SYNACKTIV
DIGITAL SECURITY

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

Date	Action
2020/12/23	Vulnerabilities details sent to psirt@cisco.com
2021/01/04	Reply from Cisco
2021/01/20	Agreed on 90 days before disclosure
2021/03/26	Cisco sent CVE IDs for each issue: • CSCvw93082 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. • CSCvw93076 Cisco SD-WAN vManage Incorrect access control on API endpoints

	– CVE-2021-1482 • CSCvw93084 Cisco SD-WAN vManage XXE injection – CVE-2021-1483 • CSCvw93086 Cisco SD-WAN vManage Commands option injection resulting in a denial of service – CVE-2021-1484 • CSCvw93066 Cisco SD-WAN vManage Cypher query injections on a few API endpoints – CVE-2021-1481
2021/04/21	Security advisories released and new version 20.5.1 published by Cisco.

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\'*

```
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, '^', '*', '/', '%', '+', '-',
'=', '~', \"<>\", \"!=\", '<', '>', \"<=\", \">=\", AND, XOR, OR or ')' (line 1, column 114
(offset: 113))\n\"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:

```
    @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()) {
```

```
            allowedPersonality = DeviceDAO.findAllowedPersonality(this.userSessionMode());
        }
        final JsonObjectBuilder builder =
JsonUtil.buildJSONObjectBuilder("DeviceActionReboot",
this.tenantComponent().deviceActionDAO().generateDeviceRebootList(groupId, deviceType,
allowedPersonality));
        return Response.ok((Object)builder.build()).build();
    }
```

Reviewing the *generateDeviceRebootList* method (*com/viptela/vmanage/server/deviceaction/DeviceActionDAO.java*), all single quotes of the *groupId* parameter are escaped with a backslash ('\'). However, when adding another backslash, the former quote is not escaped anymore, and the *groupId* is then concatenated to the query:

```
    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");
            }
```

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 {}",
```

```
(Object)deviceType);
            return Response.status(Response.Status.BAD_REQUEST).entity((Object)("Invalid
deviceType " + deviceType)).build();
        }
        final StreamingOutput responseStream =
this.tenantComponent().deviceActionDAO().vertexStreamDeviceList(groupId, deviceType,
allowedPersonality);
        return Response.ok((Object)responseStream).build();
    }
```

The *vertexStreamDeviceList* method is also declared in *com/viptela/vmanage/server/deviceaction/DeviceActionDAO.java* and performs a replace operation on single quotes:

```
    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

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:

```
    public void init(final FilterConfig filterConfig) throws ServletException {
        APIValidationFilter.LOGGER.debug("APIValidationFilter: Filter init");
        (this.filteredURIs = new
ArrayList<String>()).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");
    }

    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;
    }

    protected boolean isURIValid(final String path) {
        for (final String uri : this.filteredURIs) {
            if (path.contains(uri)) {
                return false;
            }
        }
        return true;
    }
```

```
    static {
        LOGGER = LoggerFactory.getLogger((Class)APIValidationFilter.class);
        APIValidationFilter.INVALID_CHAR = Pattern.compile("[<>%;#+`]");
        DEVICE_ACTION = Arrays.asList("/dataservice/device/action/install",
"/dataservice/device/action/reboot", "/dataservice/device/action/changepartition",
"/dataservice/device/action/removepartition",
"/dataservice/device/action/defaultpartition");
    }
```

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/rmaupdate* 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*.

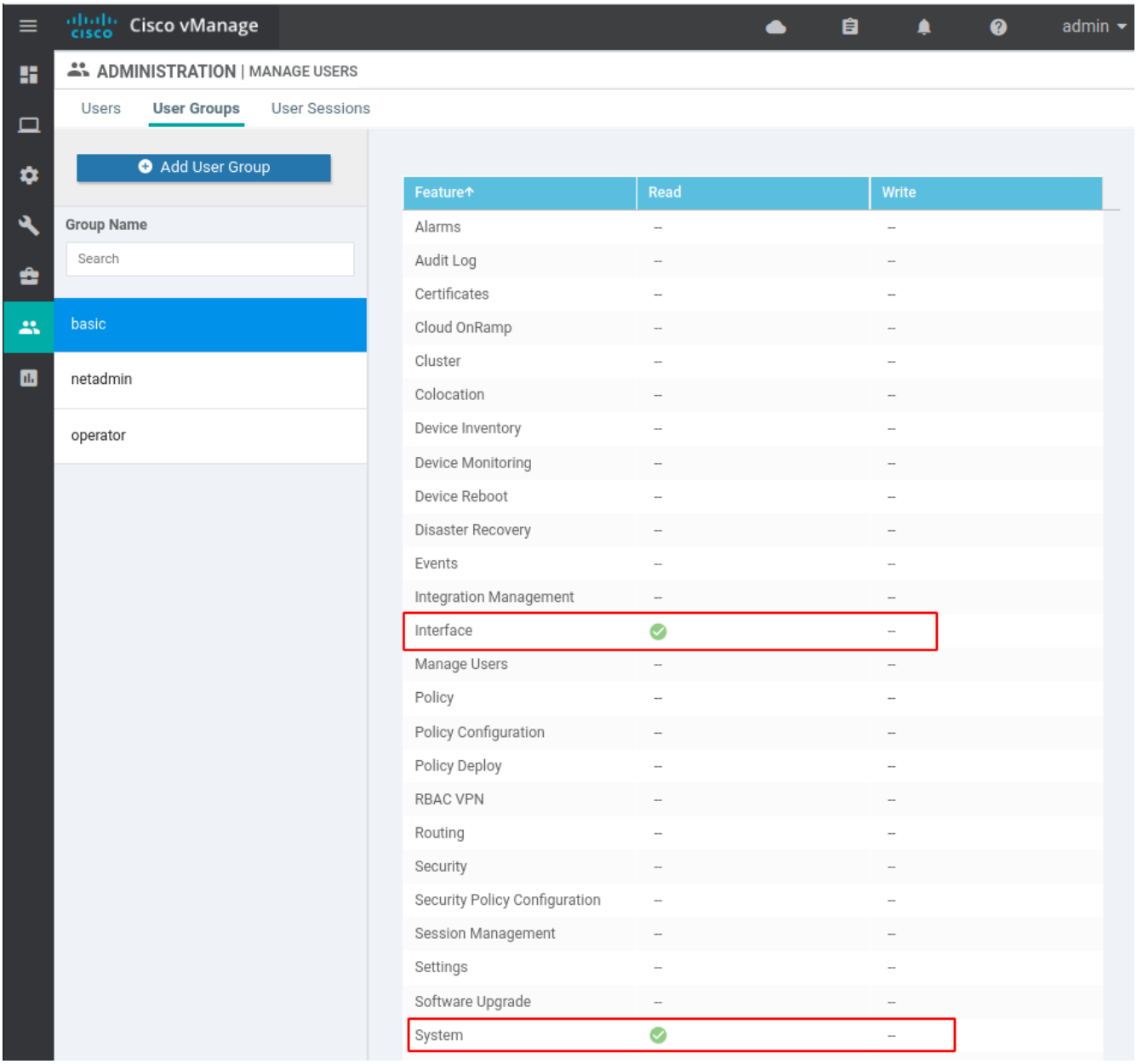
The endpoints allowing modification and actions must be reserved for users with the *write* permission.

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):*

```
    private JsonObject getUserGroupDetails(final Element ele, final String groupName)
throws Exception {
```

```
        final JsonObjectBuilder groupObject = Json.createObjectBuilder();
        groupObject.add("groupName", groupName);
        final JsonArrayBuilder taskArrayBuilder = Json.createArrayBuilder();
        final NodeSet tasks = ele.get("task");
        final Map<String, JsonObjectBuilder> taskMap = new HashMap<String,
JsonObjectBuilder>();
        for (final String task : UserGroupDAO.TASKS_CLI_LIST) {
            final JsonObjectBuilder taskobj = Json.createObjectBuilder();
            taskMap.put(task, taskobj);
            taskobj.add("feature", task);
            taskobj.add("enabled", false);
            taskobj.add("read", false);
            taskobj.add("write", false);
        }
        for (final Element task2 : tasks) {
            final String mode = this.netConfClientFactory.getValue(task2,
"mode").toString();
            final JsonObjectBuilder tobj = taskMap.get(this.getFeature(mode));
            tobj.add("enabled", true);
            final NodeSet permissions = task2.get("permission");
            for (final Element perm : permissions) {
                final String permValue = perm.getValue().toString();
                if (permValue.equals("read")) {
                    tobj.add("read", true);
                }
                if (permValue.equals("write")) {
                    if (groupName.equalsIgnoreCase("basic")) {
                        tobj.add("write", false);
                    }
                    else {
                        tobj.add("write", true);
                    }
                }
            }
        }
…
```

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:

```
$ 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

GROUP        USERS             TASK       PERMISSION
-------------------------------------------------------
basic        test              system     read write
                               interface  read write
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
```

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

vmanage(config-system)# organization-name nope
vmanage(config-system)# commit
Commit complete.
vmanage(config-system)# show full-configuration
system
 host-name             vmanage
 site-id               2
 admin-tech-on-failure
 sp-organization-name  testorg
 organization-name     nope
 vbond 192.168.1.251
```

- *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*):

```
    @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.tenantCompo
nent(), 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.tenantCompo
nent(), 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 para*meter) without disabling external entities' resolution:

```
    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));
            final DocumentBuilderFactory builderFactory =
DocumentBuilderFactory.newInstance();
            final DocumentBuilder builder = builderFactory.newDocumentBuilder();
            final Document xmlDocument = builder.parse(new InputSource(new StringReader(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*):

```
    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;
    }
```

This file is then processed by *VedgeListFileProcessor (com/viptela/vmanage/server/deviceconfig/system/device/vedgelist/VedgeListFileProcessor .java)*:

```
    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;
    }
```

If the file is a *.viptela* file, the signature will be checked in *verifyFileSignature*:

```
    private void verifyFileSignature() throws IOException, InterruptedException,
VedgeListException {
        final String command = "/usr/bin/verify_zprov_file.sh";
        if (DeviceCommInfoLog.isCertificateLoggingEnabled()) {
            VedgeListFileProcessor.LOGGER.info("Running {} for requestID {} filePath {} ",
new Object[] { command, this.requestToken, this.filePath });
        }
        final ProcessBuilder processBuilder = new ProcessBuilder(new String[] { command,
this.filePath });
        final Process action = processBuilder.start();
        action.waitFor();
        if (action.exitValue() != 0) {
            final StringBuilder errorReason = new StringBuilder();
            try (final BufferedReader reader = new BufferedReader(new
InputStreamReader(action.getErrorStream()))) {
                String errorLine = "";
                while ((errorLine = reader.readLine()) != null) {
                    errorReason.append(errorLine).append("\n");
                }
            }
            catch (Exception ex) {
                VedgeListFileProcessor.LOGGER.error("Failed to read error reason from
verify signature script for requestID " + this.requestToken, (Throwable)ex);
            }
            VedgeListFileProcessor.LOGGER.error("Error [{}] while verifying signature of
vEdge list file {} for requestID {}", new Object[] { errorReason.toString(),
this.file.getName(), this.requestToken });
            throw new
VedgeListException(VedgeListErrorCode.FILE_SIGNATURE_VERIFICATION_FAILED,
VedgeListErrorCode.FILE_SIGNATURE_VERIFICATION_FAILED.getMessage());
        }
    }
```

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

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:

```
…
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:

```
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:

```
        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.