Windows Admin GUI Model for Learning PowerShell Commands

D. Tuličić*, D. Delija*, G. Sirovatka**, M. Mrkoci**
*Visoka škola za informacijske tehnologije, Zagreb, Hrvatska
**Tehničko veleučilište u Zagrebu, Zagreb, Hrvatska
dtulicic@vsite.hr, ddelija@tvz.hr, gsirovatka@tvz.hr, mmrkoci@tvz.hr

Abstract **This paper presents the development of a platform for the education of system engineers with the aim of insight into detailed data about computers under the MS Windows operating system. The application is based on the ideas of IBM's SMIT (System Management Interface Tool) interface for AIX systems, where standard commands are used through a pre-coded semi-graphical working environment. The developed application uses PowerShell commands and is primarily focused on the security aspects of system administration.**

Keywords Windows - PowerShell, educational application, information security;

I. INTRODUCTION

This paper presents the results of a master's thesis [1] in which an educational software tool was developed as a proof of concept. Even for systems like Microsoft Windows, modern system administration still relies on the skillful and mindful use of command line utilities (CLI). While Windows is based on a graphical user interface (GUI) for ordinary users, for any kind of professional system administration, a command line tool called PowerShell is used. However, from a learning point of view, transitioning from GUI to CLI can be an unusual and challenging path for an average student. To ease this situation and aid learning and task performance, a tool that acts as a safe wrapper around CLI commands with a GUI or semi-GUI interface can be used. If designed and built properly, such a tool has additional advantages. CLI commands are executed through scripts written by seasoned administrators and system programmers. These scripts act as a safe environment around CLI tools, managing arguments, options, inputs, outputs, results, and errors, and all actions are logged and provided with detailed help data and manual pages.

An operating system (OS) is a software program that manages all the other programs and resources on a computer. It acts as an interface between the hardware of the computer and the applications that run on it. In other words, it controls how the computer's hardware interacts with software and users. Examples of operating systems include Microsoft Windows, macOS, and Linux [6]. In the context of the operating system, a system administrator is someone responsible for managing and maintaining a computer system, including its hardware, software, and network infrastructure. They are typically responsible for tasks such as installing and configuring software, monitoring system performance and security, troubleshooting issues, and performing backups and data recovery.

The educational software tool described in the text is designed to help students learn how to perform some of these tasks using command line utilities, which are commonly used by professional system administrators.

Modern system administration still depends on the skillful and mindful use of command line utilities (CLI) even for such systems as Microsoft Windows. Windows are based on a graphical user interface (GUI) for ordinary users but for any kind of professional system administration a command line tool, PowerShell is used. From a learning point, it is an unusual path with a lot of obstacles for an average student. The students, the future system administrators start with GUI tools and then go into CLI with the growth of expertise and complexity of tasks to perform.

To ease that situation and help learn and perform tasks it is possible to use a tool that is a safe wrapper around CLI commands with GUI or semi-GUI interface. Such a tool has additional advantages if is designed and built properly, for example, CLI commands are executed through scripts written by seasoned administrators and system programmers.

The result of this approach is that students can perform serious system administration tasks with confidence, without damaging the system, and with the ability to see the correct syntax and procedures to use CLI. Such an approach to system administration already exists in some older commercial operating systems like IBM AIX, where there is a "System Management Interface Tool" (SMIT) [2] with the same functionality and purpose. As many security tasks are performed by system administrators, particularly data collection required for forensic analyses in incident response situations, it is sensible to create and test SMIT-like tools in the Windows environment, oriented mostly towards forensic data collection tasks.

Many ideas have already been explored using mainly PowerShell or a combination of Python and PowerShell. There are several notable works that are worth mentioning, such as "A Comparison of PowerShell and Python for Systems Administration" by Thomas Rayner and Michael Schneider. This paper compares the use of PowerShell and Python in systems administration and offers valuable background information. Additionally, Timothy Warner's "Designing a PowerShell-based System Administration Framework" investigates the design and implementation of a PowerShell-based framework for systems administration.

System administrators can also use the same tool to obtain information or supervise the system. The application is divided into several parts, from basic commands to more advanced commands such as getting information from networks, hardware, hard drives, and processes within the Windows environment. To implement this idea in the Windows philosophy and environment, CLI will run under the Windows PowerShell Integrated Scripting Environment (ISE), while an auxiliary application called AutoPlay Menu Builder (APMB) is used for the automated use of CLI commands and provides a basis for the command execution.

Although this project may seem simple in scope and purpose, it is too complex for one student to develop. Therefore, the first step was to build a working GUI prototype to prove the ability to interface scripting and GUI environment on Windows, which are both specific in their implementation and not Posix-like or typical business environments where such tools are usually implemented. This first step of implementation is described in [1].

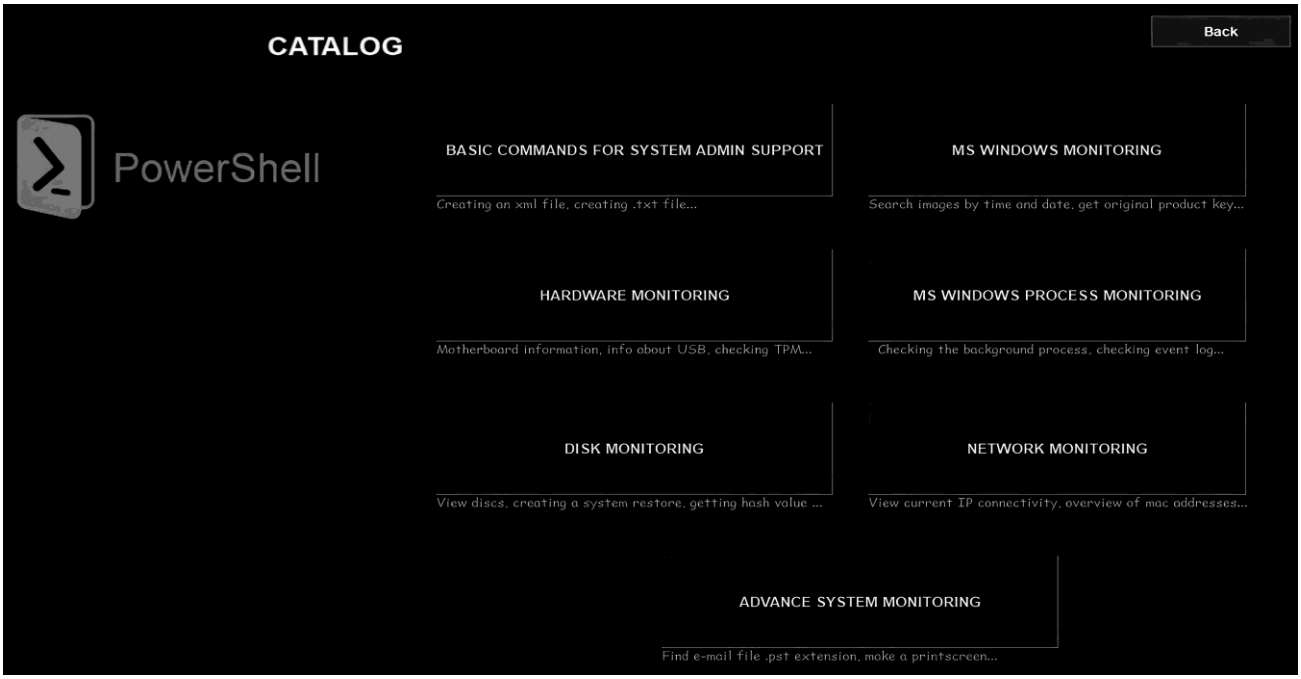

Figure 1. Main toolbar line in the application presetting the top menu, where an area of tasks can be chosen. (author's work).

This paper describes the process of developing an admin GUI application, including the tools and methods used to create the GUI. The GUI ADMIN modules and their functions will be described in detail, along with the underlying CLI commands in PowerShell. Finally, the admin GUI app will be presented, and a roadmap for its future development will be outlined.

II. TOOLS USED

A. PowerShell

PowerShell is a mixture of a command line, a functional programming language, and an object-oriented programming language [3]. It first appeared in 2006 as a replacement for various previous tools. As a tool, it is based on different languages like Python, C#, SQL, Tcl, and Puppet, with a strong resemblance to VMS CLI. The intention is to make PowerShell widely accepted, so it is ported on various other operating systems like macOS 10.12 and higher, Ubuntu 14.04, Debian 8.7+, Fedora 25, and CentOs 7. The first versions were written in the .NET Framework, while the new versions were written in the .NET Core. This intended portability makes it possibly a very useful forensic platform. Open-source software called PowerShell Core has also been released.

To implement proof-of-concept simple and complex cmdlet bar code commands representing the .Net class are used. It is important to note that cmdlets can use scripts, and the same scripts can be incorporated into modules.

With full implementation system administrators can perform tasks at remote locations but also locally because through enabled full access to Windows Management Instrumentation (WMI) and Component Object Model (COM) components tasks can be performed locally and remotely if needed.

B. Autoplay menu builder

The AutoPlay Menu Builder utility is used as a basis for application development. It enables the automatic execution of the PowerShell code in such a way that code is integrated into an executable script with the .exe extension. Implementing PowerShell code does not require programming experience, but all operations are done as drag and drop. There are security features to unauthorized prevent changes. The application supports Unicode standardization and various formats like JPG, PNG, BMP, and GIF, and it can implement video content, adobe reader, and other formats. It also supports a large font base.

C. *ISESteroids*

The environment (ISE) tool is built into the Windows operating system while The ISESteroids module is a commercially available addition that allows the encapsulation of PowerShell code into a self-contained executable. This module is not free but can be used for ten days without any restrictions. It provides many useful features in controlling the security and behavior of PowerShell code, like the application type in which code will be embedded, shell console behavior and user interaction, application signing, etc.

D. *The development process*

In the development and testing process, administrative tasks were chosen as a starting point, because of the importance of these tasks, if there is some serious misconception it is best to see them on the most important elements. Administrative tasks can be broadly separated into general system administration and operating system-specific tasks. This organization leads to three areas

1. basic commands for system admin support and advanced system monitoring;
2. MS Windows monitoring and MS Windows process monitoring;
3. hardware monitoring, disk monitoring, and network monitoring.

Because of the module functionality, each area is implemented as two or more subsets of commands. This organization is presented on the main screen of the tool "Catalog", where administrative tasks are grouped into seven different submenus, as it is shown in Figure 1.

For each chosen area tasks were defined, and appropriate commands were drafted. From this point, scripts were tested and prepared for integration.

After initial debugging and testing, there was an extended set of functional tests to explore the usability of the idea.

III. INTRODUCTION TO THE APPLICATION "EDUCATIONAL TOOL FOR SYSTEM ADMIN MONITORING "

The "Educational tool for system admin monitoring" application serves as a basis for system administrators to analyze, verify, and monitor operating systems. Also, it helps beginners in learning new commands and understands PowerShell's working environment. The application presents commands and information security aspects and a more transparent organization.

Each module presents the collection of information in a different way depending on the command that is predefined during application development (Figure 1). The modules are:

- Basic commands for system admin support,
- MS Windows Monitoring
- Hardware monitoring
- MS Windows process monitoring
- Disk monitoring
- Network monitoring
- Advance system monitoring

Each module or chapter is presented as a list of commands called "Command", followed by "Description," and an overview of commands or their execution.

The overview of commands or execution of commands is displayed in "CMD view" format, which is the basic standard text view, then "Tabular view" and "HTML View" which uses HTML elements to display the results, as it is presented in Figure 2..

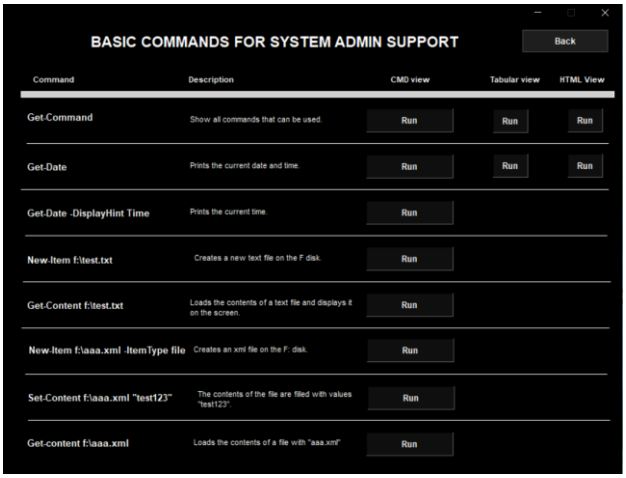

Figure 2 List of all commands on the system, cmdlet output. Each command is available as menu entry (author's work)

A. *Module "Basic commands for system admin support"*

This module will show some basic commands used for system manipulation and administration, how to display the date and time, create a new text record, display records from a text file, and retrieve and create an XML file.

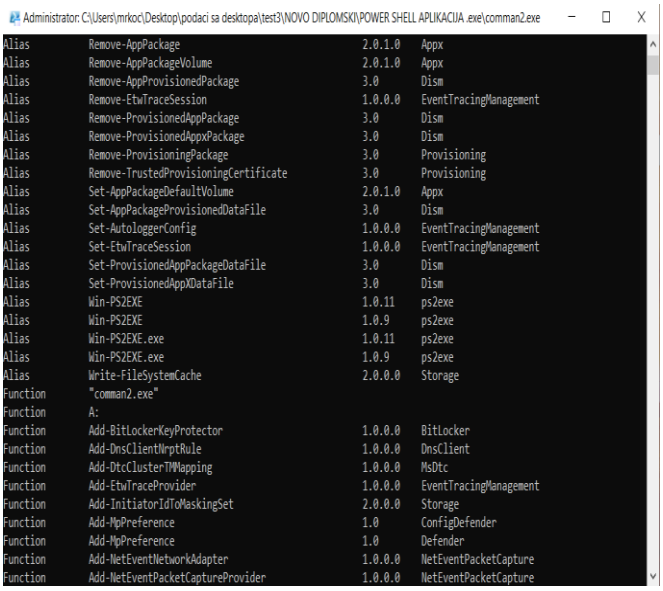

Figure 3 List off all basic commands for the system administrator support. (author's work)

As an example, Get-Command interface is presented.

1) Get-command module

Command: Get-Command

Synopsis: Displays all commands that are integrated on the local computer.

Description: Through this command, it is possible to get all command types as an alias, function, and Cmdlet commands. Results are presented in Figure 3.

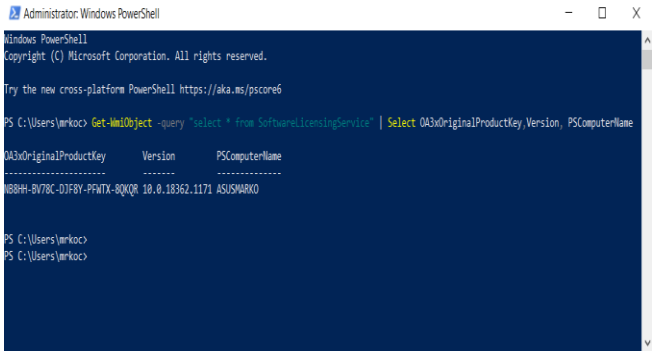

Figure 4 Windows product key as output (author's work)

Outputs: list of all available commands is printed.

In this module there are also other implemented commands:

- Get-Date – to get the system date
- Get-Content – to get file content
- Set-Content – set file content

2) Information about computer

Command: Get-WmiObject -query "select * from SoftwareLicensingService"|Select OA3xOriginalProductKey,Version, PSComputerCommand

Synopsis: This command finds the Windows key, computer name, and version.

Description: In this command, as we can see, it used the predefined commands "SoftwareLicensingService" and "OA3xOriginal ProductKey" to get the licensed key from the operating system.

Outputs: Windows product key as it is presented in Figure 4, it is in native blue background for PowerShell.

3) Hardware monitoring

Command: „Get-WmiObject Win32_Bios; Get-WmiObject win32_baseboard;Select-Object Banklabel, Manufacturer,Configuredclockspeed,Devicelocator,Capacity,Serialnumber|Get-WmiObjectwin32_physicalmemory; Get-ComputerInfo OsArchitecture,OSName“

Synopsis: Show information about the motherboard and the series and model.

Description: It can be seen that "Get-WmiObject" is used here to access digital information. With "Select-Object" we define which objects we want to display. Furthermore, the essential elements required can be seen, such as "GetComputerInfo OSName," and "OsArchitecture."

Outputs: are presented in Figure 5.

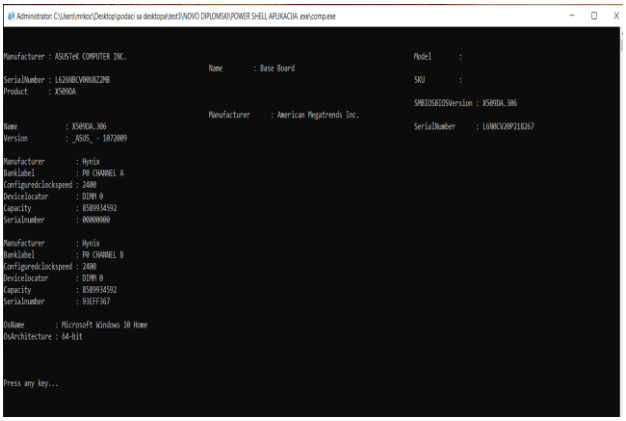

Figure ‘5 Information about the motherboard. All available info from components itselfs and registry data (author's work)

IV. FURTHER WORK

Since it is too complex to develop a whole application, it was decided to start with testing if the basic idea and needed elements are working appropriately and securely.

Examples listed here show that the concept is working and can be used for the intended purpose. There is a list of important functionalities to be implemented in the same test and proof fashion. As AIX SMIT [4] was used as a raw model additional elements needed are

- Database to store scripts, templates, and rules
- Templates for PowerShell code to be used
- Additional cmdlets for interfacing with OS and GUI
- Detailed logging facility
- Installation method
- Help subsystem
- Documentation is needed for development, usage, testing, and installation.

All these elements are essential for such a tool to be successfully and securely used, especially in incident response / digital forensic data acquisitions situation.

PowerShell as the interpreter is well suited to be sued as an interface and glue to keep all elements together. To avoid usual security problems with scripts with maximum defensive scripting and well-defined templates for the code it is possible to create secure scripts.

A. Database

The database is the backbone of such a tool, it stores code to be executed – commands, rules, and relations among this code, both with information about errors, and arguments [2]. Since the tool has to be as open as possible because of other OS support, the database should be one of the available lightweight databases with multiplatform support, also with a good interface to various programming languages [9]. The SQLite is very well suited for such a task and fulfills all these requirements

well, even though it is in extensive use on Windows from various applications and system tools [5].

B. Command templates

To keep reasonable system security and ease of maintenance command templates can be used in designing shell scripts, in this case, PowerShell commands. In such a way shape of the code, input, and command arguments can be standardized and tested before being consumed by the script itself. With this approach each script before execution is checked for validity and if approved it is executed.

C. Logging facility

Logging is essential for keeping records about events on the system and through this for general system security and reliability. As there are different logging approaches on the Windows operating system and other operating systems supported now by PowerShell, it is crucial to have a reliable and uniform interface to all logging subsystems included, to Windows logging g facility, and to syslog found on Unix-based systems.

Data that should be logged have to include:

- commands executed and their unique ids
- timestamps related to the start and stop of the commands
- arguments and inputs of the commands
- results and outputs of the commands
- debugging information if necessary
- account information under which commands were executed.

All this will provide an additional level of security and support for chain-of-evidence especially if security features like signing codes are used.

Because of the huge difference between Windows and syslog, it is possible that an additional log system should be used, where some raw data like outputs can be stored [2]. This is quite a common situation, especially on the Windows platform itself, where additional information about system events and history are stored in plain text files, while structured records are still logged into the windows logging system. The USB history is one such example. The abovementioned database can also be used as a facility to store logs, to avoid log data being too easily exposed to possible mishandling or change.

D. Utility and Interface cmdlets

To provide an efficient interface among PowerShell scripts executing tasks and the rest of the system and tool interface layer is needed, which in combination with script templates provides the required functionality. In essence, the current scripts presented in the examples in this paper require an additional wrapper layer that will provide an interface to the database and the system, while providing an environment for script execution, logging of results, presenting and preserving outputs, and all other tasks.

For example, while a task is selected through GUI, parameters, inputs, and environment data are generated for the code retrieved from the database, all this interaction is done by interface cmdlets and related utilities. The same is for handling script execution, controlling errors and results, and storing and logging generated data and script results. In essence, this layer behaves like a secure wrapper around scripts that change and modify the system keeping all tested and well-behaved.

It is important to notice that through such an approach, e.g., executable code stored in the database structures and then executed by another layer of shell code, it is possible to simplify maintenance and control of the code, resulting in more security. From a system security point of view, it adds a layer of security usually missing when system administration is done troughs scripts.

An additional benefit of such an approach is the ability to see code on request, which will run with all its arguments and variables set up, which gives huge insight into scripting practices and methods for novice system administrators, or anyone using such a system.

Development of such PowerShell code requires high programming skills and fluency in object programming, the Windows operating system, and secure coding, so it is obvious why such code was left to be implemented in further steps of development.

E. Installation Method

Installation and maintenance methods are usually overlooked in the case of system administration scripts, it is often done as simple archive extraction because of expected simplicity and “private” tools being distributed among machines. This is a huge security risk often overlooked. To avoid such a situation strict packaging into standard installation formats should be done. Since PowerShell is now available on Linux and on native Windows systems both ranges of installation packages have to be created. Again, this is a complex task same as developing the utility and interface cmdlets layer and that is why it was not implemented in the proof-of-concept tool. The current implementation is the simplest form for installation and executable which is transportable among systems.

The full implementation of the installation method will be chosen at the end of tool development, based on how other parts of the system will be implemented and how that will influence the whole tool [8]. To clarify this, the whole thing depends on the database where code and many other components are stored, so installation of the tool must incorporate the reliable installation of the database, its content, and interface layer as a set of packages. The same situation is with maintenance and upgrades where it is not yet clear which methods of updating and maintenance should be used, this heavily depends on the method of installation.

F. Help and Documentation

The help and documentation subsystem is an important part, which should be developed into a resilient and platform-independent format. Since there is a direct link between commands and help information and documentation, scripts should be developed with unique ids to ease connecting them with help files and documentation [8]. Scripts id can be based on various

identification systems, important is that each script has a unique id in each identification system, while the script still can have more than one id.

Relations among various parts of help and documentation should also be kept in the database, related to the code itself. The utility layer and GUI should be able to access help data and documentation straight from the tool itself during work, also a script code with its arguments and parameters should be visible through the GUI as part of the help system. This will also facilitate learning because the user can see the code which runs and how it interacts with the system [7].

V. CONCLUSION

As can be seen, using PowerShell and its scripts can significantly automate system processes to obtain timely information or information about a business information system. Also, the application is designed to enable the use of PowerShell commands quickly and easily, both in learning and education and in forensic research and analysis, which allows faster case resolution.

The application showed more straightforward commands such as calling only one command where an insight into the basics was obtained to complex scripts with multiple lines of code. In the following, the application is ready for possible upgrades and extensions to add more complex scripts to obtain quality information and design in the display of the same.

REFERENCES

[1] M. Mrkoci: “An educational tool for system admin monitoring using the PowerShell in the Windows environment”, Zagreb University of Applied Sciences, 2022.

[2] S. Segura: System Management Interface Tool (SMIT), IBM Redbooks, IBM, 30.11.2000

[3] C. Dent: “Mastering Windows PowerShell Scripting”, Packt Publising, 2019.

[4] IBM AIX documentation

[5] G. Allen, M. Owens: The Definitive Guide to SQLite, Apress, 2010.

[6] G.Silberschatz,,Galvin, P.,Gagne, G.: “Operating System Concepts” (10th ed.). John Wiley & Sons, 2018

[7] E.Holscher: “The Art of Documentation for Software Engineers”, Pragmatic Bookshelf, 2020

[8] W.Horton: “Developing Online Help for Computer Applications”, Wiley Year: 1991 (revised in 2003)

[9] J.Kreibich: "Using SQLite: Small. Fast. Reliable. Choose Any Three.", O'Reilly Media, 2010