside the Host Agent. From now on, this feature would be referred as "Internal Trusted Connection".

After carefully analyzing this feature, it was possible to determine how it worked:

1. Only whitelisted users (parameter **service/users_admin**) are able to use it.

2. These users are able to request, only locally, a logon file using the method RequestLogonFile exposed by the SAPHostControl web service.

3. The Host Agent generates a temporary password into a temporary file (only readable by the caller user) located in /usr/sap/hostctrl/work/sapcontrol_logon and provides the path and name to the as response of the request executed in the previous step.

4. The initial requester (user), reads the content of the file to get the password.

5. The user can perform one request, and only one, using the hardcoded username and this temporary password.

Below there is an example of how this feature can be used:
1. The whitelisted used sends a request to the SAPHostControl web service with the appropriate parameters:

```
saphost:daaadm 54> curl -skL -X POST
http://localhost:1128/SAPHostControl.cgi
-H 'Content-Type: text/xml;charset=utf-8'
--data '<?xml version="1.0"
[...]
<ns2:RequestLogonFile>
<user>daaadm</user>
</ns2:RequestLogonFile>
[...]'
| xmllint --format -
```

2. The user gathers the filename from the response of the request made in step 1.

```
<?xml version="1.0" encoding="UTF-8"?>
[...]
<SAPHostControl:RequestLogonFileResponse>
<filename>/usr/sap/hostctrl/work/sapcontrol_
logon/logon1</filename>
</SAPHostControl:RequestLogonFileResponse>
[...]
```

3. Finally, only the user is able to read that file, which has a temporary password just available to be used by that particular user:

```
saphost:daaadm 55> ls -lrht sapcontrol_logon
-rw------- 1 daaadm sapsys 40 Jun 30 logon1

saphost:daaadm 56> cat logon1
40614533500483289911294915603138102361 08
```

The conclusion of this finding is that the daaadm user can use this feature to call every function exposed by the Host Agent. Therefore, knowing the password of daaadm was not required anymore. Chaining this finding with the vulnerability explained in section 6 means that any unauthenticated attacker with access to the SolMan could finally execute every method of every Host Agent running in the same host as an SMD Agent.

7.5. Analyzing HostControl functions

Until this point it was possible to use the exposed functions using the user daaadm. However, it was important to further investigate the actual actions that those functions were able to perform. Some of the functions exposed seemed to be dangerous (StopInstance, StopDatabase, etc). Nevertheless, the main objective was to try to find a way of executing commands as root or system user.

After a careful analysis of each of the functions exposed, it was possible to identify several of them that were vulnerable to command injection. Although these functions required OS authentication, they were finally executing commands as root. Therefore this injection could lead to a privilege escalation.

Following sections will illustrate in a more detailed way just a few examples of vulnerable functions.

ExecuteOperation
This function, under certain circumstances and after a few prerequisite checks, tried to execute "**./saphostexec -upgrade**" command as root. However, the path to **saphostexec** is controlled by the attacker and it is not sanitized. Therefore, an attacker with the necessary privileges to use this function could trick this path, allowing the execution of any arbitrary script or binary as long as the name remains the same.

Log output of a successful attack:

```
[..]CommandManager::StartOSCommand: start
./saphostexec
[..]No user configured. Current user will
be used.
[..]Working directory will be change to
'/usr/sap/../../tmp/attacker'
```

ExecuteInstallationProcedure
This function executes several OS commands as root before launching the SAP installer tools, called **"sapinst"**. Again, the path to the sapinst binary could be controlled by an attacker, allowing the execution of any arbitrary script or binary as long as the name remains **"sapinst"**. Log output of a successful attack:

```
[..] PID 9162: root: Executing command
"mkdir -p -m 0770 /tmp/attacker/sapinst3"
[..] PID 9163: root: Executing command
"chown sapadm:sapinst /tmp/attacker/sapinst3"
[..] PID 9164: root: Executing command
"mv /usr/sap/hostctrl/work/eip_3HeFAw
/tmp/attacker/sapinst3/inifile.xml"
[..] PID 9165: root: Executing command
"chgrp sapinst /tmp/attacker/sapinst3/
inifile.xml"
[..] PID 9166: root: Executing command
"chmod 0660 /tmp/attacker/sapinst3/
inifile.xml"
[..] PID 9168: root: Executing command
"/tmp/attacker/sapinst [..]
```

ACOSPrepare

This function's purpose is to perform several tasks to prepare for special OS operation. Among other actions, it tries to mount a file system with administrator privileges. An attacker can control the source path of this file system and provide, for instance, a malicious one with a setuid revershell into it.

Log output of a successful attack:

```
OSP-0121: Mounting network file system
/tmp/attacker/test.fs -> /tmp/mnt
OSP-0301: Calling SAPACOSPrep platform
library function 'AcAttachNetfs'
LNX-0121: File system successfully mounted
OSP-0310: Library function returned
successfully
OSP-0200: Operation succeeded
saphostcontrol: exitcode=0
saphostcontrol: 'sapacosprep'
successfully executed
[...]
```

Afterwards checking if this file was created:

```
target:daaadm 57> ls -larht /tmp/mnt
total 20K
drwxrwxr-x  3 root root 4.0K  .
drwxrwxr-x 17 root root 4.0K  ..
-rwsrwxrwx  1 root root 8.8K revershell

target:daaadm 58> /tmp/mnt/revershell
```

All vulnerabilities and their exploitation mechanisms, work against Unix-like operating systems as well as against Windows.

7.6. Conclusion

The attacks presented in the previous section required an authenticated user that must also be part of the **service/users_admin** whitelist. Abusing the injections vulnerabilities found, this user would be able to escalate privileges and end up running commands with root privileges.

As was shown in section 7.1, **daaadm** was part of this whitelist. Furthermore, recalling section 6.5, any unauthenticated attacker was able to execute commands as **daaadm**.

Chaining both findings it is possible to conclude that any unauthenticated attacker with network access to SolMan's web server, will finally be able to execute commands as root in every server where a managed/satellite system is running.

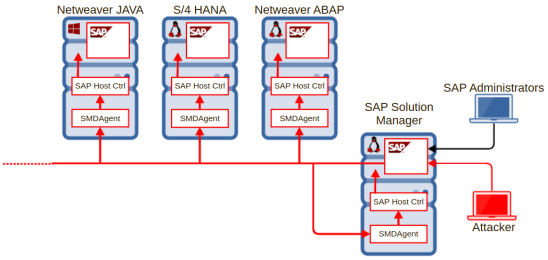

Figure 3: *Unauthenticated attacker compromises every server connected to SolMan as they are able to execute commands with root/system privileges.*

7.7. Staying protected

Applying patches

During April 2020, SAP released two patches involving the SAP Host Agent. These patches provided protection against the injections found in several functions of the Host Control web service.

Both patches provided safe versions of the affected components. In particular, for the SAP Host Agent, the provided version is:

Once this version is installed, the escalation of privileges detailed in the aforementioned section will not be able to be used anymore. Besides installing the mentioned patches, due

Note	Title	CVSS
2902645 [5]	Privilege Escalation in SAP Host Agent	7.2
2902456 [6]	Privilege Escalation in SAP Landscape Management	7.2

Table 4: *SAP Security notes related to Host Agent vulnerabilities*

Component	Support Package
SAP HOST AGENT 721	46

Table 5: *Patches related to Host Agent vulnerabilities*

to being a critical and powerful agent, it is recommended also to keep it up to date by following these SAP notes:

Note	Title
2219592	Upgrade Strategy of SAP Host Agent
2130510	SAP Host Agent 7.21

Table 6: *SAP Security notes related to Host Agent vulnerabilities*

Finally it is also important to advise that upgrading the SAP Host Agent is way more easy than upgrading an SAP System. It is a "little" technical component, without customizing and totally independent of the SAP System with its critical business data.

8. Impact

A successful attack will mean that the unauthenticated attacker will have total control over every SAP system in the landscape.

From a technical perspective, the unauthenticated attacker will have a root/system access to every server where an SAP system is running. This means that it will not only be able to compromise SAP related data, but also go beyond that and potentially compromise any other information or system running in the same server.

From a business point of view, this means a total compromise of every business data a system could hold. **On every single SAP System** connected to SolMan, the attacker could perform classical post exploitations techniques and compromise every business record. To better illustrate the impact, these are a few examples of actions that the attacker could perform:

- **Espionage:** Obtain customers, vendors or human resources data, financial planning information, balances, profits, sales information, manufacturing recipes, etc.
- **Fraud:** Modify compliance processes, modify financial information, tamper sales or purchase orders, create new vendors, modify vendor bank account numbers, etc.
- **Sabotage:** Paralyze the operation of the organization by shutting down the SAP system or the complete server, disrupting interfaces with other systems and deleting critical information, etc.

9. Conclusions

Specifically speaking about the presented attack, it was demonstrated how an unauthenticated attacker having access to SolMan's web server, was able to fully compromise every server of a system connected to the SAP landscape by being able to execute commands with system/root privileges on it .

The Solution Manager is a critical part of every SAP landscape and must be treated as it. Complementary security measures in order to protect it, such as network segregation, should be in place since its deployment. Furthermore, processes to quick and successfully apply patches for critical assets like SolMan, should be configured and maintained.

Generally speaking, hyperconnected systems play a central role in terms of security as they could act as entry points for more complex attacks. Once a system of this type is compromised, attackers could leverage their intercon-

nections in order to spread themselves through the network and extend their level of compromise.

As a final conclusion, ERP security has been improving towards a more secure state during recent years. However, as any other software, it has and will continue having flaws that may end up having critical impact. It is important to continue performing security analysis against them in order to detect and prevent them from being exploited in the wild.

References

[1] https://wiki.scn.sap.com/wiki/display/EEM/UXMonHowTo

[2] Web security: A Whitehat Perspective (199)

[3] https://launchpad.support.sap.com/#/notes/2890213

[4] https://help.sap.com/viewer/8c44f49685f44be4aa420bbf6393aeea/7.5.6/en-US/47af551efa711503e10000000a42189c.html

[5] https://launchpad.support.sap.com/#/notes/2902645

[6] https://launchpad.support.sap.com/#/notes/2902456