A picture containing icon

Description automatically generated

Tech Note:

LoadMaster STIG Script

Version 3.0 20230906

This note describes how to use the STIG script to configure a Kemp LoadMaster for use on U.S. Government networks.

Copyright © 2002-2023 Progress. All rights reserved.

.

Table of Contents

[1. Introduction 2](#_Toc147141331)

[2. Background 2](#_Toc147141332)

[3. Running the Kemp STIG Script 2](#_Toc147141333)

[4. Content of Kemp\_LoadMaster\_STIG\_V3.20230906.ps1 5](#_Toc147141334)

[5. Manually Configuring LoadMaster to meet US Gov Security Requirements 14](#_Toc147141335)

1. Introduction

This document provides information on how to apply U.S. Government STIGs and SRGs to the Kemp LoadMaster appliance. This document assumes the user has installed and licensed one or more Kemp LoadMaster appliances and has access to an administrative account for the LoadMaster.

The major change in this script is removal of the requirement to use the Kemp PowerShell Module. All commands are sent natively to the LoadMaster API interface. This script has been tested on PowerShell v5.1 and v7.3 on Windows and Linux workstations.

For information on this script, contact mike.bomba@progress.com.

1. Background

The U.S. Government publishes Security Technical Implementation Guides (STIG) and Security Recommendations and Guidelines (SRG) for equipment that connects to its networks. These guidelines cover a broad range of topics ranging from how users should authenticate the networked device to how the device should be configured to minimize its risk of compromise. Kemp LoadMaster includes a REST API and Progress offers a PowerShell wrapper that can be used to access the API to configure the appliance. The Progress Federal team has developed a PowerShell script that automates the process to harden the LoadMaster and to enable certificate login that supports both Common Access Card (CAC) or Personal Identification Verification (PIV) smartcards.

1. Running the Kemp STIG Script

The Kemp STIG script is intended to be edited before it is run. The settings section defines what portions of the script run using a series of Boolean operators (e.g. $doGOE). If these operators are set to $True, then the associated settings for that operation must be configured in the settings section of the script (e.g. $EnableGEO).

The working directory this script is set at the beginning of the configuration parameters section. Please adjust this to a location you have read/write access and where you will place certificate files (as needed). TLS certificates need to be stored in password protected PEM format. Intermediate certificates need to be stored in Base64 format.

You may enter the LoadMaster admin account into the settings section of the script or if set to “”, you will be prompted for admin account info (name and password) for an account on your LoadMaster appliances that have “All Permissions” level access.

To run this script against a list of LoadMaster appliances, you can enter a comma-separated list into the parameters section. If left blank, the script will prompt for a list of IP addresses representing the management (API) interface on the LoadMaster(s).

This script will (if configured in the parameters section) install a TLS management certificate as well as up to 4 intermediate certificates. All certificates need to be in PEM format and the TLS certificate needs to be a password protected PEM file.

This script will (if configured in the parameters section) create additional admin users and it supports both the creation of certificate-login based admin accounts as well as name-password based admin accounts.

Many of the settings in this script can be left blank and the script will either prompt you for the value or skip a section in configuration. For example, if the admin user password is left blank in the parameters file, the script will prompt you for the password. If variables associated with LDAP are left blank, the script will skip creation of an LDAP service.

This script will check connectivity (IP+Port) for the list of LoadMaster IP addresses and if it cannot connect to a specific IP address, it will skip processing for that IP address.

The script has the ability to:

* Enable session management
* Disable basic authentication
* Disable ssh access
* Enable US Gov Pre-Authentication Warning Banner for Web User Interface
* Enable US Gov Pre-Authentication Warning Banner for Console
* Enable an Out-Of-Band Management interface
* Install up to 4 Intermediate certificates (root and issuing CA certificates)
* Install and assign one TLS management certificate
* Configure Network Time Services (to include NTPv3 if required)
* Configure the LoadMaster to connect to an external LDAP(S) authentication server
* Configure a restricted FIPS algorithm (FIPS2) set (removing 3DES and SHA1)
* Assign FIPS2 to management interfaces
* Create a certificate-based local admin account
* Create a name-password local admin account
* Disable GEO load balancing (disable if service not used)
* Force Kerberos to only use AES256
* Prompt for a list of LoadMaster IP addresses to configure
* Automatically assign hostnames based on a defined filter and last octet of IP address

This script allows you to preset the admin identities for ldap-user, cert-user and password-user as well as associated passwords. To do so, edit the parameters section before running it and add this information to the associated cells.

Once you have configured the script, run the script and it will cycle thru all the IP addresses listed in the parameters section (or manually entered) and apply all associated security settings.

Some LoadMaster license types (e.g. Free, MELA, Pooled, SPLA) may generate an error related to disabling tethering to Progress. This is normal as these license types require tethering to the Progress licensing server. Cloud instances will also generate an error as these instances must report usage to the cloud management service for billing purposes. BYOL (permanent) licensed LoadMaster appliances (physical/virtual/cloud) should not generate an error, as these instances do not require tethering and the STIG script should successfully disable all call home functionality on these license types.

The Script will generate a log file that shows the same information as is presented on the screen when the script is ran.

Note on FIPS. Recent guidance changes from the US Government have deprecated algorithms originally approved as part of the FIPS 140-2 standard. The algorithms deprecated include:

3DES

SHA1

DH with keys less than 2048

To accommodate these changes, this script creates a custom cypher set called FIPS2 that starts with the FIPS cypher set and deletes these deprecated algorithms. This script then assigns the FIPS2 cypher set to the Web User Interface (management interface) for inbound TLS requests and to the LoadMaster for outbound TLS requests.

*If certificate login fails, you can bypass the certificate login process and gain access the userid/password login. Simply close the window showing available certificates to use and you will then be presented with the warning banner. After accepting the banner, you will be presented with the userid/password login option.*

*If automatic assignment of the LoadMaster management interface TLS certificate results in the inability to connect to the LoadMaster web user interface, you can use the console interface to sign in (using the “bal” account) and you can reset the web user interface. This process involved regenerating the self-signed management certificate and resetting the web user interface.*

1. Content of Kemp\_LoadMaster\_STIG\_V3.20230906.ps1

# THIS SCRIPT APPLIES SECURITY SETTINGS TO LOADMASTER APPLIANCES LMOS 7.2.54 AND NEWER

# CRITICAL NOTE: PLEASE EDIT CONFIGURATION PARAMETERS BEFORE RUNNING SCRIPT

# DESIGNED TO RUN ON WINDOWS 11 POWERSHELL 5.1 AND LINUX POWERSHELL 7.2

# CHANGE LOG

# Compensated for dirty response to get-param admincert

# Compensated for dirty response to get-param localcert

# Created custom cipher set FIPS2 to match our revised FIPS algorithm set

# Applied FIPS2 cipher set to management traffic

# Added necessary logic to support appliances set to Software FIPS Mode

###################################

# Cleanup and prepare to run script

###################################

clear-history

clear-host

Remove-Variable \* -ErrorAction SilentlyContinue

[string]$ScriptVersion = "v3.20230906"

#######################################################################

############### START OF CONFIGURATION PARAMETERS SECTION #############

[bool]$doFIPS2 = $True # ENTER $True to Create restricted FIPS cipher set removing weak ciphers ($False to skip)

[bool]$doGEO = $True # ENTER $True to Process GEO settings according to $EnableGEO value ($False to skip GEO setting)

[bool]$doPassUser = $True # ENTER $True to Create new admin account based on username and password ($False to skip)

[bool]$doCertUser = $True # ENTER $True to Create new admin account based on certificate principal name ($False to skip)

[bool]$doPEM = $True # ENTER $True to Install management certificate (.pem file) ($False to skip)

[bool]$doAdminCert = $True # ENTER $True to Assign management certificate to WUI interface ($False to skip)

[bool]$doCer = $True # ENTER $True to Install intermediate certificates (base64 .cer file) ($False to skip)

[bool]$doHostName = $True # ENTER $True to assign a hostname based on $hostbase + $IP ($False to skip)

[bool]$doNTP = $True # ENTER $True to configure LoadMaster for NTP ($False to skip)

[bool]$doNTPv3 = $False # ENTER $True to if NTP service should use NTPv3 ($False to skip)

[bool]$doLdap = $True # ENTER $True to configure an LDAP service connection ($False to skip)

[bool]$doLdapGroup = $True # ENTER $True to configure an LDAP management group ($False to skip)

[bool]$doWarningBanners = $True # ENTER $True to configure WUI and Console warning banners ($False to skip)

[bool]$EnableGEO = $False # Enter $True to enable GEO, $False to disable GEO Load Balancing service (disable port 53 listener)

[string]$hostbase = 'VLM' # ENTER a name to create hostname prefix ('' will skip process)

[string]$NewCertUser = 'Mike@KEMPTECH.BIZ' # ENTER a name to create a certificate login admin user, CASE SENSITIVE (typically certificate Principal Name)

[string]$NewPassUser = 'adminmike' # ENTER a name to create a name/password login admin user ('' will skip process)

[string]$NewPassUserPass = 'Kemp1fourall' # ENTER a password for new name/password admin user ('' will skip process)

[string]$NewUserGroup = 'Kemp' # ENTER a name to LDAP admin group ('' will skip process)

[string]$NewUserRights = 'root' # ENTER rights for new user or group objects (root - all rights) ('' will skip process)

[string]$sor = 'yes' # ENTER 'yes' to enable Subnet Originating Request appliance wide ('' will skip process)

[string]$ntpv3\_keynum = '' # ENTER NTPv3 Key Number ('' will skip process)

[string]$ntpv3\_keytype = '' # ENTER NTPv3 Key Type ('' will skip process)

[string]$ntpv3\_secret = '' # ENTER NTPv3 Key Secret ('' will skip process)

[string]$ntp\_host = 'time.nist.gov' # ENTER NTP Server name or IP address ('' will skip process)

[string]$ldap\_Name = 'KEMPTECH.BIZ' # ENTER LDAP Service name or IP address ('' will skip process)

[string]$ldap\_Server = '10.0.0.10' # ENTER LDAP Server name or IP address ('' will skip process)

[string]$ldap\_User = 'ldapuser' # ENTER LDAP Service Account Name ('' will skip process)

[string]$ldap\_UserPass = 'Kemp1fourall' # ENTER LDAP Service Account Password ('' will skip process)

[int]$ldap\_Type = 0 # ENTER LDAP Service Protocol (0 = LDAP, 1 = STARTTLS, 2 = LDAPS, '' will skip process)

[string]$WUI\_IdleTime = '600' # ENTER WUI Session Idle Logout Timer in seconds ('' will skip process)

[string]$WUI\_FailedLogins = '5' # ENTER WUI Session Max Failed Logins ('' will skip process)

[string]$WUI\_MaxLogins = '3' # ENTER WUI Session Max Concurrent Logins ('0' will not set a limit)

[string]$certpath = '.\certs\wc.pem' # ENTER filename and path for TLS Management Certificate (PEM format only) ('' will skip process)

[string]$certpass = 'password' # ENTER Password to install encrypted private key in TLS Management Certificate"

[string]$certname = 'wc' # ENTER friendly name for TLS Management Certificate"

[string]$ica1 = '.\certs\ECC.cer' # ENTER filename and path for Intermediate Certificate File (PEM format only)

[string]$ica2 = '.\certs\RSA.cer' # ENTER filename and path for Intermediate Certificate File (PEM format only) (Enter ‘none’ instead of ‘’ as needed)

[string]$ica3 = 'none' # ENTER filename and path for Intermediate Certificate File (PEM format only) (Enter ‘none’ instead of ‘’ as needed)

[string]$ica4 = 'none' # ENTER filename and path for Intermediate Certificate File (PEM format only) (Enter ‘none’ instead of ‘’ as needed)

[string]$workingDir = "~\loadmaster" # If necessary, set this to a different location you have read/write rights

#

[string]$ConsoleMsg = "USG Warning Banner - YOU ARE ACCESSING A US GOVERNMENT (USG) INFORMATION SYSTEM PROVIDED FOR AUTHORIZED USE ONLY. Communications using, or data stored on this Information System are not private, are subject to routine monitoring, interception, and search, and may be disclosed or used for any authorized purpose. USG logs all access to this system. Upon proper legal request, these logs and any other related data will be released for personnel misconduct and law enforcement purposes. Unauthorized use of this system will be prosecuted. By continuing to login, you agree to the above terms and conditions."

#

[string]$WUIMsg = "<!DOCTYPE html><html><head><title>USG Warning Banner</title></head><style>p {color: black;}.title1 {font-size: 13px;text-align: center;}.paragraph1 {font-size: 12px; text-align: left;}</style><h1>USG WARNING AND CONSENT BANNER</h1><hr><p class=title1>YOU ARE ACCESSING A UNITED STATES GOVERNMENT (USG) INFORMATION SYSTEM<br>PROVIDED FOR AUTHORIZED USE ONLY</p><br><p class=paragraph1>By using this Information System (which includes any device attached to this Information System), you consent to the following conditions:<br><br>- USG routinely intercepts and monitors communications on this Information System for purposes including, but not limited to, penetration testing, network operations and defense, and upon proper legal request for personnel misconduct and law enforcement purposes.<br><br>- At any time, the USG may inspect and seize data stored on this Information System. Communications using, or data stored on this Information System are not private, are subject to routine monitoring, interception, and search, and may be disclosed or used for any authorized purpose. This Information System includes security measures (e.g., authentication and access controls) to protect USG interests--not for your personal benefit or privacy.<br><br> - Notwithstanding the above, using this Information System does not constitute consent to legal or criminal investigative searching or monitoring of the content of privileged communications, or work product, related to personal representation or services by attorneys, psychotherapists, or clergy, and their assistants. Such communications and work product are private and confidential. <br><br> - USG requires all USG personnel to complete Information System Security training annually.<br><br>- By continuing to login, you agree to the above terms and conditions.</p></body></html>"

#

[string]$FIPS2 = "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:DHE-DSS-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA256:AES256-GCM-SHA384:AES256-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:DHE-DSS-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-SHA256:DHE-DSS-AES128-SHA256:AES128-GCM-SHA256:AES128-SHA256"

#

##### OPTIONAL PARAMETERS - IF ENTERED HERE WILL NOT BE PROMPTED FOR LATER #####

[array]$iplist = '10.0.0.36' # If set to '', you will be prompted for a list of LoadMaster IP addresses

[string]$port = '443' # If set to '', you will be prompted for the LoadMaster API port number

[string]$lmadmin = 'bal' # if set to '', you will be prompted for a LoadMaster amin account

[string]$lmadminpass = 'Kemp1fourall' # if set to '', you will be prompted for the LoadMaster admin account password

############## END OF CONFIGURATION SECTION #################

#############################################################

[string]$time = (Get-Date -Format u)

################################

# Change directory to workingDir

################################

Set-Location -Path $workingDir

#################

# DEFINE LOG FILE

#################

$logfile = "$workingdir\LoadMaster\_STIG.log"

if (test-path -Path $logfile -PathType leaf) {

Get-Date >> $logfile

$msg >> $logfile

"LoadMaster STIG Script $scriptversion log file`n" >> $logfile

}

Else {

Get-Date > $logfile

$msg >> $logfile

"LoadMaster STIG Script $scriptversion log file`n" >> $logfile

}

###########################################

# Populate $certlist for intermediate certs

###########################################

[array]$certlist = $Null

if (($ica1.length -gt 0) -and ($ica1 -ne "none") -and (test-path -path $ica1 -PathType Leaf)) {$certlist += $ica1}

if (($ica2.length -gt 0) -and ($ica2 -ne "none") -and (test-path -path $ica2 -PathType Leaf)) {$certlist += $ica2}

if (($ica3.length -gt 0) -and ($ica3 -ne "none") -and (test-path -path $ica3 -PathType Leaf)) {$certlist += $ica3}

if (($ica4.length -gt 0) -and ($ica4 -ne "none") -and (test-path -path $ica4 -PathType Leaf)) {$certlist += $ica4}

if ($certlist.length -eq 0) {$doCER = $False; $doCertUser = $False}

########################

# Provide info on script

########################

$ps = (get-host).version

$os = [System.Environment]::OSVersion.Platform

[bool]$ps7 = ($ps.major -ge 7)

[bool]$w32 = ($os -match "Win32NT")

$msg = "SCRIPT TO APPLY STIG SETTINGS TO LOADMASTER"; Write-Host -Fore cyan $msg; $msg >> $logfile

$msg = "$time INFO - Workstation Operating System ($os)"; $msg; $msg >> $logfile

$msg = "$time INFO - Workstation PowerShell Version ($ps)"; $msg; $msg >> $logfile

##############################################################################

# ENABLE WEB CONNECTIONS TO IGNORE CERTIFICATE ERRORS AND USE TLS1.2 OR NEWER

##############################################################################

if (Test-Path -Path "~\.curlrc" -pathtype Leaf) {Copy-Item -Path "~/.curlrc" "~/.oldcurlrc" -Force}

echo "--insecure" > ~/.curlrc

echo "--tls1.2" >> ~/.curlrc

$msg = "$time INFO - Created .curlrc file with --insecure and --tls1.2 params"; $msg; $msg >> $logfile

if ($ps7) {$msg = "$time INFO - Added options for Invoke-Webrequest for PowerSHell v7"; $msg; $msg >> $logfile}

Else {$msg = "$time INFO - Added options for Invoke-Webrequest for PowerSHell v5"; $msg; $msg >> $logfile}

if ($w32 -and $ps7) {

[System.Net.ServicePointManager]::ServerCertificateValidationCallback = { $True }

[System.Net.ServicePointManager]::SecurityProtocol = "tls12, tls11"

$msg = "$time INFO - Set Windows PowerShell 7 to ignore Cert Valication Errors"; $msg; $msg >> $logfile

}

if ($w32 -and -not $ps7) {

if (([System.Net.ServicePointManager]::CertificatePolicy).ToString() -ne "TrustAllCertsPolicy") {

add-type @"

using System.Net;

using System.Security.Cryptography.X509Certificates;

public class TrustAllCertsPolicy : ICertificatePolicy {

public bool CheckValidationResult(

ServicePoint srvPoint, X509Certificate certificate,

WebRequest request, int certificateProblem) {

return true;

}

}

"@

[System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy

}

[Net.ServicePointManager]::SecurityProtocol = "tls12, tls11"

$msg = "$time INFO - Set Windows PowerShell 5 to ignore Cert Validation Errors"; $msg; $msg >> $logfile

}

############################

# STANDARD GLOBAL PARAMETERS

############################

$pattern1 = '^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$'

$pattern2 = '^(\d{1,4})$'

$wait = 250 ## Recommend you leave this value as is.

######################################################################

# CHECK FOR REQUIRED PARAMETERS FOR LOGIN - PROMPT FOR VALUE AS NEEDED

######################################################################

# LOADMASTER ADMIN ACCOUNT

if ($lmadmin.length -eq 0) {

do {

$lmAdmin = Read-Host -Prompt "Enter LoadMaster Amin Account (e.g. bal)"

} while ($lmAdmin.length -lt 1)

}

# LOADMASTER ADMIN ACCOUNT PASSWORD

if ($lmadminpass.Length -eq 0) {

do {

"Enter password for $lmadmin in secure popup box"

$Pass = Read-Host "ENTER PASSWORD for $lmadmin" -AsSecureString

$Pass = [Runtime.InteropServices.Marshal]::SecureStringToBSTR($Pass)

$lmAdminPass = [Runtime.InteropServices.Marshal]::PtrToStringBSTR($Pass)

} while ($lmAdminPass.length -lt 8)

}

# LOADMASTER API INTERFACE PORT

if ($port.length -eq 0) {do {$port = read-host -Prompt "Enter LoadMaster WUI Port (e.g. 443 or 8443)"} while ($port.length -lt 1) }

# LOADMASTER API INTERFACE IP ADDRESS LIST

if (($iplist.count -eq 0) -or ($iplist[0].length -lt 5)) {

do {

$ip = read-host -Prompt "Enter all LoadMaster WUI IP Addresses (blank line to finish)"

if ($ip -match $pattern1) {$iplist += $ip}

ElseIF ($ip -ne "") {"ERROR IN IP ADDRESS FORMAT"}

} while ($ip -ne '')

$iplist = $iplist.Where({ $\_ -ne "" })

}

"################ START ##############" >> $logfile

$msg = "$time INFO - Working directory ($workingdir)"; $msg; $msg >> $logfile

###############################################

# Function to Ping a Port with variable timeout

###############################################

function Test-Port ($ip, $port, $Timeout) {

[bool]$ps7 = ((get-host).version.major -ge 7)

if ($ip.length -eq 0) {$ip = $lmip}

if ($port.length -eq 0) {$port = $lmport}

if ($Timeout.length -le 0) {$Timeout = '1000'}

if ($port.length -le 0) {$port = '443'}

if ($ps7) {test-connection -computername $ip -TcpPort $port}

else {

$tcpClient = New-Object System.Net.Sockets.TcpClient

$tcpClient.ConnectAsync($ip, $port).Wait($Timeout)

}

}

##################################

# Function to Enable API Interface

##################################

function Enable-LMapi ($ip, $port, $APIUser, $APIPass) {

[bool]$ps7 = ((get-host).version.major -ge 7)

if ($ip.length -eq 0) {$ip = $lmip}

if ($port.length -eq 0) {$port = $lmport}

if ($APIUser.Length -eq 0) {$APIUser = $lmadmin}

if ($APIPass.Length -eq 0) {$APIPass = $lmadminpass}

$ipport = $ip+":"+$port

$url = "https://$ipport/accessv2"

$cmd = 'set'

$param = 'enableapi'

$value = 'yes'

$body = '{"apiuser":"'+$APIUser+'","apipass":"'+$APIPass+'","cmd":"'+$cmd+'","param":"'+$param+'","value":"'+$value+'"}'

if ($ps7) {$rc = try {$r = (Invoke-WebRequest -SkipCertificateCheck -Method post -uri $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}}

else {$rc = try {$r = (Invoke-WebRequest -Method post -uri $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}}

if ($r.code -match '200') {$msg = "$time INFO - LoadMaster API Interface is Enabled"; $msg; $msg >> $logfile }

}

###################################

# DEFINE FUNCTION TO COLLECT APIKEY

###################################

# APIKEY can only be generated AFTER LoadMaster is licensed

function get-apikey ($ip, $port, $Credential) {

if ($ip.length -eq 0) {$ip = $lmip}

if ($port.length -eq 0) {$port = $lmport}

if ($Credential.lengh -eq 0) {$Credential = $creds}

$ipport = $ip+":"+$port

[bool]$ps7 = ((get-host).version.major -ge 7)

[string]$url = $Null; $url = "https://$ipport/access/addapikey"

if ($ps7) {$rc = try {$r = Invoke-WebRequest -SkipCertificateCheck -Method Get -Uri $url -Credential $Credential} catch {'{"code":"422"}' | ConvertFrom-Json}}

Else {$rc = try {$r = Invoke-WebRequest -Method Get -Uri $url -Credential $Credential} catch {'{"code":"422"}' | ConvertFrom-Json}}

$res = $r.RawContent.tostring()

$start = $res.indexof('<key>') + 5

$end = $res.indexof('</key>')

$apikey = $res.substring($start, ($end - $start))

$apikey

}

################

###################################################################

# CREATE SHORTCUT TO POST API COMMANDS (APIv2) SUPPORTING PS7 & PS5

###################################################################

FUNCTION post-command ($url, $body) {

[System.Uri]$u = $url

# Determine if running on PowerShell 7 or newer

[bool]$ps7 = ((get-host).version.major -ge 7)

if ($ps7) {$output = $Null; $output = Invoke-WebRequest -SkipCertificateCheck -Method post -uri $u -Body $body}

else {$output = $Null; $output = Invoke-WebRequest -Method post -uri $u -Body $body}

$output

start-sleep -Milliseconds 250

}

###################################################################

##########################################################

# CREATE FUNCTION TO SET PARAMETERS USING DIRECT API CALLS

##########################################################

FUNCTION set-param ($ip, $port, $param, $value, $APIKey, $APIUser, $APIPass) {

# Determine if running on PowerShell 7 or newer

[bool]$ps7 = ((get-host).version.major -ge 7)

if ($ip.length -eq 0) {$ip = $lmip}

if ($port.length -eq 0) {$port = $lmport}

if ($APIKey.length -eq 0) {$APIKey = $lmAPIKey}

if ($APIUser.Length -eq 0) {$APIUser = $lmadmin}

if ($APIPass.Length -eq 0) {$APIPass = $lmadminpass}

if ($value.Length -gt 60) {$val = $value.Substring(0,60)} Else {$val = $value}

$ipport = $ip+":"+$port

$url = "https://$ipport/accessv2"

$cmd = 'get'

if ($apikey.length -gt 1) {$body = '{"apikey":"'+$apikey+'","cmd":"'+$cmd+'","param":"'+$param+'"}'}

else {$body = '{"apiuser":"'+$APIUser+'","apipass":"'+$APIPass+'","cmd":"'+$cmd+'","param":"'+$param+'"}'}

if ($ps7) {$asis = $Null; $asis = (Invoke-WebRequest -SkipCertificateCheck -Method post -uri $url -Body $body | convertfrom-json).$param}

else {$asis = $Null; $asis = (Invoke-WebRequest -Method post -uri $url -Body $body | convertfrom-json).$param}

if ($asis -eq $value) {$msg = "$time SKIP - $param already set to $val"; $msg; $msg >> $logfile}

else {

$url = "https://$ipport/accessv2"

$cmd = 'set'

if ($apikey.length -gt 1) {$body = '{"apikey":"'+$apikey+'","cmd":"'+$cmd+'","param":"'+$param+'","value":"'+$value+'"}'}

else {$body = '{"apiuser":"'+$apiuser+'","apipass":"'+$apipass+'","cmd":"'+$cmd+'","param":"'+$param+'","value":"'+$value+'"}'}

if ($ps7) {$rc = $Null; $rc = (Invoke-WebRequest -SkipCertificateCheck -Method post -uri $url -Body $body | convertfrom-json).code}

else {$rc = $Null; $rc = (Invoke-WebRequest -Method post -uri $url -Body $body | convertfrom-json).code}

if ($rc -match '200') {$msg = "$time SET - $param set to $val"; $msg; $msg >> $logfile }

else {$msg = "$time ERR - Cannot set $param to $val"; $msg; $msg >> $logfile}

}

start-sleep -Milliseconds 250

}

##########################################################

##########################################################

# CREATE FUNCTION TO GET PARAMETERS USING DIRECT API CALLS

##########################################################

function Get-Param ($ip, $port, $param, $APIKEY) {

# Determine if running on PowerShell 7 or newer

[bool]$ps7 = ((get-host).version.major -ge 7)

if ($ip.length -eq 0) {$ip = $lmip}

if ($port.length -eq 0) {$port = $lmport}

if ($APIKey.length -eq 0) {$APIKey = $lmAPIKey}

$ipport = $ip+":"+$port

$url = "https://$ipport/accessv2"

$cmd = 'get'

$body = '{"apikey":"'+$apikey+'","cmd":"'+$cmd+'","param":"'+$param+'"}'

if ($ps7) {$asis = $Null; $asis = (Invoke-WebRequest -SkipCertificateCheck -Method post -uri $url -Body $body | convertfrom-json).$param}

else {$asis = $Null; $asis = (Invoke-WebRequest -Method post -uri $url -Body $body | convertfrom-json).$param}

$asis

}

##########################################################

# Run sanity checks and reset do##### parameters as needed

##########################################################

if ($certpass.length -eq 0) {$doPEM = $False}

if ($certname.length -eq 0) {$doPEM = $False}

if ($certpath.split(".")[-1] -notmatch "pem") {$doPEM = $False}

if ($NewCertUser.length -eq 0) {$doCertUser = $False}

if ($NewPassUser.length -eq 0) {$doPassUser = $False}

if ($NewPassUserPass.length -eq 0) {$doPassUser = $False}

if ($NewUserGroup.length -eq 0) {$doLdapGroup = $False}

if ($NewUserRights.length -eq 0) {$NewUserRights = 'readonly'} # need to verify this

if ($ldap\_Name.length -eq 0) {$doLdap = $False}

if ($ldap\_Server.length -eq 0) {$doLdap = $False}

if ($ldap\_User.length -eq 0) {$doLdap = $False}

if ($LDAP\_UserPass.length -eq 0) {$doLdap = $False}

if ($ldap\_Type.length -eq 0) {$ldap\_Type = '0'}

if ($ntpv3\_keynum.length -eq 0) {$doNTPv3 = $False}

if ($ntpv3\_keytype.length -eq 0) {$doNTPv3 = $False}

if ($ntpv3\_secret.length -eq 0) {$doNTPv3 = $False}

if ($ntp\_host.length -eq 0) {$doNTP = $doNTPv3 = $False}

if ($hostbase.length -eq 0) {$dohostname = $False}

if ($FIPS2.length -eq 0) {$doFIPS2 = $False}

################################

# Process all entries in $iplist

################################

$ip = $Null

foreach ($ip in $iplist) {

[string]$lmip = $ip

[string]$lmport = $port

[string]$ipport = $ip + ":" + $Port

[string]$hostname = $hostbase + '-' + $ip.split('.')[-1]

$msg = "STARTING CONFIGURATION $hostname at $ipport"; Write-Host -fore cyan $msg; $msg >> $logfile

############################################

# Test connectivity and skip entry as needed

############################################

if ((test-port)) {

$msg = "$time INFO - PASSED CONNECTION TEST TO $ipport"; $msg; $msg >> $logfile

}

Else {$msg = "$time SKIP - SKIPPING PROCESSING FOR $IP - CANNOT CONNECT"; $msg; $msg >> $logfile ; continue}

#################################

# ENSURE API INTERFACE IS ENABLED

#################################

enable-LMapi -ip $ip -port $port -APIUser $lmadmin -APIPass $lmadminpass

#########################

# Build credential object

#########################

$creds = $Null; $creds = New-Object pscredential("$lmadmin", (ConvertTo-SecureString -String "$lmadminpass" -AsPlainText -Force))

$apikey = $Null; $lmAPIKey = $apikey = get-apikey -ip $ip -port $port -Credential $creds

[string]$url = $Null; [string]$url = "https://$ipport/accessv2"

##############################

# Collect data from LoadMaster

##############################

# Collect a list of all available api commands

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"listapi"}'

[array]$allapi = $Null; [array]$allapi = (post-command -url $url -Body $body | convertfrom-json).commands

# Collect a list of all available license info

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"licenseinfo"}'

[array]$licenseinfo = $Null; [array]$licenseinfo = (post-command -url $url -Body $body | convertfrom-json)

# Collect a list of all available parameters and values

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"getall"}'

[array]$allparams = $Null; [array]$allparams = (post-command -url $url -Body $body | convertfrom-json)

# Test for FIPS Mode active

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"set","param":"WUICipherset","value":"FIPS"}'

$FIPSModeTest = $Null; $FIPSModeTest = try {(post-command -url $url -Body $body).statuscode} catch {$\_}; $FIPSModeTest = $FIPSModeTest.tostring()

$FIPSMode=$Null; $FIPSMode = $FIPSModeTest.contains("protocol violation")

if ($FIPSMode) {$doFIPS2 = $False; $msg = "$time INFO - LoadMaster is running in Software FIPS Mode. Ciphersets will be set to Default."; write-host -fore Yellow $msg; $msg >> $logfile}

###############################################

# VALIDATE INFO NEEDED FOR OPTIONS IS AVAILABLE

###############################################

#######################################################

# Check for minimum LoadMaster Operating System version

#######################################################

[string]$LM\_R = "54"

[string]$LM\_V = ($allparams.version).Split(".")[2]

if ($LM\_V -ge $LM\_R) {$msg = "$time INFO - LoadMaster LMOS equal or greater than 7.2.$LM\_R. Validation PASSED"; $msg; $msg >> $logfile }

else {$msg = "$time SKIP - LoadMaster LMOS less than 7.2.$LM\_R. Validation FAILED."; $msg; $msg >> $logfile; return }

start-sleep -Milliseconds 250

###################

# ASSIGN Cipher Set

###################

# Determine cipher set to assign

if (($LM\_V -lt 59) -and ($doFIPS2)) {[string]$cipher = 'FIPS2'}

if (($LM\_V -ge 59) -and ($doFIPS2)) {[string]$cipher = 'FIPS'; $doFIPS2 = $False}

if (-not $doFIPS2) {[string]$cipher = 'Default'}

# Test to see if FIPS2 cipher set already exists

if ($doFIPS2) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"getcipherset","name":"'+$cipher+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -body $body)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.StatusCode -match '200') {$doFIPS2 = $False; $msg = "$time SKIP - Cipherset $cipher already exists"; $msg; $msg >> $logfile }

else {$doFIPS2 = $True}

}

# Create FIPS2 cipher set

if ($doFIPS2) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"modifycipherset","name":"'+$cipher+'","value":"'+$FIPS2+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -body $body)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.statuscode -eq 200) {$msg = "$time SET - $Cipher CIPHER SET CREATED TO REMOVE WEAK FIPS ALGORITHMS"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error creating new cipher set ($Cipher)"; $msg; $msg >> $logfile}

}

start-sleep -Milliseconds 250

# SETTING OUTBOUND CIPHER

set-Param -ip $ip -port $port -APIKEY $apikey -Param 'OutboundCipherset' -Value $cipher

# SETTING INBOUND CIPHER

Set-Param -ip $ip -port $port -APIKEY $apikey -Param WUICipherset -value $cipher

start-sleep -Milliseconds 500

# Set LoadMaster hostname

if ($doHostName) {set-param -ip $ip -port $port -apikey $apikey -param "hostname" -value $hostname }

# Enable WUI Login Session Management

set-param -ip $ip -port $port -apikey $apikey -Param sessioncontrol -value "yes"

# Disable WUI Login Basic Authentication

set-param -ip $ip -port $port -apikey $apikey -Param sessionbasicauth -value $False

start-sleep -Milliseconds 250

# Set WUI Session Idle TIme

if ($WUI\_IdleTime.length -ne 0) {set-param -ip $ip -port $port -apikey $apikey -param sessionidletime -value $WUI\_IdleTime}

# Set WUI Max Failed Logins before account lock

if ($WUI\_FailedLogins.length -ne 0) {set-param -ip $ip -port $port -apikey $apikey -param sessionmaxfailattempts -value $WUI\_FailedLogins}

# Set admin interface max concurrent logins (0 - unlimited)

if ($WUI\_MaxLogins -ne '') {set-param -ip $ip -port $port -apikey $apikey -param sessionconcurrent -value $WUI\_MaxLogins}

start-sleep -Milliseconds 250

# Set USG WARNING BANNER FOR WUI Interface

if ($doWarningBanners -and ($WUIMsg.Length -gt 0)) {set-param -ip $ip -port $port -apikey $apikey -param 'WUIPreauth' -value "$WUIMsg" }

# Set USG WARNING BANNER FOR Console Interface

if ($doWarningBanners -and ($ConsoleMsg.length -gt 0)) {set-param -ip $ip -port $port -apikey $apikey -param 'SSHPreAuth' -value "$ConsoleMsg" }

# Disable Call Home

if ($licenseinfo.MandatoryTether -ne 'yes') {set-param -ip $ip -port $port -apikey $apikey -Param Tethering -Value 0}

start-sleep -Milliseconds 250

# Allows loadbalancing of servers located on a different subnet from the load balancer

set-param -ip $ip -port $port -apikey $apikey -Param nonlocalrs -Value 'yes'

# Required when LoadMaster is connected to 2 or more networks, optional when connected to 1 network

if ($sor -ne '') {set-param -ip $ip -port $port -apikey $apikey -Param subnetorigin -Value $sor}

# Disable SSL Renegotiation

$isit=$Null; $isit = get-param -ip $ip -port $port -APIKey $apikey -param "sslrenegotiate"

if (-not $isit ) {$msg = "$time SKIP - SSL renegotiation already disabled"; $msg; $msg >> $logfile}

else {set-param -ip $ip -port $port -APIKey $apikey -param "sslrenegotiate" -value 0}

start-sleep -Milliseconds 250

# Force kerberos to use AES256 and SHA1

set-param -ip $ip -port $port -apikey $apikey -Param "KcdCipherSha1" -Value 'yes'

# Enable logs to use CEF data format

set-param -ip $ip -port $port -apikey $apikey -Param "CEFMsgFormat" -Value 'yes'

# Restict Web User Interface to TLS1.2 and TLS1.3

set-param -ip $ip -port $port -apikey $apikey -Param "WUITLSProtocols" -Value "7"

start-sleep -Milliseconds 250

# Set TLS1.3 WUI Cipher Set (space separated list)

if (($LM\_V -gt 56) -and (-not $FIPSMode)) {set-param -ip $ip -port $port -apikey $apikey -Param "WUITLS13Ciphersets" -Value "TLS\_AES\_256\_GCM\_SHA384 TLS\_AES\_128\_GCM\_SHA256"}

start-sleep -Milliseconds 250

####################################################################

############### LOAD TLS CERTIFICATE (PEM format only) #############

####################################################################

if ($doPEM) {

$base64str = [convert]::ToBase64String((Get-Content -path "$certpath" -Encoding byte))

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"addcert", "cert":"'+$certname+'", "password":"'+$certpass+'", "replace":"0", "data":"'+$base64str+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - TLS Certificate ($certpath) installed as ($certname)"; $msg; $msg >> $logfile}

else {$msg = "$time SKIP - TLS Certificate ($certname) alredy installed"; $msg; $msg >> $logfile}

start-sleep -Seconds 1

}

##################################################

# LOAD INTERMEDIATE CERTIFICATES (DER format only)

##################################################

if ($doCER) {

foreach ($ica in $certlist) {

[string]$icaname=$Null; [string]$icaname = (($ica.split("\")[-1]).split(".")[-2]).toupper()

$base64str = [convert]::ToBase64String((Get-Content -path "$ica" -Encoding byte))

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"addintermediate", "replace" : "0", "cert":"'+$icaname+'", "data":"'+$base64str+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Installed Intermediate Cert ($icaname)"; $msg; $msg >> $logfile}

else {$msg = "$time SKIP - Intermediate Certificate ($ica) already installed"; $msg; $msg >> $logfile}

}

start-sleep -Milliseconds 250

}

##################################################

# Assign Management Certificate to Admin Interface

##################################################

if ($doAdminCert) {

[string]$test=$Null; [string]$test = (get-param -ip $ip -port $port -APIKEY $apikey -param 'admincert').Replace("<Data>","").Replace("</Data>","")

if ($test -ne $certname) {set-param -ip $ip -port $port -APIKEY $apikey -param admincert -value $certname}

else {$msg = "$time SKIP - admincert already set to ($certname)"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

##################################################################

# Assign Management Certificate to Local Interface (HA Pairs only)

##################################################################

[bool]$doha = (Get-Param -ip $ip -port $port -APIKEY $apikey -param 'hamode') -ne 0

if ($doAdminCert -and $doha) {

[string]$test=$Null; [string]$test = (get-param -ip $ip -port $port -APIKEY $apikey -param 'localcert').Replace("<Data>","").Replace("</Data>","")

if ($test -ne "$certname") {set-param -ip $ip -port $port -APIKEY $apikey -param localcert -value $certname}

else {$msg = "$time SKIP - localcert already set to ($certname)"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

#################

# GEO Feature Set

#################

# Test if GEO included in support level

if (-not ($EnableGEO.gettypecode() -match 'Boolean')) {$doGEO = $False}

if ($licenseinfo.Option -match "GEO") {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"isgeoenabled"}'

$isgeo = (post-command -url $url -Body $body).content -match "GEO is enabled"

if ($isgeo -ne $EnableGEO) {

if ($EnableGEO) {[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"enablegeo"}'}

else {[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"disablegeo"}'}

if ($doGEO) {

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - GEO service set to ($EnableGEO)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring GEO service"; $msg; $msg >> $logfile}

}

}

if ($isgeo -eq $EnableGEO) {$msg = "$time SKIP - Current GEO state is ($isgeo) and desired state is ($EnableGEO)"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

###############

# Configure NTP

###############

if ($doNTP) {

$null = set-param -ip $ip -port $port -apikey $apikey -Param ntpkeysecret -value ""

}

if ($doNTPv3) {

set-param -ip $ip -port $port -apikey $apikey -Param ntpkeytype -value "$NTPV3\_KeyType"

set-param -ip $ip -port $port -apikey $apikey -Param ntpkeyid -value "$NTPV3\_KeyNum"

set-param -ip $ip -port $port -apikey $apikey -Param ntpkeysecret -value "$NTPV3\_Secret"

set-param -ip $ip -port $port -apikey $apikey -Param ntphost -value "$NTP\_Host"

start-sleep -Milliseconds 250

}

if ($doNTP) {

$msg = "$time INFO - Setting NTP Host can take some time. Please be patient"; $msg

set-param -ip $ip -port $port -apikey $apikey -Param ntphost -value "$NTP\_Host"

start-sleep -Milliseconds 250

}

###########################

# Create LDAP Configuration

###########################

if ($doLDAP) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"showldapendpoint","name":"'+$LDAP\_Name+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($rc.code -eq 422) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"addldapendpoint","name":"'+$LDAP\_Name+'","ldaptype":"'+$LDAP\_Type+'","server":"'+$LDAP\_Server+'","adminuser":"'+$LDAP\_User+'","adminpass":"'+$LDAP\_UserPass+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Configured New LDAP service ($LDAL\_Name)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring LDAP service"; $msg; $msg >> $logfile}

}

Else {$msg = "$time SKIP - LDAP Service ($LDAP\_Name) already exists"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

############################

# Add Certificate based user

############################

if ($doCertUser) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"usershow","user":"'+$NewCertUser+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($rc.code -eq 422) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"useraddlocal","user":"'+$NewCertUser+'","nopass":"yes"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Configured New Certificate Admin User ($NewCertUser)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring New Certificate Admin User"; $msg; $msg >> $logfile}

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"usersetperms","user":"'+$NewCertUser+'","perms":"'+$newuserrights+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Assigned rights ($newuserrights) to user ($NewCertUser)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring rights for New Certificate Admin User ($newcertuser)"; $msg; $msg >> $logfile}

}

Else {$msg = "$time SKIP - Certificate Admin User ($NewCertUser) already exists"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

#########################

# Add Password based user

#########################

if ($doPassUser) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"usershow","user":"'+$NewPassUser+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($rc.code -eq 422) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"useraddlocal","user":"'+$NewPassUser+'","password":"'+$NewPassUserPass+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Configured New Password Admin User ($NewPassUser)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring New Certificate Admin User"; $msg; $msg >> $logfile}

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"usersetperms","user":"'+$NewPassUser+'","perms":"'+$newuserrights+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Assigned rights ($newuserrights) to user ($NewPassUser)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring rights for New Password Admin User ($newPassuser)"; $msg; $msg >> $logfile}

}

Else {$msg = "$time SKIP - Password Admin User ($NewPassUser) already exists"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

#################

# Add Local Group

#################

if ($DoLDAPGroup) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"groupshow","group":"'+$NewUserGroup+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($rc.code -eq 422) {

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"groupaddremote","group":"'+$NewUserGroup+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Configured New LDAP Group ($NewUserGroup)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring New LDAP Group ($NewUserGroup)"; $msg; $msg >> $logfile}

[string]$body=$Null; [string]$body = '{"apikey":"'+$apikey+'","cmd":"groupsetperms","group":"'+$NewUserGroup+'","perms":"'+$newuserrights+'"}'

$r=$rc=$Null; $rc = try {$r = (post-command -url $url -Body $body | convertfrom-json)} catch {'{"code":"422"}' | ConvertFrom-Json}

if ($r.code -eq '200') {$msg = "$time SET - Assigned rights ($newuserrights) to user ($NewUserGroup)"; $msg; $msg >> $logfile}

else {$msg = "$time ERR - Error configuring rights for New Password Admin User ($newPassuser)"; $msg; $msg >> $logfile}

}

Else {$msg = "$time SKIP - LDAP Group ($NewUserGroup) already exists"; $msg; $msg >> $logfile}

start-sleep -Milliseconds 250

}

###################

# SET ADMIN GATEWAY

###################

if (($admingw -eq $Null) -or ($admingw -match '')) {[string]$admingw = get-param -ip $ip -port $port -APIKEY $apikey -param dfltgw}

set-param -ip $ip -port $port -APIKEY $apikey -param admingw -value $admingw

###############################

# Enable movement of default GW

###############################

set-param -ip $ip -port $port -APIKey $apikey -param "multigw" -value 1

#########################################################

# Enable Web User Interface Certificate or Password Login

#########################################################

if ($DoCertUser) {

$r = get-param -ip $ip -port $port -APIKEY $apikey -param adminclientaccess

if ($r -ne 1) {

if ($LM\_V -ge 58) {set-param -ip $ip -port $port -APIKEY $apikey -param adminclientaccess -value "1"}

else {$lastmsg = $lastmsg = "`nCleanup for $ip. Please use WUI interface (Certificates & Security/Remote Access menu) and enable (Password or Client Certificate) login method.`n"}

}

else {$msg = "$time SKIP - Admin Login Method already set to password or certificate"; $msg; $msg >> $logfile }

start-sleep -Milliseconds 250

}

}

#################

# END OF SCRIPT #

#################

if ($lastmsg.length -gt 0) {

$msg = "`nREQUIRED MANUAL CONFIGURATION ITEMS (CLEANUP ACTIONS)"; Write-Host -fore cyan $msg; $msg >> $logfile

write-host -fore yellow $lastmsg; $lastmessage >> $logfile

}

$msg = "`nSCRIPT COMPLETED"; Write-Host -fore cyan $msg; $msg >> $logfile

"################ END ##############" >> $logfile

###########################

# Clear sensitive variables

###########################

# Remove-Variable \* -ErrorAction SilentlyContinue

###############

# Reset .curlrc

###############

if (Test-Path -Path "~\.oldcurlrc" -PathType leaf) {

Copy-Item -Path "~\.oldcurlrc" "~\.curlrc" -Force

Remove-Item -Path "~\.oldcurlrc" -Force

}

1. Manually Configuring LoadMaster to meet US Gov Security Requirements

This additional information is provided to assist anyone interested in manually configuring a LoadMaster or ECS Connection Manager to meet US Government security requirements (aka STIG Compliance). This information will include the entire process (installation, licensing, platform configuration, virtual services configuration) required to ensure you meet US Gov security requirements.

The major steps involved include:

* Downloading the appliance (virtual only)
* Preparing the network (physical or virtual)
* Downloading relevant patches, deployment guides and templates
* Importing the virtual appliance or installing the physical appliance
* Connecting the network to the appliance
* Connecting a console to the appliance
* Booting and setting appliance IP address
* Connecting to appliance web user interface and licensing the appliance
* Patching the appliance
* Creating a local admin on the appliance
* Changing the built-in (bal) account password and placing the (bal) account under 2-person control
* Creating a data plane and a dedicated management plane
* Explicitly set admin gateway
* Moving default gateway to data plane
* Configuring NTP (prefer NTPv3)
* Configuring syslogd (prefer TLS)
* Setting warning banners (web and console)
* Removing 3DES and SHA1 from FIPS algorithm set (LMOS 7.2.58 and earlier)
* Enabling Web User Interface to use FIPS cipher set
* Enabling Remote Access management services to use FIPS cipher set
* Disabling any non-used services (e.g. GEO or SSH)
* Disabling SSL Renegotiation
* Sourcing and installing an authorized TLS management certificate
* Sourcing and installing relevant intermediate certificates (CA hierarchy)
* Configuring a LDAPS authentication service for admin login
* Creating a multifactor admin user account
* Configure https virtual services to send http header [X-Forwarded-For (+VIA)]
* Configure https virtual services to use FIPS cipher set
* Configure https virtual service to use an authorized TLS certificate
* Scan using Nessus
* Scan using ZED Attack Proxy