Multiple vulnerabilities in Cisco vManage

Security advisory
2021/06/10

Julien Legras
Théo Louis-Tisserand
Vulnerabilities description

The Viptela vManage dashboard

SD-WAN is a software-defined approach to managing the wide-area network, or WAN.

The Cisco SD-WAN fabric is based on the Viptela solution, which has four main components. Each of these components has a very specific role:

- vManage – Management Dashboard.
- vEdge – The edge router at branches.
- vBond – The Orchestrator.
- vSmart – The Controller.

vManage is a GUI based Network Management System that handles the Management Plane. vManage is a single pane of glass that gives various key stats. Operations team uses vManage for doing day to day operational activities e.g. code upgrades.

The issues

Synacktiv identified multiple vulnerabilities:

- Cypher query injections on a few API endpoints – CVE-2021-1481;
- Incorrect access control on API endpoints – CVE-2021-1482;
- Write permissions willingly hidden for the basic user group – No CVE ID associated;
- XXE injection – CVE-2021-1483;
- Command's option injection resulting in a denial of service – CVE-2021-1484.

Affected versions

At the time of the audit, all versions prior to 20.5.1 are vulnerable.

Timeline

<table>
<thead>
<tr>
<th>Date</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020/12/23</td>
<td>Vulnerabilities details sent to <a href="mailto:psirt@cisco.com">psirt@cisco.com</a></td>
</tr>
<tr>
<td>2021/01/04</td>
<td>Reply from Cisco</td>
</tr>
<tr>
<td>2021/01/20</td>
<td>Agreed on 90 days before disclosure</td>
</tr>
<tr>
<td>2021/03/26</td>
<td>Cisco sent CVE IDs for each issue:</td>
</tr>
<tr>
<td></td>
<td>• CSCww93082 Cisco SD-WAN vManage Write permissions willingly hidden for the basic user group. – This is a security hardening issue and will not be receiving a CVE ID. This will be disclosed as a public release note enclosure only. This will not be disclosed via an advisory.</td>
</tr>
<tr>
<td></td>
<td>• CSCww93076 Cisco SD-WAN vManage Incorrect access control on API endpoints</td>
</tr>
<tr>
<td>2021/04/21</td>
<td>Security advisories released and new version 20.5.1 published by Cisco.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

- CVE-2021-1482
  - CSCw93084  Cisco SD-WAN vManage XXE injection – CVE-2021-1483
  - CSCw93086  Cisco SD-WAN vManage Commands option injection resulting in a denial of service – CVE-2021-1484
  - CSCw93066  Cisco SD-WAN vManage Cypher query injections on a few API endpoints – CVE-2021-1481
Technical descriptions and proofs-of-concept

**Cypher query injection inside the vManage application**

The vManage dashboard web application injects data into a Cypher query in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. This allows an attacker to send crafted data to the application and modify the original query's behavior leading to sensitive data disclosure such as device configuration and local files.

Authenticated, a vulnerable endpoint has been found while browsing the source code, and can also be found in a black-box approach thanks to error messages that can be triggered by accessing the following URL:  
https://vmanage-XXXXXX.viptela.net/dataservice/device/action/reboot/devices/router?groupId=?groupId=test'

```plaintext
HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache, no-store, must-revalidate
X-Frame-Options: DENY
Date: Mon, 02 Sep 2019 07:27:11 GMT
Connection: close
Vary: Accept-Encoding
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Type: application/json
Content-Length: 1927

{"error":{"message":"Server error","details":"Invalid input '': expected whitespace, ',', ' ', node labels, '[' , '"','\', '"', IN, STARTS, ENDS, CONTAINS, IS, ' ', '^', '+', '-', '=' , '~', '<>', '!', '<', '&', ') (line 1, column 114 (offset: 113))
"MATCH (n:vmanagedbDEVICENODE) WHERE (n.`device-model` <> 'vedge-ccm' and n.`device-type` = 'router' and 'test\\'' IN n.`groupId`) RETURN n._rid as _rid, id(n) as _id, n.`host-name-icon` as `host-name-icon` ,n.`host-name` as `host-name` ,n.`system-ip` as `system-ip` ,n.`personality` as `personality` ,n.`site-id` as `site-id` ,n.`uuid` as `uuid` ,n.`device-type` as `device-type` ,n.`version` as `version` ,n.`uptime-date` as `uptime-date` ,n.`device-model` as `device-model` ,n.`platform` as `platform` ,n.`reachability` as `reachability` ,n.`device-os` as `device-os` ,n.`local-system-ip` as `local-system-ip` ,n.`availableServices` as `availableServices` ,n.`layoutLevel` AS `layoutLevel` order by `layoutLevel` ASC,`host-name` ASC

This behavior can be explained by reviewing the source code of the endpoint (com/viptela/vmanage/server/deviceaction/DeviceActionRestfulResource.java), which uses the `generateDeviceRebootList` method, with `groupId` as a parameter:

```java
@GET
@Path("reboot/devices/{deviceType}")
@Produces({"application/json"})
@RolesAllowed({"Device Reboot-write", "Device Reboot-read"})
@ApiOperation(value = "Retrieve list of rebooted devices", notes = "Retrieve list of rebooted devices")
@ApiResponses({@ApiResponse(code = 200, message = "Success"), @ApiResponse(code = 400, message = "Bad Request", response = ErrorResponse.class), @ApiResponse(code = 403, message = "Forbidden"), @ApiResponse(code = 500, message = "Internal Server Error", response = ErrorResponse.class) })
public Response generateRebootDeviceList(@PathParam("deviceType") final String deviceType, @ApiParam(value = "Group ID", required = true) @QueryParam("groupId") final String groupId) throws Exception {
    Collection<DeviceType> allowedPersonality = null;
    if (ServerConfiguration.getInstance().isMultiTenant()) {
```
public JsonArray generateDeviceRebootList(String groupId, String deviceType, final Collection<DeviceType> allowedPersonality) throws DeviceActionException {
    final JsonArrayBuilder builder = Json.createArrayBuilder();
    try (final VGraphDataStore dataStore = this.getDatabaseManager().getGraphDataStore()) {
        if (null == deviceType || deviceType.isEmpty()) {
            deviceType = "vedge";
        }
        final DBQueryBuilder dbQueryBuilder = dataStore.createQueryBuilder();
        dbQueryBuilder.vertexLabel(new SimpleVertexLabel("Device")).properties(DeviceActionDAO.REBOOT_PROPS_LIST);
        dbQueryBuilder.has("device-model", Operator.NOT_EQUAL, DeviceModelName.CCM.getName());
        if (deviceType.equals("controller")) {
            dbQueryBuilder.has("device-type", DeviceActionDAO.CONTROLLERS_LIST);
        } else {
            dbQueryBuilder.has("device-type", deviceType);
        }
        if (allowedPersonality != null) {
            dbQueryBuilder.has("device-type", allowedPersonality);
        }
        if (groupId != null && !groupId.equals("all")) {
            groupId = groupId.replace("'", "\\'");
            dbQueryBuilder.has(groupId, Operator.IN, "groupId");
        }
    }
    return new JsonArray(builder.build());
}

The same issue affects the endpoint /dataservice/device/action/install/devices/router which uses vertexStreamDeviceList:

GET
@Path("install/devices/{deviceType}")
@Produces({ "application/json" })
@RolesAllowed({ "Software Upgrade-write", "Software Upgrade-read", "Settings-read" })
@ApiOperation(value = "Retrieve list of installed devices", notes = "Retrieve list of installed devices")
@ApiResponses({ @ApiResponse(code = 200, message = "Success"), @ApiResponse(code = 400, message = "Bad Request", response = ErrorResponse.class), @ApiResponse(code = 403, message = "Forbidden"), @ApiResponse(code = 500, message = "Internal Server Error", response = ErrorResponse.class) })
public Response generateDeviceList(@PathParam("deviceType") final String deviceType, @ApiParam(value = "Group ID", required = true) @QueryParam("groupId") final String groupId) throws Exception {
    Collection<DeviceType> allowedPersonality = null;
    if (ServerConfiguration.getInstance().isMultiTenant()) {
        allowedPersonality = DeviceDAO.findAllowedPersonality(this.userSessionMode());
    }
    if (!deviceType.matches("[a-zA-Z0-9.? -]*")) {
        DeviceActionRestfulResource.LOGGER.error("Invalid deviceType {}", deviceType);
        return Response.status(Response.Status.BAD_REQUEST).entity("Invalid deviceType").build();
    }
    return new Response().status(Response.Status.OK).entity(deviceType).build();
}
The `vertexStreamDeviceList` method is also declared in `com/viptela/vmanage/server/deviceaction/DeviceActionDAO.java` and performs a replace operation on single quotes:

```java
public StreamingOutput vertexStreamDeviceList(final String groupName, final String type, final Collection<DeviceType> allowedPersonality) throws DeviceActionException {
    ...
    if (groupName != null && !groupName.equals("all")) {
        final String groupId = groupName.replace("'", "\\'");
        dbQueryBuilder.has(groupId, Operator.IN, "groupId");
    }
}
```

An attacker could retrieve sensitive data, such as the configuration of devices and passwords hashes. Also, it is possible to use the `LOAD CSV` function to read local files or perform HTTP requests. The impact is the same as described in our previous advisory: [https://www.synacktiv.com/ressources/advisories/Cisco_SD-WAN_vManage_neo4j_injection_and_stored_xss.pdf](https://www.synacktiv.com/ressources/advisories/Cisco_SD-WAN_vManage_neo4j_injection_and_stored_xss.pdf)

It should be noted that Cisco implemented an API validation filter (`com/viptela/vmanage/server/APIValidationFilter.java`) to protect against such injections but a list of endpoints are whitelisted and thus, not protected:

```java
public void init(final FilterConfig filterConfig) throws ServletException {
    APIValidationFilter.LOGGER.debug("APIValidationFilter: Filter init");
    this.filteredURIs = new ArrayList<String>();
    this.filteredURIs.add("/dataservice/template/feature/");
    this.filteredURIs.add("/dataservice/template/device/");
    this.filteredURIs.addAll(APIValidationFilter.DEVICE_ACTION);
    this.filteredURIs.add("/dataservice/system/device/fileupload");
    this.filteredURIs.add("/dataservice/client/enable/property");
}
```

```java
public Boolean isInvalidQueryString(final HttpServletRequest httpRequest) {
    final String queryString = httpRequest.getQueryString();
    final String requestURI = httpRequest.getRequestURI();
    if (!StringUtils.isBlank((CharSequence)queryString) && this.isURIValid(requestURI)) {
        return this.checkIfIllegalCharacterPresent(queryString);
    }
    return false;
}
```

```java
protected boolean isURIValid(final String path) {
    for (final String uri : this.filteredURIs) {
        if (path.contains(uri)) {
            return false;
        }
    }
    return true;
}
```
As one can see, if the URI is contained in one of the filtered URIs, the checkIfIllegalCharacterPresent method is never called.

Also, the detection itself is not very efficient because it looks for specific bad characters:

- **URL query:**
  - "//"
  - "load csv"
  - pattern ":[<>%;#`"]"

- **POST/PUT/DELETE body:**
  - "load csv"
  - "vmanagedb"
  - "globaldb"

The "load csv" check can be bypassed by adding more spaces and the "//" is allowed in the body payloads so in case of another Cypher injection existence in body parameters, it could be exploited to read arbitrary files.

### Incorrect access control on API endpoints

Cisco SD-WAN vManage component exposes multiple APIs through the /dataservice/ endpoint. These APIs are protected using roles configured on the user's user group.

However, it was found that several endpoints allow reader roles to actually edit data or perform sensitive actions:

- **PUT /dataservice/template/config/attach/{deviceUUID}** allows uploading a configuration to a device by any users with roles Device Inventory-read, Device Inventory-write, Template Deploy-read, Template Deploy-write.

- **PUT /dataservice/template/config/maupdate** allows updating a new device by any users with roles Device Inventory-read, Device Inventory-write, Template Deploy-read, Template Deploy-write.

- **POST /dataservice/template/config/device/mode/cli** allows updating a device to CLI mode by any users with roles Device Inventory-read, Device Inventory-write, Template Deploy-read, Template Deploy-write.

- **POST /dataservice/tenantbackup/import** allows uploading and restoring a backup file by any users with roles Tenant Management-read, Tenant Management-write, Tenant Status-read, Tenant Status-write.

- **DELETE /dataservice/tenantbackup/delete** allows deleting a backup by any users with roles Tenant Management-read, Tenant Management-write, Tenant Status-read, Tenant Status-write.

- **PUT /dataservice/statistics/settings/status** allows updating the statistics settings by any users with roles Settings-read, Settings-write, dca.

- **PUT /dataservice/statistics/settings/disable/devicelist/{indexName}** allows updating the list of disabled devices for a statistics by any users with roles Settings-read, Settings-write.
• PUT /dataservice/partner/{partnerType}/{nmsId} allows updating NMS partner details by any users with roles Integration Management-read, Integration Management-write.

• POST /dataservice/partner/{partnerType} allows registering a new NMS partner by any users with roles Integration Management-read, Integration Management-write.

• DELETE /dataservice/partner/{partnerType}/{nmsId} allows deleting a NMS partner by any users with roles Integration Management-read, Integration Management-write.

• POST /dataservice/partner/{partnerType}/map/{nmsId} allows mapping devices to a NMS partner by any users with roles Integration Management-read, Integration Management-write.

• POST /dataservice/partner/{partnerType}/unmap/{nmsId} allows unmapping devices to a NMS partner by any users with roles Integration Management-read, Integration Management-write.

• DELETE /dataservice/partner/{partnerType}/map/{nmsId} allows deleting a device mapping to a NMS partner by any users with roles Integration Management-read, Integration Management-write.

• DELETE /dataservice/template/feature/{templateId} allows deleting a template by any users with roles Template Configuration-read, Template Configuration-write.

• PUT /dataservice/template/config/attach/{deviceUUID} allows attaching a configuration template to a device by any users with roles Device Inventory-read, Device Inventory-write, Template Deploy-read, Template Deploy-write.

• PUT /dataservice/template/config/rmaupdate allows updating a device by any users with roles Device Inventory-read, Device Inventory-write, Template Deploy-read, Template Deploy-write.

• POST /dataservice/template/config/device/mode/cli allows updating a device using a CLI template by any users with roles Device Inventory-read, Device Inventory-write, Template Deploy-read, Template Deploy-write.

The endpoints allowing modification and actions must be reserved for users with the write permission.

Write permissions willingly hidden for the basic user group

Cisco SD-WAN provides a few default user groups with different permissions, as stated in Cisco's documentation:

• basic - Includes users who have permission to view interface and system information.

• netadmin - Includes the admin user, by default, who can perform all operations on the vManage NMS. Other users can be added to this group.

• operator - Includes users who have permission only to view information.

This statement can be witnessed in the web interface:
However, the configuration audit revealed that the basic user group is configured with write permissions on system and interface:

```
vmanage# show aaa usergroup basic
GROUP   USERS   TASK       PERMISSION
-------------------------------------
basic   test    system     read write
interface read write
```

This can be explained if we take a look at the `getUserGroupDetails` method (com/viptela/vmanage/server/admin/UserGroupDAO.java):

```java
private JsonObject getUserGroupDetails(final Element ele, final String groupName) throws Exception {
```
The write permission is willingly removed from the result IF the user group is basic.

This permission allows basic users to perform various commands:

- system commands and modification of the organization-name parameter:

```bash
$ ssh test@192.168.1.250
viptela 20.3.1
Password: Welcome to Viptela CLI
Test connected from 192.168.1.1 using ssh on vmanage
vmanage# show aaa usergroup basic
```

<table>
<thead>
<tr>
<th>GROUP</th>
<th>USERS</th>
<th>TASK</th>
<th>PERMISSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>test</td>
<td>system</td>
<td>read write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface</td>
<td>read write</td>
</tr>
</tbody>
</table>

vmanage# config
vmanage(config-system)# help
Possible commands:

- aaa Set AAA parameters
- admin-tech-on-failure Collect admin-tech before reboot due to daemon failure
- archive Configure periodic archiving
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>Configure clock</td>
</tr>
<tr>
<td>control-session-pps</td>
<td>Control session policer rate, in packets per second</td>
</tr>
<tr>
<td>controller-group-id</td>
<td>Controller group ID</td>
</tr>
<tr>
<td>controller-group-name</td>
<td>Controller group name, typically datacentre name - OBSOLETE</td>
</tr>
<tr>
<td>description</td>
<td>System description</td>
</tr>
<tr>
<td>device-groups</td>
<td>List of vManage groups to which the device belongs</td>
</tr>
<tr>
<td>gps-location</td>
<td>GPS latitude and longitude of the device</td>
</tr>
<tr>
<td>host-name</td>
<td>Hostname</td>
</tr>
<tr>
<td>idle-timeout</td>
<td>Idle CLI timeout, in minutes</td>
</tr>
<tr>
<td>iptables-enable</td>
<td>Enable iptables for all WAN interfaces</td>
</tr>
<tr>
<td>location</td>
<td>Location description of the device</td>
</tr>
<tr>
<td>logging</td>
<td>Configure logging</td>
</tr>
<tr>
<td>ntp</td>
<td>Configure NTP</td>
</tr>
<tr>
<td>organization-name</td>
<td>Organization name</td>
</tr>
<tr>
<td>port-hop</td>
<td>Enable port hopping for all tlocs</td>
</tr>
<tr>
<td>port-offset</td>
<td>Port offset (unique value; use only if multiple Viptela devices are behind the same NAT)</td>
</tr>
<tr>
<td>radius</td>
<td>Set RADIUS server parameters</td>
</tr>
<tr>
<td>site-id</td>
<td>Site ID</td>
</tr>
<tr>
<td>sp-organization-name</td>
<td>Service Provider Organization name</td>
</tr>
<tr>
<td>system-ip</td>
<td>System IP address</td>
</tr>
<tr>
<td>system-tunnel-mtu</td>
<td>Control tunnel MTU</td>
</tr>
<tr>
<td>tacacs</td>
<td>Set TACACS server parameters</td>
</tr>
<tr>
<td>timer</td>
<td>Set various timer timeouts</td>
</tr>
<tr>
<td>track-default-gateway</td>
<td>Enable/Disable default gateway tracking</td>
</tr>
<tr>
<td>track-transport</td>
<td>Enable transport tracking</td>
</tr>
<tr>
<td>upgrade-confirm</td>
<td>Configure software upgrade confirmation timeout</td>
</tr>
<tr>
<td>vbond</td>
<td>Configure remote vBond or local IPv4 vbond address</td>
</tr>
</tbody>
</table>

```
vmanage(config-system)# organization-name nope
vmanage(config-system)# commit
Commit complete.
vmanage(config-system)# show full-configuration
```

```
<table>
<thead>
<tr>
<th>System</th>
<th>Configuration Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>host-name</td>
<td>vmanage</td>
</tr>
<tr>
<td>site-id</td>
<td>2</td>
</tr>
<tr>
<td>admin-tech-on-failure</td>
<td></td>
</tr>
<tr>
<td>sp-organization-name</td>
<td>testorg</td>
</tr>
</tbody>
</table>
```

```
• interface commands and modification of the VPN name:

```
vmanage(config-vpn-0)# help
Possible commands:
  dns          Configure DNS server
  host         Configure static DNS mapping
  interface    Interface
  name         VPN description
  nat64        NAT64 configuration commands
```

```
vmanage(config-vpn-0)# name test
vmanage(config-vpn-0)# commit
Commit complete.
vmanage(config-vpn-0)# show full-configuration
```

```
vpn 0
  name test
  interface eth0
  ip dhcp-client
  ipv6 dhcp-client
  no shutdown
```
As one can see, the commands listed above could be used to alter the running configuration of devices.

**XXE injection**

The DNAC SDA API allows pushing configurations through NETCONF (com/viptela/vmanage/server/partner/dnac/SDARestfulResource.java):

```java
@POST
@Path("config/{partnerId}")
@Consumes("application/json")
@Produces("application/json")
@RolesAllowed({ "Policy Configuration-write", "Integration Management-write" })
@ApiOperation(value = "Device SDA configuration", notes = "Device SDA configuration")
@ApiResponses({ @ApiResponse(code = 200, message = "Success"), @ApiResponse(code = 400, message = "Bad request"), @ApiResponse(code = 403, message = "Forbidden"), @ApiResponse(code = 500, message = "Internal Server Error") })
public Response createSDAConfig(@PathParam("partnerId") final String partnerId, @ApiParam(value = "SDA configurartion Json for device", required = true) final JsonObject definitionJson, @Context final HttpServletRequest httpServletRequest) throws Exception {
    SDARestfulResource.LOGGER.info("Config from DNAC {}", (Object)definitionJson);
    return Response.ok((Object)this.tenantComponent().sdaDataDAO().pushNetconfConfigs(this.tenantComponent(), partnerId, definitionJson, false, UserSessionInfo.createUserSessionInfo(httpServletRequest, this.userSessionMode())).build());
}

@POST
@Path("netconfconfig/{partnerId}")
@Consumes("application/json")
@Produces("application/json")
@RolesAllowed({ "Policy Configuration-write", "Integration Management-write" })
@ApiOperation(value = "Device SDA configuration", notes = "Device SDA configuration")
@ApiResponses({ @ApiResponse(code = 200, message = "Success"), @ApiResponse(code = 400, message = "Bad request"), @ApiResponse(code = 403, message = "Forbidden"), @ApiResponse(code = 500, message = "Internal Server Error") })
public Response createSDAConfigFromNetconf(@PathParam("partnerId") final String partnerId, @ApiParam(value = "SDA configurartion Json for device", required = true) final JsonObject definitionJson, @Context final HttpServletRequest httpServletRequest) throws Exception {
    SDARestfulResource.LOGGER.info("Config from DNAC {}", (Object)definitionJson);
    return Response.ok((Object)this.tenantComponent().sdaDataDAO().pushNetconfConfigs(this.tenantComponent(), partnerId, definitionJson, true, UserSessionInfo.createUserSessionInfo(httpServletRequest, this.userSessionMode())).build());
}
```

However, the pushNetconfConfigs method (com/viptela/vmanage/server/partner/dnac/SDADataDAO.java) is parsing the provided XML file (inside the definitionJson parameter) without disabling external entities' resolution:

```java
public JsonObject pushNetconfConfigs(final TenantComponent tc, String partnerId, final JsonObject definitionJson, final boolean isNetconfConfig, final UserSessionInfo userSession) throws PartnerException, Exception {
    if (DeviceCommInfoLog.isConfigurationLoggingEnabled()) {
        SDADataDAO.LOGGER.info("DNAC pushNetconfConfigs partnerId {}", (Object)partnerId);
    }
```
partnerId = partnerId.trim();
final List<String> listOfDevicesAttached = this.listOfPartnerDevices(partnerId);
final JsonArray devices = definitionJson.getJsonArray("data");
final Map<String, List<String>> mapOfDeviceIdToConfig = new HashMap<String, List<String>>();
final List<String> deviceList = new ArrayList<String>();
String truncatedConfig = null;
for (int i = 0; i < devices.size(); ++i) {
    final JsonObject deviceIdValues = devices.getJsonObject(i);
    final String deviceId = deviceIdValues.getString("deviceId");
    if (!listOfDevicesAttached.contains(deviceId)) {
        SDADataDAO.LOGGER.error("Device {} is not managed by the partner {}", (Object)((JsonValue)devices.get(i)).toString(), (Object)partnerId);
        throw new PartnerException(PartnerErrorCode.PARTNER_INVALID_CONFIG, "Partner doesn't map to this device " + deviceId);
    }
    deviceList.add(deviceId);
    final String config = deviceIdValues.getString("deviceConfig");
    final byte[] decodedByteArray = Base64.decodeBase64(config);
    SDADataDAO.LOGGER.info("Decoded config from DNAC", (Object)new String(decodedByteArray));
}
deviceList.add(config);
final JsonObject deviceValues = devices.getJsonObject(i);
final String deviceId = deviceValues.getString("deviceId");
if (!listOfDevicesAttached.contains(deviceId)) {
    SDADataDAO.LOGGER.error("Device {} is not managed by the partner {}", (Object)((JsonValue)devices.get(i)).toString(), (Object)partnerId);
    throw new PartnerException(PartnerErrorCode.PARTNER_INVALID_CONFIG, "Partner doesn't map to this device " + deviceId);
}
deviceList.add(deviceId);
final String config = deviceValues.getString("deviceConfig");
final byte[] decodedByteArray = Base64.decodeBase64(config);
SDADataDAO.LOGGER.info("Decoded config from DNAC", (Object)new String(decodedByteArray, "UTF-8"));
}

Synacktiv was not able to trigger this vulnerability as the function listOfPartnerDevices returns an empty list if no device is managed by partners. External entities resolution should be disabled like on other XML parsing functions:

builderFactory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);

Command's option injection resulting in a denial of service

The vManage web interface provides a way to upload WAN Edge List. By reading the code, Synacktiv discovered that there are two kinds of allowed files:

- .viptela files: “wan-edge-upload-signed-file”
- .csv files: “wan-edge-upload-csv-user-created-file”

The type identifier is implemented in the findUploadType method (com/viptela/vmanage/server/deviceconfig/system/device/vedgelist/VedgeListFileUploadHandler.java):

```java
private String findUploadType(final String uploadedFilename) {
    String uploadType = "wan-edge-upload-invalid-file";
    final int lastIndexOfDot = uploadedFilename.lastIndexOf(46);
    String fileExt = (lastIndexOfDot > 0) ? uploadedFilename.substring(lastIndexOfDot + 1, uploadedFilename.length()) : ".";
    fileExt = fileExt.trim().toLowerCase();
    if ("viptela".equals(fileExt)) {
        uploadType = "wan-edge-upload-signed-file";
    } else if ("csv".equals(fileExt)) {
        uploadType = "wan-edge-upload-csv-user-created-file";
    }
    return uploadType;
}
```
public JsonObject process() throws IOException, InterruptedException, VedgeListException, SystemDeviceException {
    JsonObject jsonObj = null;
    if (this.uploadType.equals("wan-edge-upload-signed-file")) {
        this.verifyFileSignature();
        jsonObj = this.parseFile();
    } else if (this.uploadType.equals("wan-edge-upload-csv-user-created-file")) {
        VedgeListFileProcessor.LOGGER.info("CSV File Uploaded: {}", (Object)this.file.getName());
        jsonObj = this.parseCsvFile();
    }
    return jsonObj;
}

The bash script /usr/bin/verify_zprov_file.sh is in charge of the signature verification and proceeds in the simplified following steps:

1. Execute the command /usr/bin/verify_zprov_file.sh with the file path.
2. Check the exit status of the process.
3. If the exit status is not 0, read the error reason from the error stream and log the error.
4. Throw a VedgeListException with the appropriate message.

If the file is a .viptela file, the signature will be checked in verifyFileSignature:
1. Extract files from the archive as a .tar.gz file.

2. Call the basename command on *.sig and *.cer files.

3. Use these .cer files to validate the signed files (.sig).

However, the basename calls are performed without prior sanitization of the filenames:

```bash
validate()
{
  local tmpdir=$1; shift
  first_file=`basename $1`; shift
  ...
  if [ $(ls -1 ${tmpdir}/*.cer 2>/dev/null | wc -l) != 0 ]; then
    echo 'PnP'
    pushd ${tmpdir} > /dev/null
    cert=`basename *cer`
  ...
```

As a result, if a filename in the archive starts with a - (dash), the basename command will try to use it as an option and will fail if the option does not exist:

```bash
vmanage:/tmp$ touch -- -oupsie.cer
vmanage:/tmp$ tar cvfz test.viptela -- *.cer -oupsie.cer
vmanage:/tmp$ bash /usr/bin/verify_zprov_file.sh test.viptela
basename: invalid option -- 'o'
Try 'basename --help' for more information.
file: /tmp/tmp.tFQfNiAq1Y/
PnP
basename: invalid option -- 'o'
Try 'basename --help' for more information.
```

And then, the process hangs. If this file is uploaded through the web interface, it will block the upload handler, resulting in a denial of service of this feature.

This is mostly due to the lack of timeout configuration in the process creation:

```java
final ProcessBuilder processBuilder = new ProcessBuilder(new String[] { command, this.filePath });
final Process action = processBuilder.start();
action.waitFor();
```

The waitFor function can take a timeout parameter that would limit the denial of service.