**TP2 – Configuration Collector Script**

Team lead: Louis Delcourt

Vasile-Alexandru Ifrim

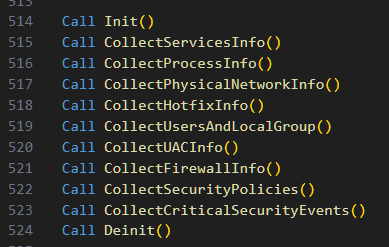
1. **Scope**

The purpose of this script is to collect information about different parts of a Windows machine that can be later used for cold analysis. A resulting CAB archive is output to the root of the C: drive. It is written in Visual Basic 6.0, self-supported and compatible down to Windows Server 2008 R2. This version was tested on Windows Server 2008 R2 and 2012 R2. The current script is capable of collecting information about:

* Generic system information
* Processes
* Physical network
* Applied hotfixes
* Users and local groups
* User Access Control
* Firewall
* Account Policies
* Security Events

1. **How to use**
2. Login as Administrator.
3. Launch the script.
4. From Windows Explorer, navigate to C:\ and find „audit.cab” – this is the archive with collected results.
5. Send the archive to the auditing team.
6. **Architecture**

The script’s architecture is pretty forward, an iterative set of calls that can be seen in the next figure:



Each step from line 515 to 523 involves:

* Opening a new file for the output of the current security objective
* Prepending a header to the file
* Extract relevant information
* When done, closing the file

1. **Implementation**

* Init() and Deinit(): Functions meant to initialize the output folder and, at the end of the script, to output a final .cab file of the collected results and then safely clean any temporary resources.
* CollectServiceInfo(): Query the Win32\_Service class: Service name and display name, start mode, current state. We want to be able to make a list of possibly misconfigured services, outdated or unnecessary.
* CollectProcessInfo(): The function queries the Win32\_Process class to gather data about all processes running on the machine: name, caption, process id, virtual/physical memory size, with reason to get an overview of any excessive resource consumption and suspicious activity.
* CollectPhysicalNetworkInfo(): retrieves data about network adapters, IP configurations, and other network settings.

First, the script queries the Win32\_NetworkAdapter class to gather details about the physical network adapters: name, manufacturer, description, type (e.g., ethernet, wireless), MAC address (unique physical adress), NetConnectionID (ID useful for identifying the adapter in other logs). We want to know what hardware is currently in use to help verify proper network configuration or find out if there is a faulty or rogue component.

The query of Win32\_NetworkAdapterConfiguration collects information for IP enabled adapters: description, DHCP Server (network server that automatically provides IP addresses and other important parameters to client devices) if applicable, IP address if available, subnet mask (helps identify networks), default IP gateway (where the outgoing traffic should be directed), DNS server ( translates URLs into IP adresses), MAC address, WINS server if any ( DNS and WINS can be exploited for man-in-the-middle attacks); all traffic needs to be routed through trusted servers. This information is useful mainly in troubleshooting connectivity problems but it can also show critical network misconfigurations and potential for data leakage or unwanted exposure.

The machine’s network configuration must be secure, correctly set up, and compliant with the set standards.

* CollectHotfixInfo(): hotfixes (patches) are updates meant to fix a specific problem or vulnerability in a software application, often released on short notice to address an urgent issue. By querying Win32\_QuickFixEngineering we gather the computer name, a brief description, hotfix ID, who installed it (whether it was automatic or manually applied) and when it was installed. Being up-to-date with released hotfixes eliminates possibilities of not being compliant with organizational policies and their documentation makes sure we stay transparent in the security management.
* CollectUsersAndLocalGroup(): The query of Win32\_UserAccount retrieves entries of username, domain, status (active, disabled), and the query of Win32\_Group logs group names and the domains associated with the groups. Then, for each group, we query Win32\_GroupUser and log its members (name, domain).

Local groups are used to manage user privileges. Unauthorized or unexpected users in privileged groups can lead to unauthorized access to sensitive resources and functions. Keeping and audit trail of users and group memberships checks the principle of accountability.

* CollectUACInfo(): UAC is a core security feature in Windows that helps prevent unauthorized changes to the system by prompting for administrative permissions. In short, the function reads relevant registry values to determine if the User Access Control feature is enabled, what level it is set to, whether secure desktop is used (a feature through which credentials are being presented only in secure memory locations, and only to trusted system processes).

The function queries parses reg keys to read the values

* EnableLUA - UAC was formerly known as Limited User Account
* ConsentPromptBehaviorAdmin – controls how strictly UAC prompts for administrative changes; values range from 0 - never notify, to 5 - notify when an app tries to make changes.
* PromptOnSecureDesktop – isolate prompts from other running applications; Possible values: 0 - Disables this policy and secure desktop prompting, meaning all credential or consent prompting will occur on the interactive user's desktop; 1 - force all UAC prompts to happen on the user's secure desktop.

At the end, based on the values of ConsentPromptBehaviorAdmin and Prompt-OnSecureDesktop, the script logs a short conclussion of the policies put in place.

* CollectFirewallInfo(): capture general firewall settings and all specific rules (allowed/ blocked apps, protocols, ports).

The function first gets access to the firewall policy interface (Set objFirewall = CreateObject("HNetCfg.FwPolicy2")) from the COM object for Home Net Config Manager, component of Windows. Then, for each profile (1 – Domain, 2 – Private, 4 – Public) the script logs which one is active, if the firewall’s enabled, whether all inbound traffic is blocked, and what is the profile’s default actions for inbound and outbound traffic.

In the second part, we iterate over the firewall rules and log Action ( allow / block), direction, protocol, local ports, profiles to which the rule applies, and application name linked to the rule, if any.

The firewall is a key element in controlling network access, it supports the principle of least privilege and reduces the attack surface.

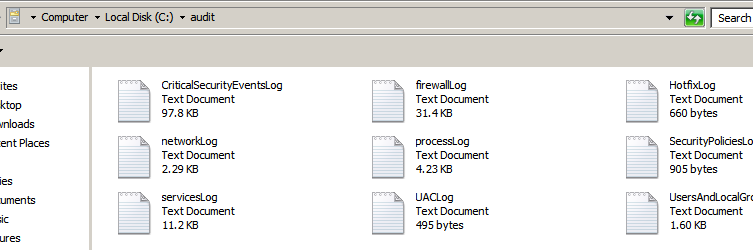
* CollectSecurityPolicies(): system's password and account policies, such as password expiration, password minimum/maximum age, minimum length, and complexity, lockout policies and parameters. This collected data is meant for verifying strong password and account management settings.
* CollectCriticalSecurityEvents(): retrieves and logs security-related events from the Windows Event Log, specifically from the Security log, which contains events such as login attempts, failed authentication, permission or system policies changes.

For each event, we select its event ID, source name ( what app or component generated the event), category, the event description, the time it was generated, the associated user, and the computer name where the event occured. All this information is collected to detect suspicious behaviour, being a source for forensic analysis. Many regulatory frameworks demand monitoring and maintaning a record of security events.

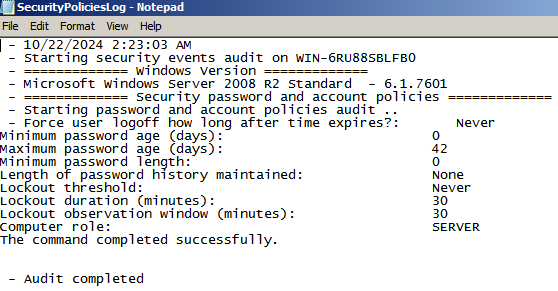
Helper functions:

* Header(): add a header to the currently opened file (objLogFile).
* openFile(file): open and assign a new file to the global variable objLogFile.
* ConvertProfileType(profileType): convert profile types from integer to strings.
* SafeIIf(condition, truePart, falsePart): function for ease of treating if conditions in a shorter inline manner.
* LogMessage(message): function for outputting messages.

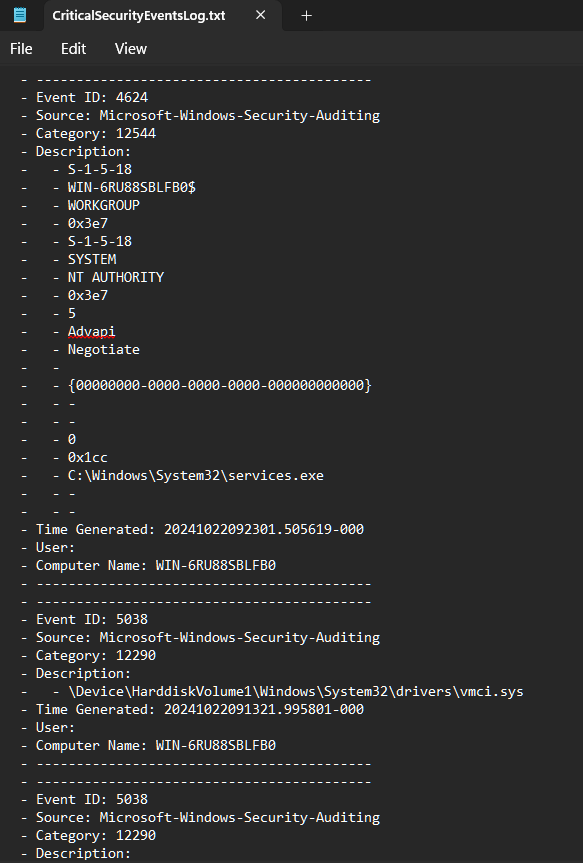
1. **Examples**



Content of the output „audit.cab” file



Example of extracted accounts policies



Extracted security events

1. **References**

* <https://support.microsoft.com/en-us/windows/system-configuration-tools-in-windows-f8a49657-b038-43b8-82d3-28bea0c5666b> Windows configuration tools.
* <https://learn.microsoft.com/en-us/windows/win32/wmisdk/invoking-a-synchronous-query> How to query WMI.
* <https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/win32-quickfixengineering> WMI class for extracting CBS hotfixes.
* [https://learn.microsoft.com/en-us/previous-versions/windows/desktop/eventlogprov/ win32-ntlogevent](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/eventlogprov/%20win32-ntlogevent) WMI class of the Windows Event log.
* <https://learn.microsoft.com/en-us/windows/win32/api/netfw/nn-netfw-inetfwpolicy2> Firewall policies interface.
* [https://github.com/s0h3ck/waps/blob/ master/waps.ps1](https://github.com/s0h3ck/waps/blob/%20master/waps.ps1) – curiosity, later in development, brought us upon WAPS, a modular Powershell script meant to enumerate components and parse information of a Windows machine with scope of a secure audit. It presents many features, in our schema for a collector script we managed to think about and cover only some of them, because our interest was to provide a different solution, written in Visual Basic, taking into consideration a much more extended backwards compatibility. Further improvements can be made, like better error handling and responsability delegation, better design (using decorator coding patterns, RAII principle), as well as development of missing features (BIOS, disk& shares information, DLLs, and more).
* <https://gist.github.com/rheid/c00a3b18ba35aabaecf0> How to create CAB archives.