linuxprivchecker / linuxprivchecker.py

jtpereyda add ps -w option to support systems where default width is c... 000fc51 · 4 years ago History

Code Blame Executable File · 649 lines (518 loc) · 36.3 KB Raw

```
#!/usr/bin/python

"""
################################################################################
## [Title]: linuxprivchecker.py -- a Linux Privilege Escalation Check Script
## [Original Author]: Mike Czumak (T_v3rn1x) -- @SecuritySift
##  [Maintainer]: Michael Contino -- @Sleventyeleven
##------------------------------------------------------------------------------
## [Details]:
## This script is intended to be executed locally on a Linux box to enumerate basic sys
## search for common privilege escalation vectors such as world writable files, misconf
## passwords and applicable exploits.
##------------------------------------------------------------------------------
## [Modification, Distribution, and Attribution]:
## You are free to modify and/or distribute this script as you wish.  I only ask that y
## author attribution and not attempt to sell it or incorporate it into any commercial
## worth anything anyway :)
################################################################################
TODO:
Add search for writable and/or missing library files
Add detection and enumeration for systemd
Add search for accessiable ssh sockets
Add search for ssh keys
Add search for know access tokens
Expand Sudo support to include rules in sudoers.d
Add more high profile exploit checks (ie shellshock)
"""

# conditional import for older versions of python not compatible with subprocess
try:
    import subprocess as sub
    compatmode = 0  # newer version of python, no need for compatibility mode
except ImportError:
    import os  # older version of python, need to use os instead
    compatmode = 1


def execute_cmd(cmddict):
    """
    Execute Command (execute_cmd)
    loop through dictionary, execute the commands, store the results, return updated di

    :param cmddict: Dictionary of commands to execute and results
    :return: The command Dictionary with the commands results included
    """

    for item in cmddict:
        cmd = cmddict[item]["cmd"]
        if compatmode == 0:  # newer version of python, use preferred subprocess
            out, error = sub.Popen([cmd], stdout=sub.PIPE, stderr=sub.PIPE, shell=True)
            results = out.split('\n')
        else:  # older version of python, use os.popen
            echo_stdout = os.popen(cmd, 'r')
            results = echo_stdout.read().split('\n')

        # write the results to the command Dictionary for each command run
```

```python
        cmddict[item]["results"] = results

    return cmddict


def print_results(cmddict):
    """
    Print Results (printResults)
    Print results for each previously executed command, no return value

    :param cmddict: Dictionary of commands to execute and results
    :return: None
    """

    for item in cmddict:
        msg = cmddict[item]["msg"]
        results = cmddict[item]["results"]
        print "[+] " + msg

        for result in results:
            if result.strip() != "":
                print "    " + result.strip()
    print


def enum_system_info():
    """
    Basic System Info (get_system_info)
    Enumerate Basic System Information by executing simple commands than saving the res

    :return: Dictionary of system information results
    """

    print "[*] GETTING BASIC SYSTEM INFO...\n"

    sysinfo = {
        "OS": {"cmd": "cat /etc/issue", "msg": "Operating System", "results": []},
        "KERNEL": {"cmd": "cat /proc/version", "msg": "Kernel", "results": []},
        "HOSTNAME": {"cmd": "hostname", "msg": "Hostname", "results": []}
    }

    sysinfo = execute_cmd(sysinfo)
    print_results(sysinfo)

    return sysinfo


def enum_network_info():
    """
    Basic Network Info (get_network_info)
    Enumerate Basic Network Information by executing simple commands

    :return: Dictionary of Network information with results
    """

    print "[*] GETTING NETWORKING INFO...\n"

    netinfo = {
        "netinfo": {"cmd": "/sbin/ifconfig -a", "msg": "Interfaces", "results": []},
        "ROUTE": {"cmd": "route", "msg": "Route(s)", "results": []},
        "NETSTAT": {"cmd": "netstat -antup | grep -v 'TIME_WAIT'", "msg": "Netstat", "r
      }
```

```
    # import sys for io redirection
    import sys

    class Logger(object):
```

```
                def __init__(self):
                    self.terminal = sys.stdout
                    self.log = open(args.outfile, 'a')

                def write(self, message):
                    self.terminal.write(message)
                    self.log.write(message)
            sys.stdout = Logger()

    except ImportError:
        print 'Arguments could not be processed, defaulting to print everything'
        processsearches = True

    # title / formatting
    bigline = "========================================================================
    print bigline
    print """
        __    _                  ____       _       ________              __
       / /   (_)___  __  ___  __/ __ \_____(_)   __/ ____/ /_  ___  _____/ /_____  ____
      / /   / / __ \/ / / / |/_/ /_/ / ___/ / | / / /   / __ \/ _ \/ ___/ //_/ _ \/ ___
     / /___/ / / / / /_/ />  </ ____/ /  / /| |/ / /___/ / / /  __/ /__/ ,< /  __/ /
    /_____/_/_/ /_/\__,_/_/|_/_/   /_/  /_/ |___/\____/_/ /_/\___/\___/_/|_|\___/_/

    """
    print bigline

    # Enumerate Basic User Information
    userinfo = enum_user_info()

    # Enumerate Basic System Information
    sysinfo = enum_system_info()

    # Enumerate Basic Network Information
    enum_network_info()

    # Enumerate User History Files
    enum_user_history_files()

    # Enumerate Basic RC Files
    enum_rc_files()

    # Enumerate Basic Filesystem Information
    driveinfo = enum_filesystem_info()

    # Enumerate List of all Cron jobs
    enum_cron_jobs()

    # Enumerate Package and Process information
    pkgsandprocs = enum_procs_pkgs()

    # Enumerate Possible Root/superuser packages or processes
    enum_root_pkg_proc(pkgsandprocs, userinfo)

    # Emumerate Available Development Tools
    devtools = enum_dev_tools()

    # Enumerate Possible Shell Escapes
    enum_shell_esapes(devtools)

    find_likely_exploits(sysinfo, devtools, pkgsandprocs, driveinfo)

    if processsearches:
        # Search for Insecure File/Folder Permissions
        search_file_perms()

        # Search for files with potential credentials
        search_file_passwords()

    print "Finished"
    print bigline
```