Security Auditor Program - User Manual

Contents

Introduction

The Security Auditor Program is a comprehensive toolset designed for performing security audits on Windows systems. It allows you to perform remote audits on multiple systems concurrently and process audit results locally, providing an in-depth understanding of your security posture.

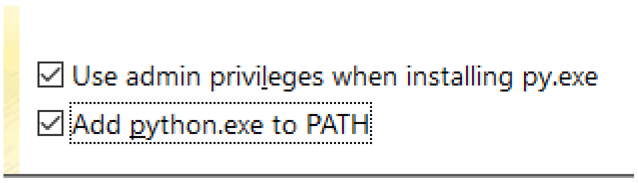
Program Directory Structure

```
Security Auditor Program/
        |- config/
        |       |- config.xlsx
        |- out/
        |       |- script/
        |       |       |- CIS_Microsoft_Windows_Server_2016_Benchmark_v2.0.0_L1_MS.ps1
        |       |       |- CIS_Microsoft_Windows_Server_2019_Benchmark_v2.0.0_L1_DC.ps1
                |       |       |- ……
        |       |- local_win_11.xlsx
        |       |- local_win_2016.xlsx
        |       |- ……
        |- src/
        |       |- Audit/
        |       |       |- CIS_Microsoft_Windows_Server_2016_Benchmark_v2.0.0_L1_MS.xlsx
        |       |       |- CIS_Microsoft_Windows_Server_2019_Benchmark_v2.0.0_L1_DC.xlsx
        |       |       |- ……
        |       |- CIS/
        |       |       |- CIS_Microsoft_Windows_Server_2016_Benchmark_v2.0.0_L1_MS.audit
        |       |       |- CIS_Microsoft_Windows_Server_2019_Benchmark_v2.0.0_L1_DC.audit
        |       |       |- ……
        |- utilities/
        |       |- __init__.py
        |       |- getAnonySID.py
        |       |- getAuditPolicy.py
        |       |- getBannerCheck.py
        |       |- getCheckAccount.py
        |       |- getLockoutPolicy.py
        |       |- getPwdPolicy.py
        |       |- getRegCheck.py
        |       |- getRegValue.py
        |       |- getUserRights.py
        |       |- getWMIPolicy.py
        |- audit_file_parser.py
        |- local_audit_command_generator.py
        |- local_audit_results_processor.py
        |- remote_audit_executor.py
```

Installation Guide

Python Installation

Install Python version > 3.7


☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

Remember to select “Add python.exe to PATH”

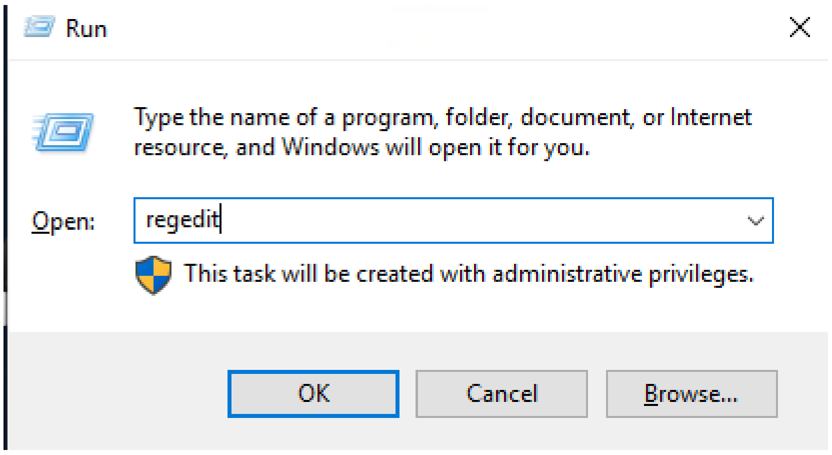
Python Packages Installation

Install Python packages using pip

- pip install bs4 lxml pandas argparse openpyxl regex pypsexec smbprotocol

Remote Host Requirements (Remote version only)

- User Account Control (UAC):
  - Open Windows Registry Editor (Press Win+R, enter ‘regedit’)

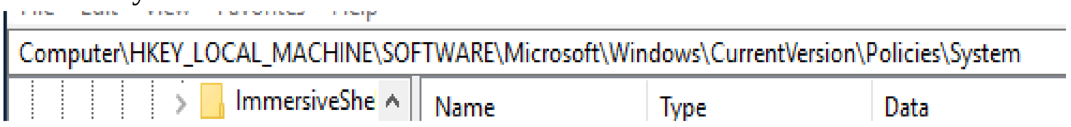

Run

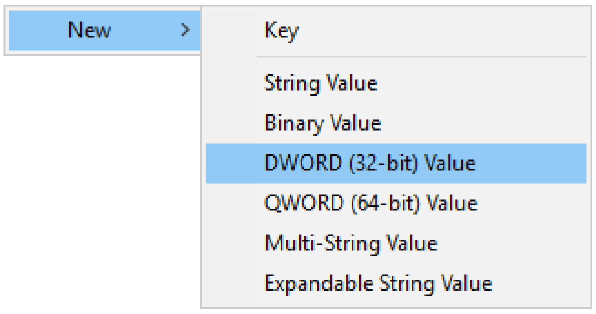
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

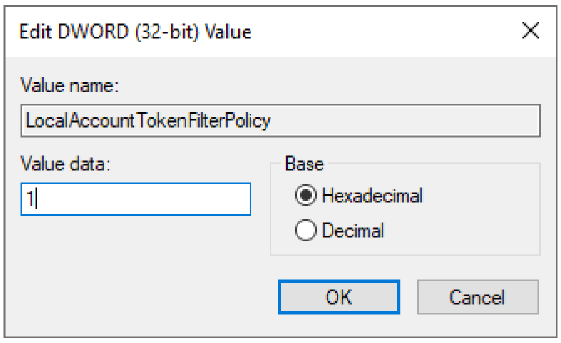
Open: regedit

This task will be created with administrative privileges.

  - Add a new registry DWORD named LocalAccountTokenFilterPolicy and setting its value to 1. Create this key in the following registry:
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\.


Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System




Edit DWORD (32-bit) Value

Value name: LocalAccountTokenFilterPolicy

Value data: 1

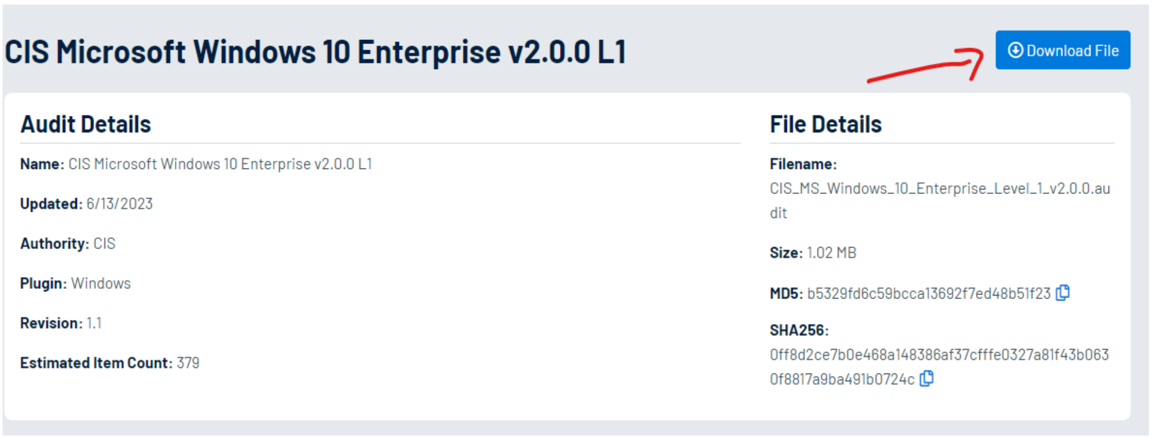
Base: [✓ Hexadecimal] Decimal

- Host Firewall:
    - Use the Run prompt (Win+R), run gpedit.msc and enable Group Policy Object Editor. Navigate to Local Computer Policy > Administrative Templates > Network > Network Connections > Windows Defender Firewall > Standard Profile > Windows Firewall: Allow inbound file and printer exception and enable it.
    - While in the Group Policy Object Editor, navigate to Local Computer Policy > Administrative Templates > Network > Network Connections > Prohibit use of Internet connection firewall on your DNS domain. Set this option to either Disabled or Not Configured.
    - Open any host firewalls to allow connections from local to File and Printer Sharing on TCP ports 139 and 445.

Running the Program

Audit File Preparation

1. Download the latest CIS benchmark Windows Level 1 .audit file from Nessus (https://www.tenable.com/audits/search?q=windows+L1+AND+type%3A%28CIS%29+AND+display_name%3A%28L1%29+AND+plugin%3A%28Windows%29&sort=&page=1).
   For example:
   https://www.tenable.com/audits/CIS_MS_Windows_10_Enterprise_Level_1_v2.0.0



CIS Microsoft Windows 10 Enterprise v2.0.0 L1

Audit Details

Name: CIS Microsoft Windows 10 Enterprise v2.0.0 L1

Updated: 6/13/2023

Authority: CIS

Plugin: Windows

Revision: 1.1

Estimated Item Count: 379

File Details

Filename: CIS_MS_Windows_10_Enterprise_Level_1_v2.0.0.audit

Size: 1.02 MB

MD5: b5329fd6c59bcca13692f7ed48b51f23

SHA256: 0ff8d2ce7b0e468a148386af37cfffe0327a81f43b0630f8817a9ba491b0724c

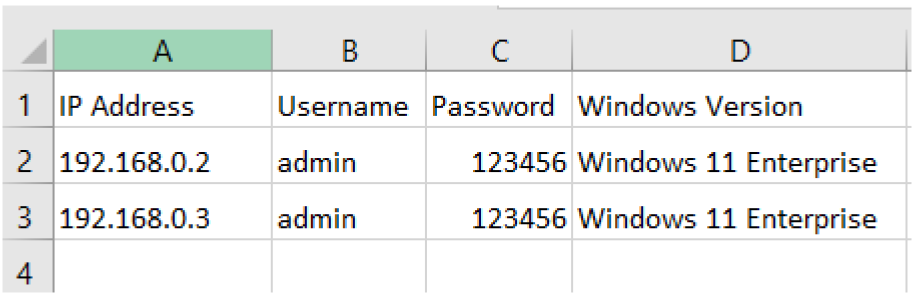
2. Run `**audit_file_parser.py**` from the command line, passing the path of the .audit file. For example:
   - python audit_file_parser.py --audit CIS_MS_Windows_11_Enterprise_Level_1_v1.0.0.audit

3. Wait for the script to finish running, then check the results in the output file. This script will generate a new Excel file with the same name as the input audit file at `/src/Audit`.

4. At this point, you can customize this Excel file to meet your specific requirements. It will serve as the input for the following audit process.

How to Run Remotely

1. Prepare the configuration file, which is located at `config/config.xlsx`. It contains the IP address, username, password, and Windows version for the security audit.



	A	B	C	D
1	IP Address	Username	Password	Windows Version
2	192.168.0.2	admin	123456	Windows 11 Enterprise
3	192.168.0.3	admin	123456	Windows 11 Enterprise
4				

   Make sure to use an administrator account. Also, you should only select the same Windows version each time it runs.

2. Run `**remote_audit_executor.py**` from the command line, passing the path of the configuration file and output file.
   For example:
   - python Main_remote_multiprocess.py --config config.xlsx --output output.xlsx
   - Where, `config.xlsx` is the configuration file, `output.xlsx` is the output file.

3. Wait for the script to finish running, then check the results in the output file.

How to Run Locally

1. Run `**local_audit_command_generator.py**` from the command line, passing the path of the audit file.
   For example:
   - python local_audit_command_generator.py --audit audit_file.xlsx

2. This script will generate a PowerShell script (.ps1) with the same name as the input audit file at `/out/script`.

3. Copy the PowerShell script to the target host. Run the script in the PowerShell (Administrator mode) and export the result into a txt file.
   For example:
   - ./script_name.ps1 > audit_result.txt

   If you encounter a permission error, try to run this command:
   - Set-ExecutionPolicy Unrestricted

4. Copy the output file (result.txt) to local host.

5. Run `**local_audit_results_processor.py**` from the command line, passing the path of the result file, audit file and output file.
   For example:
   - python local_audit_results_processor.py --audit audit_file.xlsx --result audit_result.txt --output output.xlsx
   - Where, `audit_file.xlsx` is the audit file, `audit_result.txt` is the file copied from the target host, and `output.xlsx` is the output for the script.

6. Wait for the script to finish running, then check the results in the output file.

Common Troubleshoots

The following guidelines provide solutions to some common issues that you might encounter while running the remote audit script. They cover problems ranging from initial setup and file access to network issues and performance concerns. These suggestions are designed to assist in resolving issues in a systematic way.

Script fails to run

- Make sure Python is installed and properly configured on your machine.
- Ensure that all required Python packages are installed. These might include pandas, openpyxl, and any others imported in the script.

Script cannot read the configuration or audit files

- Make sure the paths to the files are correct.
- Ensure the files are formatted correctly.
- Make sure the script has read permissions for these files.
- Make sure all the target systems specified in a single Excel configuration file share the same Windows version

Script cannot write the output file

- Make sure the script has write permissions in the directory where it's trying to write the output file.
- If the file is already open, close it before running the script.

Script fails to connect to the target systems

- Ensure the target systems are up and running.
- Check the network connection between the machine running the script and the target systems.
- Make sure the IP addresses, usernames, and passwords in the configuration file are correct.
- Make sure the account used is administrator account. (remote only)
- Make sure the registry and firewall settings are correct.
- Error messages:
    - “[WinError 10060] A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.”
    - “smbprotocol.exceptions.LogonFailure: Received unexpected status from the server: The attempted logon is invalid. This is either due to a bad username or authentication information.”
    - “AttributeError: 'NoneType' object has no attribute 'open_service_w'.”

Script returns incorrect or unexpected audit results

- Verify the audit commands in the Excel file and in the gen_ps_args function.
- Make sure the target systems have the required features enabled and permissions set to allow the script to perform its tasks.

Script runs slowly or hangs

- Try reducing the number of target systems or the number of audit tasks.

- Check the system resources on the machine running the script. It may need more memory or CPU power to handle the tasks.

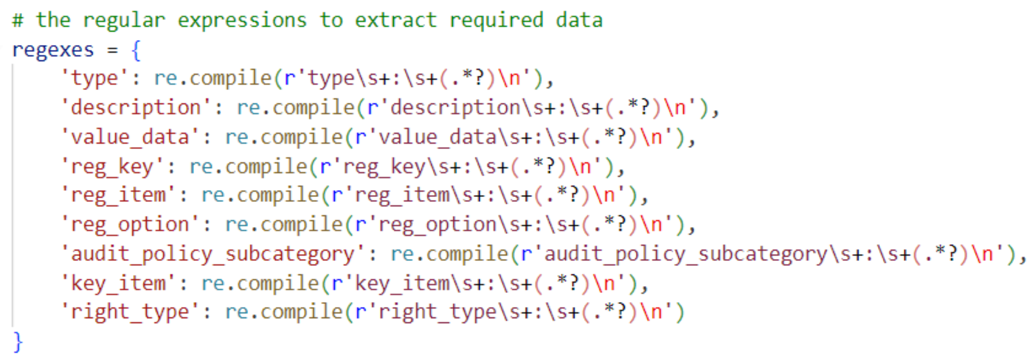
Place for Updates

This section is designed to guide future developers who might wish to modify or expand the functionality of these scripts. It identifies key areas in the code that are likely to require updates or modifications to meet changing requirements or to add new features. In this section, detailed explanations will be provided on the areas in each Python script that may require modifications.

audit_file_parser.py:

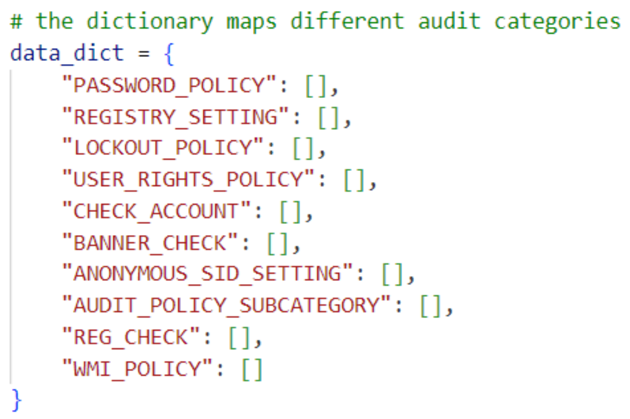
If Nessus updates the format of the `.audit` file, the following places are likely to require updates.

- This is the place to update regular expressions based on the `.audit` file.



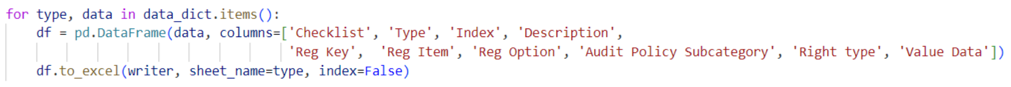
```python
# the regular expressions to extract required data
regexes = {
    'type': re.compile(r'type\s+:\s+(.*?)\n'),
    'description': re.compile(r'description\s+:\s+(.*?)\n'),
    'value_data': re.compile(r'value_data\s+:\s+(.*?)\n'),
    'reg_key': re.compile(r'reg_key\s+:\s+(.*?)\n'),
    'reg_item': re.compile(r'reg_item\s+:\s+(.*?)\n'),
    'reg_option': re.compile(r'reg_option\s+:\s+(.*?)\n'),
    'audit_policy_subcategory': re.compile(r'audit_policy_subcategory\s+:\s+(.*?)\n'),
    'key_item': re.compile(r'key_item\s+:\s+(.*?)\n'),
    'right_type': re.compile(r'right_type\s+:\s+(.*?)\n')
}
```

- This is the place to update dictionary that maps different audit categories.



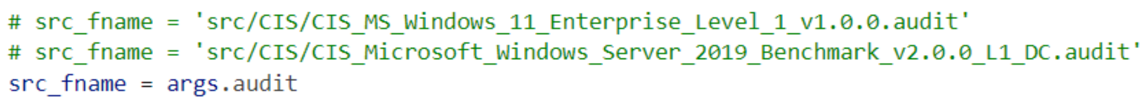
```python
# the dictionary maps different audit categories
data_dict = {
    "PASSWORD_POLICY": [],
    "REGISTRY_SETTING": [],
    "LOCKOUT_POLICY": [],
    "USER_RIGHTS_POLICY": [],
    "CHECK_ACCOUNT": [],
    "BANNER_CHECK": [],
    "ANONYMOUS_SID_SETTING": [],
    "AUDIT_POLICY_SUBCATEGORY": [],
    "REG_CHECK": [],
    "WMI_POLICY": []
}
```

- This is the place to modify the column name of the output file.



```python
for type, data in data_dict.items():
    df = pd.DataFrame(data, columns=['Checklist', 'Type', 'Index', 'Description',
                                     'Reg Key',  'Reg Item', 'Reg Option', 'Audit Policy Subcategory', 'Right type', 'Value Data'])
    df.to_excel(writer, sheet_name=type, index=False)
```

- This comment is for testing purposes.



```python
# src_fname = 'src/CIS/CIS_MS_Windows_11_Enterprise_Level_1_v1.0.0.audit'
# src_fname = 'src/CIS/CIS_Microsoft_Windows_Server_2019_Benchmark_v2.0.0_L1_DC.audit'
src_fname = args.audit
```

- This is the place to modify the output file name if needed. (Not recommended)

```python
# save the data into an Excel file
# out_fname = 'src\win_server_2022_ms_v1.xlsx'
out_fname = 'src\\Audit\\' + \
    src_fname.split("\\")[-1].replace("audit", "xlsx")
```

- These are the places to update the rule of cleaning the data. If you miss some irregular data, you can also manually modify the data in the output `.xlsx` file.

```python
if type == "AUDIT_POWERSHELL":
    continue
else:
    type = type.strip()

description = regexes['description'].search(item_str)
description = description.group(1) if description else None
description = description.replace('"', '')

if description[0].isdigit():
    index = re.search(r'(.*?)\s', description)
    index = index.group(1) if index else None
    description = description.replace(index, '').strip()
else:
    index = 0

index = str(index).strip()

value_data = regexes['value_data'].search(item_str)
value_data = value_data.group(1) if value_data else None
value_data = str(value_data).replace('"', '')
value_data = str(value_data).replace('&amp;&amp;', '&&')

reg_key = regexes['reg_key'].search(item_str)
reg_key = (reg_key.group(1)).replace('"', '') if reg_key else None
```

```python
reg_item = regexes['reg_item'].search(item_str)
reg_item = (reg_item.group(1)).replace('"', '') if reg_item else None

reg_option = regexes['reg_option'].search(item_str)
reg_option = (reg_option.group(1)).replace(
    '"', '') if reg_option else None

key_item = regexes['key_item'].search(item_str)
key_item = key_item.group(1) if key_item else None

if key_item:
    reg_item = key_item.replace('"', '')

audit_policy_subcategory = regexes['audit_policy_subcategory'].search(
    item_str)
audit_policy_subcategory = (audit_policy_subcategory.group(
    1)).replace('"', '') if audit_policy_subcategory else None

right_type = regexes['right_type'].search(item_str)
right_type = (right_type.group(1)).replace(
    '"', '') if right_type else None
```

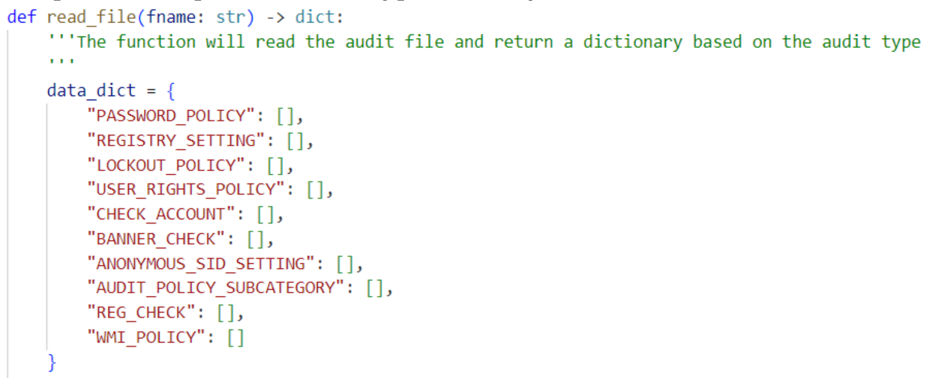
```python
# Clean the data
if type == 'BANNER_CHECK':
    value_data = ''
elif type == 'ANONYMOUS_SID_SETTING':
    value_data = '0'
elif type == 'REG_CHECK':
    reg_key = value_data
    value_data = ''
elif type == 'CHECK_ACCOUNT':
    if 'Rename administrator account' in description:
        value_data = 'Administrator'
```

```python
    elif 'Disabled' in description:
        value_data = 'No'
elif type == 'PASSWORD_POLICY':
    if value_data == 'Enabled':
        value_data = 1
    elif value_data == 'Disabled':
        value_data = 0
    elif value_data == '@PASSWORD_HISTORY@':
        value_data = 24
    elif value_data == '@MAXIMUM_PASSWORD_AGE@':
        value_data = 365
    elif value_data == '@MINIMUM_PASSWORD_AGE@':
        value_data = 1
    elif value_data == '@MINIMUM_PASSWORD_LENGTH@':
        value_data = 14
elif type == 'REGISTRY_SETTING':
    if index == '0':
        value_data = 'Windows'
    elif 'Lock Workstation' in description:
        value_data = '1 || 2 || 3'
    elif 'None' in description:
        value_data = 'Null'
    elif ' Remotely accessible registry paths' in description:
        value_data = value_data.replace(' && ', '')
    elif 'Screen saver timeout' in description:
        value_data = '[0..900]'
```

local_audit_command_generator.py

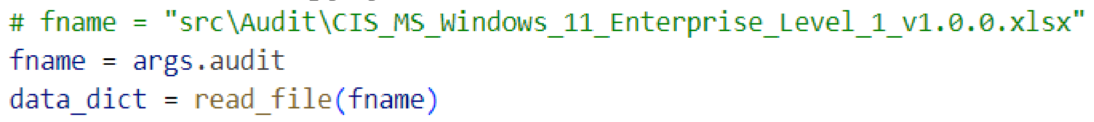
If you updated the audit type name or column name, you will have to modify the name in this script as well. If you need to update PowerShell commands, please refer to the details in ‘PowerShell Commands’ section.

- This place is to update the audit type name if you have modified before.



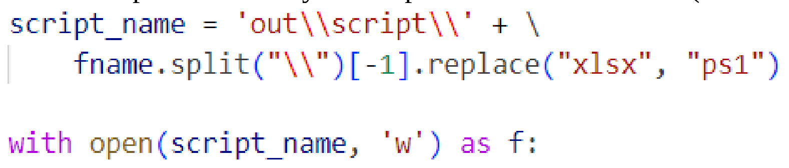
```
def read_file(fname: str) -> dict:
    '''The function will read the audit file and return a dictionary based on the audit type
    '''
    data_dict = {
        "PASSWORD_POLICY": [],
        "REGISTRY_SETTING": [],
        "LOCKOUT_POLICY": [],
        "USER_RIGHTS_POLICY": [],
        "CHECK_ACCOUNT": [],
        "BANNER_CHECK": [],
        "ANONYMOUS_SID_SETTING": [],
        "AUDIT_POLICY_SUBCATEGORY": [],
        "REG_CHECK": [],
        "WMI_POLICY": []
    }
```

- This comment is for testing purposes.



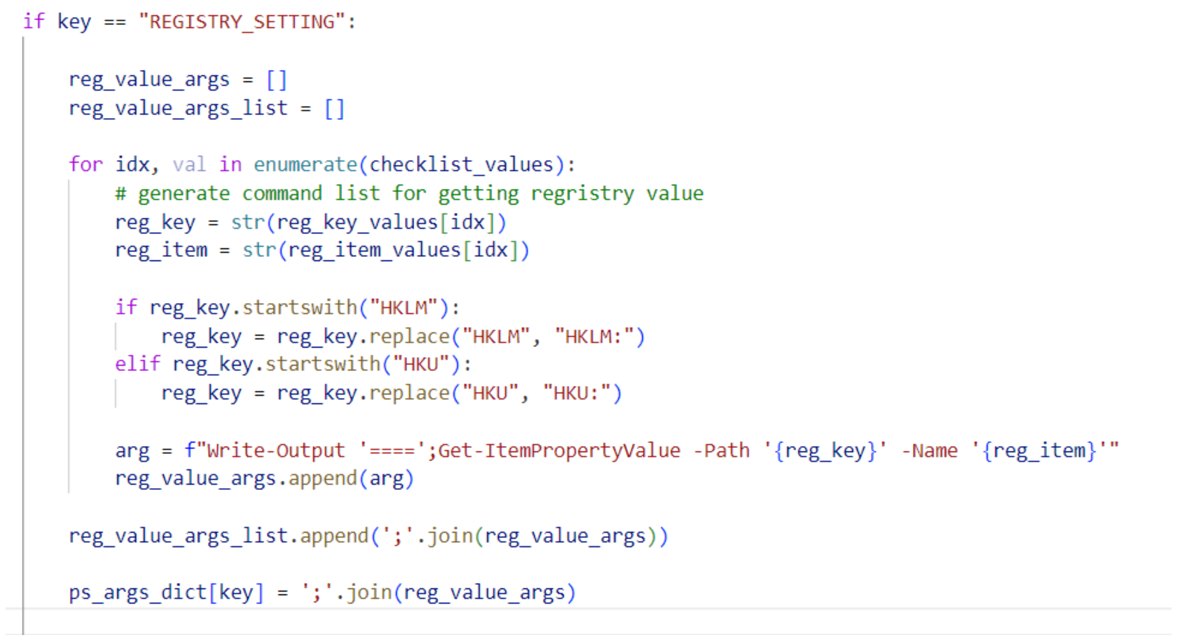
```
# fname = "src\Audit\CIS_MS_Windows_11_Enterprise_Level_1_v1.0.0.xlsx"
fname = args.audit
data_dict = read_file(fname)
```

- This is the place to modify the output file name if needed. (Not recommended)



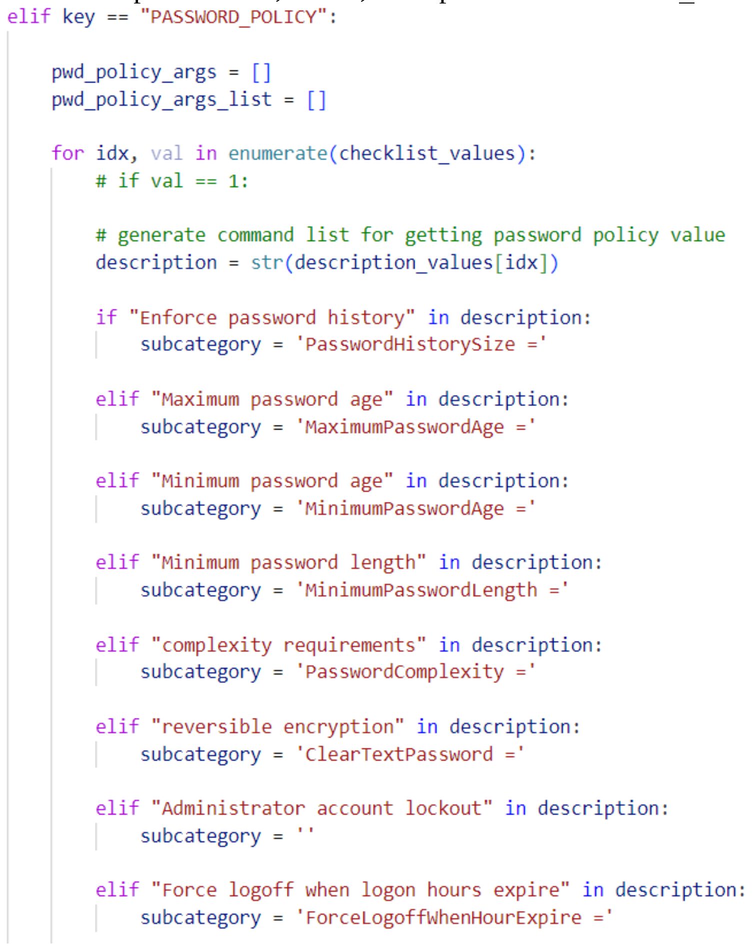
```
script_name = 'out\\script\\' + \
    fname.split("\\")[-1].replace("xlsx", "ps1")

with open(script_name, 'w') as f:
```

- This is the place to add, delete, and update ‘REGISTRY_SETTING’ commands.



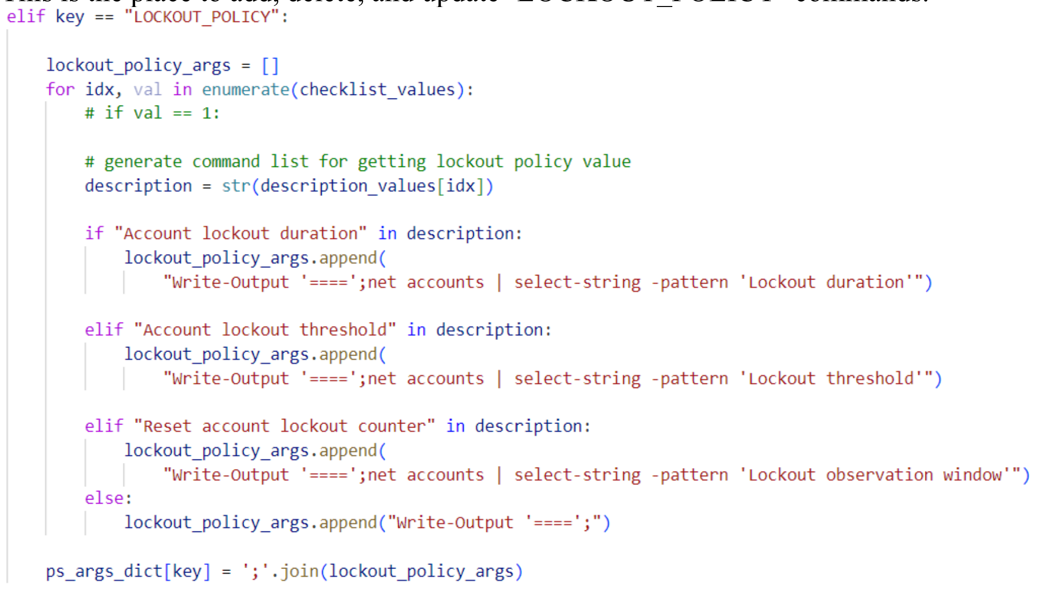
```
if key == "REGISTRY_SETTING":

    reg_value_args = []
    reg_value_args_list = []

    for idx, val in enumerate(checklist_values):
        # generate command list for getting regristry value
        reg_key = str(reg_key_values[idx])
        reg_item = str(reg_item_values[idx])

        if reg_key.startswith("HKLM"):
            reg_key = reg_key.replace("HKLM", "HKLM:")
        elif reg_key.startswith("HKU"):
            reg_key = reg_key.replace("HKU", "HKU:")

        arg = f"Write-Output '===';Get-ItemPropertyValue -Path '{reg_key}' -Name '{reg_item}'"
        reg_value_args.append(arg)

    reg_value_args_list.append(';'.join(reg_value_args))

    ps_args_dict[key] = ';'.join(reg_value_args)
```

- This is the place to add, delete, and update ‘PASSWORD_POLICY’ commands.



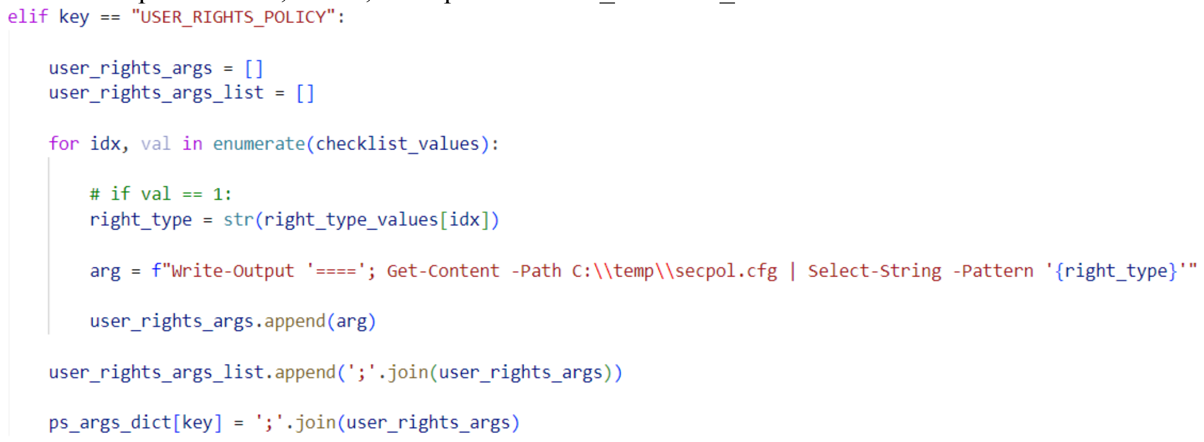
```
elif key == "PASSWORD_POLICY":

    pwd_policy_args = []
    pwd_policy_args_list = []

    for idx, val in enumerate(checklist_values):
        # if val == 1:

        # generate command list for getting password policy value
        description = str(description_values[idx])

        if "Enforce password history" in description:
            subcategory = 'PasswordHistorySize ='

        elif "Maximum password age" in description:
            subcategory = 'MaximumPasswordAge ='

        elif "Minimum password age" in description:
            subcategory = 'MinimumPasswordAge ='

        elif "Minimum password length" in description:
            subcategory = 'MinimumPasswordLength ='

        elif "complexity requirements" in description:
            subcategory = 'PasswordComplexity ='

        elif "reversible encryption" in description:
            subcategory = 'ClearTextPassword ='

        elif "Administrator account lockout" in description:
            subcategory = ''

        elif "Force logoff when logon hours expire" in description:
            subcategory = 'ForceLogoffWhenHourExpire ='
```

- This is the place to add, delete, and update ‘LOCKOUT_POLICY’ commands.



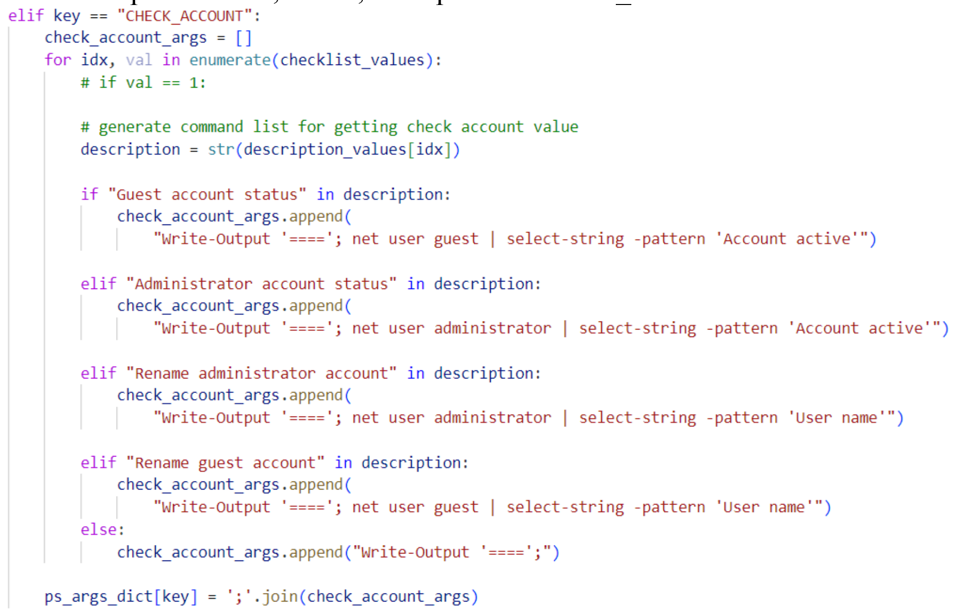
```
elif key == "LOCKOUT_POLICY":

    lockout_policy_args = []
    for idx, val in enumerate(checklist_values):
        # if val == 1:

        # generate command list for getting lockout policy value
        description = str(description_values[idx])

        if "Account lockout duration" in description:
            lockout_policy_args.append(
                "Write-Output '===';net accounts | select-string -pattern 'Lockout duration'")

        elif "Account lockout threshold" in description:
            lockout_policy_args.append(
                "Write-Output '===';net accounts | select-string -pattern 'Lockout threshold'")

        elif "Reset account lockout counter" in description:
            lockout_policy_args.append(
                "Write-Output '===';net accounts | select-string -pattern 'Lockout observation window'")
        else:
            lockout_policy_args.append("Write-Output '===';")

    ps_args_dict[key] = ';'.join(lockout_policy_args)
```

- This is the place to add, delete, and update ‘USER_RIGHTS_POLICY’ commands.



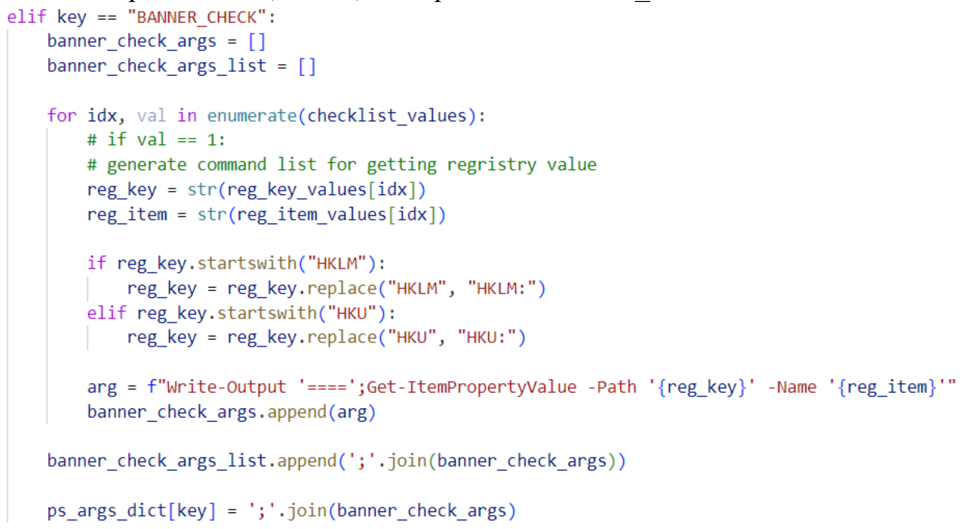
```
elif key == "USER_RIGHTS_POLICY":

    user_rights_args = []
    user_rights_args_list = []

    for idx, val in enumerate(checklist_values):

        # if val == 1:
        right_type = str(right_type_values[idx])

        arg = f"Write-Output '===='; Get-Content -Path C:\\temp\\secpol.cfg | Select-String -Pattern '{right_type}'"

        user_rights_args.append(arg)

    user_rights_args_list.append(';'.join(user_rights_args))

    ps_args_dict[key] = ';'.join(user_rights_args)
```

- This is the place to add, delete, and update ‘CHECK_ACCOUNT’ commands.



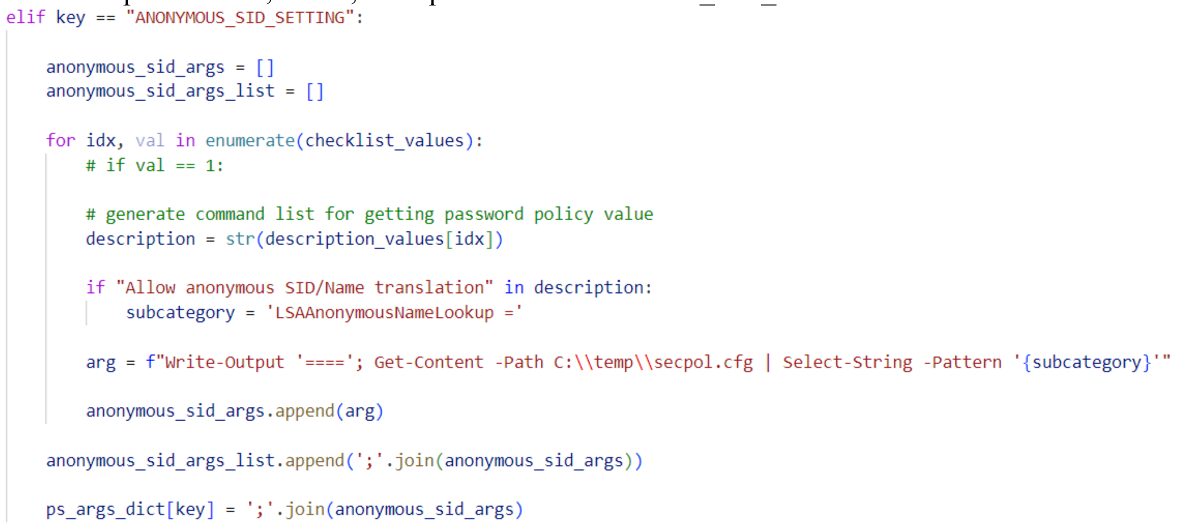
```
elif key == "CHECK_ACCOUNT":
    check_account_args = []
    for idx, val in enumerate(checklist_values):
        # if val == 1:

        # generate command list for getting check account value
        description = str(description_values[idx])

        if "Guest account status" in description:
            check_account_args.append(
                "Write-Output '===='; net user guest | select-string -pattern 'Account active'")

        elif "Administrator account status" in description:
            check_account_args.append(
                "Write-Output '===='; net user administrator | select-string -pattern 'Account active'")

        elif "Rename administrator account" in description:
            check_account_args.append(
                "Write-Output '===='; net user administrator | select-string -pattern 'User name'")

        elif "Rename guest account" in description:
            check_account_args.append(
                "Write-Output '===='; net user guest | select-string -pattern 'User name'")
        else:
            check_account_args.append("Write-Output '====';")

    ps_args_dict[key] = ';'.join(check_account_args)
```

- This is the place to add, delete, and update ‘BANNER_CHECK’ commands.



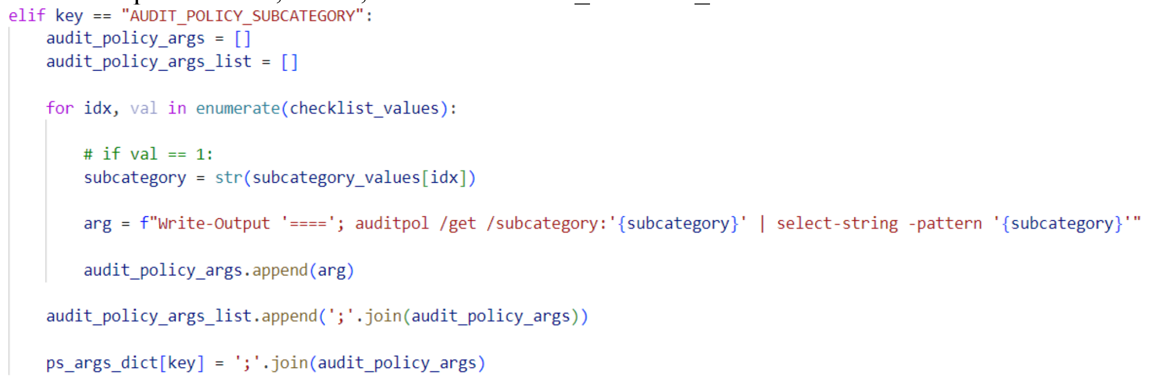
```
elif key == "BANNER_CHECK":
    banner_check_args = []
    banner_check_args_list = []

    for idx, val in enumerate(checklist_values):
        # if val == 1:
        # generate command list for getting regristry value
        reg_key = str(reg_key_values[idx])
        reg_item = str(reg_item_values[idx])

        if reg_key.startswith("HKLM"):
            reg_key = reg_key.replace("HKLM", "HKLM:")
        elif reg_key.startswith("HKU"):
            reg_key = reg_key.replace("HKU", "HKU:")

        arg = f"Write-Output '====';Get-ItemPropertyValue -Path '{reg_key}' -Name '{reg_item}'"
        banner_check_args.append(arg)

    banner_check_args_list.append(';'.join(banner_check_args))

    ps_args_dict[key] = ';'.join(banner_check_args)
```

- This is the place to add, delete, and update ‘ANONYMOUS_SID_SETTING’ commands.



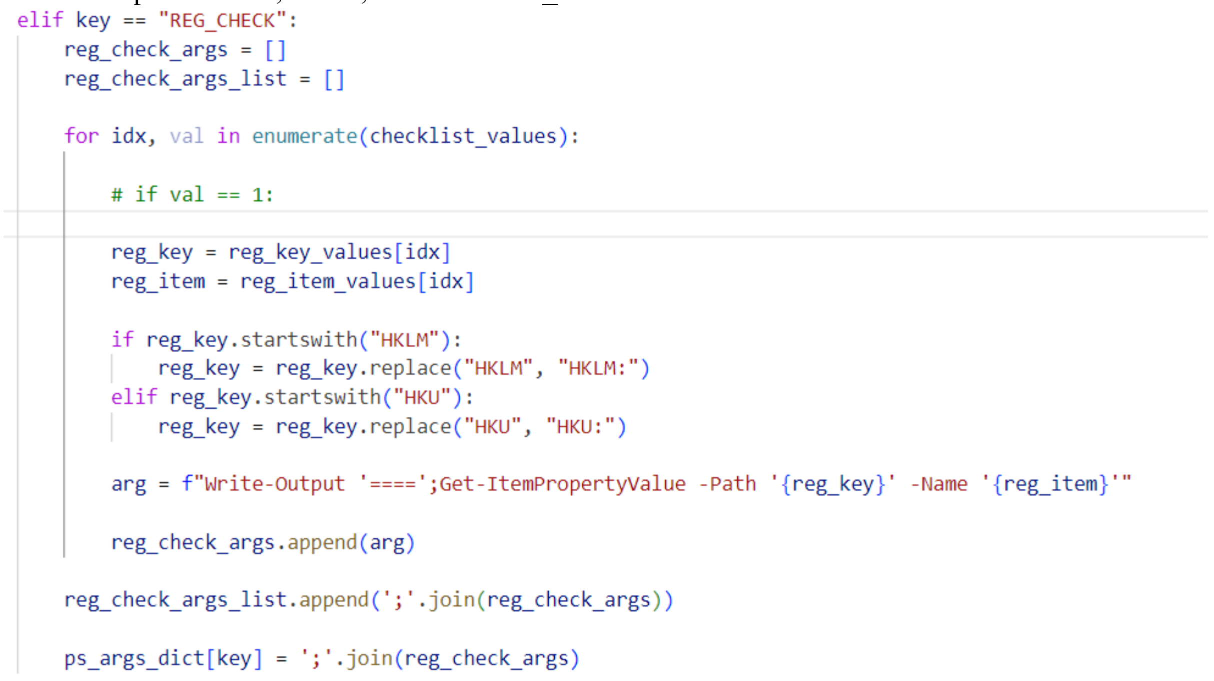
```python
elif key == "ANONYMOUS_SID_SETTING":

    anonymous_sid_args = []
    anonymous_sid_args_list = []

    for idx, val in enumerate(checklist_values):
        # if val == 1:

        # generate command list for getting password policy value
        description = str(description_values[idx])

        if "Allow anonymous SID/Name translation" in description:
            subcategory = 'LSAAnonymousNameLookup ='

        arg = f"Write-Output '===='; Get-Content -Path C:\\temp\\secpol.cfg | Select-String -Pattern '{subcategory}'"

        anonymous_sid_args.append(arg)

    anonymous_sid_args_list.append(';'.join(anonymous_sid_args))

    ps_args_dict[key] = ';'.join(anonymous_sid_args)
```

- This is the place to add, delete, and add ‘AUDIT_POLICY_SUBCATEGORY’ commands.



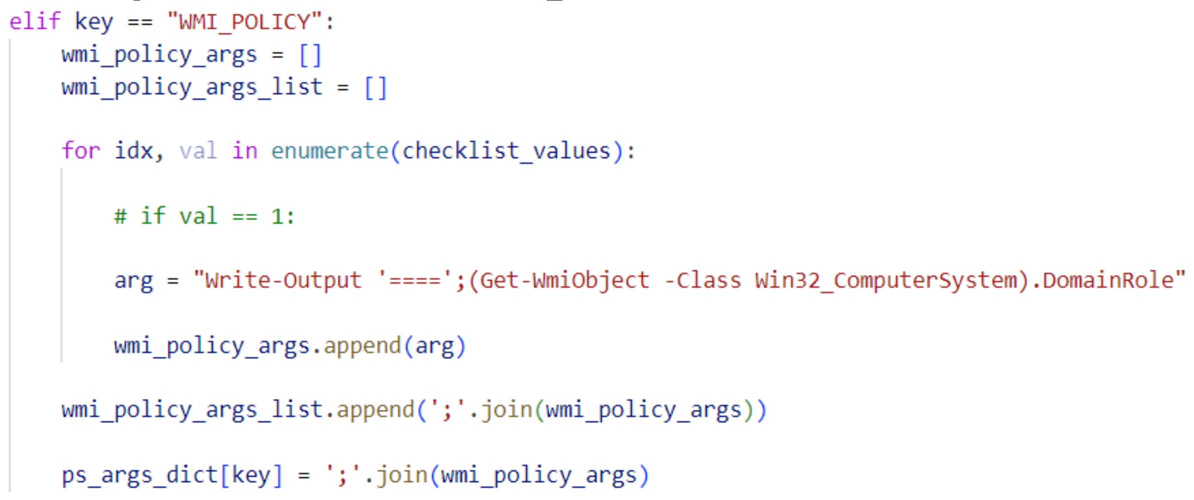
```python
elif key == "AUDIT_POLICY_SUBCATEGORY":
    audit_policy_args = []
    audit_policy_args_list = []

    for idx, val in enumerate(checklist_values):

        # if val == 1:
        subcategory = str(subcategory_values[idx])

        arg = f"Write-Output '===='; auditpol /get /subcategory:'{subcategory}' | select-string -pattern '{subcategory}'"

        audit_policy_args.append(arg)

    audit_policy_args_list.append(';'.join(audit_policy_args))

    ps_args_dict[key] = ';'.join(audit_policy_args)
```

- This is the place to add, delete, and add ‘REG_CHECK’ commands.



```python
elif key == "REG_CHECK":
    reg_check_args = []
    reg_check_args_list = []

    for idx, val in enumerate(checklist_values):

        # if val == 1:

        reg_key = reg_key_values[idx]
        reg_item = reg_item_values[idx]

        if reg_key.startswith("HKLM"):
            reg_key = reg_key.replace("HKLM", "HKLM:")
        elif reg_key.startswith("HKU"):
            reg_key = reg_key.replace("HKU", "HKU:")

        arg = f"Write-Output '====';Get-ItemPropertyValue -Path '{reg_key}' -Name '{reg_item}'"

        reg_check_args.append(arg)

    reg_check_args_list.append(';'.join(reg_check_args))

    ps_args_dict[key] = ';'.join(reg_check_args)
```

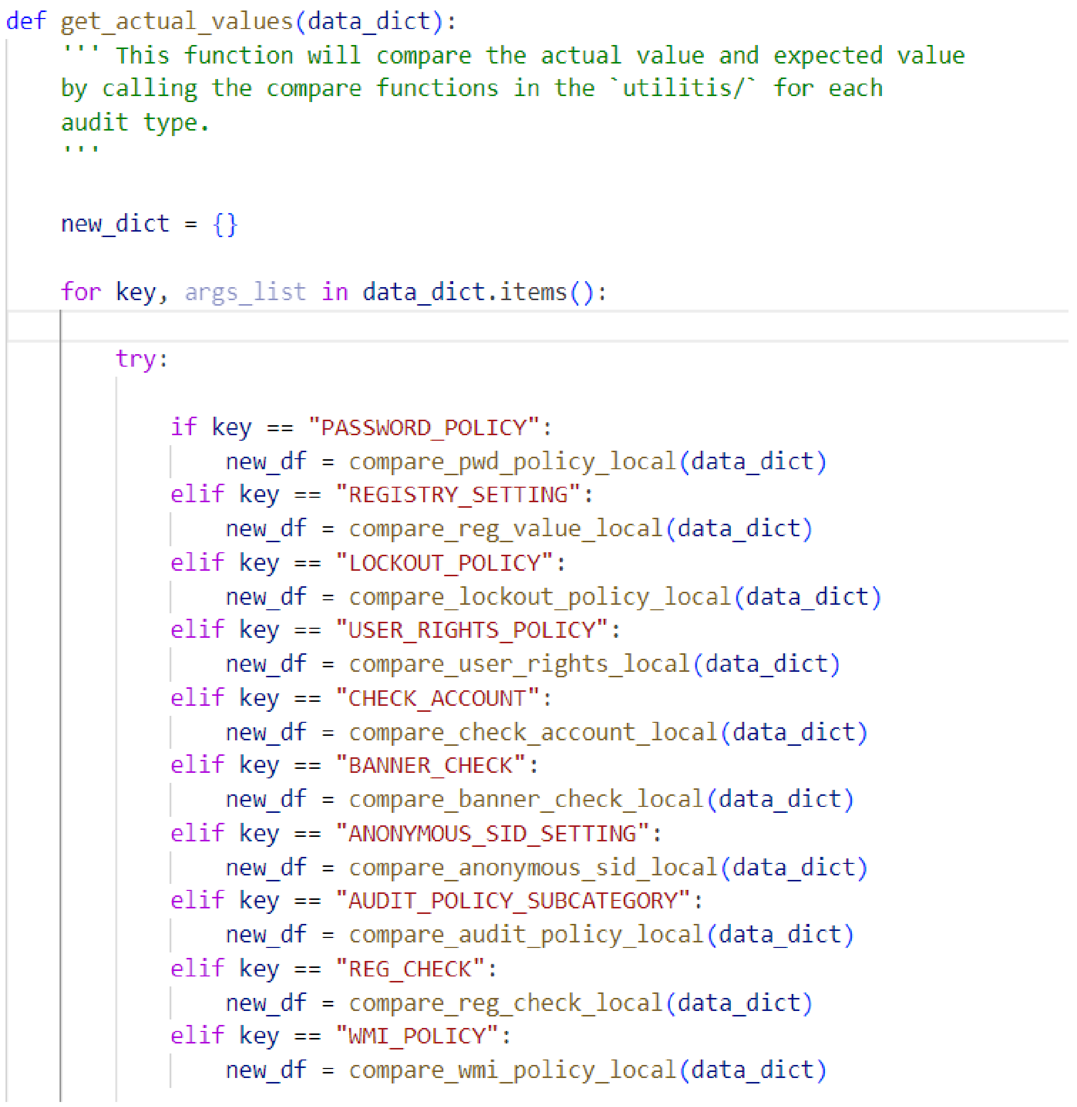
- This is the place to add, delete, and add ‘WMI_POLICY’ commands.



```
elif key == "WMI_POLICY":
    wmi_policy_args = []
    wmi_policy_args_list = []

    for idx, val in enumerate(checklist_values):

        # if val == 1:

        arg = "Write-Output '===='；(Get-WmiObject -Class Win32_ComputerSystem).DomainRole"

        wmi_policy_args.append(arg)

    wmi_policy_args_list.append(';'.join(wmi_policy_args))

    ps_args_dict[key] = ';'.join(wmi_policy_args)
```

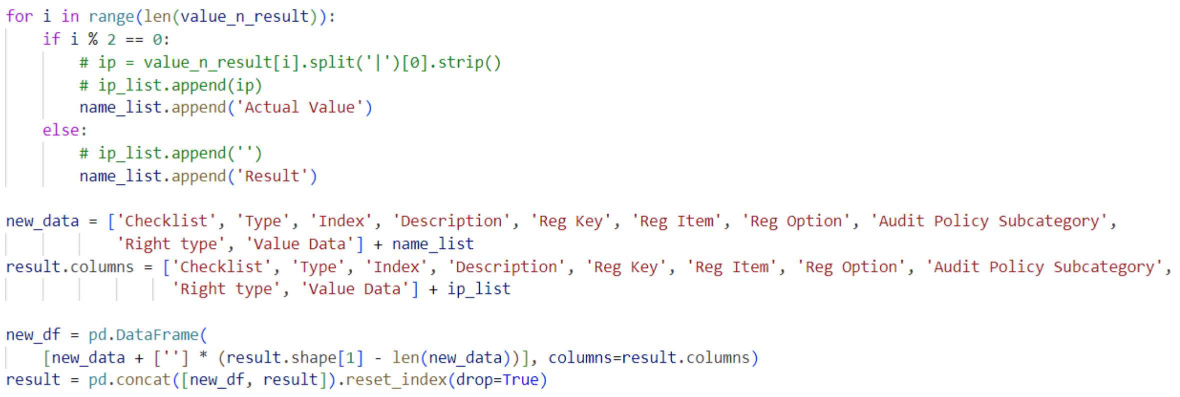
local_audit_results_processor.py

- This is the place to modify the audit type name, if it has been updated in `audit_file_parser.py`.



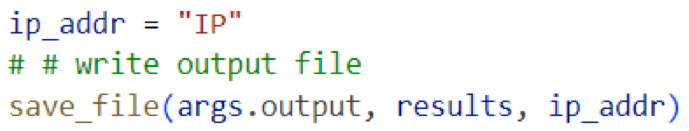
```python
def get_actual_values(data_dict):
    ''' This function will compare the actual value and expected value
    by calling the compare functions in the `utilitis/` for each
    audit type.
    '''

    new_dict = {}

    for key, args_list in data_dict.items():

        try:

            if key == "PASSWORD_POLICY":
                new_df = compare_pwd_policy_local(data_dict)
            elif key == "REGISTRY_SETTING":
                new_df = compare_reg_value_local(data_dict)
            elif key == "LOCKOUT_POLICY":
                new_df = compare_lockout_policy_local(data_dict)
            elif key == "USER_RIGHTS_POLICY":
                new_df = compare_user_rights_local(data_dict)
            elif key == "CHECK_ACCOUNT":
                new_df = compare_check_account_local(data_dict)
            elif key == "BANNER_CHECK":
                new_df = compare_banner_check_local(data_dict)
            elif key == "ANONYMOUS_SID_SETTING":
                new_df = compare_anonymous_sid_local(data_dict)
            elif key == "AUDIT_POLICY_SUBCATEGORY":
                new_df = compare_audit_policy_local(data_dict)
            elif key == "REG_CHECK":
                new_df = compare_reg_check_local(data_dict)
            elif key == "WMI_POLICY":
                new_df = compare_wmi_policy_local(data_dict)
```

- This is the place to modify the column name of the output Excel file. It must be modified if the number of columns has been changed.



```python
for i in range(len(value_n_result)):
    if i % 2 == 0:
        # ip = value_n_result[i].split('|')[0].strip()
        # ip_list.append(ip)
        name_list.append('Actual Value')
    else:
        # ip_list.append('')
        name_list.append('Result')

new_data = ['Checklist', 'Type', 'Index', 'Description', 'Reg Key', 'Reg Item', 'Reg Option', 'Audit Policy Subcategory',
            'Right type', 'Value Data'] + name_list
result.columns = ['Checklist', 'Type', 'Index', 'Description', 'Reg Key', 'Reg Item', 'Reg Option', 'Audit Policy Subcategory',
                  'Right type', 'Value Data'] + ip_list

new_df = pd.DataFrame(
    [new_data + [''] * (result.shape[1] - len(new_data))], columns=result.columns)
result = pd.concat([new_df, result]).reset_index(drop=True)
```

- This is the place to update the IP address column of the output Excel file. (Not necessary)

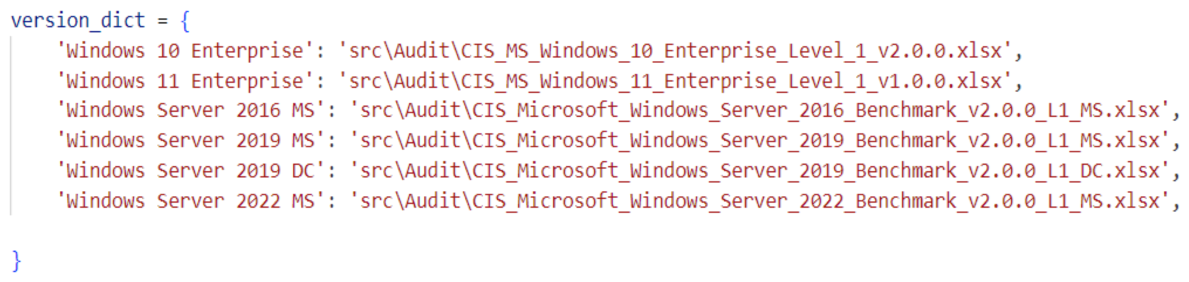


```python
ip_addr = "IP"
# # write output file
save_file(args.output, results, ip_addr)
```

-

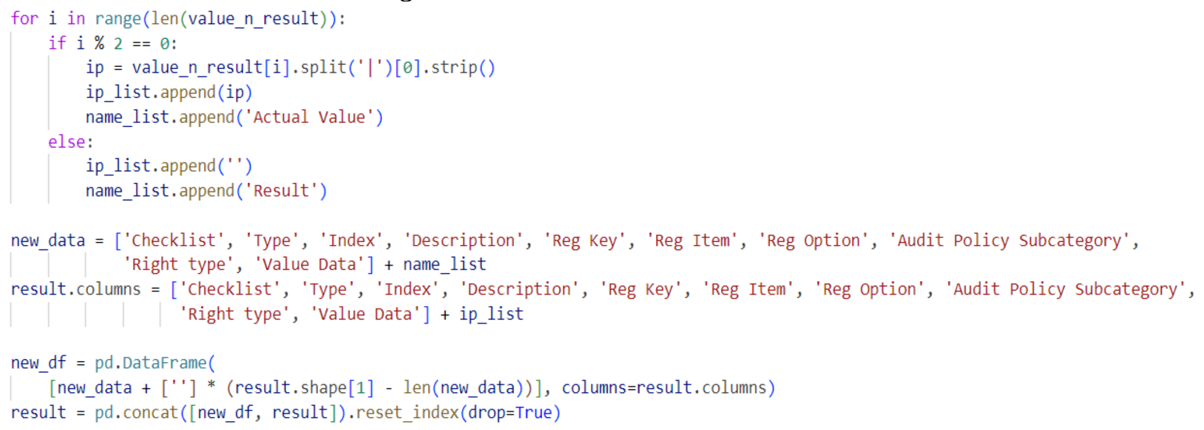
remote_audit_executor.py

- For the modification on `gen_ps_args()`, please refer to `local_audit_command_generator.py`
- This is the place to update Windows version and its corresponding audit file path.



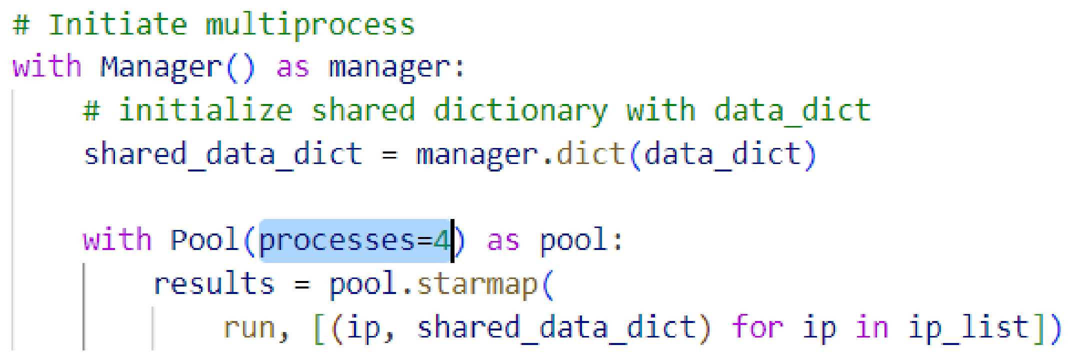
```
version_dict = {
    'Windows 10 Enterprise': 'src\Audit\CIS_MS_Windows_10_Enterprise_Level_1_v2.0.0.xlsx',
    'Windows 11 Enterprise': 'src\Audit\CIS_MS_Windows_11_Enterprise_Level_1_v1.0.0.xlsx',
    'Windows Server 2016 MS': 'src\Audit\CIS_Microsoft_Windows_Server_2016_Benchmark_v2.0.0_L1_MS.xlsx',
    'Windows Server 2019 MS': 'src\Audit\CIS_Microsoft_Windows_Server_2019_Benchmark_v2.0.0_L1_MS.xlsx',
    'Windows Server 2019 DC': 'src\Audit\CIS_Microsoft_Windows_Server_2019_Benchmark_v2.0.0_L1_DC.xlsx',
    'Windows Server 2022 MS': 'src\Audit\CIS_Microsoft_Windows_Server_2022_Benchmark_v2.0.0_L1_MS.xlsx',

}
```

- This is the place to modify the column name of the output Excel file. It must be modified if the number of columns has been changed.



```
for i in range(len(value_n_result)):
    if i % 2 == 0:
        ip = value_n_result[i].split('|')[0].strip()
        ip_list.append(ip)
        name_list.append('Actual Value')
    else:
        ip_list.append('')
        name_list.append('Result')

new_data = ['Checklist', 'Type', 'Index', 'Description', 'Reg Key', 'Reg Item', 'Reg Option', 'Audit Policy Subcategory',
            'Right type', 'Value Data'] + name_list
result.columns = ['Checklist', 'Type', 'Index', 'Description', 'Reg Key', 'Reg Item', 'Reg Option', 'Audit Policy Subcategory',
                  'Right type', 'Value Data'] + ip_list

new_df = pd.DataFrame(
    [new_data + [''] * (result.shape[1] - len(new_data))], columns=result.columns)
result = pd.concat([new_df, result]).reset_index(drop=True)
```

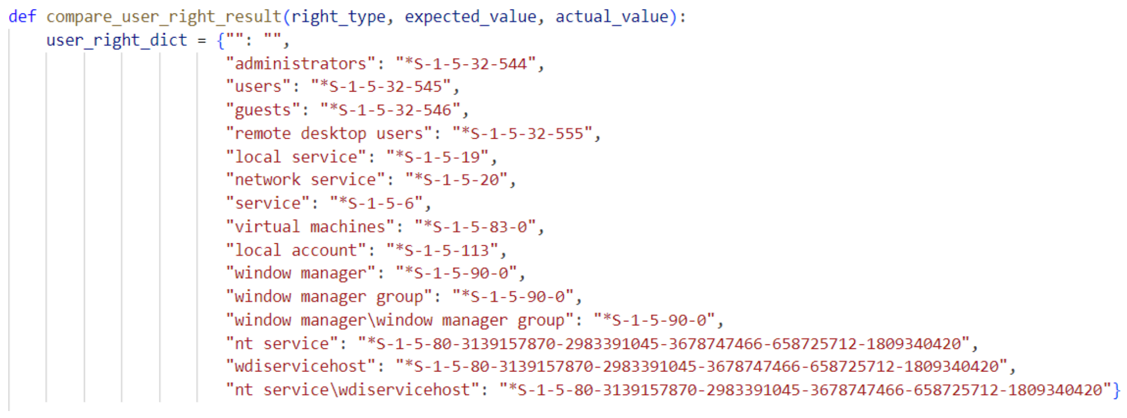
- This is the place to modify the number of processes.



```
# Initiate multiprocess
with Manager() as manager:
    # initialize shared dictionary with data_dict
    shared_data_dict = manager.dict(data_dict)

    with Pool(processes=4) as pool:
        results = pool.starmap(
            run, [(ip, shared_data_dict) for ip in ip_list])
```

utilities\getUserRights.py

- This the place to update use right security identifiers. Please refer to https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers



```python
def compare_user_right_result(right_type, expected_value, actual_value):
    user_right_dict = {"": "",
                       "administrators": "*S-1-5-32-544",
                       "users": "*S-1-5-32-545",
                       "guests": "*S-1-5-32-546",
                       "remote desktop users": "*S-1-5-32-555",
                       "local service": "*S-1-5-19",
                       "network service": "*S-1-5-20",
                       "service": "*S-1-5-6",
                       "virtual machines": "*S-1-5-83-0",
                       "local account": "*S-1-5-113",
                       "window manager": "*S-1-5-90-0",
                       "window manager group": "*S-1-5-90-0",
                       "window manager\window manager group": "*S-1-5-90-0",
                       "nt service": "*S-1-5-80-3139157870-2983391045-3678747466-658725712-1809340420",
                       "wdiservicehost": "*S-1-5-80-3139157870-2983391045-3678747466-658725712-1809340420",
                       "nt service\wdiservicehost": "*S-1-5-80-3139157870-2983391045-3678747466-658725712-1809340420"}
```

PowerShell Commands

This section provides a brief overview of the PowerShell commands used in the security audit. Each audit type uses specific PowerShell commands to retrieve the required information from the target server. Here are some examples:

- REGISTRY_SETTING / BANNER_CHECK / REG_CHECK
    - Get-ItemPropertyValue -Path '{reg_key}' -Name '{reg_item}'
    - e.g., Get-ItemPropertyValue -Path 'HKLM:\\Software\\Microsoft\\Windows NT\\CurrentVersion' -Name "ProductName"

- PASSWORD_POLICY / ANONYMOUS_SID_SETTING
    - if (!(Test-Path -Path C:\temp)) { New-Item -ItemType directory -Path C:\temp }
    - secedit /export /cfg C:\temp\secpol.cfg /areas SECURITYPOLICY
    - $secpol = Get-Content -Path C:\temp\secpol.cfg
    - $secpol | Select-String -Pattern '{subcategory}'
    - e.g., $secpol | Select-String -Pattern "PasswordHistory"

- USER_RIGHTS_POLICY
    - if (!(Test-Path -Path C:\temp)) { New-Item -ItemType directory -Path C:\temp }
    - secedit /export /cfg C:\temp\secpol.cfg /areas user_rights
    - $secpol = Get-Content -Path C:\temp\secpol.cfg
    - $secpol | Select-String -Pattern " right_type"
    - e.g., $secpol | Select-String -Pattern "SeNetworkLogonRight"

- LOCKOUT_POLICY
    - net accounts
    - net accounts | Select-String -Pattern "{subcategory}"

- CHECK_ACCOUNT
    - net user guest
    - net user administrator
    - net user administrator | select-string -pattern "{subcategory}"

- AUDIT_POLICY_SUBCATEGORY
    - auditpol /get /subcategory:'{subcategory}'
    - e.g., auditpol /get /subcategory:"Special Logon"

- WMI_POLICY
    - (Get-WmiObject -Class Win32_ComputerSystem).DomainRole