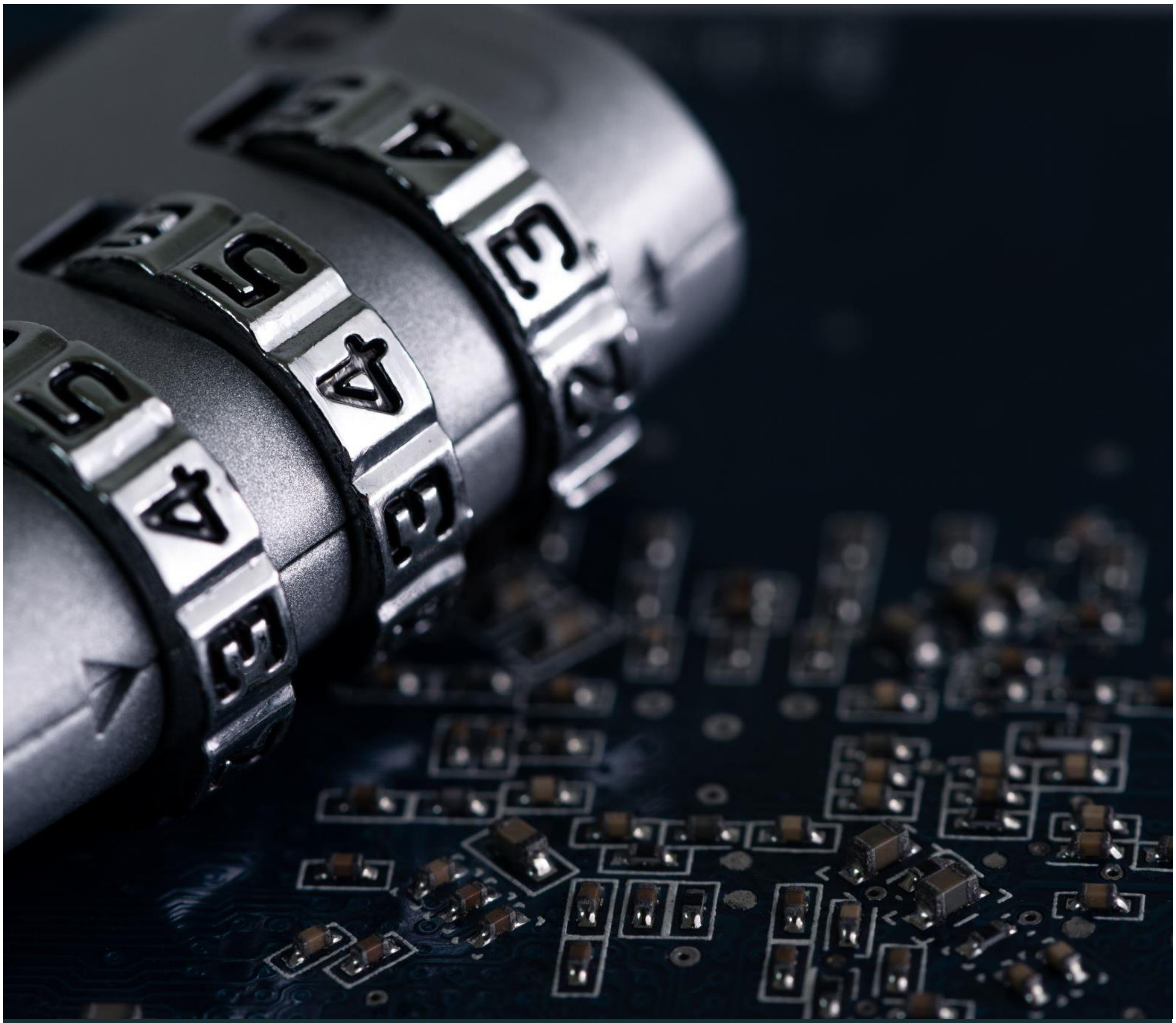


# Continuous verification of security baselines using a CI/CD pipeline.

A proof of concept to verify the different security configurations of a SMB infrastructure using a CI/CD pipeline.

Matthias Neuville

11/11/21

PACS – Application-level security

# Abstract

During the proof of concept (PoC) certain aspects of a SMB infrastructure are investigated to ensure a secure setup. CI/CD pipelines allow the use of centralized scripts which can be run on explicitly defined runners, for example installed inside the SMB network.

The PoC consist of four stages, starting with a verification of the public IPs. By querying Shodan.io, the open inbound ports and publicly available vulnerabilities are identified. This is then followed by querying portquiz.net from the internal runner. Thirdly, certain configurations in Azure and Microsoft 365 (M365) are verified. The last stage will make a call to a reference host to verify the registry values.

These four stages provide a centralized overview of the current security health of an organization. The results of all the jobs allow the SMB and/or their IT partner to act accordingly. Running the CI/CD pipeline on regular basis allows for a continuous verification and follow up of the SMB's cybersecurity health and risk state.

All files used and created during this proof of concept are also available on the following GitHub repository: https://github.com/CaptainBlackbutter/ContinuesSecurityConfigurationVerificationUsingPipelines.

## Table of Contents

## Use case

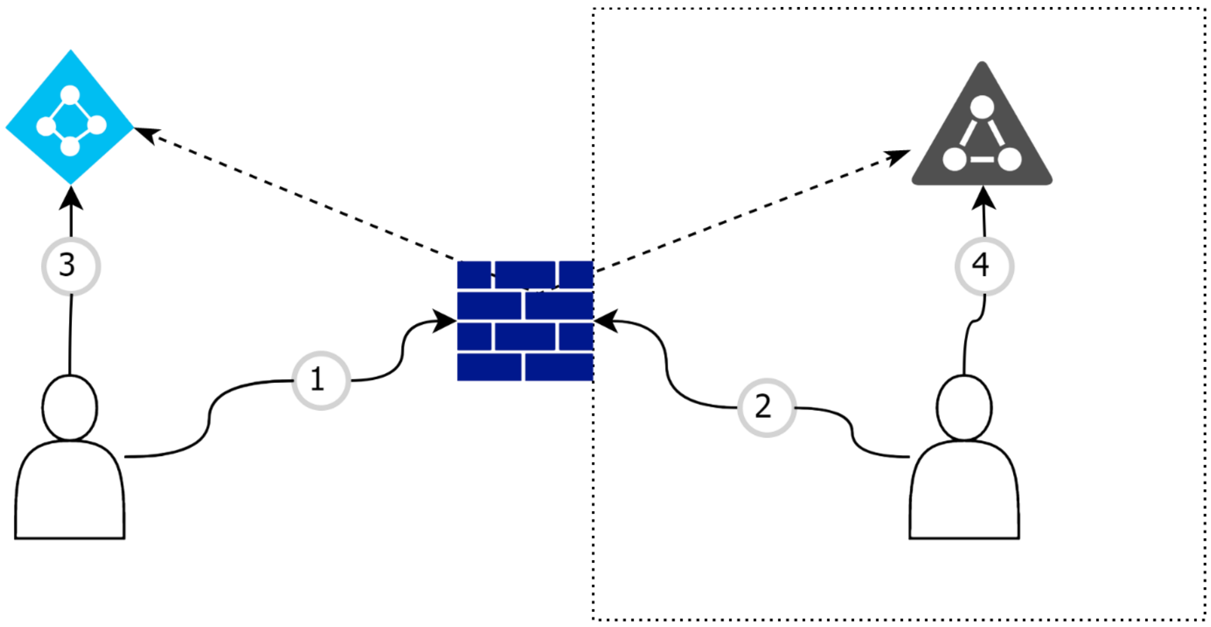
A growing MSSP is often expected to monitor and maintain the complete infrastructure of a customer. They often believe that their current situation is correctly configured and is protecting them from outside threats. During the product/solution lifecycle a lot of changes happen to the configuration either by the MSSP, the customer or other (malicious) actors. To answer these possible risks and expectations a proof of concept has been defined using the automation possibilities found in CI/CD pipelines, hereinafter shortened to pipeline. Implementation of the pipeline should use a key management solution to securely handle customer credentials.

The pipeline should monitor the following aspects of the infrastructure of a small and midsize business (SMB), as visualized in Figure 1.

1. Inbound ports should be monitored to verify changes in the public facing attack surface. Additionally public faced vulnerabilities should be identified.
2. Outbound traffic possibilities should be monitored as well as the effectiveness of the security services on a firewall.
3. Configuration of an Azure Active Directory (AAD) and other M365 solutions to identity if best-practices are configured.
4. Configuration of Active Directory (AD) to identify if best practices are configured. These configurations should be verified using a reference host.

Figure 1

*Attack surface of a SMB, limited to the current use case.*



*Note.* Right side represents internal infrastructure. Dotted line between AAD and AD represents the synchronization. By M. Neuville, 2021

## Research

During the research we will identify the technology used to answer the requirements and to build the pipeline. An attentive reader might notice that some test are not all inclusive, this is due to the limitation of the scope of this PoC.

### Inbound ports and vulnerabilities

Multiple tools are available to detect open ports and vulnerabilities on the public facing side of a firewall. For this paper any tool requiring a subscription or other form of payment has been left out of scope (for example tenable.io).

Shodan.io is a public search engine which crawls the internet on a regular basis. This search engine provides us with information around open ports, detected vulnerabilities and other values. (Shodan, n.d.)

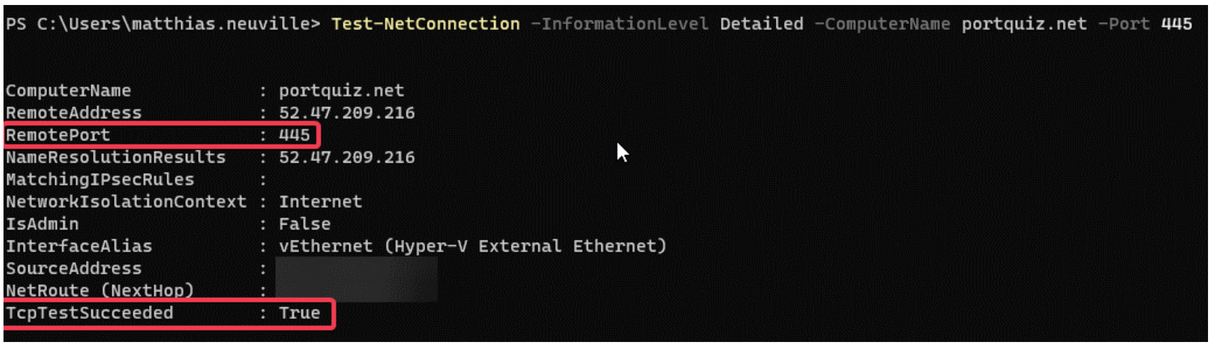
### Outbound connections

To test outbound ports a publicly available listener is required. A public destination which we can use is portquiz.net. Portquiz.net listens to all Transmission Control Protocol (TCP) ports except for TCP port 445. (Maurice, n.d.)

Verification of the server and its capabilities resulted in a positive response on TCP port 445, this is visualized in Figure 2. Therefore it can be included in the test.

Figure 2

*Successful connection to TCP port 445 on portquiz.net.*



```
PS C:\Users\matthias.neuville> Test-NetConnection -InformationLevel Detailed -ComputerName portquiz.net -Port 445


ComputerName             : portquiz.net
RemoteAddress            : 52.47.209.216
RemotePort               : 445
NameResolutionResults    : 52.47.209.216
MatchingIPsecRules       :
NetworkIsolationContext  : Internet
IsAdmin                  : False
InterfaceAlias           : vEthernet (Hyper-V External Ethernet)
SourceAddress            : [redacted]
NetRoute (NextHop)       :
TcpTestSucceeded         : True
```

*Note.* By M. Neuville, 2021

### Verification of security services

Firewall security services are expected to prevent the download of malware and other malicious actions. For the scope of this paper, the job will try to download malware using a multitude of different protocols. The samples used in this paper are publicly available on GitHub: https://github.com/ytisf/theZoo.

As a precaution, the Linux based GitLab runners will integrate with Docker to provide an isolated container. This will ensure that the downloaded malware is removed from the environment after the verification.

### Configuration of Azure and M365 solutions

As the cloud solution of Microsoft are in an ever-evolving development lifecycle, multiple solutions are required to connect and maintain these products. In the scope of this paper, we will use two different modules to connect and gather the required configuration.

### Azure Active Directory PowerShell for Graph module 2.0

Using this module, we can use different commands to perform administrative tasks and configurations for the Azure Active Directory. (de Jong et al., 2021)

### Exchange Online PowerShell

This module allows the management of Microsoft Exchange Online. It allows to get the required information from the Exchange Online configuration, the Advanced Threat Protection and other components. (Raya & Davis, 2021)

## Configuration of Active Directory

In the scope of this paper, we will verify the configuration baseline on a reference host. This host is supposed to represent the configuration baseline of the SMB. To verify the configuration of the Active Directory group policies, the pipeline will query the registry using the remote registry service and default PowerShell queries.

## Key management in CI Pipeline

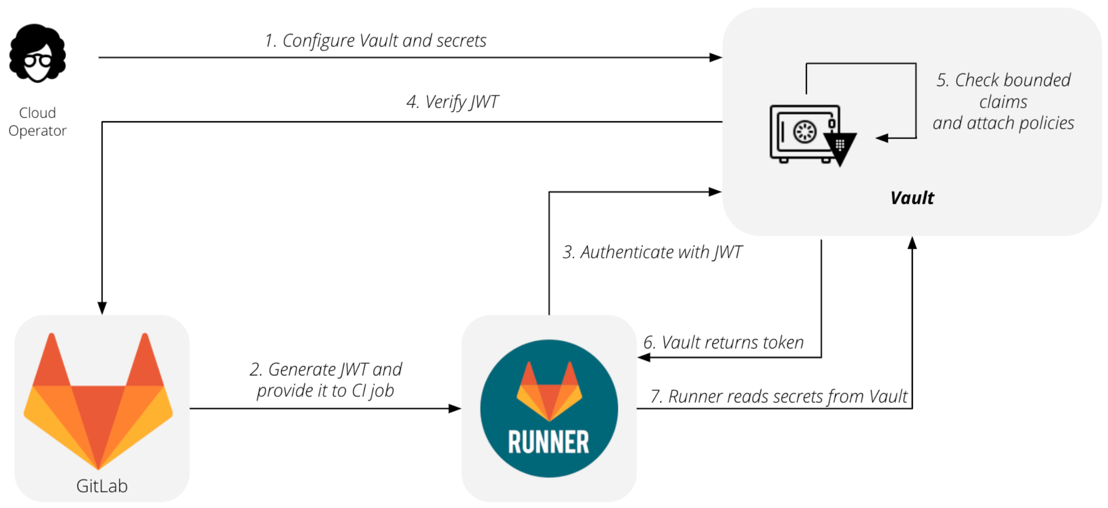
As the pipeline will handle multiple credentials, with often high-level access to different systems, it is necessary to use a trustworthy and secure key management solution. HashiCorp Vault provides a solution for this problem.

Each job in the pipeline has a JSON Web Token (JWT) which can be used to authenticate with the Vault. This combined with the necessary policies will allow us to implement a least-privileged approach for the jobs and pipeline, where possible. (GitLab, 2021)

As shown in Figure 3, it's the runner who requests the secret directly from the vault. This will allow us to prevent the secret from being passed through the MSSP infrastructure.

**Figure 3**

*Diagram to visualize the flow between GitLab and HashiCorp Vault.*


1. Configure Vault and secrets
Cloud Operator
4. Verify JWT
5. Check bounded claims and attach policies
Vault
3. Authenticate with JWT
6. Vault returns token
2. Generate JWT and provide it to CI job
7. Runner reads secrets from Vault
GitLab
RUNNER

*Note. (GitLab, 2021)*

# Implementation

## Lab setup

The lab will exist out of the following components:

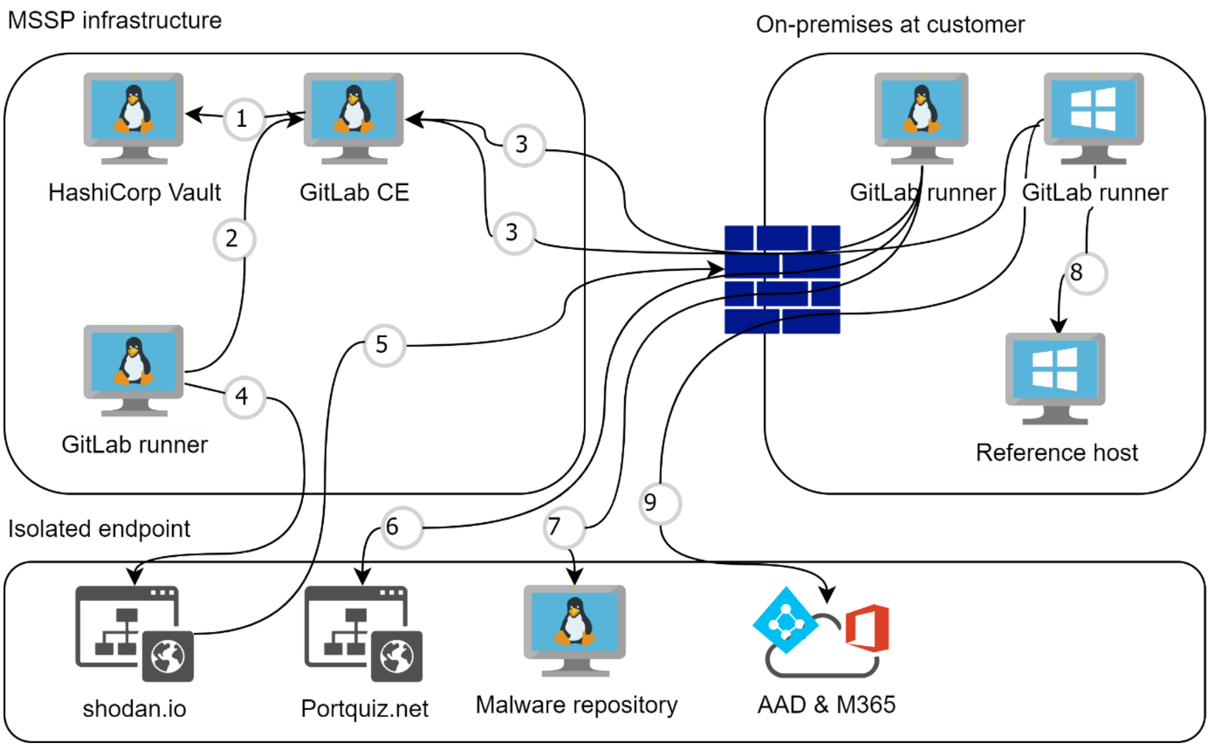
- MSSP environment
  - GitLab community Edition (see Appendix B)
  - GitLab runner with Docker integration (see Appendix C)
  - HashiCorp Vault (see Appendix A)
- Customer environment
  - GitLab runner on Linux with Docker integration (see Appendix C)
  - GitLab runner on Windows host (see Appendix C)
  - Windows reference host
- Isolated endpoints
  - Portquiz.net
  - Shodan.io
  - Malware repository
  - Azure and M365 environment

As visible in Figure 3 multiple dataflows are defined in the setup.

1. GitLab CE access data in the HashiCorp Vault using the JWT.
2. MSSP GitLab runner communicates with GitLab CE.
3. On-premises GitLab runners communicates with GitLab CE.
4. MSSP GitLab runner communicates with Shodan.io.
5. Shodan.io performs scan on public IP's.
6. On-premises GitLab runner tests outbound ports by setting up connections to portquiz.net.
7. On-premises GitLab runner tests security services by downloading from a repository.
8. On-premises GitLab runner queries reference host registry.
9. On-premises GitLab runner queries Azure and M365 environment.

Figure 4

*Data flow in lab.*



*Note.* Numbering is not corresponding with data flow order. By M. Neuville, 2021

## Project basic setup

### Directory structure

Create a new project.

Create the following directory structure.

- Lib (will contain base files for checks)
- Customers (will contain our customer specific information)
    - customer_a (information of customer_a)

The scripts used in this lab are available in the different appendixes. These should be placed in the lib directory. The scripts are also available on the following GitHub repository:

### Define pipeline (.gitlab-ci.yml)

In the root of the project create a file named ".gitlab-ci.yml" with the following content.

```
stages:
    - WAN
    - LAN
    - AAD
    - AD

include:
    - 'customers/**/*.yml'
```

The include statement allows to include all "*.yml" files, in our customer directories, defined for each customer. This allows us to create a more customer defined approach and provides more readability in the code.

In the subfolder of each customer create a file “<customer>.yml” with the following content.

```
stages:
    - WAN
    - LAN
    - AAD
    - AD
```

## Setup HashiCorp Vault

All information provided here is available from the GitLab docs. (GitLab, 2021)

Connect with SSH to the vault server. (All these can also be done using the web UI)
Set VAULT_ADDR to http (in a production environment the vault must be configured to use https).

```
Export VAULT_ADDR=’http://127.0.0.1:8200’
```

Enable the use of JSON Web Tokens (JWT).

```
Vault auth enable jwt
```

Create a secrets engine for the global secrets.

```
Vault secrets enable -version=2 -path=global kv
```

Create a secrets engine for each customer.

```
Vault secrets enable -version=2-path=customer_a kv
```

Create the Shodan.io API token.

```
Vault kv put global/shodan shodan_api_token=<token>
```

Verify if token has been created.

```
Vault kv get global/shodan
```

This returns all keys defined under the secret engine “global”, for the secret “shodan”.
Create access policy for specific job.

```
Vault policy write shodan_api_token - <<EOF
# Policy name: shodan_api_token
#
# Read-only permission on ‘global/data/*’ path
path “global/data/*” {
  capabilities = [ “read” ]
}
EOF
```

Create the role required for the JWT token. Make sure the correct “project_id” is defined correctly. This can be found on the GitLab web interface, on the project root.

```
vault write auth/jwt/role/shodan_api_token - <<EOF
{
  "role_type": "jwt",
  "policies": ["shodan_api_token"],
  "token_explicit_max_ttl": 60,
  "user_claim": "user_email",
  "bound_claims": {
    "project_id": "34",
    "ref": "main",
    "ref_type": "branch"
  }
}
EOF
```

Configure the JWT authentication method.

```
vault write auth/jwt/config \
    jwks_url="https://gitlab.example.com/-/jwks" \
    bound_issuer="gitlab.example.com"
```

## Stage: WAN

During this stage the pipeline will verify if any changes have been made to the inbound ports and if any vulnerabilities have been detected. This verification will be done using the API from Shodan.io. The token used to authenticate has been stored in the HashiCorp Vault during the previous chapter.

### Evaluating the Shodan.io result

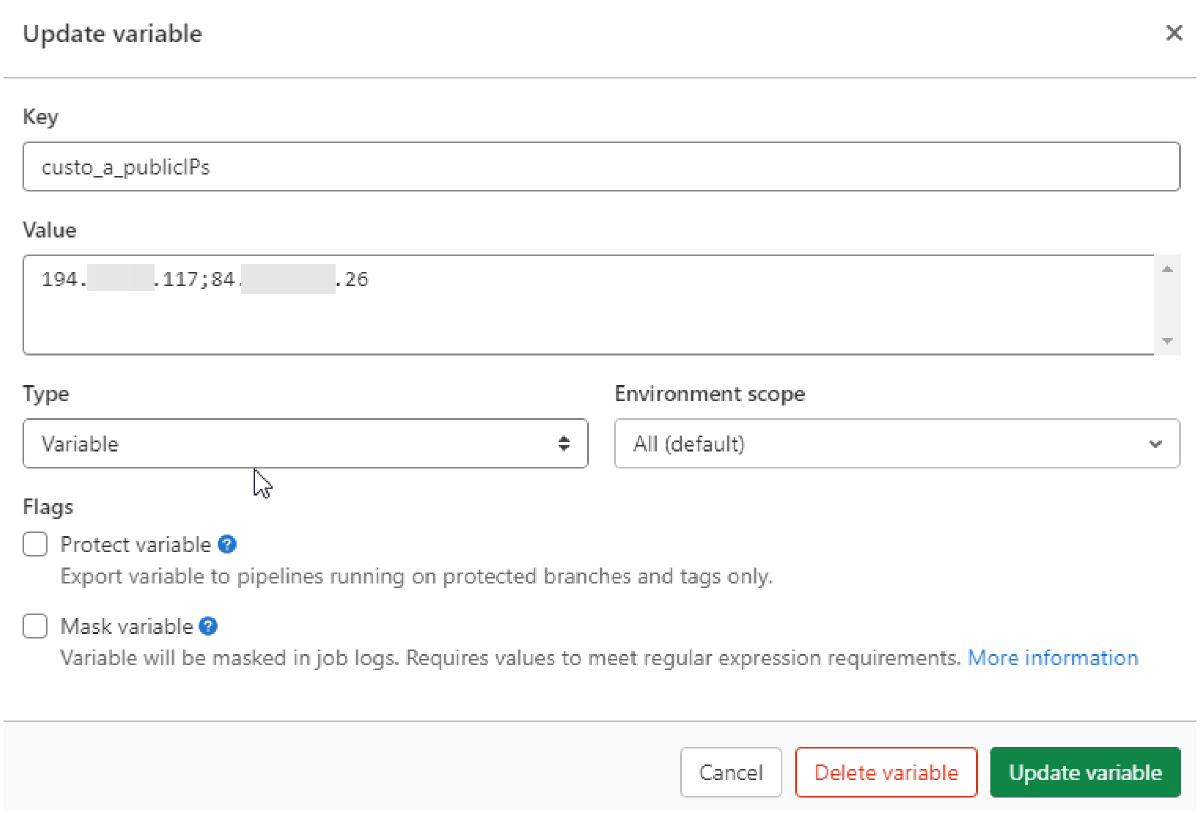
This stage, and all other stages, are defined in the customer-specific directory and will require a secondary file called “approved_inbound_ports.csv”. The name of this file is the same for every customer. The content of this file will use the following structure, where “xxx.xxx.xxx.xxx” is the public IP for which the file approves the defined port “yyy”.

```
public_ip;approved_port
xxx.xxx.xxx.xxx;yyy
```

Secondly, the stage also requires a list of the public IPs of the customer. These are provided using the GitLab pipeline variables. The Python script requires one key with a list of the IPs separated by a “;”. An example is provided in Figure 5.

**Figure 5**

*Screenshot of public IP variable.*


Update variable

Key

custo_a_publicIPs

Value

```
194.[redacted].117;84.[redacted].26
```

Type: Variable

Environment scope: All (default)

Flags

- [ ] Protect variable
  Export variable to pipelines running on protected branches and tags only.
- [ ] Mask variable
  Variable will be masked in job logs. Requires values to meet regular expression requirements. More information

*Note.* By M. Neuville, 2021

In the "*.yml" file of the customers directory the following job must be created.

- Stage: WAN
- Tags: mssp (to use the runners located at the MSSP)
- Script
    - Get API key from HashiCorp Vault
    - Trigger a Python script with the required variables
- Allow_failure: true (if not defined the pipeline would not test other configurations/stages)
- Image: to specify the docker image required for the script.

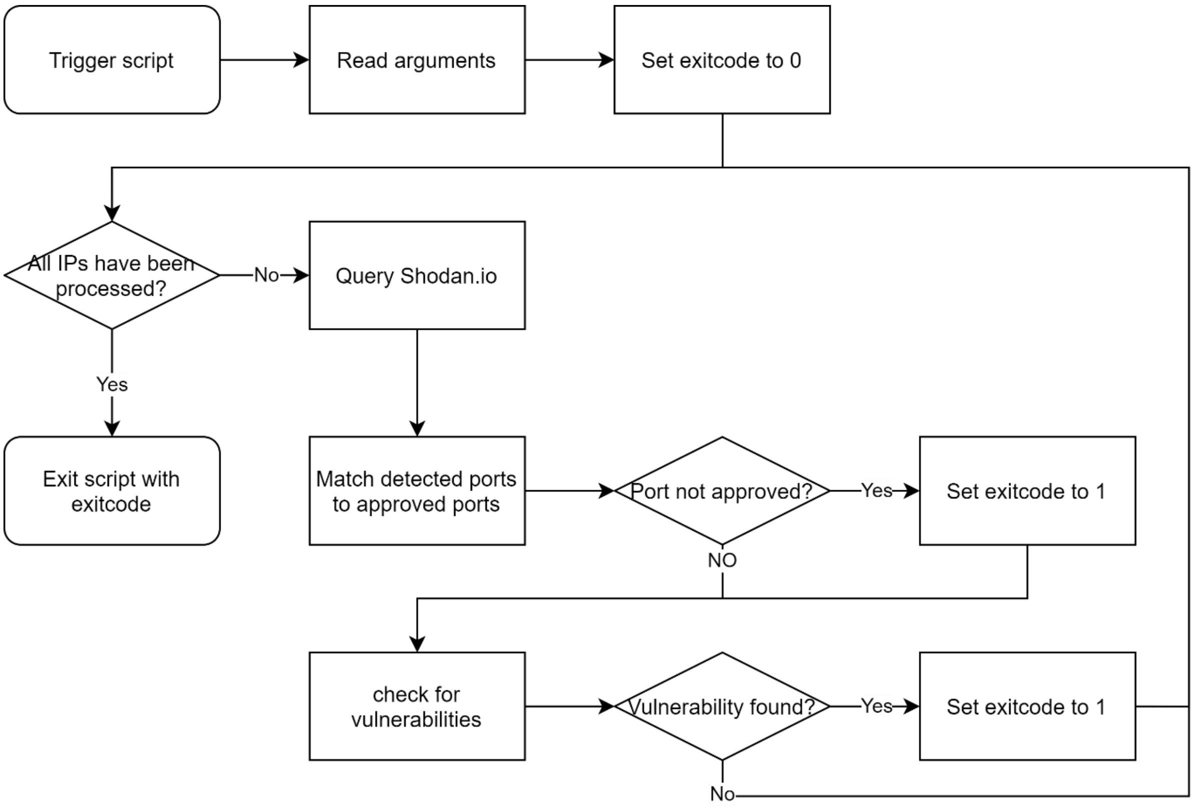
```
custo_a_WAN:
    image: python:3
    stage: WAN
    tags:
        - mssp
    script:
        - apt-get update && apt-get install -y
        - apt-get install -y python3-pip
        - apt-get install software-properties-common -y
        - curl -fsSL https://apt.releases.hashicorp.com/gpg | apt-key add -
        - apt-add-
repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
        - apt-get update
        - apt-get install --reinstall -y vault
        - setcap cap_ipc_lock= /usr/bin/vault
        - export VAULT_ADDR=http://ucdp-ci-vault.westeurope.cloudapp.azure.com:8200/
        - export VAULT_TOKEN="$(vault write -
field=token auth/jwt/login role=shodan_api_token jwt=$CI_JOB_JWT)"
        - export API_TOKEN="$(vault kv get -field=shodan_api_token global/shodan)"
        - python3 -m pip install requests
        - python3 ./lib/wan_check_shodan.py --public_ip=$custo_a_publicIPs --
shodan_api_token=$API_TOKEN --customer=customer_a
```

```
    allow_failure: true
```

As visualized in figure 6, the Python script (see Appendix D) will make an API call to Shodan.io for every IP provided in the variable. These open ports from this result will then be matched against the previously defined approved list. If any port has been found that is not approved the script will exit with code 1, resulting in a failed job. Secondly, the result will also be reviewed for detected vulnerabilities. If any are found, the script will exit with 1 and fail the job.

Figure 6

*Diagram of Python script to detect WAN misconfigurations and vulnerabilities.*



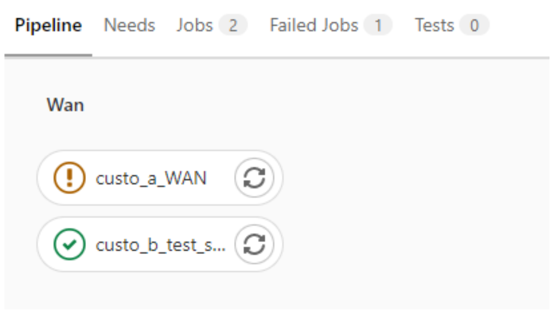
*Note.* By M. Neuville, 2021

## Result of stage WAN implementation

Because of the "allowed to fail tag" the stage will result in a warning if a job has failed. In Figure 7 the job failed for customer_a because of detected vulnerabilities and succeeded for customer_b. Figure 8 shows the result for a configuration with approved ports (customer_b) and Figure 9 shows the result for a configuration with an unapproved port and a vulnerability.

Figure 7

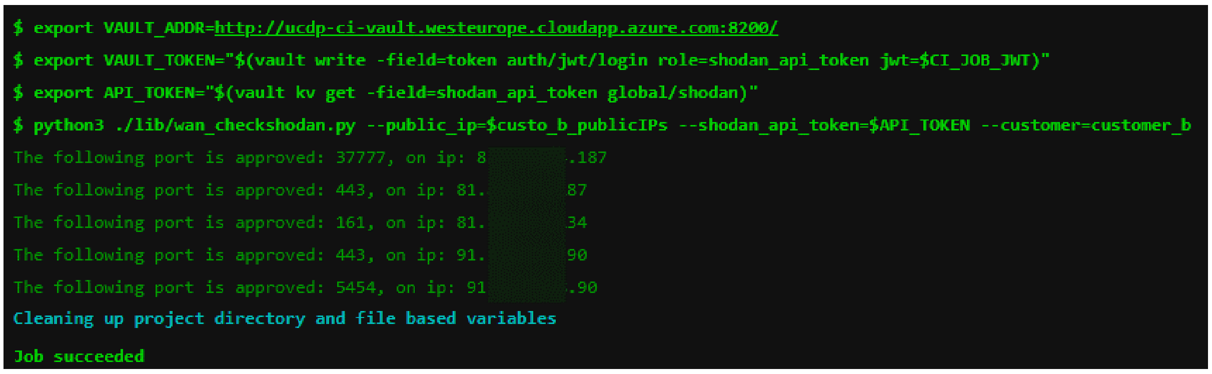
*Pipeline of WAN stage.*



*Note.* By M. Neuville, 2021

Figure 8

*Successful WAN stage of customer_b.*

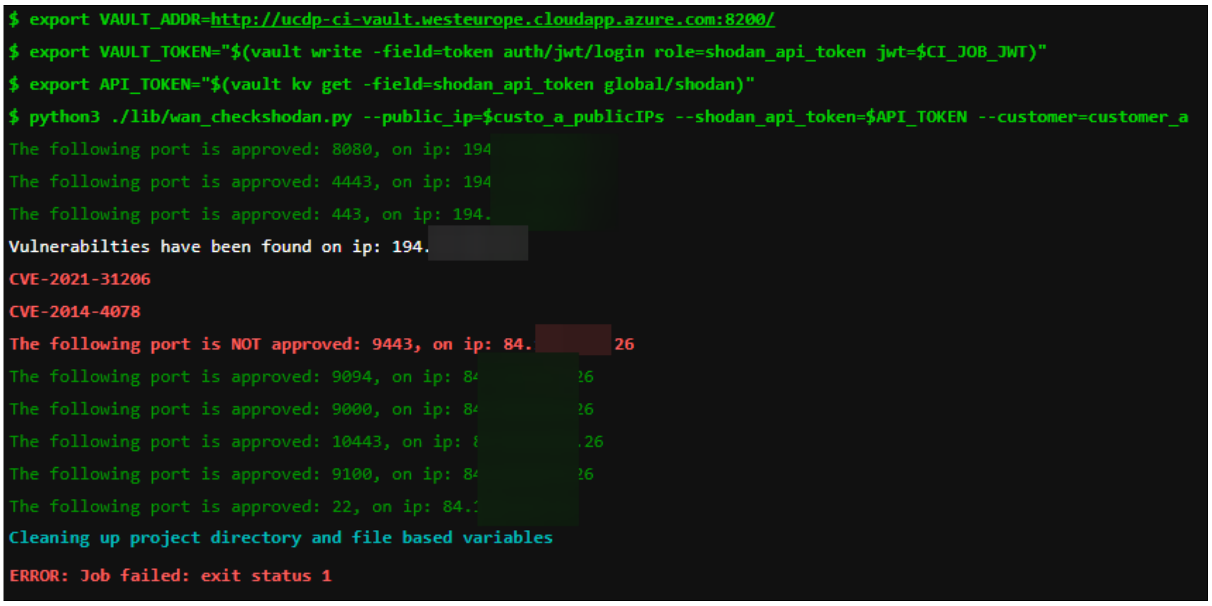


```
$ export VAULT_ADDR=http://ucdp-ci-vault.westeurope.cloudapp.azure.com:8200/
$ export VAULT_TOKEN="$(vault write -field=token auth/jwt/login role=shodan_api_token jwt=$CI_JOB_JWT)"
$ export API_TOKEN="$(vault kv get -field=shodan_api_token global/shodan)"
$ python3 ./lib/wan_checkshodan.py --public_ip=$custo_b_publicIPs --shodan_api_token=$API_TOKEN --customer=customer_b
The following port is approved: 37777, on ip: 8[redacted].187
The following port is approved: 443, on ip: 81.[redacted]87
The following port is approved: 161, on ip: 81.[redacted]34
The following port is approved: 443, on ip: 91.[redacted]90
The following port is approved: 5454, on ip: 91[redacted].90
Cleaning up project directory and file based variables

Job succeeded
```

*Note.* By M. Neuville, 201

Figure 9

*Failed WAN stage of customer_a.*



```
$ export VAULT_ADDR=http://ucdp-ci-vault.westeurope.cloudapp.azure.com:8200/
$ export VAULT_TOKEN="$(vault write -field=token auth/jwt/login role=shodan_api_token jwt=$CI_JOB_JWT)"
$ export API_TOKEN="$(vault kv get -field=shodan_api_token global/shodan)"
$ python3 ./lib/wan_checkshodan.py --public_ip=$custo_a_publicIPs --shodan_api_token=$API_TOKEN --customer=customer_a
The following port is approved: 8080, on ip: 194[redacted]
The following port is approved: 4443, on ip: 194[redacted]
The following port is approved: 443, on ip: 194.[redacted]
Vulnerabilties have been found on ip: 194.[redacted]
CVE-2021-31206
CVE-2014-4078
The following port is NOT approved: 9443, on ip: 84.[redacted]26
The following port is approved: 9094, on ip: 84[redacted]26
The following port is approved: 9000, on ip: 84[redacted]26
The following port is approved: 10443, on ip: [redacted].26
The following port is approved: 9100, on ip: 84[redacted]26
The following port is approved: 22, on ip: 84.[redacted]
Cleaning up project directory and file based variables

ERROR: Job failed: exit status 1
```

*Note.* By M. Neuville, 2021

## Stage: LAN

The LAN stage will exist of two jobs in which the firewall configuration will be verified. The first job will query portquiz.net to check which outbound port is open. The second job will download malicious files to verify the security services of the firewall. Both jobs are executed from within the local network of the customer. Both are executed on a Docker container to limit exposure risk to production environment.

### Verifying outbound ports

To verify the outbound ports, a list of approved ports is required. Under the customer specific directory create a new file called: “approved_outbound_ports.csv”. This file will contain the list of approved outbound ports, using the following structure, where 443 is an example of an approved port.

```
port
443
```

The pipeline configuration (*.yml) for each customer must be updated with a new job. For these jobs the grouping option will be used to combine the two jobs in this stage in one visualization job.

Secondly the job may start to run even before the previous stage has been complete. To do this, the job will be configured with an empty array for the “needs” keyword.

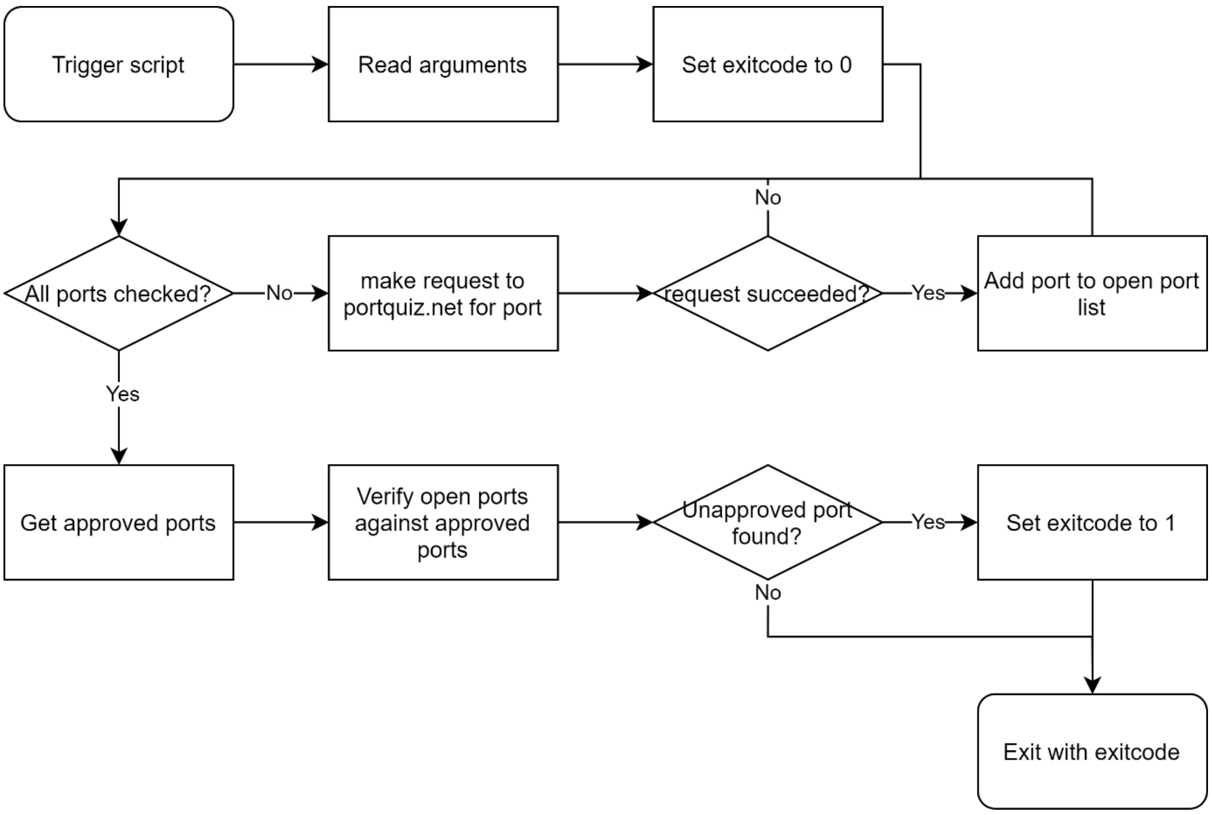
Thirdly this job will be tagged with the same tag used for the on-premises runner. This way we force the job to be run on the runner installed in the customer’s local network.

Finally, the script part will trigger a Python script (see Appendix E) which will verify if a port is open by making a query to portquiz.net. These results are then mapped against an approved list. See Figure 10 for a diagram explaining the Python script.

```
custo_a_LAN 1/2:
    image: python:3
    stage: LAN
    tags:
        - custo_a-001
    needs: []
    script:
        - apt-get update && apt-get install -y
        - apt-get install -y python3-pip
        - python3 -m pip install requests
        - python3 ./lib/lan_check_outbound_ports.py --customer=customer_a
    allow_failure: true
```

**Figure 10**

*Diagram of Python script to verify outbound ports.*



*Note.* By M. Neuville, 2021

## Downloading malware to verify security services

In this PoC the firewall services will be validated by downloading a malicious file from a public repository. This file will be downloaded using a HTTP request and a HTTPS request. As HTTPS is encrypted, the firewall should implement a decryption solution to monitor this traffic.

The second job in the pipeline configuration will again trigger a Python script and will use the same job settings as the previous job.

```yaml
custo_a_LAN 2/2:
    image: python:3
    stage: LAN
    tags:
        - custo_a-001
    needs: []
    script:
        - apt-get update && apt-get install -y
        - apt-get install -y python3-pip
        - python3 -m pip install requests
        - python3 ./lib/lan_check_firewall_security_services.py
    allow_failure: true
```

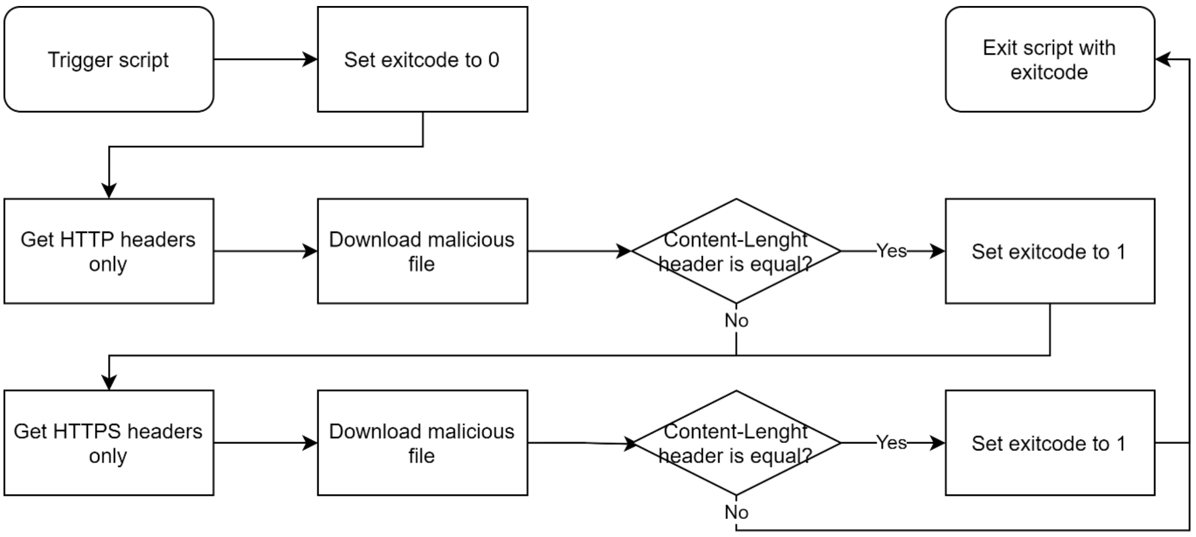
This script (see Appendix F) will do the same steps for HTTP and HTTPS and will fail if any method allows to download the malicious file. The verification of this will be done using the content-size header.

Firstly, a request will be made which will only request the headers and which simulates the GET request. This allows the job to get the original file size of the malware.

Secondly the file will be download in the memory of the running process. The file will not be saved on the file system of the runner and will be removed of the system when the runner finishes the job. Both headers of the request will be matched and if the content is similar, the file has been successfully downloaded. This process is shown in Figure 11.

Figure 11

*Diagram of Python script to verify security services on the firewall.*



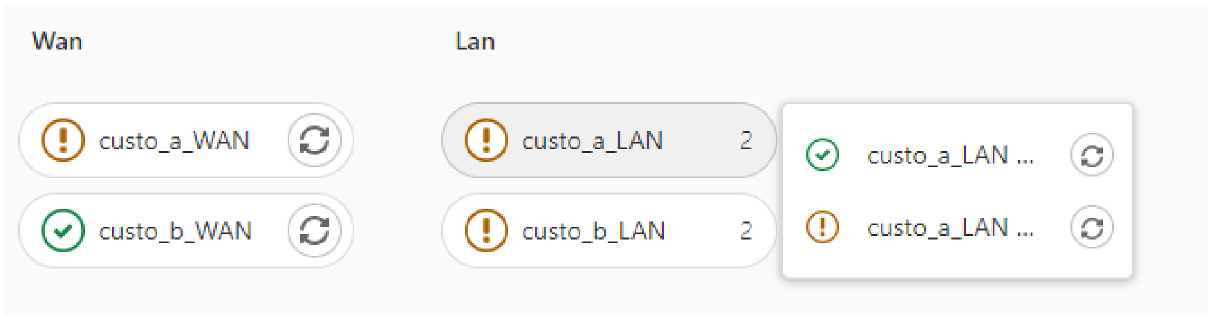
*Note.* By M. Neuville, 2021

## Result of stage LAN implementation

The pipeline will now consist out of two stages with the three jobs, for each company, grouped at the LAN stage, as shown in Figure 12.

Figure 12

*Pipeline of WAN and LAN stage with grouped jobs.*

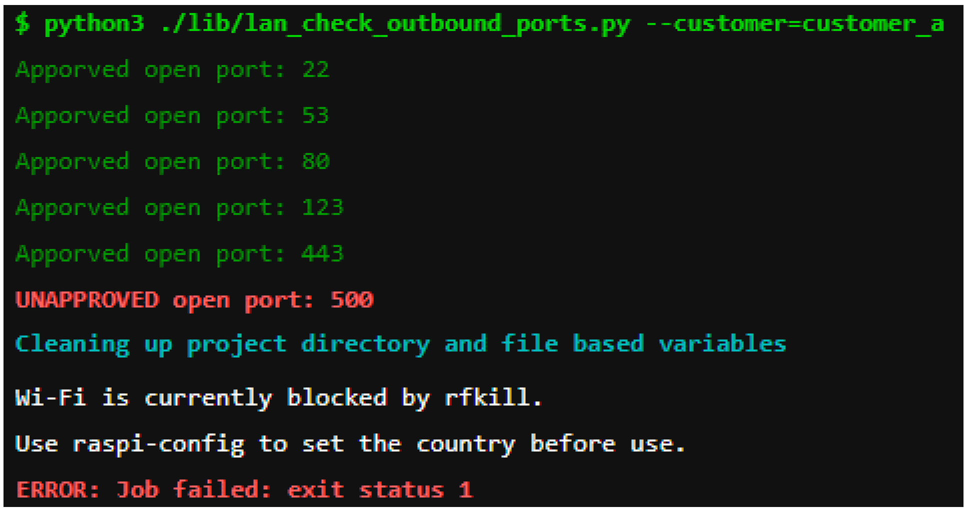


*Note.* By M. Neuville, 2021

When looking in the result of the two jobs the reason is again visible in the output. Figure 13 illustrates an unapproved open port 500, while Figure 14 illustrates the successful download of the malware.

Figure 13

*Failed job detecting open outbound ports.*

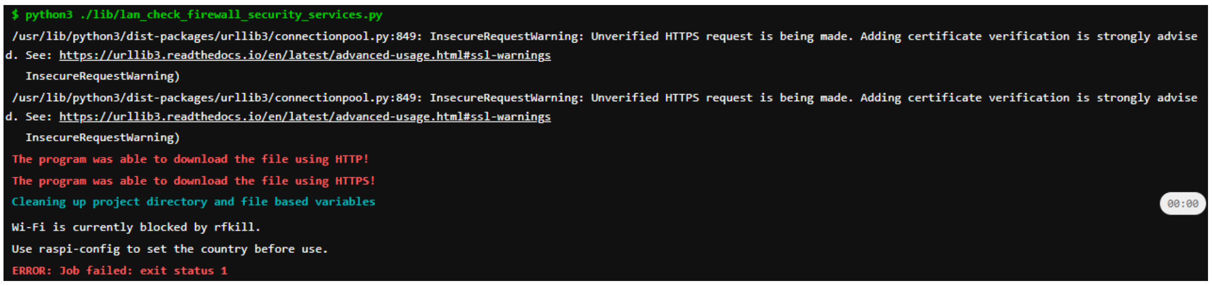


```
$ python3 ./lib/lan_check_outbound_ports.py --customer=customer_a
Apporved open port: 22
Apporved open port: 53
Apporved open port: 80
Apporved open port: 123
Apporved open port: 443
UNAPPROVED open port: 500
Cleaning up project directory and file based variables
Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.
ERROR: Job failed: exit status 1
```

*Note.* By M. Neuville, 2021

Figure 14

*Failed job detecting the possibility to download malware.*



```
$ python3 ./lib/lan_check_firewall_security_services.py
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:849: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advise
d. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:849: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advise
d. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
The program was able to download the file using HTTP!
The program was able to download the file using HTTPS!
Cleaning up project directory and file based variables                                   00:00
Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.
ERROR: Job failed: exit status 1
```

*Note.* The errors regarding the unverified HTTPS request are because the repository is using a self-signed certificate. By M. Neuville, 2021

## Stage AAD

### Adding extra value to the HashiCorp Vault

To authenticate against the Azure Active Directory, the CI job requires a User Principal Name (UPN) and a password. As this might be a privileged account, and even if it wasn't, these credentials must be safely stored.

Add the username and password to the previously created secret engine.

```
vault kv put customer_a/aad aad_upn='<upn>' aad_pw='password>'
```

Create the access policy.

```
vault policy write customer_a_aad - <<EOF
# Policy name: customer_a_aad
#
# Read-only permission on customer_a/data/*' path
path "customer_a/data/*" {
  capabilities = [ "read" ]
}
```

```
EOF
```

Create the required role.

```
vault write auth/jwt/role/customer_a_aad - <<EOF
{
  "role_type": "jwt",
  "policies": ["customer_a_aad"],
  "token_explicit_max_ttl": 60,
  "user_claim": "user_email",
  "bound_claims": {
    "project_id": "34",
    "ref": "main",
    "ref_type": "branch"
  }
}
EOF
```

## Using PowerShell modules to communicate with Azure/M365.

This job will make several requests to Azure and M365 solutions to verify certain configuration topics. If the result is not what we want, not secure or not best practice, the script will exit using 1 and failing the job. Some modules are only supported in PowerShell 7, so the runner will need to use “pwsh” instead of PowerShell as the shell variable. This can be set in the config.toml file of the runner.

```
[[runners]]
  name = "<runner identification name>"
  url = "https://<gitlab url>/"
  token = "<runner token"
  executor = "shell"
  shell = "pwsh"
  [runners.custom_build_dir]
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
```

To securely store the username and password to connect to the tenant, the job will request the data from the HashiCorp Vault. Unlike the job running in a Linux environment, this job will run from a Windows based solution. This approach limits the job in using any vault binaries and will request the credentials using API calls. These calls are coded in the “aad_check.ps1” script (see Appendix G). Following is a code snippet showing how to make the API call and get the secret.

```
# Query Vault API for token
$postdata = '{\"role\": \"'+$customer+'_aad\", \"jwt\": \"' + $JWT + '\"}'
$vaultlogin = curl --request POST --data $postdata http://<vault-url>/v1/auth/jwt/login -s
$jsonlogin = ConvertFrom-Json $vaultlogin
$vaulttoken = $jsonlogin.auth.client_token
# Query vault API for secret
$secret = curl --header "X-Vault-Token: $vaulttoken" http://<vault-url>/v1/$customer/data/aad -s
$secretjson = ConvertFrom-Json $secret
```

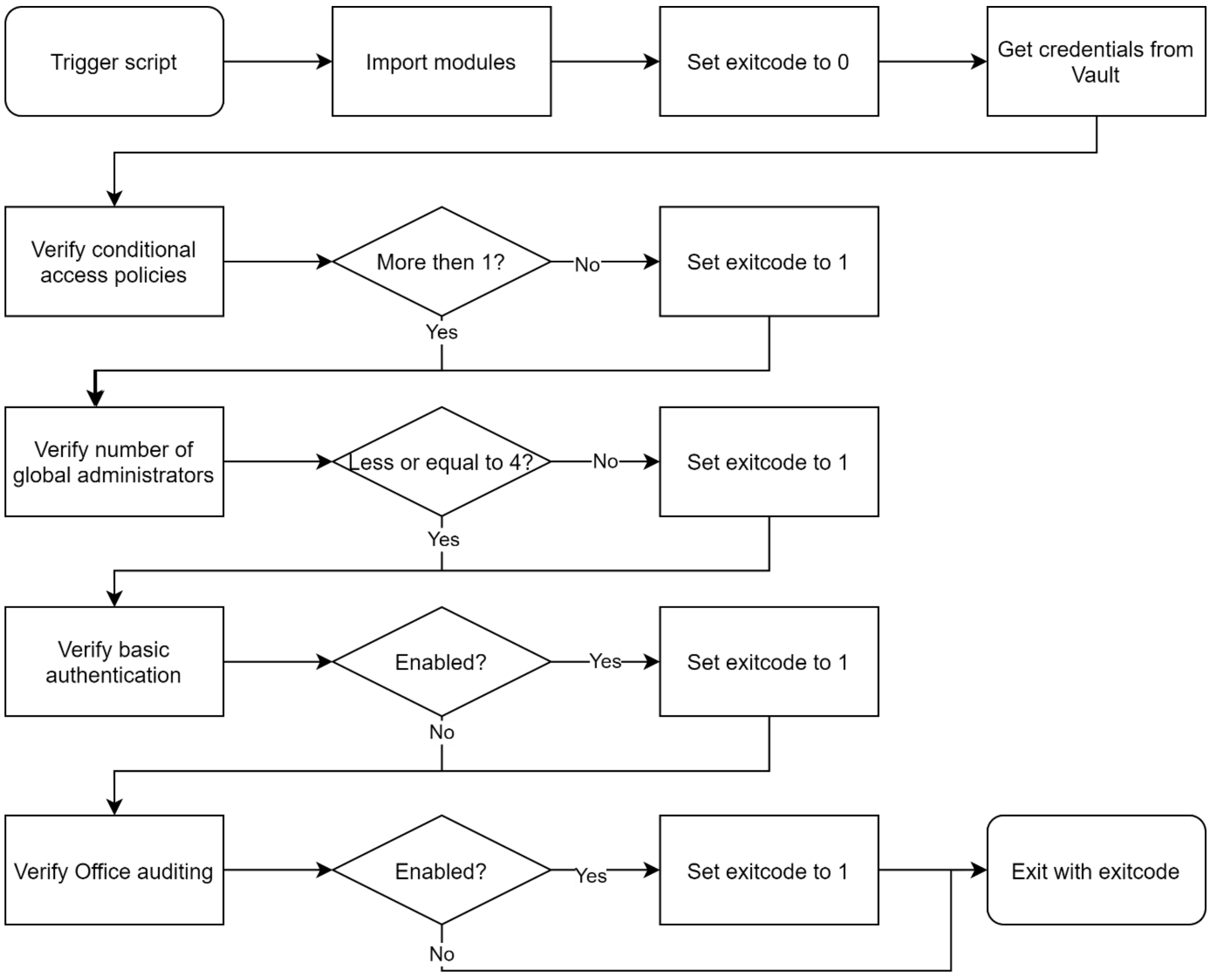
Add the following code to the customer “*.yml” file. Do note that the tag now links to a Windows based GitLab runner.

```
custo_a_AAD:
    stage: AAD
    tags:
        - custo_a-003
    needs: []
    script:
        - pwsh -File ./lib/aad_check.ps1 -JWT $CI_JOB_JWT -customer customer_a
    allow_failure: true
```

In Figure 15 the flow used by the script is visualized. Four different parameters are checked using two different modules.

**Figure 15**

*Diagram of aad_check.ps1 script.*



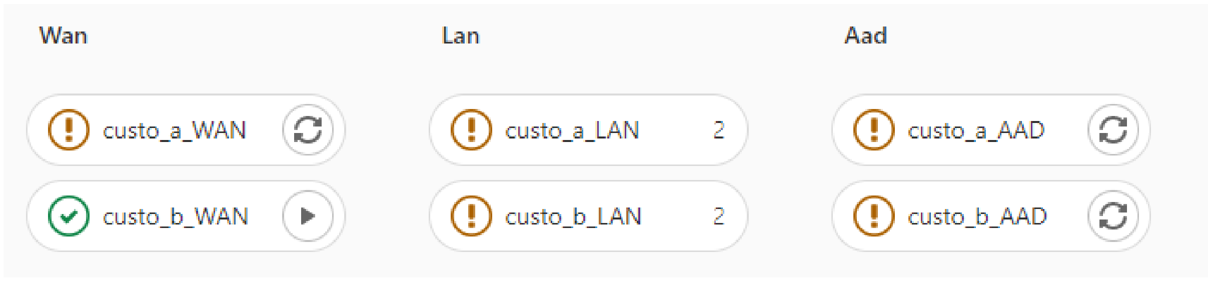
*Note.* By M. Neuville, 2021

## Result of stage AAD implementation

After configuring the AAD stage in the “*.yml" and running the test against the Azure and M365 infrastructure the following job is added to the pipeline. This is illustrated in Figure 16.

Figure 16

*Pipeline containing WAN, LAN, and AAD stage.*

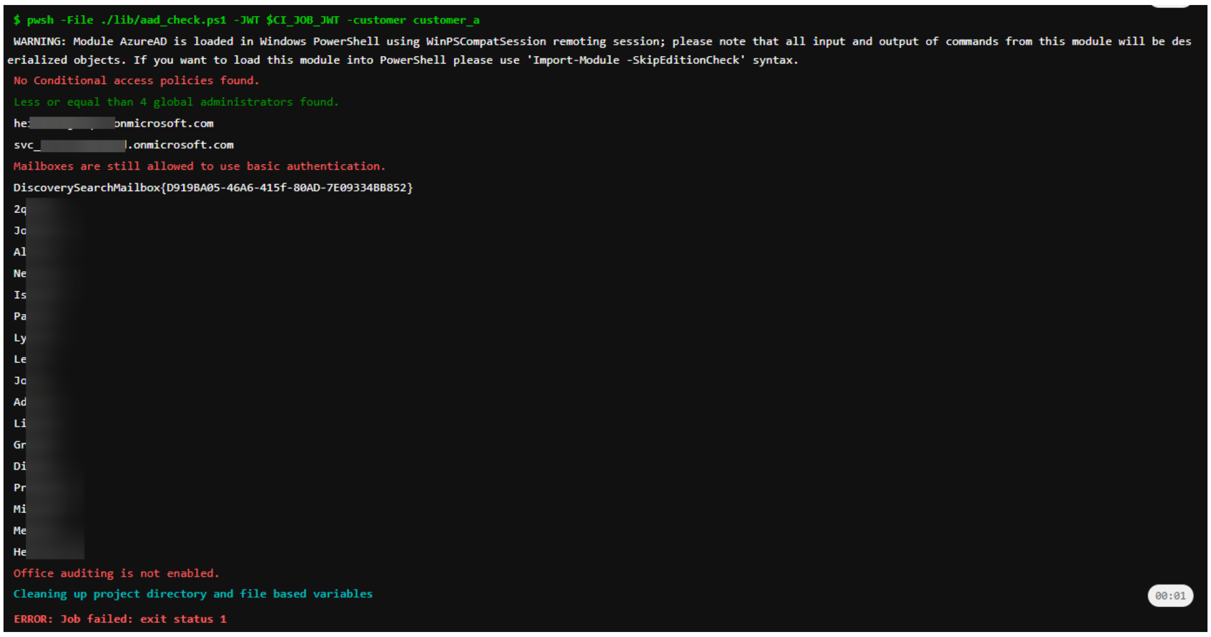


*Note.* By M. Neuville, 2021

Looking into the details why the job failed, see Figure 17, the MSSP can act to enhance the configuration of the customers Azure and M365 infrastructure. The result can include the required information to provide a targeted approach if needed.

Figure 17

*Result of a failed job in the AAD stage.*



```
$ pwsh -File ./lib/aad_check.ps1 -JWT $CI_JOB_JWT -customer customer_a
WARNING: Module AzureAD is loaded in Windows PowerShell using WinPSCompatSession remoting session; please note that all input and output of commands from this module will be des
erialized objects. If you want to load this module into PowerShell please use 'Import-Module -SkipEditionCheck' syntax.
No Conditional access policies found.
Less or equal than 4 global administrators found.
he[redacted]onmicrosoft.com
svc_[redacted].onmicrosoft.com
Mailboxes are still allowed to use basic authentication.
DiscoverySearchMailbox{D919BA05-46A6-415f-80AD-7E09334BB852}
2q[redacted]
Jo
Al
Ne
Is
Pa
Ly
Le
Jo
Ad
Li
Gr
Di
Pr
Mi
Me
He
Office auditing is not enabled.
Cleaning up project directory and file based variables                                                    00:01
ERROR: Job failed: exit status 1
```

*Note.* By M. Neuville, 2021

## Stage AD

### Adding extra value to the HashiCorp Vault

Using the same method as in stage AAD add the username and password to the HashiCorp Vault. Place them under the customer\ad path.

```
vault kv put customer_a/ad ad_upn='<upn>' ad_pw='password>'
```

Similar as the previous steps, create the required policy and role.

```
vault policy write customer_a_ad - <<EOF
# Policy name: customer_a_ad
#
# Read-only permission on customer_a/data/*' path
path "customer_a/data/*" {
  capabilities = [ "read" ]
}
EOF
```

```
vault write auth/jwt/role/customer_a_ad - <<EOF
{
  "role_type": "jwt",
  "policies": ["customer_a_ad"],
  "token_explicit_max_ttl": 60,
  "user_claim": "user_email",
  "bound_claims": {
    "project_id": "34",
    "ref": "main",
    "ref_type": "branch"
  }
}
EOF
```

## Verifying registry keys

To verify the configuration on the reference host we can setup a remote PowerShell session and invoke the commands using that session. Similar with the AAD stage, the script (see Appendix H) will gather the credentials from the HashiCorp Vault using the API. This job requires that the reference host allows incoming remote PowerShell sessions.

```
Enable-PSRemoting
```

This job will run on a Windows GitLab runner. Do note that in this lab the reference host and the runner where in the same Active Directory Domain. If this is not the case it might be possible that the TrustedHosts must be configured correctly on both the runner and the reference host.

```
Set-Item WSMan:\localhost\Client\TrustedHosts -Value *
```

Lastly because the job uses Invoke-Command commands to gather the data from the reference host, this command must return the correct exit value, which must be processed separately after each command. This is illustrated in the following code snippet.

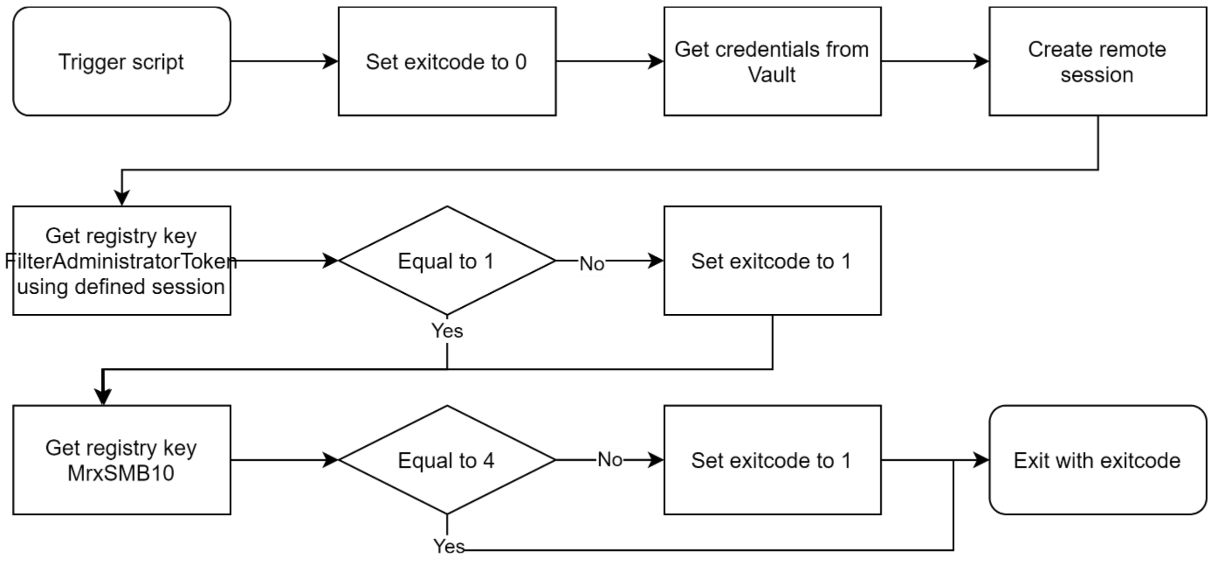
```
$result = Invoke-Command -Session $remote -ScriptBlock {
     $exitcode_inBlock = 0
     $value = Get-ItemPropertyValue -
Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System -
Name "FilterAdministratorToken"
     if ($value -eq 1)
     {
         Write-Host "$([char]27)[92mAdmin approval mode for the built-
in administrator account is enabled.$([char]27)[0m"
     }
     else
     {
        Write-Host "$([char]27)[91mAdmin approval mode for the built-
in administrator account is NOT enabled.$([char]27)[0m"
        $exitcode_inBlock = 1
     }
    return $exitcode_inBlock
}
```

```
if ($result -eq 1) {$exitcode = 1}
```

Figure 18 illustrates a diagram which represents the flow of the script. In the script two registry keys are verified based on the CIS benchmark recommendations. (Ferguson, et al., 2021)

**Figure 18**

*Diagram of ad_check.ps1.*



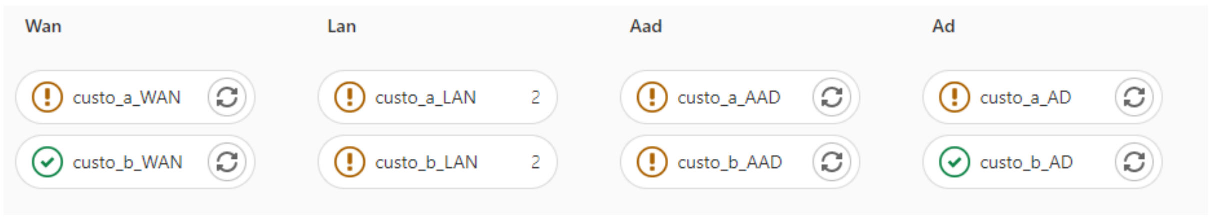
*Note.* By M. Neuville, 2021

## Result of stage AD implementation

After implementing the last stage, the MSSP now has a good overview of the health of the customers infrastructure. This is visible in Figure 19.

**Figure 19**

*Pipeline after implementing all stages.*

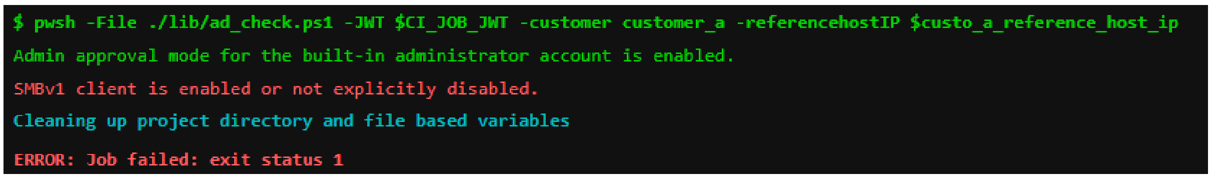


*Note.* By M. Neuville, 2021

When looking into the details why the script failed the MSSP has more information on which checks failed. This is illustrated in Figure 20.

Figure 20

*Failed AD verification script.*



```
$ pwsh -File ./lib/ad_check.ps1 -JWT $CI_JOB_JWT -customer customer_a -referencehostIP $custo_a_reference_host_ip
Admin approval mode for the built-in administrator account is enabled.
SMBv1 client is enabled or not explicitly disabled.
Cleaning up project directory and file based variables

ERROR: Job failed: exit status 1
```

*Note.* By M. Neuville, 2021

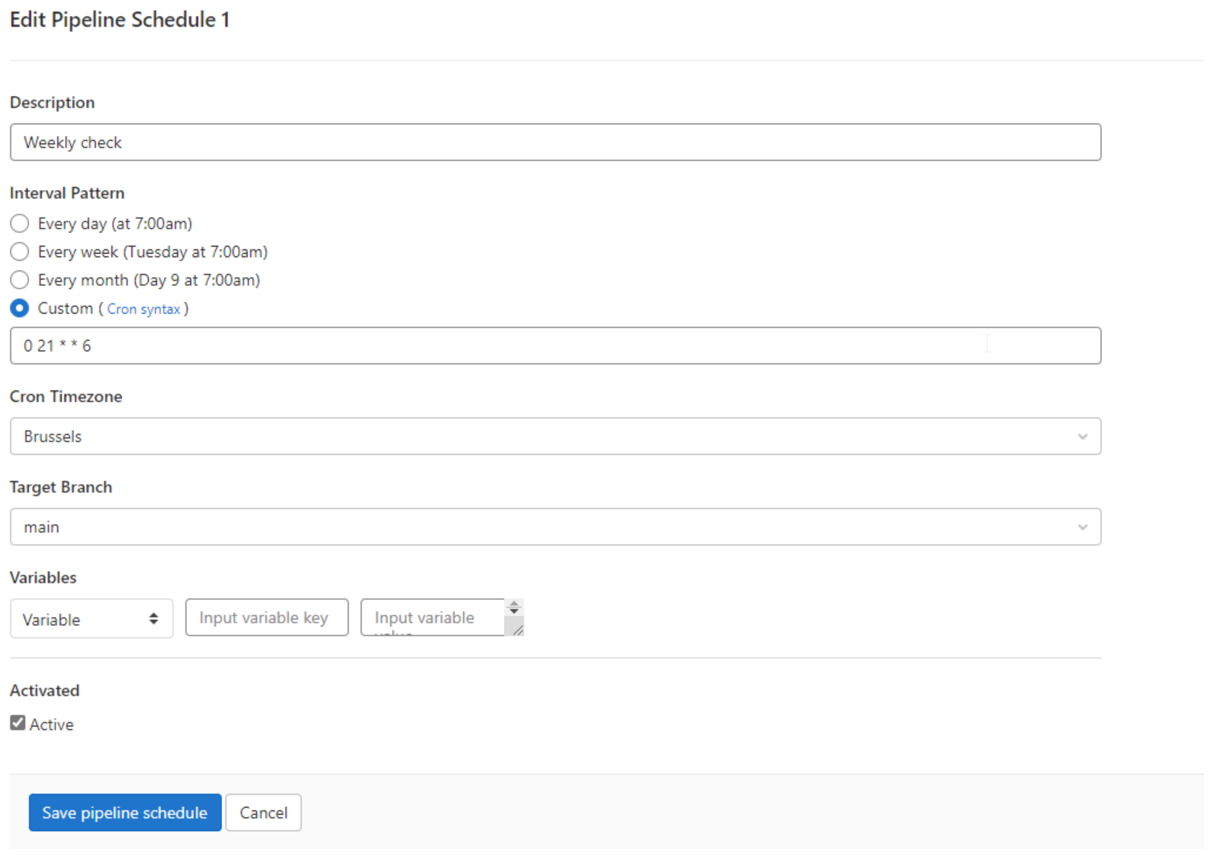
## Setting the schedule

Now that the pipeline is working as expected, it should run on a regular basis to verify the configuration on a regular basis. This can be done using the following steps. (GitLab, 2021)

Navigate to the project's CI/CD > Schedules page and click the New schedule button.
Fill in the Schedule a new pipeline form as illustrated in Figure 21.
Click the Save pipeline schedule button.

Figure 21

*Pipeline schedule every Saturday at 21h00.*



Edit Pipeline Schedule 1

Description: Weekly check

Interval Pattern
- Every day (at 7:00am)
- Every week (Tuesday at 7:00am)
- Every month (Day 9 at 7:00am)
- [✓ Custom ( Cron syntax )]

0 21 * * 6

Cron Timezone: Brussels

Target Branch: main

Variables: Variable | Input variable key | Input variable value

Activated: ✓ Active

*Note.* By M. Neuville, 2021

## Remarks on the setup

During the development of this proof of concept several different remarks must be made on the setup. Most of these, if not all, are related to the security of this setup and privacy issues. To provide more guidance, a reference has been made to the Mitre ATT&CK framework. Following is a list which summarizes the remarks.

1. The HashiCorp Vault must be configured to use HTTPS instead of HTTP. (Mitre ATT&CK ID T1040)
2. The HashiCorp Vault policies should be more restrictive to only provide access to the required keys. (Mitre ATT&CK ID T1212)
3. The JWT tokens do not differentiate between the different customers. The token could be misused to gather sensitive information of other customers. (Mitre ATT&CK ID T1212)
4. The GitLab runner cache on Windows is not removed after the job, caching should be disabled or Docker containers should be used. The GitLab runner cache includes sensitive data from all customers in the project. (MITRE ATT&CK ID T1083)
5. The GitLab runner directory should be protected with the least-privileged NTFS rights. This to prevent any unauthorized user from manipulating the runner instance. (MITRE ATT&CK ID T1543.003)
6. The GitLab runner should run with a dedicated service account instead of the system account. Doing so limits the privileges of the service which is setup with the account system. (MITRE ATT&CK ID T1068)
7. PowerShell logging on the GitLab runner might expose the used credentials. (MITRE ATT&CK ID T1005)
8. Setting the TrustedHosts to include all ("*") exposes the device to unnecessary risks. (MITRE ATT&CK ID T1021.006)
9. If the customer is using Geo-IP filtering on its firewall, it's possible that Shodan won't detect anything. (MITRE ATT&CK ID T1593.002)

This list, which is not limitative, is provided as a handle to mitigate some important issues when using this solution in production environments.

## Conclusion

During this proof of concept, it became clear that CI pipelines does provide certain advantages. Starting with a centralized repository that gets matched against all onboarded customers, which provides a quick overview of the security health of these infrastructures. The centralized repository also allows to add verifications directly for all customers. During the next verification cycle these are then matched and flagged as incorrectly configured.

A second advantage of using the pipelines is an easy onboarding, which can be (partially) performed by colleagues who have no knowledge of the solution. This spreads the cost and load of the initial investment over other services which are already in place.

A final advantage to note is the use of a scheduled verification check. By running the pipeline on a predefined schedule, a continuous assessment is provided. If any changes, against the defined baseline, are made by either the customer, the MSSP or another source it will be flagged on the next run. This provides some peace of mind for both the MSSP and the customer.

To conclude, a more in-depth development of this proof of concept could provide a valuable solution or service to offer. The solution also allows to be extended with other verifications and integrations. Before it can be implemented, it must be setup using the best practices of each product and the remarks mentioned should be addressed accordingly.

## References

Ferguson, W., Hunt, J., Markl, K., Sarinana, S., Edgerton, H., Mehrotara, H., Zhang, K, White,P, Braum, J, Woods, M, Jarose, J, Moten, C, Margosis, A, Munck, R & Harris, M. (2021). *CIS Microsoft Intune for Windows 10 Release 2004 Benchmark [pdf].* CIS. https://downloads.cisecurity.org/#/

GitLab. (2021, July 28). *Authenticating and reading secrets with HashiCorp Vault*. GitLab Docs. https://docs.gitlab.com/ee/ci/examples/authenticating-with-hashicorp-vault/

GitLab. (2021, October 6). *Install GitLab Runner on Windows.* GitLab Docs. https://docs.gitlab.com/runner/install/windows.html

GitLab. (2021, October 19). *Pipeline schedules*. GitLab Docs. https://docs.gitlab.com/ee/ci/pipelines/schedules.html

GitLab. (2021, September 17). *Using external secrets in CI*. GitLab Docs. https://docs.gitlab.com/ee/ci/secrets/

GitLab. (n.d.). *Install self-managed GitLab*. GitLab. https://about.gitlab.com/install/#debian

HashiCorp. (n.d.). *Install Vault*. HashiCorp Learn. https://learn.hashicorp.com/tutorials/vault/getting-started-install

Maurice, M. (n.d.). Outgoing port tester. From http://portquiz.net/

Raya, T., & Davis, C. (2021, September 9). *Exchange Online PowerShell*. Microsoft Docs: https://docs.microsoft.com/en-us/powershell/exchange/exchange-online-powershell?view=exchange-ps

Shodan. (n.d.). *Shodan*. Shodan: https://www.shodan.io/

van Stijn, S., Kester, A., m-qXimilian, ProgramComputer, P., Hello, P. D., denis-roy, & linuxmetel. (2021, October 20). *Install Docker Engine on Debian*. from Docker docs: https://docs.docker.com/engine/install/debian/

## Table of figures

# Appendix A – Installation of HashiCorp Vault

This installation manual is based on the tutorials on HashiCorp Learn, more in-depth information is available there. The content of this tutorial is also limited to the scope and requirements of the paper and does not follow any hardening guidelines. (HashiCorp, n.d.)

## Prerequisites

- Ubuntu or Debian (can be installed on other distributions)

## Installation

Install gnupg2.

```
sudo apt-get install gnupg2
```

Install the software-properties-common package.

```
sudo apt-get install software-properties-common -y
```

Add the HashiCorp GPG key.

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add
```

Add the official HashiCorp Linux repository.

```
sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs)
main"
```

Update and install.

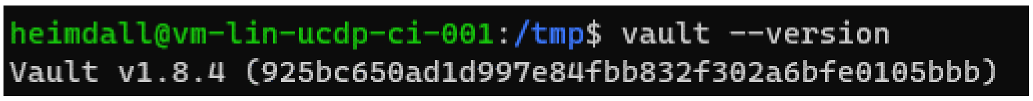
```
sudo apt-get update && sudo apt-get install vault
```

Verify if vault is installed.

```
Vault --version
```

Figure 22

*Screenshot of verification that vault has been installed.*



```
heimdall@vm-lin-ucdp-ci-001:/tmp$ vault --version
Vault v1.8.4 (925bc650ad1d997e84fbb832f302a6bfe0105bbb)
```

*Note. By M. Neuville,2021*

## Basic configuration

Create a config folder in the /opt/vault folder.

```
cd /opt/vault
sudo mkdir config
```

Create a config file config.hcl with the following content.

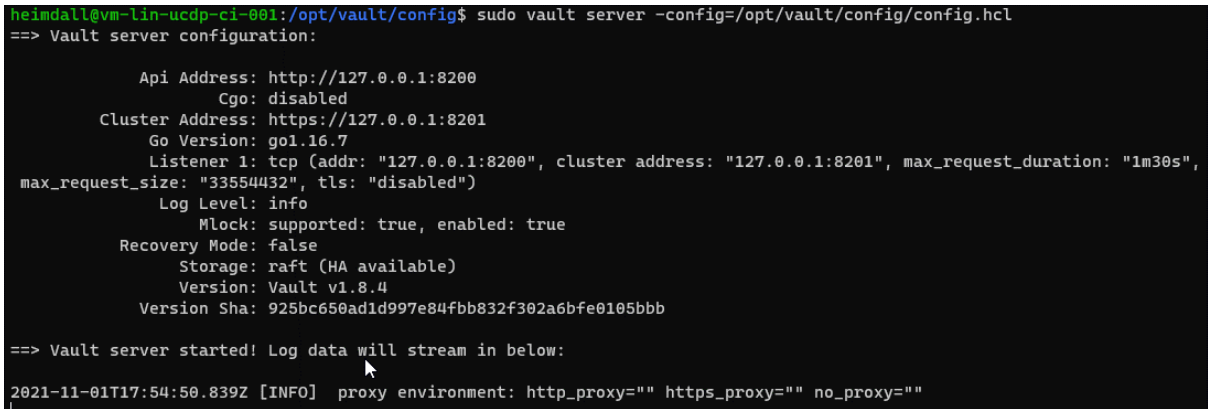
```
storage "raft" {
  path    = "/opt/vault/data"
  node_id = "node1"
}

listener "tcp" {
  address     = "0.0.0.0:8200"
  tls_disable = "true"
}

api_addr = "http://127.0.0.1:8200"
cluster_addr = "https://127.0.0.1:8201"
ui = true
```

Start the vault server.

```
vault server -config=/opt/vault/config/config.hcl
```

Figure 23

*Screenshot of a running vault server.*



```
heimdall@vm-lin-ucdp-ci-001:/opt/vault/config$ sudo vault server -config=/opt/vault/config/config.hcl
==> Vault server configuration:

             Api Address: http://127.0.0.1:8200
                     Cgo: disabled
         Cluster Address: https://127.0.0.1:8201
              Go Version: go1.16.7
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201", max_request_duration: "1m30s",
 max_request_size: "33554432", tls: "disabled")
               Log Level: info
                   Mlock: supported: true, enabled: true
           Recovery Mode: false
                 Storage: raft (HA available)
                 Version: Vault v1.8.4
             Version Sha: 925bc650ad1d997e84fbb832f302a6bfe0105bbb

==> Vault server started! Log data will stream in below:

2021-11-01T17:54:50.839Z [INFO]  proxy environment: http_proxy="" https_proxy="" no_proxy=""
```

*Note.* By M. Neuville, 2021

Stop the vault service by pressing ctrl+c.
Create the vault service.

```
sudo touch /etc/systemd/system/vault.service
```

Add the following configuration to the vault.service file.

```
[Unit]
Description="HashiCorp Vault - A tool for managing secrets"
Documentation=https://www.vaultproject.io/docs/
Requires=network-online.target
After=network-online.target
ConditionFileNotEmpty=/etc/vault.d/vault.hcl
StartLimitIntervalSec=60
StartLimitBurst=3

[Service]
User=root
Group=root
ProtectSystem=full
```

```
ProtectHome=read-only
PrivateTmp=yes
PrivateDevices=yes
SecureBits=keep-caps
AmbientCapabilities=CAP_IPC_LOCK
Capabilities=CAP_IPC_LOCK+ep
CapabilityBoundingSet=CAP_SYSLOG CAP_IPC_LOCK
NoNewPrivileges=yes
ExecStart=vault server -config=/opt/vault/config/config.hcl
ExecReload=/bin/kill --signal HUP $MAINPID
KillMode=process
KillSignal=SIGINT
Restart=on-failure
RestartSec=5
TimeoutStopSec=30
StartLimitInterval=60
StartLimitIntervalSec=60
StartLimitBurst=3
LimitNOFILE=65536
LimitMEMLOCK=infinity
LogRateLimitIntervalSec=0
LogRateLimitBurst=0

[Install]
WantedBy=multi-user.target
```

Start the service.

```
sudo systemctl start vault
```

Set local vault address.

```
export VAULT_ADDR='http://127.0.0.1:8200'
```

Initialize the vault.

```
vault operator init
```

Save the keys and root tokens somewhere secure and preferably not together.
Unseal the vault (enter a key from the step above).

```
vault operator unseal
```

Perform the unseal step three times.
Login with the root token.

```
vault login <Initial_Root_Token>
```

*Note: when using the vault in production, it should be configured according to the best practices.*

## Unsealing vault after reboot

Set local vault address.

```
export VAULT_ADDR='http://127.0.0.1:8200'
```

Verify that service is running.

```
systemctl status vault
```

Unseal the vault three times using the unseal keys.

```
vault operator unseal
```

# Appendix B – GitLab CE Installation

This installation manual is based on the tutorials on GitLab, more in-depth information is available there. The content of this tutorial is also limited to the scope and requirements of the paper and does not follow any hardening guidelines. (GitLab, n.d.)

## Prerequisites

- Debian 10

## Installation

Install dependencies.

```
sudo apt-get update
sudo apt-get install -y curl openssh-server ca-certificates perl
```

Add the GitLab repository.

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo bash
```

Install GitLab.
Change the URL to which the installation will be reachable.

```
sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ee
```

The GitLab is now reachable at the previously defined URL, the password root password can be found using the following command. This password is only available for 24 hours after creation.

```
cat /etc/gitlab/initial_root_password
```

# Appendix C – GitLab runner installation

## With Docker support on Linux

This installation manual is based on the tutorials on GitLab, more in-depth information is available there. The content of this tutorial is also limited to the scope and requirements of the paper and does not follow any hardening guidelines. (GitLab, n.d.)

Installation of Docker is based upon the manual provided by Docker docs. (van Stijn, et al., 2021)

### Prerequisites

- Debian 10

### Install prerequisites

```
sudo apt-get install ca-certificates curl gnupg lsb-release
```

### Docker installation

Configure the repository and install the latest version.

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
keyring.gpg] https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Verify Docker installation.

```
sudo docker run hello-world
```

Install docker-compose.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

### GitLab runner Installation

Add the official GitLab repository.

```
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" |
sudo bash
```

Install latest version of the GitLab Runner.

```
sudo apt-get update
sudo apt-get install gitlab-runner
```

#### Register the runner

Register the runner with the token that can be found under https://<fqdn>/admin/runners.

```
sudo gitlab-runner register --url https://<FQDN>/ --registration-token $REGISTRATION_TOKEN
```

### On Windows without Docker support

The following procedure is available on the official GitLab Docs and is only provided here as a reference for this proof of concept. (GitLab Docs, 2021)

#### Prerequisites

- Windows 11, with PowerShell 7

#### Install Git

Download from the official source (https://git-scm.com/download/win). Leave the default options during install unless otherwise required by personal choice.

#### Install GitLab runner on windows

Download the runner from the official source (https://docs.gitlab.com/runner/install/windows.html).

Create the following folder “c:\GitLab-Runner”.

Move the downloaded installer to the “c:\GitLab-Runner” folder and rename the file to “gitlab-runner.exe”.

Register the runner from an elevated prompt.

```
c:\GitLab-Runner\gitlab-runner.exe register
```

Install the runner as a service.

```
cd C:\GitLab-Runner
.\gitlab-runner.exe install
.\gitlab-runner.exe start
```

## Appendix D – WAN Detection script

This is the script used to query and evaluate the result of the Shodan.io API.

https://github.com/CaptainBlackbutter/ContinuesSecurityConfigurationVerificationUsingPipelines

```python
import sys
import requests
import os
import csv
import argparse
from colors import *

#define project arguments
parser = argparse.ArgumentParser()
parser.add_argument("--public_ip", help="Provide a comma seperated list of public ip's")
parser.add_argument("--shodan_api_token", help="Provide the Shodan.io API token")
parser.add_argument("--customer", help="Provide the customer name, as in the project structure")
args = parser.parse_args()
#set exit value
exitcode = 0
# getting IPs from arguments
ips = args.public_ip.split(";")
# checking ip status
for ip in ips:
    # query and parse result
    query = "https://api.shodan.io/shodan/host/"+ip+"?key="+args.shodan_api_token
    data = requests.get(query)
    jsondata = data.json()
    openports = jsondata.get("ports","[]")
    vulns = jsondata.get("vulns","[]")
    # define approved list path
    approvedlistpath = "./customers/"+args.customer+"/approved_inbound_ports.csv"
    # check if list exists otherwise fail because at least an empty approved list is required.
    if os.path.exists(approvedlistpath):
        approvedports = []
        # read approved list to verify later.
        with open(approvedlistpath, newline='') as csvfile:
                portreader = csv.reader(csvfile, delimiter=";")
                row = 0
                for port in portreader:
                    if (row != 0):
                        if (port[0] == ip):
                            approvedports.append(port[1])
                    row = row + 1
        # check if detected ports are in approved ports, fail if not
        for x in range(len(openports)):
            porttocheck = str(openports[x])
            if porttocheck in approvedports:
                sys.stdout.write(GREEN)
                print(f"The following port is approved: {porttocheck}, on ip: {ip}")
            else:
                sys.stdout.write(RED)
                print(f"The following port is NOT approved: {porttocheck}, on ip: {ip}")
                exitcode = 1
    else:
        sys.stdout.write(RED)
        print("file not found, provide an approved CSV file: "+approvedlistpath )
        exitcode = 1
    # check if any vulnerabilities have been found, fail if found
    if vulns != "[]":
        sys.stdout.write(RESET)
        print(f"Vulnerabilties have been found on ip: {ip}")
        for vuln in vulns:
            sys.stdout.write(RED)
            print(vuln)
        exitcode = 1

exit(exitcode)
```

## Appendix E – LAN outbound port script

This script is used to verify the open outbound ports using portquiz.net. Do note that the detected port range is set to the first 1024 ports and does not include all 65535 ports. In a production environment this can be set by changing the end parameter of the while function to 65536. When implementing this range, set the job timeout value to twenty-four hours.

https://github.com/CaptainBlackbutter/ContinuesSecurityConfigurationVerificationUsingPipelines

```
import sys
import csv
import os
import argparse
import requests
from colors import *

# define project arguments
parser = argparse.ArgumentParser()
parser.add_argument("--customer", help="Provide the customer name, as in the project structure")
args = parser.parse_args()
# set exit value
exitcode = 0
# define openports array
openports=[]
# detect open ports 1-65535
# This will take a long time!
porttoverify = 1
while porttoverify < 1024:
    query = "http://portquiz.net:"+str(porttoverify)
    # try to make the call, if it times out, it's not open.
    try:
        result = requests.get(query, timeout=0.2)
        openports.append(porttoverify)
    except:
        pass
    porttoverify = porttoverify + 1

# define approved list path
approvedlistpath = "./customers/"+args.customer+"/approved_outbound_ports.csv"
# define approvedports array
approvedports = []
# get approved open ports
with open(approvedlistpath, newline='') as csvfile:
                portreader = csv.reader(csvfile, delimiter=";")
                row = 0
                for port in portreader:
                    if (row != 0):
                            approvedports.append(port[0])
                    row = row + 1
# check for unapproved open ports
for openport in openports:
    if str(openport) in approvedports:
        sys.stdout.write(GREEN)
        print(approved open port: {openport}")
    else:
        sys.stdout.write(RED)
        print(f"UNAPPROVED open port: {openport}")
        exitcode=1

exit(exitcode)
```

## Appendix F – LAN firewall security services verification

This script is used to verify if the security services of the firewall are enabled and effective.

https://github.com/CaptainBlackbutter/ContinuesSecurityConfigurationVerificationUsingPipelines

```
import requests
import sys
from colors import *

# define exitcode
exitcode = 0
# define repo url
repourl = "pentest-malware-repo.westeurope.cloudapp.azure.com/repo"
# check if HTTP download is allowed
query_http = "http://"+repourl+"/unzipped/Ransomware.WannaCry/ed01ebfbc9eb5bbea545af4d01bf5f1071
661840480439c6e5babe8e080e41aa.exe"
# try to download the file on HTTP
try:
    # define content-size
    headeronly = requests.head(query_http)
    result = requests.get(query_http, timeout=5)
    if (result.headers.get('Content-Length') == headeronly.headers['Content-Length'] ):
        exitcode = 1
        sys.stdout.write(RED)
        print("The program was able to download the file using HTTP!")
        sys.stdout.write(RESET)
except:
    pass

query_https = "https://"+repourl+"/unzipped/Ransomware.WannaCry/ed01ebfbc9eb5bbea545af4d01bf5f10
71661840480439c6e5babe8e080e41aa.exe"
# try to download the file on HTTP
try:
    # define content-size
    headeronly = requests.head(query_https, verify=False)
    result = requests.get(query_https, timeout=5, verify=False)
    if (result.headers.get('Content-Length') == headeronly.headers['Content-Length'] ):
        exitcode = 1
        sys.stdout.write(RED)
        print("The program was able to download the file using HTTPS!")
        sys.stdout.write(RESET)
except:
    pass

exit(exitcode)
```

## Appendix G – AAD configuration verification script

This script uses different PowerShell modules to gather data from the Azure and M365 solutions. To authenticate against these services an API is used to gather the credentials from the HashiCorp Vault.

https://github.com/CaptainBlackbutter/ContinuesSecurityConfigurationVerificationUsingPipelines

```
# defining parameters
param ($JWT, $customer)
# import required modules
Import-Module WindowsCompatibility
Import-Module AzureAD -UseWindowsPowerShell
Import-Module ExchangeOnlineManagement
# Set exitcode
$exitcode = 0
# -----------------------------------------------------------------------------------------------
# Get credentials using Vault API and CI_JOB_JWT
# -----------------------------------------------------------------------------------------------
# Query Vault API for token
$postdata = '{\"role\": \"'+$customer+'_aad\", \"jwt\": \"' + $JWT + '\"}'
$vaultlogin = curl --request POST --data $postdata http://ucdp-ci-
vault.westeurope.cloudapp.azure.com:8200/v1/auth/jwt/login -s
$jsonlogin = ConvertFrom-Json $vaultlogin
$vaulttoken = $jsonlogin.auth.client_token
# Query vault API for secret
$secret = curl --header "X-Vault-Token: $vaulttoken" http://ucdp-ci-
vault.westeurope.cloudapp.azure.com:8200/v1/$customer/data/aad -s
$secretjson = ConvertFrom-Json $secret
# Set credential object
$secpassword = ConvertTo-SecureString $secretjson.data.data.aad_pw -AsPlainText -Force
$credObject = New-
Object System.Management.Automation.PSCredential ($secretjson.data.data.aad_upn, $secpassword)
# Connect to AzureAD
Connect-AzureAD -Credential $credObject
# -----------------------------------------------------------------------------------------------
# check if conditional access policies exist
# -----------------------------------------------------------------------------------------------
$CA_policies = Get-AzureADMSConditionalAccessPolicy
if ($CA_policies.Count -lt 1) {
    Write-Host "$([char]27)[91mNo Conditional access policies found.$([char]27)[0m"
    $exitcode = 1
}
else {
    Write-
Host "$([char]27)[92mConditional access policies found, no detailled processing.$([char]27)[0m"
    Write-Host $CA_policies
}
# -----------------------------------------------------------------------------------------------
# check number of global administrators
# -----------------------------------------------------------------------------------------------
$globaladminroleid = (Get-AzureADDirectoryRole | Where-Object {$_.DisplayName -
eq "Global Administrator"}).ObjectId
$globaladminmembers = Get-AzureADDirectoryRoleMember -ObjectId $globaladminroleid
if ($globaladminmembers.Count -ge 4) {
    Write-Host "$([char]27)[91mMore then three global administrators found.$([char]27)[0m"
   foreach ($member in $globaladminmembers){
        Write-Host $member.UserPrincipalName
    }
    $exitcode = 1
}
else {
    Write-Host "$([char]27)[32mLess or equal than 4 global administrators found.$([char]27)[0m"
    foreach ($member in $globaladminmembers){
        Write-Host $member.UserPrincipalName
```

```powershell
    }
}
# ----------------------------------------------------------------------------------------------
# Connect to Exchange Online
# ----------------------------------------------------------------------------------------------
Connect-ExchangeOnline -Credential $credObject  -ShowBanner:$false
# ----------------------------------------------------------------------------------------------
# Find mailboxes enable for basic authentication
# ----------------------------------------------------------------------------------------------
$basicAuthEnabled = Get-CASMailbox -Filter {ImapEnabled -eq "true" -or PopEnabled -eq "true" }
if ($basicAuthEnabled.Count -ge 0){
     Write-
Host "$([char]27)[91mMailboxes are still allowed to use basic authentication.$([char]27)[0m"
   foreach ($member in $basicAuthEnabled){
        Write-Host $member.Name
    }
    $exitcode = 1
}
Else {
    Write-Host "$([char]27)[32mNo mailboxes allow to use basic authentication.$([char]27)[0m"
}
# ----------------------------------------------------------------------------------------------
# Check if auditing is enabled (disabled by default)
# ----------------------------------------------------------------------------------------------
if (Get-AdminAuditLogConfig | FL UnifiedAuditLogIngestionEnabled){
     Write-Host "$([char]27)[91mOffice auditing is not enabled.$([char]27)[0m"
}
else {
    Write-Host "$([char]27)[32mOffice auditing is enabled.$([char]27)[0m"
}
# ----------------------------------------------------------------------------------------------
# exit the script with the required result
# ----------------------------------------------------------------------------------------------
exit $exitcode
```

## Appendix H – AD Configuration verification script

This script uses remote PowerShell to get the required verification from the registry.

https://github.com/CaptainBlackbutter/ContinuesSecurityConfigurationVerificationUsingPipelines

```
# defining parameters
param ($JWT, $customer, $referencehostIP)
# Set exitcode
$exitcode = 0
# ------------------------------------------------------------------------------------------------
# Get credentials using Vault API and CI_JOB_JWT
# ------------------------------------------------------------------------------------------------
# Query Vault API for token
$postdata = '{\"role\": \"'+$customer+'_ad\", \"jwt\": \"' + $JWT + '\"}'
$vaultlogin = curl --request POST --data $postdata http://ucdp-ci-
vault.westeurope.cloudapp.azure.com:8200/v1/auth/jwt/login -s
$jsonlogin = ConvertFrom-Json $vaultlogin
$vaulttoken = $jsonlogin.auth.client_token
# Query vault API for secret
$secret = curl --header "X-Vault-Token: $vaulttoken" http://ucdp-ci-
vault.westeurope.cloudapp.azure.com:8200/v1/$customer/data/ad -s
$secretjson = ConvertFrom-Json $secret
# Set credential object
$secpassword = ConvertTo-SecureString $secretjson.data.data.ad_pw -AsPlainText -Force
$credObject = New-
Object System.Management.Automation.PSCredential ($secretjson.data.data.ad_upn, $secpassword)
# ------------------------------------------------------------------------------------------------
# Create remote PowerShell session
# ------------------------------------------------------------------------------------------------
 $remote = New-PSSession -ComputerName $referencehostIP -Credential $credObject
# ------------------------------------------------------------------------------------------------
# Verify CIS 2.3.17.1 (L1) Ensure 'User Account Control: Admin Approval Mode for the Built-
in Administrator account' is set to 'Enabled' (Automated)
# ------------------------------------------------------------------------------------------------
 $result = Invoke-Command -Session $remote -ScriptBlock {
     $exitcode_inBlock = 0
     $value = Get-ItemPropertyValue -
Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System -
Name "FilterAdministratorToken"
     if ($value -eq 1)
     {
         Write-Host "$([char]27)[92mAdmin approval mode for the built-
in administrator account is enabled.$([char]27)[0m"
     }
     else
     {
        Write-Host "$([char]27)[91mAdmin approval mode for the built-
in administrator account is NOT enabled.$([char]27)[0m"
        $exitcode_inBlock = 1
     }
    return $exitcode_inBlock
}
if ($result -eq 1) {$exitcode = 1}
# ------------------------------------------------------------------------------------------------
# Verify CIS 18.3.2 (L1) Ensure 'Configure SMB v1 client driver' is set to 'Enabled: Disable dri
ver (recommended)' (Automated)
# ------------------------------------------------------------------------------------------------
 $result = Invoke-Command -Session $remote -ScriptBlock {
     $exitcode_inBlock = 0
     $value = 0
     $ErrorActionPreference = "stop"
     try {

         $value = Get-ItemPropertyValue -Path HKLM:\SYSTEM\CurrentControlSet\Services\MrxSmb10 -
Name "Start"
```

```
     }
     catch{
         $value = 0
     }
     if ($value -eq 4)
     {
         Write-Host "$([char]27)[92SMBv1 client is disabled.$([char]27)[0m"
     }
     else
     {
        Write-
Host "$([char]27)[91mSMBv1 client is enabled or not explicitly disabled.$([char]27)[0m"
        $exitcode_inBlock = 1
     }
     return $exitcode_inBlock
}
if ($result -eq 1) {$exitcode = 1}
exit $exitcode
```