МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ ТА НАЛАШТУВАННЯ РОБОЧИХ СТАНЦІЙ ЗА ДОПОМОГОЮ СКРИПТІВ POWERSHELL

Виконав:
студент групи 2К-22
спеціальності
123 Комп’ютерна інженерія
Денис БРОВКО
Керівник:
Наталя ФАЛЬЧЕНКО

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій
Спеціальність 123 «Комп’ютерна інженерія»
Освітня програма Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувачка циклової комісії КІ та ІТ
Наталя ФАЛЬЧЕНКО
«_____» _____2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Бровку Денису Дмитровичу

1. Тема кваліфікаційної роботи: Автоматизація розгортання та налаштування робочих станцій за допомогою скриптів powershell

Керівник роботи: Фальченко Наталя Григорівна, викладач вищої категорії затверджені наказом закладу фахової перед вищої освіти від «6» жовтня 2025 року № 84/1-у.

2. Строк подання студентом кваліфікаційної роботи: 08.06.2026

3. Вихідні дані до кваліфікаційної роботи: Наукова та технічна література з автоматизації ІТ-інфраструктури; характеристики та архітектурні особливості корпоративних операційних систем Windows 10/11; технічна документація щодо архітектури PowerShell.

4. Зміст кваліфікаційної роботи: Поняття та архітектура систем автоматизованого розгортання робочих станцій; основні причини часових та ресурсних втрат при ручному адмініструванні; аналіз методів та технологій оптимізації та захисту операційних систем у корпоративному середовищі; розробка комплексу модульних скриптів автоматизації та логування на базі оболонки PowerShell.

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	Розділ 1 Аналіз сучасних методів та засобів керування парком робочих станцій	09.12.2025	
3	Розділ 2 Проєктування архітектури системи автоматизованого розгортання	10.03.2026	
4	Розділ 3 Практична реалізація скриптів автоматизації та налаштування	28.04.2026	
5	Висновки	12.05.2026	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	05.06.2026	
8	Подання кваліфікаційної роботи на затвердження завідувачу ЦК (за 7 дніів до захисту)	08.06.2026	

Студент Денис БРОВКО

Керівник роботи Наталя ФАЛЬЧЕНКО

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці та впровадженню комплексної системи автоматизації розгортання робочих станцій у корпоративному IT-середовищі. Метою роботи є створення відмовостійкого програмного комплексу на базі скриптів PowerShell для оперативного та стандартизованого конфігурування комп'ютерної техніки. Зосереджено увагу на надійності виконання команд, автоматизації процесів інсталяції, а також забезпеченні жорстких корпоративних політик кібербезпеки.

Робота містить 62 сторінок, 5 таблиць, 11 рисунків, 3 додатки та 20 джерел за переліком посилань.

У процесі проєктування створено гнучку модульну архітектуру, що включає головний керуючий скрипт та набір незалежних конфігураційних блоків: модуль базового налаштування ОС, встановлення програмного забезпечення, оптимізації інтерфейсу та зміцнення захисту. Процес інсталяції прикладних програм реалізовано через нативний пакетний менеджер Winget із підтримкою JSON-конфігурацій, що дозволило досягти високої стабільності без ручного втручання. Для аудиту результатів розроблено інтегровану систему структурованого логування подій.

Проведено тестування роботи розробленого комплексу в ізольованому віртуалізованому середовищі, що імітує типові сценарії експлуатації. Підтверджено високу відмовостійкість архітектури за рахунок перехоплення помилок, перевірено автоматичну активацію механізмів захисту, а також здійснено оптимізацію роботи ОС шляхом видалення вбудованих надлишкових компонентів. Визначено, що впровадження комплексу кардинально скорочує час підготовки робочих місць, гарантує стандартизацію інфраструктури та усуває ризики помилок ручного адміністрування.

Ключові слова: автоматизація розгортання, PowerShell, інфраструктура як код, системне адміністрування, скриптинг, Winget, кібербезпека кінцевих точок, модульна архітектура.

ABSTRACT

This thesis focuses on the development and implementation of a comprehensive system for automating the deployment of workstations within a corporate IT environment. The aim of the thesis is to create a fault-tolerant software suite based on PowerShell scripts for the rapid and standardised configuration of computer hardware. The focus is on the reliability of command execution, the automation of installation processes, and the enforcement of strict corporate cybersecurity policies.

The thesis comprises 62 pages, 5 tables, 11 figures, 3 appendices and 20 references.

During the design process, a flexible modular architecture was created, comprising a main control script and a set of independent configuration blocks: a module for basic OS configuration, software installation, interface optimisation, and security hardening. The application installation process is implemented via the native Winget package manager with support for JSON configurations, which has enabled high stability to be achieved without manual intervention. An integrated structured event logging system has been developed to audit the results.

The developed system was tested in an isolated virtualised environment that simulates typical operational scenarios. The architecture’s high fault tolerance was confirmed through error handling, the automatic activation of security mechanisms (including BitLocker and Microsoft Defender configuration) was verified, and the operating system’s performance was optimised by removing built-in redundant components (bloatware). It has been established that the implementation of the system drastically reduces the time required to prepare workstations, ensures infrastructure standardisation, and eliminates the risks of errors associated with manual administration.

Keywords: deployment automation, PowerShell, infrastructure as code, system administration, scripting, Winget, endpoint cybersecurity, modular architecture.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та цифровізації бізнес-процесів ефективне управління IT-інфраструктурою стало вирішальним фактором успіху будь-якої організації. Основу цієї інфраструктури становлять робочі станції користувачів, які потребують регулярного розгортання, оновлення, налаштування та постійної технічної підтримки. З розширенням підприємств, переходом до гібридних моделей роботи та постійною плинністю кадрів традиційні методи ручної інсталяції операційних систем (ОС) та прикладного програмного забезпечення виявляються вкрай неефективними.

Ручна конфігурація сотень кінцевих точок не тільки вимагає колосальних затрат часу від IT-фахівців, але й неминуче призводить до так званого “людського фактора”. Це призводить до помилок, невідповідностей у налаштуваннях і, як наслідок, до появи критичних вразливостей у системі інформаційної безпеки підприємства. Крім того, в умовах постійно зростаючих кіберзагроз швидкість стандартизованої реакції служби підтримки має першочергове значення. Ручна зміна політик на кожному комп’ютері окремо є неприйнятно повільним процесом, що суперечить фундаментальним принципам надання високоякісних IT-послуг.

Вирішення цієї проблеми полягає у відході від парадигми ручного адміністрування на користь концепції “Інфраструктура як код”. Серед широкого спектру інструментів автоматизації, доступних для екосистеми Microsoft Windows, безперечним лідером є оболонка та мова сценаріїв PowerShell. Завдяки об’єктно-орієнтованій архітектурі, глибокій інтеграції з ядром ОС, доступу до Windows Management Instrumentation (WMI) та .NET Framework, PowerShell надає адміністраторам безпрецедентні можливості для управління конфігурацією. Використання PowerShell дозволяє реалізувати

підхід “Light-Touch Deployment”, що забезпечує узгодженість налаштувань на всіх робочих станціях та мінімізує втручання людини.

Об’єктом дослідження є розгортання, базової конфігурації та системного адміністрування парку робочих станцій під управлінням операційних систем на базі Windows у корпоративному IT-середовищі.

Предметом дослідження є методи, алгоритми та програмні засоби для автоматизації рутинних завдань технічної підтримки з використанням мови сценаріїв PowerShell.

Мета кваліфікаційної роботи полягає у розробці та практичному впровадженні комплексної системи скриптів на базі PowerShell, призначеної для автоматизації процесів підготовки робочих станцій, що дозволить підвищити надійність інфраструктури, стандартизувати робочі місця та оптимізувати витрати часу IT-відділу.

Для досягнення поставленої мети в роботі сформульовано та вирішено такі основні завдання:

– Провести аналіз сучасного стану проблеми управління парком комп'ютерної техніки та оцінити недоліки існуючих методів ручного конфігурування кінцевих точок.

– Дослідити архітектурні особливості технології PowerShell та протоколів віддаленого керування (WinRM) як ключових інструментів автоматизації.

– Сформувати вимоги до цільової конфігурації стандартизованої робочої станції з урахуванням сучасних політик безпеки та потребам користувачів.

– Розробити модульну структуру сценаріїв для автоматичного виконання етапів налаштування ОС, встановлення необхідного програмного забезпечення (з використанням пакетних менеджерів) та конфігурування мережевих параметрів.

– Реалізувати надійні механізми обробки помилок та логування дій скрипта для забезпечення контролю за процесом розгортання.

– Створити тестовий стенд у віртуалізованому середовищі для перевірки працездатності розроблених рішень та проведення оцінки їхньої економічної і часової ефективності.

У процесі виконання роботи застосовувався комплекс загальнонаукових та спеціальних методів: системний аналіз - для вивчення архітектури ОС; методи алгоритмізації та структурного програмування - під час написання коду сценаріїв; експериментальні методи та моделювання - для розгортання віртуального тестового середовища (на базі гіпервізорів) та практичної перевірки ефективності створеного програмного продукту.

Практичне значення отриманих результатів. Розроблений комплекс PowerShell-сценаріїв має високий ступінь готовності до впровадження. Запропоновані рішення можуть бути безпосередньо використані фахівцями технічної підтримки та системними адміністраторами для оптимізації щоденної роботи. Впровадження результатів дозволяє скоротити час підготовки одного робочого місця на 70-80%, гарантує відповідність конфігурації корпоративним стандартам і значно знижує навантаження на IT-персонал.

Апробація результатів кваліфікаційної роботи здійснювалася шляхом їх представлення на XVIII Всеукраїнській науково-практичній конференції «Тенденції розвитку IT-технологій в Україні», 23-24 квітня, м. Черкаси. Тема доповіді: "Використання сценаріїв powershell для налаштування робочих станцій "

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ КЕРУВАННЯ ПАРКОМ РОБОЧИХ СТАНЦІЙ

1.1 Проблематика ручного налаштування систем у корпоративному середовищі

У сучасному корпоративному середовищі IT-інфраструктура є основою безперебійної роботи будь-якої організації. У міру зростання компанії її парк комп'ютерного обладнання неминуче розширюється, що вимагає регулярного розгортання, обслуговування, оновлення та стандартизації. Традиційний підхід до налаштування нових робочих станцій передбачав ручну інсталяцію операційної системи, необхідних драйверів, пакетів оновлень та програмного забезпечення. Однак, з огляду на сучасні вимоги до швидкості бізнес-процесів та інформаційної безпеки, ручна конфігурація робочих станцій стала критичним вузьким місцем, що породжує низку серйозних інфраструктурних проблем [1].

Низька ефективність використання ресурсів та часу. Першою та найочевиднішою проблемою є вкрай неефективне використання робочого часу IT-фахівців. Процес ручного розгортання та виконання базової конфігурації однієї робочої станції може зайняти від кількох годин до цілого робочого дня. Це включає встановлення операційної системи, завантаження останніх оновлень, налаштування регіональних параметрів, приєднання комп'ютера до домену Active Directory та ручне встановлення конкретних корпоративних програм. Коли виникає потреба налаштувати десятки робочих станцій одночасно (наприклад, під час відкриття нового відділення), ручний підхід призводить до значних фінансових витрат для бізнесу [4]. Замість виконання складних архітектурних завдань кваліфікований інженерний персонал витрачає цінний час на монотонні операції.

Вплив "людського фактора" та зміщення конфігурації. Другою критичною проблемою є неминучий вплив людського фактора. Під час

виконання однакових дій, що повторюються сотні разів, увага адміністратора неминуче знижується. Це призводить до помилок: пропущених налаштувань конфіденційності, неправильно вказаних мережевих параметрів або встановлення некоректних версій ПЗ. Це породжує явище, відоме як “зміщення конфігурації” [2]. Зміщення означає, що з часом налаштування комп'ютерів, які спочатку мали бути ідентичними, починають суттєво відрізнятися. Відсутність стандартизації робить інфраструктуру непередбачуваною: скрипт або групова політика (GPO) може коректно спрацювати на половині комп'ютерів і викликати критичні збої на іншій через непомітні розбіжності в їхніх локальних налаштуваннях.

Третій аспект проблематики тісно пов'язаний з кібербезпекою підприємства. Робочі станції користувачів традиційно залишаються найуразливішою ланкою мережі. У разі ручного конфігурування неможливо гарантувати стовідсоткове застосування всіх політик безпеки: вимкнення вразливих протоколів (наприклад, SMBv1), активації шифрування дисків BitLocker чи встановлення актуальних засобів Endpoint Detection and Response (EDR) [3]. Пропуск хоча б одного критичного оновлення під час ручного розгортання створює "дірку" в безпеці, через яку зловмисники можуть скомпрометувати всю корпоративну мережу. Більше того, неможливість швидкого масового реагування на нові вразливості нульового дня робить компанію фактично беззахисною.

Обмеження масштабованості та виклики в гібридному робочому середовищі. Варто звернути увагу на проблеми, пов’язані з сучасними тенденціями до гібридних та віддалених моделей роботи [6]. Співробітники можуть перебувати за межами периметра корпоративної мережі, що унеможливлює фізичний доступ адміністратора до їхніх пристроїв для ручного налаштування. Крім того, паперові або текстові інструкції (переліки) щодо розгортання швидко застарівають. Процес адаптації нових системних адміністраторів ускладнюється відсутністю єдиного джерела достовірної

інформації, оскільки знання про конкретні деталі конфігурації часто зберігаються лише в пам’яті окремих співробітників.

Підсумовуючи, можна стверджувати, що ручна конфігурація системи є застарілим підходом, який знижує рівень кібербезпеки та перешкоджає масштабуванню бізнесу. Подолання цих викликів вимагає обов’язкового впровадження концепції “Інфраструктура як код”, за якої всі процеси розгортання описуються програмно та виконуються автоматично, що гарантує повну узгодженість результату [2].

1.2 Огляд існуючих систем автоматизації

Для подолання недоліків ручного адміністрування, описаних у попередньому підрозділі, індустрія інформаційних технологій розробила низку спеціалізованих систем керування конфігураціями та розгортання операційних систем. Вибір конкретного інструменту залежить від масштабів підприємства, архітектури мережі (локальна, хмарна або гібридна) та наявного бюджету. Розглянемо найбільш поширені рішення корпоративного рівня [19].

Класичний і найбільш потужний інструмент від Microsoft для керування великими локальними інфраструктурами на базі ОС Windows - Microsoft Endpoint Configuration Manager (MECM, раніше SCCM). MECM забезпечує повний цикл автоматизації: від розгортання ОС "на голому залізі" за допомогою середовища попереднього завантаження до встановлення оновлень, керування ліцензіями та розповсюдження прикладного ПЗ [6]. Незважаючи на колосальні можливості, MECM має суттєві недоліки (табл. 1.1). По-перше, це висока вартість ліцензування. По-друге, система вимагає розгортання складної серверної інфраструктури (сервери баз даних, точки розповсюдження тощо) та наявності вузькопрофільних спеціалістів для її обслуговування.

Таблиця 1.1 - Порівняльний аналіз систем керування конфігураціями

Назва системи	Архітектура / Модель роботи	Основні переваги	Головні недоліки
MECM (SCCM)	Локальна, агентна	Повний цикл розгортання ОС, глибокий контроль	Висока вартість, складна серверна інфраструктура
Microsoft Intune	Хмарна (MDM)	Ідеально для гібридної роботи, Zero Trust, Autopilot	Залежність від підписки, обмеження в ізольованих мережах
Ansible	Локальна, безагентна	Декларативний підхід (YAML), відсутність агентів	Вимагає керуючого сервера на базі ОС Linux
Puppet	Локальна, агентна	Суворий контроль зміщення конфігурацій	Навантаження від фонових агентів, специфічна мова
Власні скрипти PowerShell	Локальна / Віддалена (WinRM)	Безкоштовність, абсолютна гнучкість, немає агентів	Вимагає експертизи в написанні та підтримці коду

У міру переходу підприємств на хмарні технології та модель “Zero Trust” компанія Microsoft активно розвиває Intune - хмарний сервіс для управління мобільними пристроями та додатками. Intune ідеально підходить для сучасної гібридної моделі роботи, оскільки дозволяє налаштовувати робочі станції незалежно від їхнього фізичного розташування, використовуючи лише підключення до Інтернету [6]. Технологія Windows Autopilot, інтегрована з Intune, дозволяє автоматично налаштовувати пристрої одразу після розпакування. Однак до недоліків Intune належать залежність від хмарної передплати, обмежені можливості при роботі в ізольованих мережах, а також

іноді менша гнучкість у виконанні складних багатоетапних сценаріїв порівняно з локальними скриптами.

Це популярна безагентна система автоматизації з відкритим вихідним кодом, розроблена компанією Red Hat. Ansible використовує декларативний підхід: адміністратор описує бажаний стан системи у файлах формату YAML, а система автоматично приводить цільовий вузол до цього стану [19]. Для керування Windows-машинами Ansible використовує протокол віддаленого керування WinRM. Головною перевагою є відсутність необхідності встановлювати агенти на кінцеві точки. Однак, традиційно керуючим вузлом для Ansible має бути сервер на базі ОС Linux, що створює додатковий бар'єр для IT-відділів, які працюють виключно в екосистемі Microsoft.

Puppet є потужним інструментом керування конфігураціями, який зазвичай працює за агентною моделлю. На кожну робочу станцію встановлюється спеціальна програма-агент, яка періодично звертається до центрального сервера, завантажує актуальні конфігураційні маніфести та застосовує їх [19]. Це гарантує суворий контроль за дотриманням корпоративних стандартів і запобігає зміщенню конфігурацій. Основним недоліком Puppet є необхідність вивчення специфічної мови програмування та додаткове навантаження на систему через роботу фонових агентів.

Хоча корпоративні рішення, такі як MECM або Intune, пропонують широкий функціонал, їх впровадження не завжди є економічно вигідним для малих та середніх підприємств. Крім того, важливо відзначити фундаментальний факт: при виконанні складних, нестандартних завдань MECM, Intune та Ansible “під капотом” використовують скрипти PowerShell [2]. Таким чином, розробка власної системи автоматизації на основі модульних скриптів PowerShell забезпечує гнучке, безкоштовне рішення, яке є настільки ж ефективним, як і комерційні альтернативи, повністю задовольняючи потреби IT-інфраструктури у стандартизації та швидкому розгортанні робочих станцій.

1.3 Переваги та можливості PowerShell як інструменту системного адміністрування

Еволюція засобів системного адміністрування в екосистемі Microsoft пройшла довгий шлях від простих пакетних файлів та VBScript до створення потужного, сучасного інструментарію. Сучасний етап управління конфігураціями та автоматизації неможливо уявити без PowerShell. Спочатку концептуалізований у маніфесті “Monad” архітектором Microsoft Джеффрі Сновером, PowerShell був створений з метою подолання фундаментальних обмежень традиційних командних оболонок та надання адміністраторам інструменту, що за потужністю не поступається мовам програмування високого рівня [1]. Сьогодні PowerShell є не просто оболонкою, а комплексною платформою автоматизації та управління конфігураціями, тісно інтегрованою з операційною системою Windows та платформою .NET. Використання PowerShell для автоматизації розгортання та налаштування робочих станцій обґрунтовується низкою його унікальних архітектурних, функціональних та безпекових переваг.

Фундаментальною і найбільш значущою перевагою PowerShell є його об'єктно-орієнтована природа. Традиційні оболонки (наприклад, Bash у UNIX-подібних системах або cmd.exe) функціонують на основі обробки неструктурованого тексту. Коли одна команда передає результати своєї роботи іншій через конвеєр, передається звичайний текстовий потік. Для вилучення корисної інформації адміністратор змушений використовувати утиліти для синтаксичного аналізу тексту, використання регулярних виразів та обробки рядків [5]. Цей підхід є вразливим до будь-яких змін у форматі виводу (наприклад, зміна локалі або оновлення утиліти може "зламати" скрипт).

PowerShell кардинально змінює цю парадигму. Кожен фрагмент даних, що генерується або обробляється в PowerShell, є повноцінним об’єктом .NET. Цей об’єкт інкапсулює як дані (властивості), так і поведінку (методи) [7]. Коли

виконується команда (cmdlet), наприклад Get-Process, вона повертає не текстову таблицю процесів, а масив об’єктів System.Diagnostics.Process. Коли ці об’єкти передаються через конвеєр до наступної команди, наприклад Stop-Process, остання отримує прямий доступ до ідентифікаторів та інших атрибутів без будь-якого розбору тексту. Це забезпечує безпрецедентну надійність, гнучкість та стійкість скриптів автоматизації до змін у системному середовищі, що є критичним критерієм для систем розгортання на підприємстві [2].

PowerShell вирішує цю проблему за допомогою концепції провайдерів (PowerShell Providers) та віртуальних дисків (PSDrives). Провайдер - це програма .NET, яка представляє доступ до спеціалізованого сховища даних у вигляді віртуальної файлової системи [7]. Завдяки цьому адміністратор може здійснювати навігацію системним реєстром за допомогою тих самих командлетів (наприклад, Set-Location, Get-ChildItem), які використовуються для роботи зі звичайними файлами та папками. Така уніфікація інтерфейсу значно знижує поріг входження для IT-спеціалістів та прискорює розробку скриптів для конфігурування системних параметрів.

Крім того, PowerShell має нативну інтеграцію з Інструментарієм управління Windows (WMI - Windows Management Instrumentation) та Загальною інформаційною моделлю (CIM - Common Information Model). За допомогою командлетів сімейства Get-CimInstance можна отримати доступ до тисяч класів, які описують апаратне забезпечення (процесор, пам'ять, BIOS) та програмні компоненти ОС, що дозволяє реалізувати складну логіку прийняття рішень під час розгортання (наприклад, встановлювати певні драйвери лише на пристрої конкретного виробника) [2].

Процес розгортання та управління комп’ютерами в корпоративній мережі неможливий без надійного механізму віддаленого доступу. PowerShell використовує технологію PowerShell Remoting, яка базується на галузевому

стандарті WS-Management (реалізованому в Windows за допомогою служби WinRM - Windows Remote Management) [8].

Цей механізм дозволяє виконувати скрипти на віддалених машинах у режимі “Fan-out” - адміністратор ініціює з’єднання з однієї керуючої машини та одночасно виконує команди на десятках або сотнях цільових робочих станцій. У цьому випадку всі обчислення відбуваються на цільових системах, а через мережу передаються лише серіалізовані об’єкти результатів (у форматі XML). Ця технологія підтримує надійну аутентифікацію (Kerberos у доменному середовищі) та шифрування трафіку, забезпечуючи конфіденційність і цілісність даних під час віддаленого налаштування. Важливим нововведенням стала технологія JEA (Just Enough Administration), яка дозволяє створювати спеціальні кінцеві точки з’єднання з обмеженими правами: адміністратор може дозволити користувачеві виконувати лише певний набір скриптів або командлетів від імені привілейованого облікового запису, не надаючи йому повних прав адміністратора на комп’ютері [10].

Для реалізації парадигми “Інфраструктура як код” (Infrastructure as Code) PowerShell пропонує платформу Desired State Configuration (DSC). Традиційні скрипти є імперативними: вони вказують системі як виконати дію (наприклад, "перевір, чи є ключ реєстру; якщо немає - створи його"). Натомість DSC використовує декларативний підхід: адміністратор описує кінцевий результат (наприклад, "цей ключ реєстру повинен існувати і мати таке значення").

Менеджер локальної конфігурації (LCM) на цільовій робочій станції відповідає за перевірку поточного стану та автоматичне застосування необхідних змін для досягнення бажаного результату [11]. Головною перевагою DSC є його ідемпотентність - здатність застосовувати одну й ту саму конфігурацію кілька разів без ризику пошкодити систему (якщо система вже перебуває в бажаному стані, LCM просто не вноситиме жодних змін). Це робить DSC ідеальним інструментом для запобігання “відхиленню конфігурації” протягом усього життєвого циклу робочої станції.

З огляду на зростаючі кіберзагрози, інструмент автоматизації з глибоким доступом до ОС повинен мати бездоганну систему безпеки. Microsoft інтегрувала в PowerShell набір багаторівневих заходів безпеки. Хоча політики виконання слугують лише базовим захистом від випадкового виконання неперевірених скриптів, справжній захист забезпечується глибокою інтеграцією з Antimalware Scan Interface (AMSI). AMSI дозволяє антивірусному програмному забезпеченню (такому як Microsoft Defender) перевіряти код скрипта або команди безпосередньо в оперативній пам’яті комп’ютера перед його виконанням, навіть якщо код був сильно заплутаний або згенерований динамічно [12].

Крім того, PowerShell надає вичерпні можливості для аудиту. Функції Module Logging (логування модулів) та Script Block Logging (логування блоків скриптів) дозволяють записувати кожен фрагмент виконаного коду у журнал подій Windows. Завдяки цьому служба інформаційної безпеки підприємства завжди має повну видимість того, які саме сценарії автоматизації виконувалися на робочих станціях, що критично важливо для розслідування інцидентів та дотримання політик комплаєнсу (compliance).

Підсумовуючи, можна стверджувати, що об'єктна архітектура, глибока інтеграція з Windows API, потужні інструменти віддаленого керування та безкомпромісні механізми безпеки роблять PowerShell не просто зручною утилітою, а стратегічною платформою для побудови високоефективних, безпечних та масштабованих систем автоматизованого розгортання робочих станцій.

1.4 Порівняльний аналіз PowerShell з іншими мовами сценаріїв у середовищі Windows

Історично вибір інструментів для системного адміністрування був тісно пов’язаний з операційною системою: адміністратори UNIX-подібних систем використовували оболонки командного рядка сімейства Bourne (переважно Bash), тоді як адміністратори Windows покладалися на пакетні файли (Batch-скрипти, cmd.exe) та VBScript [5]. Однак із розвитком концепцій DevOps, гібридних хмарних інфраструктур та програмного забезпечення з відкритим кодом межі між екосистемами почали стиратися. Сьогодні в середовищі Windows технічно можливо використовувати не тільки вбудований PowerShell, але й мову програмування Python та оболонку Bash (завдяки Windows Subsystem for Linux - WSL - або портированим утилітам, таким як Git Bash) для автоматизації рутинних завдань.

Щоб обґрунтувати вибір PowerShell як основного інструменту для розгортання та налаштування робочих станцій, необхідно провести глибокий порівняльний аналіз цих трьох технологій на основі ключових критеріїв: архітектурних парадигм, глибини інтеграції з ядром ОС, підходів до розгортання середовища виконання та механізмів інформаційної безпеки.

Найважливіша відмінність між PowerShell та традиційними оболонками, такими як Bash, полягає в архітектурі обробки даних у конвеєрі (pipeline). Bash, дотримуючись класичної філософії UNIX, оперує потоками неструктурованого або напівструктурованого тексту. Коли одна команда (наприклад, ps для виводу процесів) передає дані іншій команді через конвеєр (|), наступна утиліта отримує просто набір символів [2]. Щоб витягти потрібну інформацію (наприклад, ідентифікатор процесу - PID), адміністратор змушений використовувати інструменти синтаксичного аналізу тексту, такі як grep, awk або cut. Цей підхід чудово працює в Linux, але в середовищі Windows, де конфігурації зберігаються в складних структурах (реєстр, бази

даних WMI, об'єкти Active Directory), текстовий парсинг стає вкрай ненадійним і схильним до помилок при найменшій зміні формату виводу.

Натомість PowerShell було розроблено з нуля на платформі .NET Framework. Кожна команда (cmdlet) у PowerShell повертає не текст, а повноцінний об’єкт .NET [7]. Цей об’єкт містить властивості (дані) та методи (дії, які можна виконати над цим об’єктом). Наприклад, при запиті списку служб (Get-Service) адміністратор отримує масив об’єктів служб. Щоб зупинити всі служби, імена яких починаються з “Update”, достатньо передати ці об’єкти через конвеєр до команди Stop-Service. PowerShell автоматично розуміє, який метод викликати для кожного об’єкта. Відсутність необхідності розбору тексту робить скрипти PowerShell значно надійнішими, стійкішими до змін в ОС та простішими в обслуговуванні.

Python також є об’єктно-орієнтованою мовою високого рівня, яка обробляє складні структури даних (словники, списки, класи). З точки зору архітектури програмування Python є більш універсальним і потужним інструментом, ніж спеціалізовані мови скриптів [17]. Однак у контексті адміністрування кінцевих точок Windows його об’єктна модель існує ізольовано від системних об’єктів Windows, що вимагає додаткових рівнів перекладу (наприклад, використання сторонніх бібліотек для перетворення даних WMI у формати, зрозумілі Python).

Глибина доступу до внутрішніх компонентів операційної системи є критичним фактором при виборі інструменту автоматизації. Робоча станція під управлінням Windows - це складна система, адміністрування якої вимагає взаємодії з реєстром (Windows Registry), журналами подій (Event Logs), інструментарієм управління (WMI/CIM), сертифікатами безпеки та компонентами Component Object Model (COM).

PowerShell має нативну, вбудовану підтримку всіх перелічених технологій. Адміністратор може підключитися до реєстру як до звичайного логічного диска (через механізм PSDrive) і маніпулювати ключами так само,

як і файлами [7]. Для PowerShell розроблені тисячі офіційних модулів від Microsoft, які дозволяють керувати всіма аспектами інфраструктури: від налаштування мережевих адаптерів до керування сервісами Microsoft 365, Hyper-V та Active Directory. Це створює єдине, уніфіковане середовище для вирішення будь-яких завдань.

Спроби виконати подібні завдання за допомогою Bash у Windows (через WSL) натрапляють на принципові обмеження. WSL створює ізольоване середовище Linux у Windows. Хоча WSL дозволяє запускати виконувані файли Windows (наприклад, файли .exe), Bash не має прямого структурованого доступу до WMI або реєстру без виклику зовнішніх утиліт (таких як команди PowerShell), що зводить нанівець саму ідею використання Bash для налаштування хостової ОС Windows [5]. Bash залишається чудовим інструментом для розробників, але не для адміністраторів кінцевих точок Windows.

Python займає проміжне положення. Для взаємодії з API Windows було розроблено потужну бібліотеку pywin32, яка дозволяє викликати функції COM та керувати процесами. Однак робота з цією бібліотекою вимагає значно вищих навичок програміста та написання більшого обсягу коду порівняно з лаконічними командлетами PowerShell [2]. Те, що в PowerShell робиться одним рядком (наприклад, `Get-CimInstance Win32_OperatingSystem`), у Python вимагає імпортування модулів, ініціалізації з’єднання та обробки винятків COM-об’єктів.

Тут PowerShell має беззаперечну перевагу, оскільки він інтегрований у ядро ОС Windows починаючи з версії Windows 7. На будь-якій сучасній корпоративній робочій станції середовище виконання PowerShell (разом із .NET Framework) присутнє за замовчуванням [1]. Це означає, що адміністратор може миттєво запускати скрипти розгортання або використовувати PowerShell Remoting (WinRM) без будь-якої попередньої підготовки комп'ютера (так званий "Zero-touch" або "Light-touch" підхід).

Використання Python для керування кінцевими точками Windows наражається на серйозну перешкоду: інтерпретатор Python не постачається з Windows. Щоб запустити Python-скрипт для конфігурування ПК, необхідно спочатку якимось чином розгорнути на ньому сам Python, налаштувати змінні середовища (PATH), встановити систему управління пакетами pip та завантажити необхідні залежності (зовнішні бібліотеки) [1]. У масштабах підприємства з тисячами робочих станцій це створює додатковий, непотрібний шар адміністративних витрат (overhead). Замість того, щоб керувати інфраструктурою, IT-відділ змушений керувати життєвим циклом інтерпретаторів Python на кожному ПК.

В епоху постійних кіберзагроз інструмент автоматизації з високими привілеями (на рівні SYSTEM або адміністратора) повинен мати надійні механізми контролю. PowerShell було розроблено з урахуванням концепції “Безпека за замовчуванням”. Він реалізує політики виконання, які за замовчуванням забороняють виконання непідписаних скриптів [7]. Існує режим обмеженої мови, який обмежує доступ до небезпечних API. Крім того, функція реєстрації блоків скриптів дозволяє записувати кожен фрагмент виконаного коду в журнал подій Windows (навіть якщо скрипт був зашифрований або виконувався виключно в оперативній пам'яті), що є золотим стандартом для цифрової криміналістики та реагування на інциденти (DFIR).

Підсумовуючи порівняльний аналіз (табл. 1.2), можна стверджувати, що незважаючи на величезну популярність Bash у світі UNIX та беззаперечне лідерство Python у сфері машинного навчання, мережевої автоматизації та кросплатформної розробки, для завдань конфігурування та розгортання робочих станцій під управлінням Windows ці мови є субоптимальними. PowerShell, завдяки своїй об'єктній моделі, абсолютній інтеграції з Windows API, відсутності потреби у встановленні додаткових агентів та потужним

системам аудиту, залишається безальтернативним стандартом, що забезпечує найвищий рівень надійності та безпеки корпоративної інфраструктури.

Таблиця 1.2 - Порівняння мов сценаріїв у середовищі ОС Windows

Критерій порівняння	PowerShell	Bash (через WSL / Git Bash)	Python
Архітектура даних	Об'єктно-орієнтована (.NET)	Обробка неструктурованого тексту	Об'єктно-орієнтована
Інтеграція з Windows API	Нативна (WMI, CIM, Реєстр, COM)	Відсутня (вимагає зовнішніх утиліт)	Потребує сторонніх бібліотек (pywin32)
Середовище виконання	Вбудовано в ОС за замовчуванням	Потребує встановлення WSL	Потребує встановлення інтерпретатора
Засоби аудиту та безпеки	Script Block Logging, AMSI	Обмежені	Базові (на рівні процесів)

У першому розділі доведено, що традиційні методи ручного налаштування робочих станцій є економічно неефективними, оскільки вони спричиняють явище зміщення конфігурацій (configuration drift) та підвищують ризики кіберзагроз через неминучий «людський фактор». Водночас аналіз індустріальних Enterprise-систем (MECM, Microsoft Intune, Ansible, Puppet) показав, що попри їхній вичерпний функціонал, впровадження таких рішень часто ускладнюється високою вартістю ліцензування, залежністю від хмарних підписок та необхідністю підтримки складної серверної інфраструктури. Це створює об'єктивну потребу в розробці гнучкої, безкоштовної та ефективної альтернативи на базі засобів скриптової автоматизації.

На основі порівняльного аналізу обґрунтовано вибір PowerShell як безальтернативного інструменту для реалізації такого рішення в екосистемі Windows. Завдяки унікальній об'єктно-орієнтованій архітектурі (.NET), глибокій нативній інтеграції з системним реєстром та WMI/CIM, а також відсутності потреби у встановленні додаткових агентів, PowerShell забезпечує абсолютний та надійний контроль над операційною системою. Разом із

потужними вбудованими механізмами безпеки (AMSI, Script Block Logging) та підтримкою віддаленого керування через WinRM, цей технологічний стек підтверджує доцільність переходу до парадигми «Інфраструктура як код» (IaC) і формує надійне підґрунтя для проєктування модульної архітектури розгортання у другому розділі.

РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ

2.1 Формування вимог до цільової конфігурації робочої станції

Розробка будь-якої системи автоматизації інфраструктури кінцевих точок неможлива без чіткого визначення та формалізації кінцевого стану системи, до якого її необхідно довести. У сучасній практиці системного адміністрування цей етап називають створенням стандартного робочого середовища (SOE) [1]. Визначення вимог до цільової конфігурації робочої станції є першим кроком, який визначає логіку майбутніх сценаріїв автоматизації, структуру модулів розгортання та критерії перевірки успішного виконання скриптів. Без детального еталонного стану автоматизація втрачає сенс, оскільки неможливо забезпечити ідемпотентність процесів та усунути явище дрейфу конфігурації [2].

Цільова конфігурація корпоративного комп'ютера під управлінням операційної системи Windows має бути комплексною та охоплювати кілька взаємопов'язаних рівнів: абстракцію апаратного забезпечення, параметри операційної системи, мережеві налаштування, базовий набір прикладного програмного забезпечення та багаторівневу систему інформаційної безпеки.

Перший рівень вимог (табл. 1.2) стосується базових параметрів самої операційної системи. Для корпоративного середовища стандартом є використання редакцій Windows Pro або Enterprise (версії 10 або 11), оскільки лише вони мають повний набір інструментів для віддаленого керування, шифрування та інтеграції в доменну структуру [17]. Автоматизоване налаштування має забезпечувати уніфікацію регіональних стандартів та мовних пакетів (зокрема, встановлення української локалізації як основної для інтерфейсу та введення даних), конфігурування єдиного часового поясу та синхронізацію системного часу з корпоративним або публічним NTP-сервером.

Таблиця 2.1 - Специфікація стандартизованого операційного середовища (SOE)

Рівень конфігурації	Опис цільових налаштувань
Базова ОС	Редакція Windows Pro/Enterprise, українська локалізація, уніфіковане ім'я ПК (напр. CORP-PC-XXXX).
Мережа та ідентифікація	Отримання IP через DHCP, статичні DNS, інтеграція в домен Active Directory.
Програмне забезпечення	Тихе встановлення базового стеку (веббраузери, архіватори, .NET) через пакетний менеджер Winget.
Інформаційна безпека	Шифрування BitLocker (TPM), вимкнення SMBv1/LLMNR, Defender ASR, відсутність прав локального адміністратора.

Окрему увагу слід приділити схемі іменування робочих станцій (Computer Name). Ручне або випадкове присвоєння імен ускладнює інвентаризацію та моніторинг. Цільова конфігурація вимагає впровадження стандартизованого шаблону, який може базуватися на серійному номері пристрою (отриманому нативно з BIOS/UEFI через CIM-екземпляри) або включати буквений код підрозділу компанії (наприклад, CORP-DEV-XXXXX).

Рівень налаштування мережі визначає здатність робочої станції безпечно та надійно працювати в локальних і глобальних мережах. Основними вимогами тут є автоматичне отримання IP-адреси та налаштувань DNS за допомогою протоколу DHCP з урахуванням особливостей внутрішньої маршрутизації. Скрипти автоматизації повинні забезпечувати правильну конфігурацію мережевих адаптерів, відключення застарілих і небезпечних протоколів, таких як IPv6 (якщо вони не використовуються в інфраструктурі), а також реєстрацію комп'ютера в корпоративній службі каталогів Active Directory Domain Services (AD DS) або Azure Active Directory (Entra ID) [6]. Інтеграція в домен є критично важливою вимогою, оскільки вона дозволяє в подальшому застосовувати централізовані об'єкти групової політики (GPO).

На рівні прикладного програмного забезпечення формується так званий “базовий стек”, необхідний для роботи середньостатистичного користувача підприємства. До цього стеку входять сучасні веббраузери (Google Chrome, Microsoft Edge), офісні пакети (Microsoft Office або безкоштовні альтернативи), засоби перегляду документів (Adobe Reader), архіватори та системні компоненти, такі як .NET Framework та Visual C++ Redistributables [18].

Вимога до цільової конфігурації полягає не просто в наявності цього програмного забезпечення, а в тому, щоб воно було встановлено в режимі автоматичної інсталяції без відображення будь-яких графічних вікон для користувача. Дистрибутиви необхідно отримувати з офіційних репозиторіїв менеджерів пакетів (наприклад, менеджера пакетів Windows - Winget), що гарантує безпеку та актуальність версій програмного забезпечення на момент розгортання [9].

Найбільш критичним та об'ємним блоком вимог є рівень інформаційної безпеки кінцевих точок. Згідно з рекомендаціями провідних інституцій у сфері кібербезпеки, зокрема Center for Internet Security (CIS) та Національного інституту стандартів і технологій (NIST), операційна система “з коробки” має надлишкову кількість увімкнених служб та функцій, які розширюють поверхню потенційної атаки [14]. Тому до цільової конфігурації висуваються суворі вимоги щодо загартування (hardening) ОС:

— Вимкнення застарілих протоколів: Повне деактивування вразливого протоколу загального доступу до файлів SMBv1, а також протоколів виявлення мережі LLMNR та NetBIOS, які часто використовуються зловмисниками для атак типу індукування (spoofing) та перехоплення облікових даних [18].

— Керування локальними обліковими записами: Перейменування або повне вимкнення вбудованого запису “Адміністратор”, встановлення унікальних складних паролів для локальних користувачів та обмеження їхніх

прав до рівня стандартного користувача (Standard User) для запобігання несанкціонованій зміні системних параметрів.

— Криптографічний захист: Обов'язкова активація вбудованого інструменту повнодискового шифрування BitLocker для захисту конфіденційних даних у разі фізичної крадіжки або втрати пристрою, з автоматичним збереженням ключів відновлення в безпечному централізованому сховищі.

— Захист від шкідливого коду: Конфігурування антивірусного захисту (Microsoft Defender Antivirus) у режим постійного моніторингу в реальному часі, увімкнення хмарного захисту та налаштування щоденного автоматичного оновлення сигнатурних баз [18].

Щоб забезпечити можливість управління розгорнутою робочою станцією та її прозорість для IT-відділу, цільова конфігурація повинна містити налаштування інструментів моніторингу та телеметрії. Необхідно обмежити обсяг діагностичних даних, що надсилаються до Microsoft, до необхідного мінімуму (з метою збереження конфіденційності), а також увімкнути та налаштувати службу Windows Remote Management (WinRM). Увімкнення WinRM та створення відповідних винятків у брандмауері Windows є обов’язковим технічним критерієм, оскільки саме цей протокол дозволяє адміністраторам підключатися віддалено через PowerShell Remoting для обслуговування системи без фізичної присутності на місці [8].

2.2 Вибір стратегії розгортання

Процес інтеграції нової комп'ютерної техніки до корпоративної інфраструктури або перевстановлення операційної системи на існуючих пристроях вимагає обрання оптимальної методології конфігурування. У сучасній практиці системного адміністрування та управління інфраструктурою виділяють три фундаментальні стратегії підготовки кінцевих точок: розгортання на чисте обладнання (Bare-metal deployment),

використання післяінсталяційних сценаріїв (Post-install scripts або Provisioning) та гібридний підхід (Hybrid deployment). Вибір конкретної стратегії є критичним архітектурним рішенням, оскільки воно безпосередньо впливає на завантаженість локальної мережі підприємства, швидкість підготовки одного робочого місця та загальну складність підтримки кодової бази сценаріїв автоматизації системними адміністраторами [1].

Стратегія розгортання “з нуля” передбачає повне очищення (форматування) накопичувачів цільової робочої станції та встановлення операційної системи Windows з нуля. Цей підхід традиційно базується на використанні середовища попередньої інсталяції Windows (WinPE) та мережевого завантаження за протоколом PXE (Preboot Execution Environment). У рамках цієї стратегії адміністратори часто створюють так звані “товсті образи” у форматі WIM (Windows Imaging Format), які вже містять розпаковану операційну систему, встановлені корпоративні програмні пакети, інтегровані драйвери пристроїв та необхідні оновлення безпеки [6]. Перевагою цього методу є гарантована відсутність будь-якого небажаного попередньо встановленого програмного забезпечення від виробника обладнання (OEM bloatware) та абсолютна уніфікація файлової системи на всіх корпоративних пристроях після завершення процесу створення образу.

Проте стратегія Bare-metal має низку суттєвих недоліків (табл. 2.2) у контексті сучасних гнучких інфраструктур. Найголовнішим викликом є необхідність постійної підтримки та оновлення еталонних образів. Кожного разу, коли виходить критичне оновлення для Windows або нова версія базового веббраузера, IT-відділ змушений перезбирати багатогігабайтний файл образу, тестувати його та завантажувати на сервери розповсюдження. Крім того, передача “товстого образу” обсягом від десяти до двадцяти гігабайтів по локальній мережі на десятки комп'ютерів одночасно створює критичне навантаження на мережеве обладнання, що може призвести до деградації пропускної здатності для інших бізнес-процесів підприємства [17].

Цей підхід також є вкрай неефективним для віддалених працівників, оскільки завантаження масивного WIM-файлу через віртуальну приватну мережу (VPN) з домашнього інтернет-з'єднання є практично неможливим.

Таблиця 2.2 - Аналіз стратегій розгортання робочих станцій

Стратегія розгортання	Принцип роботи	Переваги	Недоліки
Bare-metal (“Товстий образ”)	Встановлення ОС з образу (WIM), що вже містить ПЗ та налаштування.	Гарантія чистої ОС, ідентичність файлової системи.	Важко оновлювати образ, велике навантаження на мережу (PXE).
Post-install скрипти	Налаштування заводської (OEM) ОС після першого запуску (OOBE).	Незалежність від "заліза", швидка доставка конфігурацій.	Ризик конфліктів із вбудованим ПЗ від виробника (bloatware).
Гібридна (“Тонкий образ”)	Встановлення чистого образу ISO + застосування PowerShell скриптів.	Гнучкість підтримки ПЗ, швидкість мережевого розгортання.	Вимагає високої якості та відмовостійкості скриптів.

Альтернативою розгортанню на основі образів є стратегія, що використовує скрипти післяінсталяційного налаштування. Цей метод ґрунтується на концепції динамічного розгортання. Замість форматування диска та переінсталяції ОС ця стратегія використовує операційну систему, яка вже була попередньо встановлена виробником обладнання (OEM-версія Windows) або розгорнута з найпростішого оригінального дистрибутива Microsoft [6]. Втручання IT-інфраструктури починається на етапі першого запуску комп'ютера користувачем, так званого Out-Of-Box Experience (OOBE). На цьому етапі виконуються спеціальні скрипти PowerShell, що перетворюють стандартну версію ОС Home або Basic Professional на повноцінне корпоративне робоче середовище.

Сценарії Post-install автоматично видаляють непотрібні вбудовані додатки, змінюють ключ продукту для підвищення редакції до Windows Enterprise, приєднують пристрій до служби каталогів Active Directory, завантажують актуальні версії корпоративного програмного забезпечення через мережу Інтернет та застосовують жорсткі політики безпеки. Головною перевагою цього підходу є його абсолютна незалежність від апаратного забезпечення: адміністраторам не потрібно інтегрувати специфічні драйвери для різних моделей материнських плат або мережевих адаптерів у скрипти, оскільки заводська операційна система вже містить усе необхідне для коректної роботи пристрою. Крім того, обсяг даних, які необхідно передати на кінцеву точку для її конфігурування (самі скрипти та дистрибутиви ПЗ), є в десятки разів меншим порівняно з передачею повноцінного образу диска [2].

Незважаючи на високу гнучкість, стратегія виключно післяінсталяційних скриптів не завжди є ідеальною для масштабних підприємств із високими вимогами до безпеки. OEM-версії Windows часто містять глибоко інтегровані утиліти від виробників обладнання (наприклад, програми для оновлення драйверів або налаштування звуку), які можуть мати власні вразливості та конфліктувати з корпоративними системами захисту. Спроби повністю вичистити систему від таких компонентів лише за допомогою скриптів можуть бути непередбачуваними та призводити до нестабільної роботи операційної системи в майбутньому [17].

Оптимальним компромісом, який фактично став галузевим стандартом для сучасних IT-інфраструктур, є стратегія гібридного розгортання. Гібридний підхід органічно поєднує надійність методів на основі образів із гнучкістю автоматизації за допомогою скриптів. Основою цієї стратегії є використання так званого “тонкого образу”. Тонкий образ - це повністю чиста версія операційної системи Windows, без будь-якого стороннього програмного забезпечення або додаткових драйверів, створена безпосередньо з оригінального образу Microsoft (ISO). Цей образ не містить ні офісних пакетів,

ні корпоративних налаштувань. Його єдина мета - забезпечити надійне, стандартизоване та чисте базове середовище, гарантуючи відсутність небажаних компонентів від виробників обладнання [1].

Завдяки своєму мінімальному розміру тонкий образ надзвичайно швидко розгортається на кінцевих точках через мережу. Одразу після завершення цього базового етапу управління перехоплює потужна система сценаріїв PowerShell. У гібридному підході скрипти беруть на себе сто відсотків завдань із кастомізації середовища. Вони динамічно визначають модель комп'ютера через запити до Windows Management Instrumentation (WMI), завантажують та інсталюють відповідні драйвери з центрального репозиторію, встановлюють необхідне прикладне програмне забезпечення за допомогою пакетних менеджерів та формують цільову конфігурацію безпеки. Завдяки такому розподілу ролей IT-відділ підприємства звільняється від необхідності підтримувати десятки різних “товстих образів” для кожного типу обладнання. Усі зміни, пов'язані з оновленням версій програм або політиками безпеки, вносяться виключно у текстові файли PowerShell-сценаріїв, що займає лічені хвилини та повністю відповідає парадигмі “Інфраструктура як код”[2].

Для реалізації завдань даної кваліфікаційної роботи стратегія гібридного розгортання з акцентом на складні післяінсталяційні сценарії PowerShell визнається найбільш доцільною та технічно обґрунтованою. Розробка комплексної автоматизації саме на етапі Post-install дозволяє продемонструвати всю потужність об'єктно-орієнтованої архітектури PowerShell. Такий підхід дає змогу створити універсальний програмний продукт - набір модульних скриптів, які зможуть коректно відпрацювати на будь-якій версії Windows 10 або 11, незалежно від того, як саме операційна система була попередньо встановлена на комп'ютер. Це забезпечує високу портативність розробленого рішення, дозволяючи впроваджувати його як для

підготовки нових робочих місць у локальній мережі, так і для віддаленого переналаштування комп'ютерів співробітників.

2.3 Використання технології PowerShell Remoting та WinRM для віддаленого керування

Масштабування процесів налаштування кінцевих точок у сучасному корпоративному середовищі вимагає впровадження надійних, безпечних та високопродуктивних механізмів віддаленої взаємодії з операційною системою. Традиційні підходи, засновані на технологіях віддаленого виклику процедур (RPC) або застарілих інструментах сімейства PsExec, мають серйозні обмеження в плані безпеки, слабкі можливості аутентифікації та створюють надмірне навантаження на мережеву інфраструктуру. Сучасним галузевим стандартом для управління системами Windows є технологія PowerShell Remoting, яка працює на основі системної служби Windows Remote Management (WinRM). Цей набір інструментів дозволяє здійснювати централізоване управління конфігурацією, виконувати складні сценарії розгортання одночасно на тисячах робочих станцій та повністю інтегрувати процеси адміністрування кінцевих точок у загальну систему автоматизації інфраструктури [8].

Фундаментом віддаленого керування в оболонці PowerShell є служба WinRM, яка є нативною реалізацією відкритого міжнародного стандарту Web Services for Management (WS-Management). Цей протокол базується на технологіях вебсервісів і використовує для передачі інформації стандартний протокол прикладного рівня HTTP або його захищену версію HTTPS, де керуючі команди та результати їх виконання інкапсулюються у структуровані повідомлення Simple Object Access Protocol (SOAP). Використання вебстандартів замість специфічних бінарних протоколів минулих поколінь забезпечує високу сумісність із мережевим обладнанням, оскільки трафік WinRM, що за замовчуванням проходить через TCP-порти 5985 для HTTP та 5986 для HTTPS, легко піддається маршрутизації, фільтрації та інспектуванню

сучасними міжмережевими екранами (Firewalls). Це усуває необхідність відкриття широкого діапазону динамічних портів, як це було у випадку з технологіями DCOM або RPC, суттєво звужуючи поверхню потенційних атак на корпоративну мережу [8].

Архітектурна унікальність PowerShell Remoting полягає у способі обробки та передачі даних між консоллю управління системного інженера та цільовою робочою станцією. На відміну від класичних текстових протоколів віддаленого доступу, таких як Secure Shell (SSH) у системах UNIX, PowerShell Remoting повністю зберігає об’єктно-орієнтований характер оболонки під час мережевого зв’язку. Коли адміністратор запускає команду на віддаленому комп’ютері, локальний сеанс PowerShell не просто надсилає текстовий рядок, а серіалізує об’єкти параметрів у структурований XML-формат, відомий як CLIXML. Цей XML-потік передається через транспортний рівень WinRM на цільову машину, де локальна служба WinRM передає його фоновому процесу хоста сеансу PowerShell. На стороні клієнта дані десеріалізуються, код командлету виконується безпосередньо, а отримані об’єкти потім знову серіалізуються у формат CLIXML і повертаються на робочу станцію адміністратора. Хоча отримані об’єкти десеріалізуються (тобто вони втрачають свої активні методи, але повністю зберігають усі властивості та структуру даних), вони дозволяють здійснювати подальшу логіку конфігурації, фільтрування та агрегацію інформації на вузлі управління без додаткових мережевих запитів [7].

Безпека віддаленого підключення в PowerShell Remoting забезпечується інтеграцією з підсистемою безпеки операційної системи Windows та використанням сучасних протоколів автентифікації. У доменному середовищі Active Directory за замовчуванням застосовується протокол Kerberos, який гарантує взаємну автентифікацію як керуючого сервера, так і цільової робочої станції без передачі паролів користувачів у відкритому чи навіть зашифрованому вигляді по мережі. Для середовищ, які функціонують поза

межами домену (робочі групи, Workgroups), підтримується автентифікація за допомогою протоколу NTLM або використання сертифікатів шифрування. Увесь трафік сесії PowerShell Remoting, навіть при використанні базового протоколу HTTP на порті 5985, піддається обов'язковому криптографічному шифруванню на рівні самого протоколу WS-Management за допомогою ключів сесії автентифікації (Kerberos або NTLMSSP), що повністю виключає можливість перехоплення даних, модифікації команд або проведення атак типу “людина посередині”[8].

Однією з найскладніших інженерних проблем при проектуванні масштабних систем автоматизації розгортання є так звана проблема “подвійного переходу”. Вона виникає, коли адміністратор підключається за допомогою PowerShell Remoting до першої віддаленої робочої станції та намагається отримати доступ до ресурсів на третьому комп’ютері з цієї робочої станції - наприклад, завантажити інсталяційний пакет програмного забезпечення з центрального мережевого сховища (папки SMB) або підключитися до бази даних інвентаризації. З міркувань безпеки протокол Kerberos за замовчуванням не дозволяє віддаленому комп’ютеру делегувати отримані ним облікові дані адміністратора далі по мережі. Для вирішення цієї проблеми в архітектурах автоматизації застосовують кілька технічних рішень: налаштування обмеженого делегування Kerberos, використання Credential Security Support Provider (CredSSP) або впровадження сеансів із фіксованими обліковими даними в конфігураціях кінцевих точок сеансу [7].

Ефективність масового розгортання конфігурацій безпосередньо залежить від здатності системи виконувати операції паралельно. PowerShell Remoting реалізує модель паралельного виконання “Fan-Out”, яка дозволяє за допомогою одного командлета Invoke-Command ініціювати одночасний запуск сценаріїв на сотнях цільових пристроїв. Керуючий комп'ютер створює пул підключень і розподіляє ресурси таким чином, що виконання коду відбувається локально на кожній цільовій станції, мінімізуючи обчислювальне

навантаження на робоче місце адміністратора. За допомогою параметра обмеження інтенсивності (ThrottleLimit) інженер може динамічно регулювати кількість одночасних мережевих сесій, балансуючи між швидкістю розгортання та пропускною здатністю каналів зв'язку, що є критично важливим для великих корпоративних мереж [2].

Для оптимізації багаторазових або тривалих процедур налаштування технологія дозволяє створювати постійні віддалені сесії за допомогою об'єктів PSSession. Замість того, щоб встановлювати нове криптографічне з'єднання та проходити процес автентифікації для кожної окремої команди, що створює значні часові затримки, адміністратор один раз ініціює об'єкт PSSession за допомогою командлета New-PSSession. Ця сесія залишається активною у фоновому режимі, дозволяючи послідовно виконувати різні модулі скриптів розгортання, передавати змінні між локальним та віддаленим середовищем та контролювати стан системи в реальному часі. Після завершення всіх конфігураційних процедур сесія коректно закривається, звільняючи системні ресурси на обох сторонах підключення [7].

Найвищий рівень безпеки та контролю під час використання віддаленого управління забезпечує технологія Just Enough Administration (JEA), інтегрована в платформу PowerShell Remoting. Концепція JEA реалізує принцип мінімальних привілеїв та контроль доступу на основі ролей (RBAC) на рівні операційної системи Windows. Замість надання фахівцям технічної підтримки повних прав локального адміністратора на робочих станціях, що створює значні ризики для безпеки, JEA дозволяє створювати конкретні обмежені кінцеві точки підключення WinRM. Конфігурація такої кінцевої точки визначає точний перелік командлетів, функцій, зовнішніх утиліт або навіть конкретних параметрів команд, які підключений користувач має право виконувати. Коли скрипт автоматизації або оператор підключається до сеансу JEA, всі команди виконуються у віртуальному, ізольованому середовищі від імені спеціального тимчасового облікового запису з підвищеними привілеями

(Virtual Account), але користувачеві фізично не дозволяється виходити за межі дозволеного файлу конфігурації, що мінімізує ризики компрометації системи або випадкового пошкодження критичних параметрів ОС [10].

Додатковим елементом надійності є впровадження жорстких політик безпеки для самої служби WinRM за допомогою групових політик (GPO). Централізоване налаштування корпоративної мережі повинно включати обмеження списку IP-адрес, з яких дозволено приймати вхідні підключення WinRM (параметр TrustedHosts), примусове використання виключно зашифрованих HTTPS-лісенерів із валідацією сертифікатів внутрішнього центру сертифікації (CA), а також активацію обов'язкового розширеного логування всіх дій віддалених сесій. Функції аудиту операційної системи, що взаємодіють із PowerShell Remoting, забезпечують фіксацію кожного виконаного блоку коду в системних журналах Event Logs, що дозволяє службі кібербезпеки підприємства здійснювати безперервний моніторинг процесів автоматизації та гарантувати повну прозорість дій у межах життєвого циклу управління робочими станціями [16].

Таким чином, інтеграція технології PowerShell Remoting та інфраструктури WinRM у загальну архітектуру системи автоматизованого розгортання забезпечує створення швидкого, безпечного та масштабованого каналу управління. Використання об'єктного конвеєра, паралельного виконання команд, механізмів подолання обмежень автентифікації та технологій обмеження привілеїв дозволяє повністю відмовитися від локального налаштування комп'ютерів, перетворюючи процес конфігурування парку робочих станцій на централізовану, керовану кодом інженерну процедуру корпоративного рівня [2].

2.4 Проєктування структури модульного скрипта та системи логування

Побудова надійної системи автоматизованого розгортання вимагає переходу від написання простих лінійних сценаріїв до застосування повноцінних інженерних практик розробки програмного забезпечення. Історично багато адміністраторів розпочинали автоматизацію зі створення монолітних скриптів, у яких усі команди для налаштування мережі, встановлення програмного забезпечення та застосування політик безпеки записувалися послідовно в один масивний файл. Такий підхід, відомий як процедурне програмування без структури (spaghetti code), є абсолютно неприйнятним для корпоративного середовища. Монолітні скрипти вкрай важко підтримувати, тестувати та масштабувати. Будь-яка зміна, наприклад, оновлення версії прикладного пакета, може випадково порушити логіку виконання наступних етапів налаштування операційної системи. Тому фундаментальною вимогою до проєктування архітектури є застосування принципу єдиної відповідальності (Single Responsibility Principle) та розбиття загального процесу на ізольовані, незалежні логічні блоки - модулі [2].

Модульна архітектура скриптів PowerShell передбачає створення ієрархічної структури файлів, де кожен окремий файл відповідає виключно за одне конкретне завдання або область налаштувань. Наприклад, окремі модулі створюються для налаштування основних параметрів ОС (регіональні налаштування, ім’я комп’ютера, приєднання до домену), встановлення системних компонентів і драйверів, встановлення програмного забезпечення за допомогою менеджера пакетів Winget, а також для застосування корпоративних політик безпеки та зміцнення захисту операційної системи. Такий розподіл дозволяє різним фахівцям IT-відділу паралельно працювати над відповідними частинами коду без створення конфліктів. Крім того, модульність забезпечує високий рівень повторного використання коду: якщо конкретному підрозділу компанії потрібен інший набір програмного

забезпечення, системному інженеру достатньо замінити або модифікувати лише один модуль інсталяції програми, залишивши незмінними решту етапів базової конфігурації та налаштування безпеки [2].

Для керування цією множиною незалежних модулів проєктується головний керуючий сценарій, який виконує роль оркестратора (Orchestrator або Master Script). Головний скрипт не містить жодної прямої логіки налаштування системи. Його єдиною метою є ініціалізація середовища виконання, перевірка базових передумов (наприклад, наявності прав локального адміністратора, активного підключення до мережі Інтернет та доступності корпоративного репозиторію) і послідовний виклик підпорядкованих модулів. Виклик модулів у PowerShell може реалізовуватися різними методами, проте найбільш надійним для систем розгортання є використання механізму dot-sourcing або явного імпорту модулів (Import-Module), що дозволяє завантажувати функції у глобальну область видимості, а потім виконувати їх із передачею необхідних параметрів. Оркестратор також відповідає за управління загальним станом процесу: він відстежує, які модулі вже були успішно виконані, що дозволяє реалізувати механізм відновлення (Resume) розгортання у випадку непередбачуваного перезавантаження комп'ютера або тимчасової втрати живлення [16]. Детальну структуру розроблених модулів автоматизації та їхні функції наведено у таблиці 2.3.

Таблиця 2.3 - Архітектура PowerShell-модулів автоматизації

Назва модуля / Файлу	Призначення та основні функції	Механізм обробки стану
00-MasterScript.ps1	Оркестратор: ініціалізація, виклик модулів, керування логуванням (JSON/Event Log).	Глобальний Try-Catch, зчитування маркерів виконання.
01-BaseConfig.ps1	Перейменування ПК (серійний номер), локалізація, введення в домен AD.	Reboot/RunOnce маркер у реєстрі.

02-InstallApps.ps1	Встановлення ПЗ згідно з корпоративним JSON-файлом через Winget.	Перевірка кодів виходу ($LASTEXITCODE), ідемпотентність.

Продовження таблиці 2.3

03-RemoveBloatware.ps1	Видалення UWP-сміття, оптимізація служб (телеметрія, планувальник завдань).	Звірка поточного статусу служб перед зміною.
04-SecurityConfig.ps1	Активація BitLocker, вимкнення вразливих протоколів, налаштування брандмауера.	Перевірка апаратного статусу vTPM перед шифруванням.

Успішна автоматизація неможлива без глибоко продуманої системи обробки помилок та виключних ситуацій. У середовищі PowerShell більшість командлетів за замовчуванням генерують так звані некритичні помилки (Non-terminating errors), які виводять червоний текст у консоль, але дозволяють скрипту продовжувати виконання наступних рядків коду. Для процесу конфігурування робочої станції така поведінка є вкрай небезпечною, оскільки ігнорування помилки на етапі налаштування мережі призведе до каскадного провалу всіх наступних етапів завантаження програмного забезпечення. Для вирішення цієї проблеми на рівні оркестратора встановлюється глобальна змінна ErrorActionPreference у значення Stop. Це примусово перетворює всі помилки на критичні (Terminating errors), зупиняючи виконання поточного потоку. Далі в кожному модулі використовується конструкція Try-Catch-Finally. Блок Try містить безпосередній код налаштування, блок Catch перехоплює будь-які згенеровані винятки, дозволяючи коректно обробити їх (наприклад, записати детальну інформацію про збій у базу даних або журнал), а блок Finally виконує гарантоване очищення ресурсів, закриваючи мережеві з'єднання та видаляючи тимчасові файли незалежно від того, успішно завершився код чи з помилкою [7].

Оскільки процес автоматизованого розгортання часто відбувається у фоновому режимі (без взаємодії з користувачем) або дистанційно через протокол WinRM, традиційне виведення інформації на екран за допомогою командлетів на кшталт Write-Host втрачає свій сенс. Адміністратор фізично не бачить консолі під час виконання скрипта, а після його завершення вся виведена інформація безслідно зникає з оперативної пам'яті. Саме тому критично важливою складовою архітектури є розробка комплексної, стійкої системи логування та аудиту. Базовим інструментом для цього є нативна функція транскрибування PowerShell (Start-Transcript), яка автоматично записує абсолютно всі команди, що виконуються, та їхній текстовий вивід у вказаний текстовий файл. Транскрибування є незамінним інструментом для глибокого налагодження (debugging), оскільки воно фіксує точний хронометраж виконання кожної операції та допомагає виявити приховані синтаксичні помилки [16].

Однак транскрипція генерує надто об’ємні та неструктуровані масиви тексту, які важко аналізувати за допомогою автоматизованих систем моніторингу. Тому паралельно розробляється спеціальна система структурованого логування. Розробляється спеціалізована функція (наприклад, Write-DeploymentLog), яку викликають оркестратор та всі модулі. Ця функція приймає повідомлення, рівень серйозності (INFO, WARNING, ERROR) та назву компонента, який її викликав. Отримана інформація форматується у стандартизований формат з обов’язковим додаванням точного часового штампу та записується у структуровані формати, такі як CSV (Comma-Separated Values) або JSON. Форматування журналів у форматі JSON є сучасним галузевим стандартом, оскільки такі файли можна легко імпортувати та аналізувати за допомогою систем SIEM (Security Information and Event Management) або інструментів агрегації журналів, таких як ELK Stack (Elasticsearch, Logstash, Kibana) або Splunk [16].

Найвищим рівнем надійності системи аудиту є інтеграція функцій логування безпосередньо з системними журналами подій операційної системи (Windows Event Logs). Використання командлета Write-EventLog або сучаснішого New-WinEvent дозволяє реєструвати спеціальне джерело подій (Custom Event Source), специфічне для системи розгортання підприємства. Запис критичних етапів конфігурування (наприклад, “Успішно приєднано до домену”, “Помилка встановлення антивірусного ПЗ”) безпосередньо в Event Log гарантує, що ця інформація не буде випадково видалена звичайним користувачем, як це може статися зі звичайними текстовими файлами на диску [20]. Крім того, корпоративні системи моніторингу (наприклад, Zabbix або Microsoft System Center) мають вбудовані агенти, які вміють автоматично зчитувати журнали подій Windows і миттєво генерувати сповіщення (Alerts) для інженерів технічної підтримки у разі фіксації критичного збою розгортання.

Окремою архітектурною задачею є реалізація механізму управління станом для забезпечення ідемпотентності скриптів. Ідемпотентність означає, що один і той самий скрипт можна безпечно запускати на комп’ютері кілька разів, і він вноситиме зміни лише тоді, коли фактична конфігурація відрізняється від цільової. Для цього система ведення журналів та оркестрування використовує механізм маркерів (своєрідних “контрольних точок”). Після успішного завершення кожного важливого етапу (наприклад, встановлення Office) скрипт створює певний ключ у системному реєстрі Windows або записує відповідний статус у локальний файл конфігурації XML/JSON. Під час наступного запуску оркестратор спочатку зчитує ці маркери. Якщо він бачить, що етап уже успішно завершено, він миттєво пропускає цей модуль і переходить до наступного. Це значно зменшує мережевий трафік, дозволяє уникнути повторного завантаження багатогігабайтних дистрибутивів, які вже встановлені, та робить процес

налаштування комп’ютера стійким до будь-яких перебоїв у мережі або перезавантажень апаратного забезпечення [2].

Впровадження описаних принципів проєктування перетворює набір розрізнених команд на професійний, відмовостійкий програмний продукт. Модульна архітектура забезпечує гнучкість та легкість масштабування кодової бази, сувора обробка виключних ситуацій за допомогою конструкцій Try-Catch запобігає непередбачуваній поведінці системи, а багаторівнева система структурованого логування та фіксації стану гарантує абсолютну прозорість процесу для адміністраторів і відповідність корпоративним стандартам безперервного аудиту. Такий комплексний архітектурний підхід є необхідним фундаментом для подальшого етапу безпосереднього кодування сценаріїв та їх інтеграції в IT-інфраструктуру.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СКРИПТІВ АВТОМАТИЗАЦІЇ ТА НАЛАШТУВАННЯ

3.1 Розробка головного керуючого скрипта

Процес практичної реалізації комплексної системи автоматизованого розгортання робочих станцій розпочався з проєктування та розробки головного керуючого модуля. У контексті обраної модульної архітектури цей сценарій виконує функцію Оркестратора. Його головне призначення полягає не в безпосередньому виконанні конфігураційних змін в операційній системі, а в координації роботи всіх інших підпорядкованих модулів, забезпеченні надійного середовища виконання та веденні централізованого журналу аудиту (логування).

На етапі ініціалізації скрипта 00-MasterScript.ps1 першочерговим завданням є забезпечення безпеки та перевірка достатнього рівня привілеїв. Оскільки подальші модулі виконують глибокі зміни у системному реєстрі, керують службами та встановлюють програмне забезпечення, їх виконання можливе виключно з правами локального адміністратора. Для автоматизації цієї перевірки у скрипті було використано звернення до класів платформи .NET: [Security.Principal.WindowsIdentity] та [Security.Principal.WindowsPrincipal]. Розроблений логічний блок перевіряє, чи належить поточний користувач до вбудованої групи Administrator. У разі відсутності необхідних прав виконання скрипта миттєво блокується за допомогою команди Break, а користувачеві виводиться попередження через командлет Write-Warning. Це запобігає частковому або некоректному застосуванню налаштувань.

Наступним критично важливим архітектурним рішенням стала розробка системи структурованого логування. Оскільки розгортання часто відбувається в автоматичному режимі, адміністратор потребує надійного джерела інформації про результати виконання кожного етапу. Для цього в тілі

Оркестратора було задекларовано створення окремої директорії C:\DeploymentLogs. Назва лог-файлу генерується динамічно за допомогою командлета Get-Date із форматуванням yyyyMMdd_HHmmss, що унеможливлює перезапис попередніх звітів та дозволяє вести історичний аудит розгортань.

Для стандартизації записів було розроблено кастомну функцію Write-Log. Вона приймає два обов'язкові параметри: текст повідомлення ($Message) та рівень події ($Level). Функція автоматично додає мітку часу до кожного повідомлення. Під час тестування модуля в реальному середовищі було виявлено проблему збереження кириличних символів - стандартні командлети PowerShell зберігали текст в кодуванні ANSI, що призводило до появи нечитабельних символів та порушення синтаксису у згенерованих звітах. Цю проблему було успішно вирішено шляхом додавання параметра -Encoding UTF8 до командлета Add-Content, що гарантувало коректну підтримку української локалізації у файлах журналів. Одночасно з записом у файл, функція дублює інформацію в консоль за допомогою Write-Host, використовуючи кольорове кодування (зелений для успіху, червоний для помилок) для зручності візуального контролю.

Основна логіка керування модулями реалізована через використання масиву $ScriptsToRun, до якого внесено точні назви всіх чотирьох робочих сценаріїв у порядку їх виконання: налаштування бази, встановлення ПЗ, очищення системи та застосування політик безпеки. Оркестратор використовує цикл foreach для послідовної ітерації цього масиву. Перед запуском кожного модуля скрипт перевіряє його фізичну наявність у робочій директорії за допомогою командлета Test-Path. Запуск модулів здійснюється за допомогою оператора виклику (Call operator - &).

Для візуалізації алгоритму роботи головного керуючого скрипта було розроблено UML-діаграму діяльності (рис. 3.1). Вона демонструє розгалуження процесу на етапі перевірки привілеїв, послідовну ітерацію

масиву конфігураційних модулів та роботу механізму інкапсуляції помилок (Try-Catch), який забезпечує безперервність процесу розгортання навіть у разі локальних збоїв.

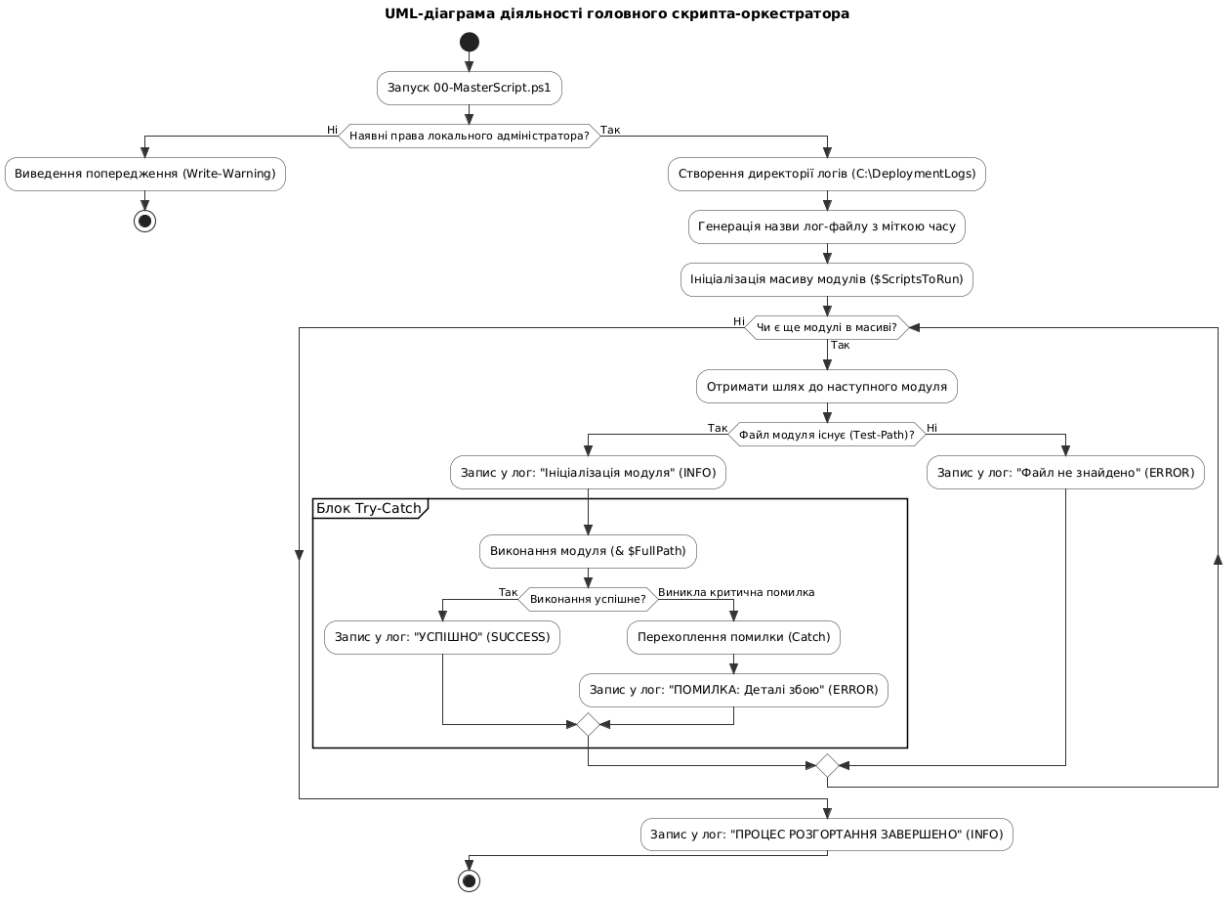


Рисунок 3.1 - UML-діаграма діяльності головного скрипта-оркестратора

Детальний алгоритм функціонування керуючого скрипта 00-MasterScript.ps1 наведено на UML-діаграмі діяльності (рис. 3.1). Робота Оркестратора починається з обов'язкової перевірки прав локального адміністратора, без яких процес миттєво зупиняється. Далі система ініціалізує середовище (створює директорію логів, формує послідовний масив модулів) і переходить до їх циклічної обробки. Кожен файл перевіряється на наявність та виконується всередині захищеного блоку Try-Catch. Це архітектурне рішення гарантує відмовостійкість розгортання: якщо в будь-якому модулі

виникає критична помилка, Оркестратор перехоплює її, фіксує збій у централізованому журналі та безперебійно продовжує запуск наступних етапів системи.

3.2 Розробка скрипта базового конфігурування ОС

Другим етапом практичної реалізації системи автоматизації стало створення модуля 01-BaseConfig.ps1. Цей сценарій відповідає за фундаментальне налаштування операційної системи одразу після розгортання “тонкого образу”. Головною метою модуля є усунення рутинних операцій з первинної ідентифікації пристрою, налаштування мережевих та регіональних параметрів, а також інтеграція робочої станції до корпоративної служби каталогів.

Процес конфігурування розпочинається зі стандартизації імені комп'ютера (Hostname). У корпоративних мережах ручне або випадкове присвоєння імен є неприпустимим, оскільки це ускладнює інвентаризацію та призводить до конфліктів у системі доменних імен (DNS). Для вирішення цього завдання у скрипті реалізовано динамічну генерацію імені на основі апаратних ідентифікаторів. За допомогою командлета Get-CimInstance -ClassName Win32_BIOS скрипт звертається до інтерфейсу управління Windows (WMI/CIM) та зчитує унікальний серійний номер материнської плати (Service Tag). Отримане значення конкатенується зі стандартизованим корпоративним префіксом (наприклад, "CORP-PC-"). Після формування унікального рядка скрипт викликає командлет Rename-Computer для застосування нового імені на рівні операційної системи.

Наступний логічний блок модуля присвячений налаштуванню регіональних стандартів та синхронізації часу. Точність системного годинника є критично важливою вимогою для коректного функціонування криптографічних протоколів автентифікації (зокрема Kerberos), які відхиляють підключення при розбіжності часу між клієнтом та контролером

домену. Для автоматизації цього процесу використовується командлет Set-TimeZone -Id "FLE Standard Time", який встановлює єдиний східноєвропейський часовий пояс. Одночасно з цим скрипт примусово застосовує українську локалізацію для всього інтерфейсу та форматів введення даних за допомогою групи командлетів: Set-WinSystemLocale -SystemLocale uk-UA, Set-WinUserLanguageList та Set-WinHomeLocation. Це гарантує, що всі системні журнали (Event Logs) генеруватимуться з коректними часовими мітками, а кінцевий користувач отримає стандартизоване робоче середовище без необхідності ручного втручання в налаштування мови.

Критичним етапом базового конфігурування є інтеграція робочої станції до локальної мережі підприємства та служби Active Directory. Після успішної перевірки доступності мережевого адаптера (через Get-NetAdapter) скрипт ініціює приєднання до домену за допомогою командлета Add-Computer -DomainName. Оскільки архітектура безпеки категорично забороняє зберігання паролів адміністратора домену у відкритому тексті всередині коду, скрипт використовує попередньо згенерований об'єкт PSCredential або інтегрується із захищеним сховищем секретів (SecretManagement), що гарантує безпечну автентифікацію в мережі.

Оскільки зміна мережевого імені комп'ютера та приєднання до домену вимагають обов'язкового перезавантаження операційної системи, архітектура модуля передбачає коректне керування станом (State Management). Перед завершенням своєї роботи 01-BaseConfig.ps1 створює спеціальний ключ-маркер у системному реєстрі Windows (у гілці HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce). Цей маркер містить команду для автоматичного повторного запуску головного Оркестратора (00-MasterScript.ps1) одразу після перезавантаження комп'ютера. Завдяки цьому процес автоматизації не переривається: після рестарту система самостійно продовжує роботу, зчитує статус успішного

виконання першого модуля і миттєво переходить до наступного етапу - встановлення прикладного програмного забезпечення.

Для наочної демонстрації логіки роботи модуля базового конфігурування було побудовано UML-діаграму діяльності (Додаток А). На схемі відображено розподіл завдань на логічні блоки: ідентифікація, налаштування регіональних параметрів, мережева інтеграція та механізм управління станом через системний реєстр, який завершує виконання модуля ініціалізацією перезавантаження.

3.3 Розробка скрипта автоматизованого встановлення програмного забезпечення

Наступним логічним кроком у побудові стандартизованого операційного середовища (SOE) є насичення операційної системи необхідним прикладним інструментарієм. Для вирішення цього завдання було спроєктовано та розроблено модуль 02-InstallApps.ps1. Головною метою цього сценарію є повна відмова від застарілих та ненадійних методів ручного завантаження інсталяторів (у форматах .exe або .msi) на користь сучасного підходу - використання нативного пакетного менеджера Windows Package Manager (Winget).

Відповідно до пара дигми «Інфраструктура як код» (IaC), архітектура даного модуля передбачає суворе відокремлення програмної логіки від конфігураційних даних. Перелік необхідного програмного забезпечення не інтегрується безпосередньо у вихідний код PowerShell. Натомість було створено зовнішній конфігураційний файл apps.json, який містить структурований масив ідентифікаторів пакетів (наприклад, Google.Chrome, 7zip.7zip). Сценарій на етапі ініціалізації використовує командлет Get-Content для зчитування цього файлу та ConvertFrom-Json для перетворення текстових даних у масив об'єктів PowerShell. Такий підхід забезпечує максимальну гнучкість: системний адміністратор може додавати або видаляти програми з

корпоративного образу просто редагуючи текстовий JSON-файл, не ризикуючи пошкодити логіку самого скрипта розгортання.

Основний робочий процес модуля реалізовано через цикл foreach, який послідовно ітерує кожен ідентифікатор програми з отриманого масиву. Для безпосередньої інсталяції скрипт формує виклик системної утиліти winget.exe. Оскільки виклик зовнішніх консольних програм із PowerShell може мати непередбачувану поведінку щодо потоків виведення, було використано командлет Start-Process. До утиліти пакетного менеджера передається набір аргументів: install, --id, --silent (для приховування графічних вікон інсталятора) та --accept-package-agreements (для автоматичного прийняття ліцензійних угод без участі користувача).

Критично важливою архітектурною особливістю цього виклику є обов'язкове використання параметра -Wait у командлеті Start-Process. Служба Windows Installer (msiexec), яка працює «під капотом» багатьох пакетів Winget, має фундаментальне обмеження: вона не дозволяє запускати кілька процесів інсталяції одночасно. Спроба паралельного встановлення неминуче призводить до фатальної помилки з кодом 1618 («Інсталяція вже виконується»). Параметр -Wait змушує інтерпретатор PowerShell призупинити виконання скрипта і дочекатися повного завершення поточного процесу Winget, перш ніж переходити до наступного пакета у циклі. Це гарантує абсолютну стабільність масового розгортання.

Для забезпечення прозорості та аудиту модуль містить глибоку інтеграцію з системою логування Оркестратора. Після завершення роботи Start-Process скрипт зчитує значення автоматичної системної змінної $LASTEXITCODE. Якщо пакетний менеджер повертає код 0 (що свідчить про успішну інсталяцію), скрипт генерує подію зі статусом SUCCESS. Будь-який інший код виходу класифікується як ERROR, після чого детальне повідомлення записується до централізованого журналу. Завдяки використанню конструкції Try-Catch, помилка встановлення однієї конкретної

програми не призводить до аварійної зупинки всього модуля, дозволяючи системі продовжити інсталяцію наступних пакетів за списком.

Для наочної демонстрації логіки роботи модуля автоматизованого встановлення програмного забезпечення було побудовано UML-діаграму діяльності (Додаток Б). На схемі відображено процеси зчитування конфігураційних даних, циклічної обробки пакетів та механізм синхронізації процесів інсталяції з перевіркою кодів завершення.

3.4 Розробка скрипта оптимізації та очищення системи

Наступним важливим кроком у формуванні корпоративного стандарту робочої станції є глибока оптимізація операційної системи. Базові дистрибутиви Windows, навіть у редакціях Pro та Enterprise, містять значний обсяг попередньо встановленого програмного забезпечення (так званого bloatware), орієнтованого на домашніх користувачів: ігрові сервіси, розважальні платформи та рекламні інтеграції. Окрім цього, система за замовчуванням запускає фонові процеси збору телеметрії. Для автоматизованого видалення цих компонентів та налаштування мінімалістичного інтерфейсу було розроблено модуль 03-RemoveBloatware.ps1.

Процес очищення системи від небажаних додатків Universal Windows Platform (UWP) вимагає комплексного підходу, оскільки просте видалення програми в поточному сеансі не гарантує її відсутності в майбутньому. У розробленому скрипті імплементовано дворівневий механізм видалення. Спочатку формується масив регулярних виразів із назвами небажаних пакетів (наприклад, *BingWeather*, *XboxApp*, *ZuneVideo*). Далі скрипт у циклі використовує конвеєр PowerShell для передачі об'єктів від командлета Get-AppxPackage -AllUsers до Remove-AppxPackage, що деінсталює додатки для всіх існуючих профілів. На другому рівні використовується командлет Get-AppxProvisionedPackage -Online у поєднанні з Remove-

AppxProvisionedPackage. Ця операція фізично вирізає інсталяційні пакети зі сховища базового образу (Windows Image), гарантуючи, що при створенні профілю для нового співробітника ці додатки більше не будуть встановлені автоматично.

Другим логічним блоком модуля є підвищення рівня конфіденційності та вивільнення апаратних ресурсів шляхом відключення систем збору діагностичних даних. За допомогою командлетів Stop-Service та Set-Service скрипт ідентифікує та переводить у стан примусового відключення (Disabled) службу Connected User Experiences and Telemetry (внутрішня назва DiagTrack), а також службу підтримки роздрібної демонстрації (RetailDemo). Оскільки маніпуляції зі службами можуть викликати виключення, якщо служба вже зупинена, код обгорнуто в конструкцію Try-Catch із параметром -ErrorAction SilentlyContinue. Крім того, скрипт дублює блокування телеметрії на рівні системного реєстру, створюючи ключ AllowTelemetry зі значенням 0 у гілці політик HKLM:\SOFTWARE\Policies\Microsoft\Windows\DataCollection.

Особливе місце в архітектурі модуля займає конфігурування інтерфейсу користувача (UI), зокрема оптимізація панелі завдань Windows 11. У корпоративному середовищі системні віджети погоди, інтегрований пошук новин та персональний чат Teams часто відволікають працівників та споживають оперативну пам'ять. Оскільки Microsoft на рівні ядра операційної системи обмежує можливості скриптів безпосередньо змінювати закріплені ярлики активної сесії (в рамках політики Zero Trust), модуль використовує метод модифікації реєстру поточного користувача.

За допомогою командлета Set-ItemProperty скрипт звертається до гілки HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced та змінює значення параметрів типу DWORD. Встановлення параметра TaskbarDa у нуль вимикає панель віджетів, а TaskbarMn - приховує іконку чату. Аналогічним чином через гілку Search вимикається інтегрований рядок пошуку (SearchboxTaskbarMode). Щоб застосовані зміни інтерфейсу набули

чинності миттєво, без необхідності перезавантаження всієї робочої станції, фінальним кроком модуля є примусовий перезапуск процесу Провідника Windows за допомогою команди Stop-Process -Name explorer -Force.

Для кращого розуміння та візуалізації алгоритму роботи модуля оптимізації було побудовано UML-діаграму діяльності (рис. 3.4). Схема демонструє послідовність виконання завдань: дворівневе видалення UWP-пакетів, блокування служб телеметрії та фінальну модифікацію ключів реєстру для налаштування корпоративної панелі завдань.

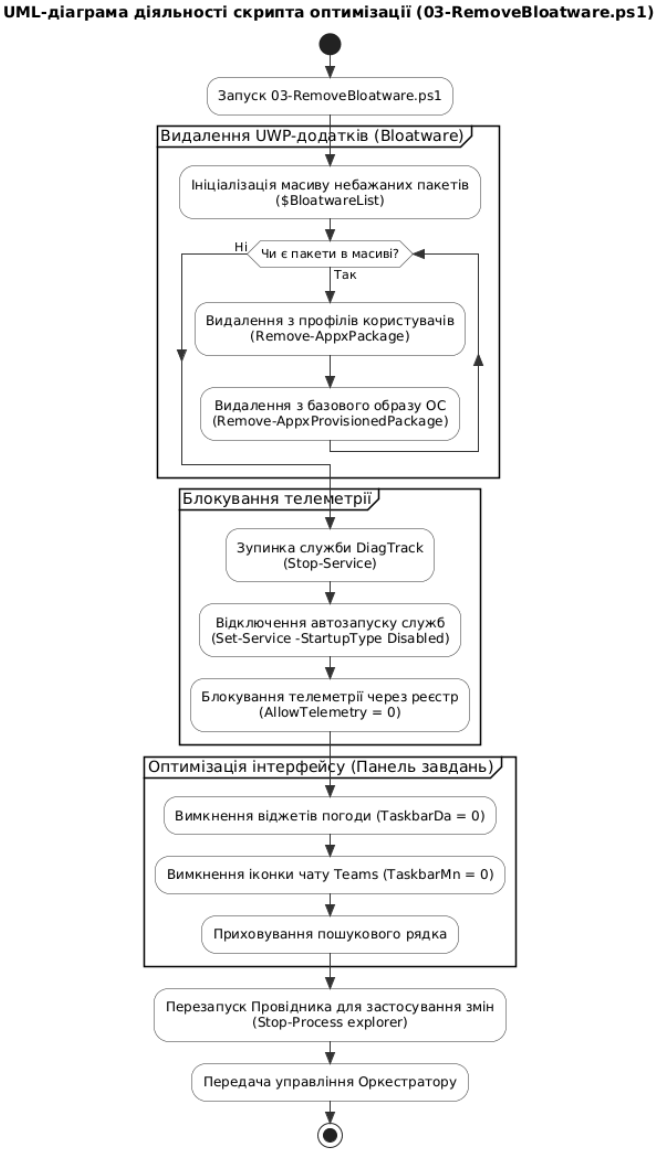


Рисунок 3.4 - UML-діаграма діяльності скрипта оптимізації та очищення системи

3.5 Розробка скрипта налаштування політик кібербезпеки

Завершальним, але найбільш критичним етапом формування стандартизованого операційного середовища (SOE) є імплементація політик інформаційної безпеки (Hardening). Операційна система Windows «з коробки» налаштована на забезпечення максимальної зворотної сумісності зі старим обладнанням та протоколами, що суттєво розширює площу потенційної кібератаки. Для автоматизації процесу загартування системи та приведення її у відповідність до корпоративних стандартів безпеки було розроблено фінальний модуль - 04-SecurityConfig.ps1.

Перший логічний блок скрипта спрямований на усунення вразливостей на рівні мережевих протоколів. Найбільшу загрозу в сучасних локальних мережах становить застарілий протокол спільного доступу до файлів Server Message Block версії 1 (SMBv1), який неодноразово використовувався як основний вектор поширення програм-вимагачів (зокрема вірусу WannaCry). У скрипті реалізовано деактивацію цього протоколу за допомогою командлета Set-SmbServerConfiguration -EnableSMB1Protocol $false -Force. Крім того, для захисту від атак типу індукування (spoofing) та перехоплення облікових даних, скрипт здійснює пряму модифікацію системного реєстру для відключення протоколів Link-Local Multicast Name Resolution (LLMNR) та NetBIOS over TCP/IP, які є надлишковими в мережах із повноцінно функціонуючими DNS-серверами.

Другим етапом є програмне конфігурування вбудованого захисту кінцевих точок - Microsoft Defender Antivirus. Замість покладання на стандартні налаштування, скрипт використовує командлет Set-MpPreference для активації посилених механізмів виявлення. Через цей командлет примусово вмикається хмарний захист (Cloud-delivered protection), автоматичне надсилання зразків підозрілих файлів для аналізу (MAPS) та захист від потенційно небажаних програм (PUA). Особлива увага приділена правилам скорочення поверхні атаки (Attack Surface Reduction, ASR). Скрипт

застосовує специфічні GUID-ідентифікатори правил ASR, щоб заборонити офісним додаткам (наприклад, Word чи Excel) створювати дочірні виконувані процеси, що на рівні ядра блокує виконання шкідливих макросів із фішингових електронних листів.

Найбільш технологічно складним завданням у межах даного модуля стала реалізація автоматизованої активації повнодискового шифрування BitLocker. З огляду на те, що скрипт може виконуватися на різних апаратних конфігураціях (включаючи віртуальні машини), прямий виклик команди шифрування без попередніх перевірок неминуче призвів би до критичних помилок. Тому алгоритм розпочинається зі звернення до командлета Get-Tpm. Скрипт перевіряє властивості TpmPresent та TpmReady, щоб переконатися в наявності та працездатності апаратного (або віртуального vTPM) криптографічного модуля.

У разі успішної перевірки TPM скрипт викликає командлет Get-BitLockerVolume -MountPoint "C:" для визначення поточного статусу системного тому. Якщо диск ще не зашифровано, ініціюється процес шифрування командою Enable-BitLocker з використанням параметра -UsedSpaceOnly, що значно прискорює процес на щойно розгорнутих системах. З метою запобігання безповоротній втраті корпоративних даних у випадку апаратного збою материнської плати, скрипт налаштований на автоматичну генерацію 48-значного цифрового пароля відновлення. Цей ключ експортується та зберігається у захищений текстовий файл у централізованій директорії логів (C:\DeploymentLogs), звідки він згодом може бути безпечно переміщений адміністратором у доменне сховище Active Directory.

Уся логіка роботи з криптографією обгорнута в конструкцію Try-Catch. Якщо активація BitLocker неможлива (наприклад, через відсутність TPM 2.0 на старій робочій станції), скрипт перехоплює виключення, записує помилку рівня ERROR у структурований журнал Оркестратора, але дозволяє системі

завершити свою роботу штатно, передаючи управління назад головному процесу.

Для детальної візуалізації алгоритму застосування політик безпеки та обробки криптографічних функцій було створено UML-діаграму діяльності (рис. 3.5). Вона наочно показує етапи мережевого загартування, налаштування антивірусного захисту та умовні переходи, пов'язані з перевіркою апаратного модуля TPM перед початком шифрування диска.

Рисунок 3.5 - UML-діаграма діяльності скрипта налаштування політик кібербезпеки

3.6 Тестування та верифікація результатів автоматизованого розгортання

Для практичної перевірки працездатності, відмовостійкості та ефективності розробленого комплексу PowerShell-модулів було проведено повноцінне тестування у віртуалізованому ізольованому середовищі. Тестовий стенд розгорнуто на базі гіпервізора з інстальованим чистим «тонким образом» (Thin Image) операційної системи Windows 11.

На початку експерименту операційна система мала стандартний вигляд після встановлення (Out-of-Box Experience). Панель завдань містила системні віджети, іконку чату Teams та інші компоненти, які підлягають подальшій оптимізації (рис. 3.6).

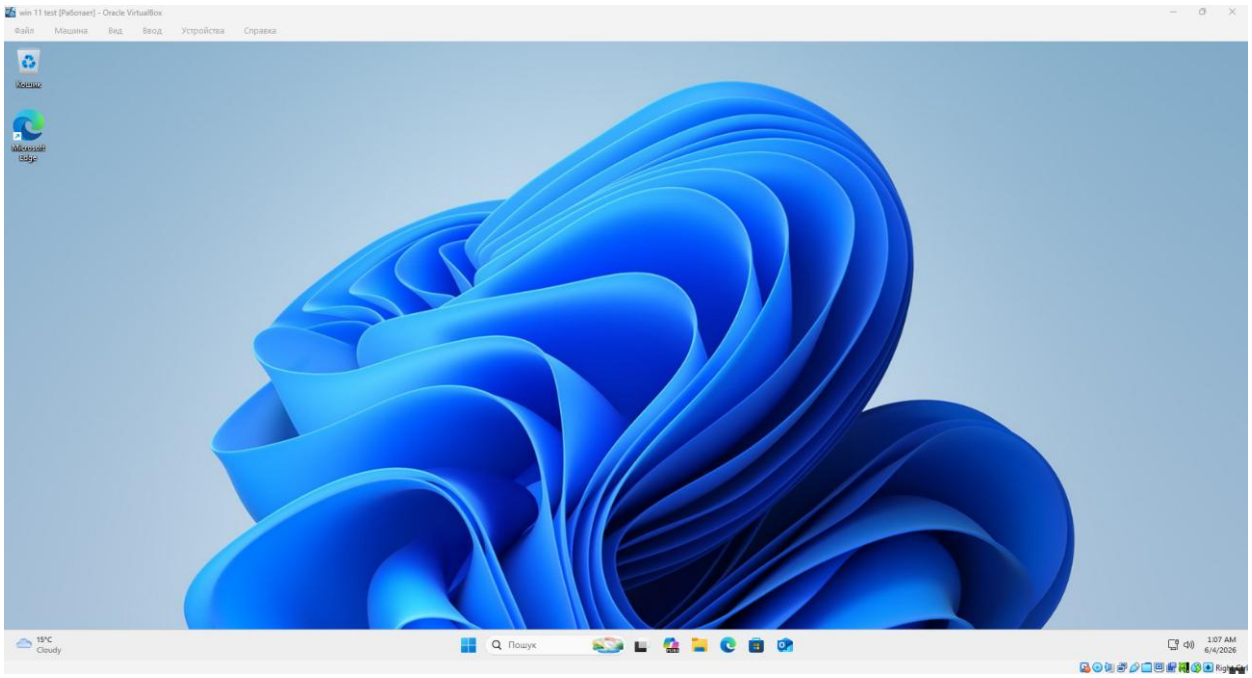


Рисунок 3.6 - Початковий стан операційної системи Windows 11 перед початком автоматизованого розгортання

Підготовлені скрипти автоматизації разом із конфігураційним файлом apps.json, який містить перелік необхідного корпоративного програмного забезпечення, були скопійовані до локальної робочої директорії C:\skript на цільовій машині. Структура файлів повністю відповідає спроєктованій

модульній архітектурі, де кожен скрипт відповідає за свою специфічну область конфігурування (рис. 3.7).

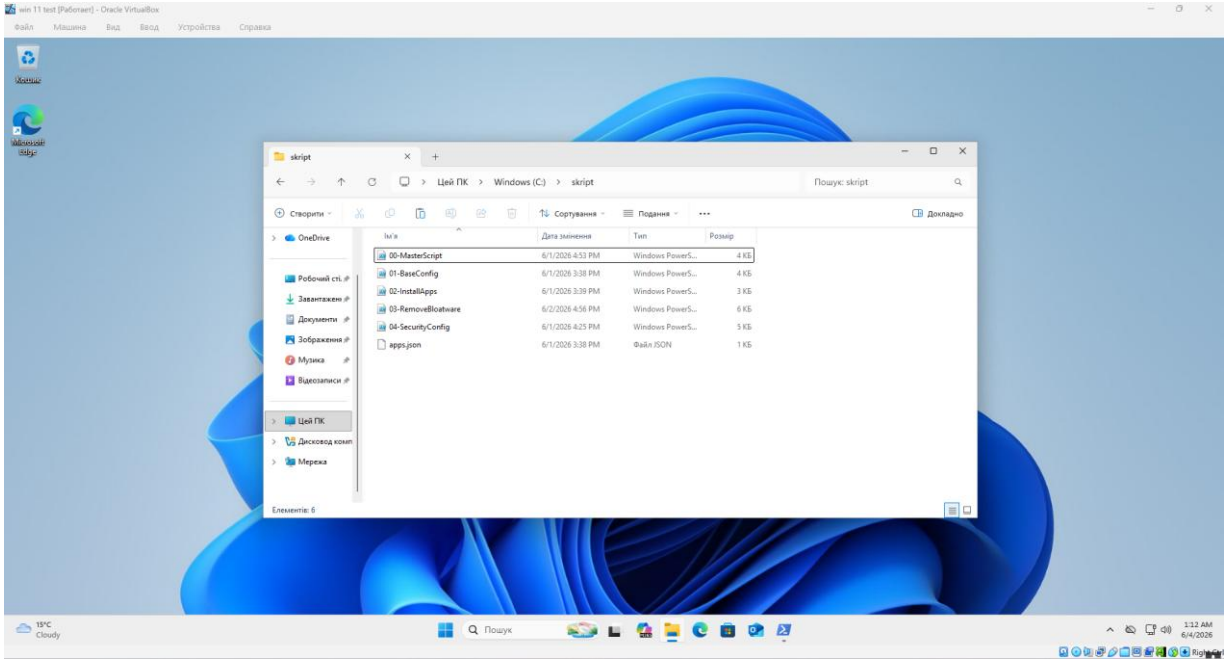


Рисунок 3.7 - Робоча директорія з підготовленими модулями розгортання та конфігураційним JSON-файлом

Процес автоматизації ініціюється виключно з консолі PowerShell, запущеної з правами локального адміністратора. Оскільки за замовчуванням у Windows виконання сторонніх скриптів заблоковано заради безпеки, першим кроком адміністратора є тимчасове послаблення політики виконання за допомогою командлета Set-ExecutionPolicy RemoteSigned -Force. Після цього запускається головний керуючий модуль (Оркестратор) - 00-MasterScript.ps1, який перехоплює подальше управління системою (рис. 3.8).

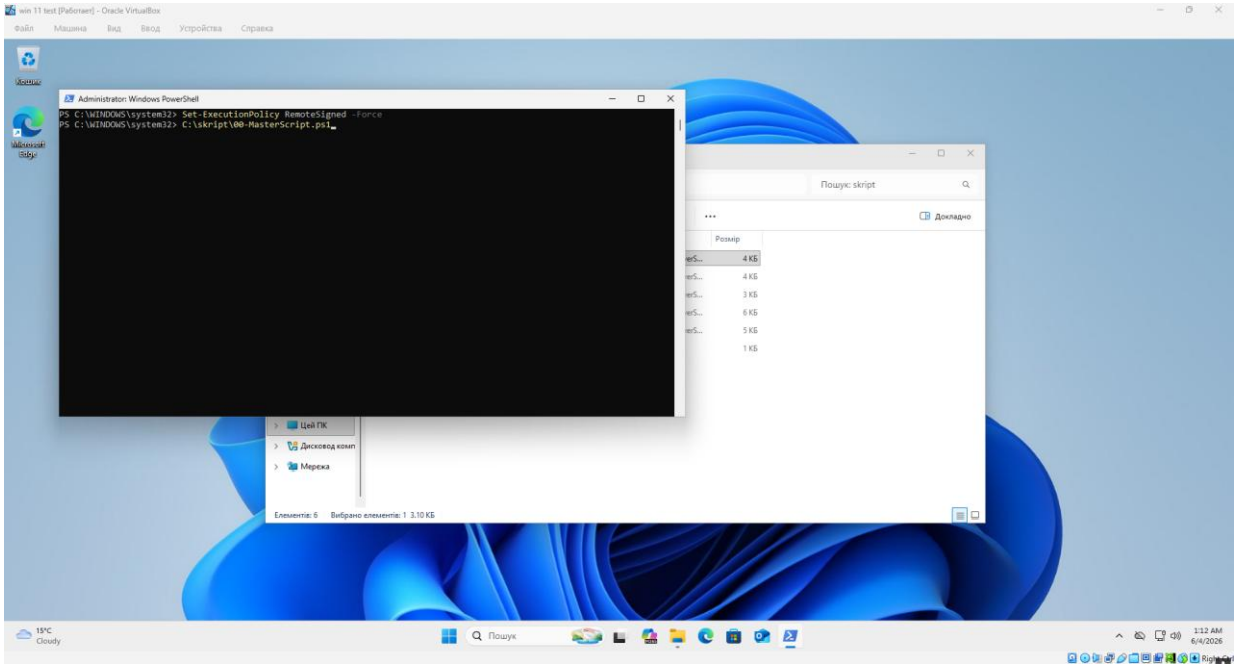


Рисунок 3.8 - Запуск головного керуючого скрипта (Оркестратора) в консолі адміністратора

Під час виконання Оркестратор послідовно викликає підпорядковані модулі. На рисунку 3.9 зафіксовано процес роботи системи: модуль базового конфігурування 01-BaseConfig.ps1 успішно встановив українську локалізацію та змінив ім'я комп'ютера. Далі управління перейшло до модуля 02-InstallApps.ps1, який у фоновому режимі ініціював завантаження та інсталяцію веббраузера Google Chrome через пакетний менеджер Winget, зчитуючи параметри з файлу apps.json. Усі дії супроводжуються інформативними повідомленнями в консолі з точними мітками часу.

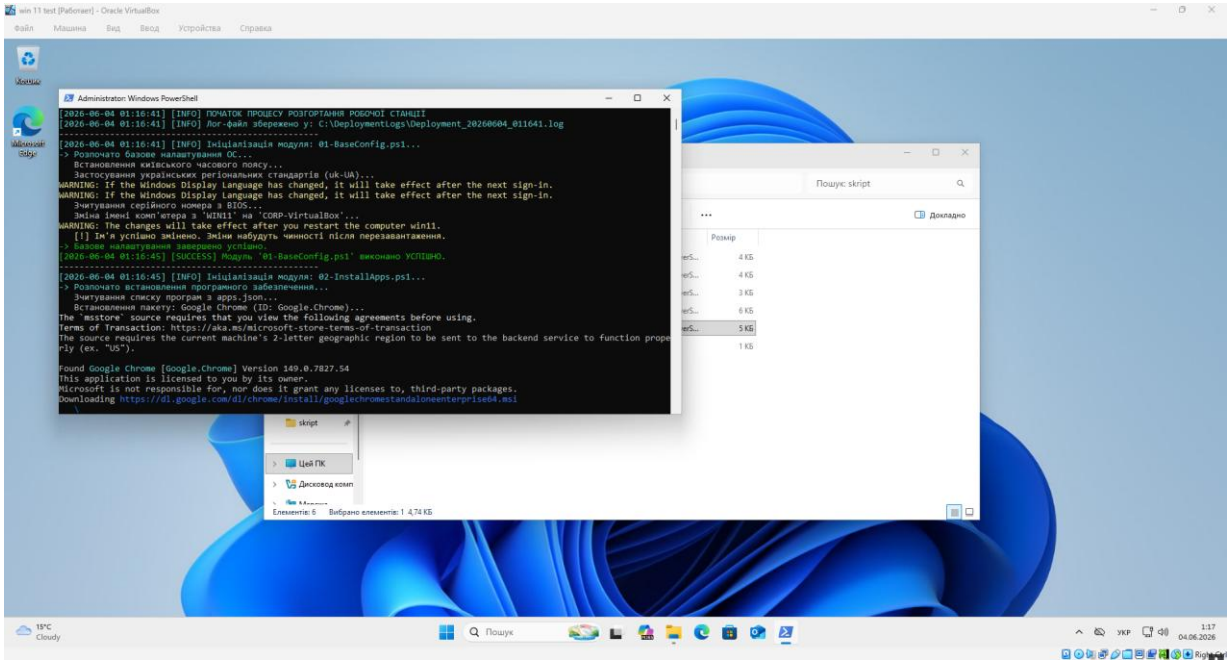


Рисунок 3.9 - Процес виконання базового конфігурування та автоматизованого встановлення ПЗ

Після завершення встановлення прикладного програмного забезпечення скрипт успішно виконує модулі 03-RemoveBloatware.ps1 та 04-SecurityConfig.ps1. На рисунку 3.10 продемонстровано фінальний вивід консолі, який підтверджує успішне блокування телеметрії, видалення системних віджетів та конфігурування антивірусного захисту Defender. Візуальним підтвердженням роботи скрипта є поява ярлика встановленого браузера Chrome на робочому столі та оптимізована панель завдань, з якої автоматично зникли віджети погоди та іконка чату Teams.

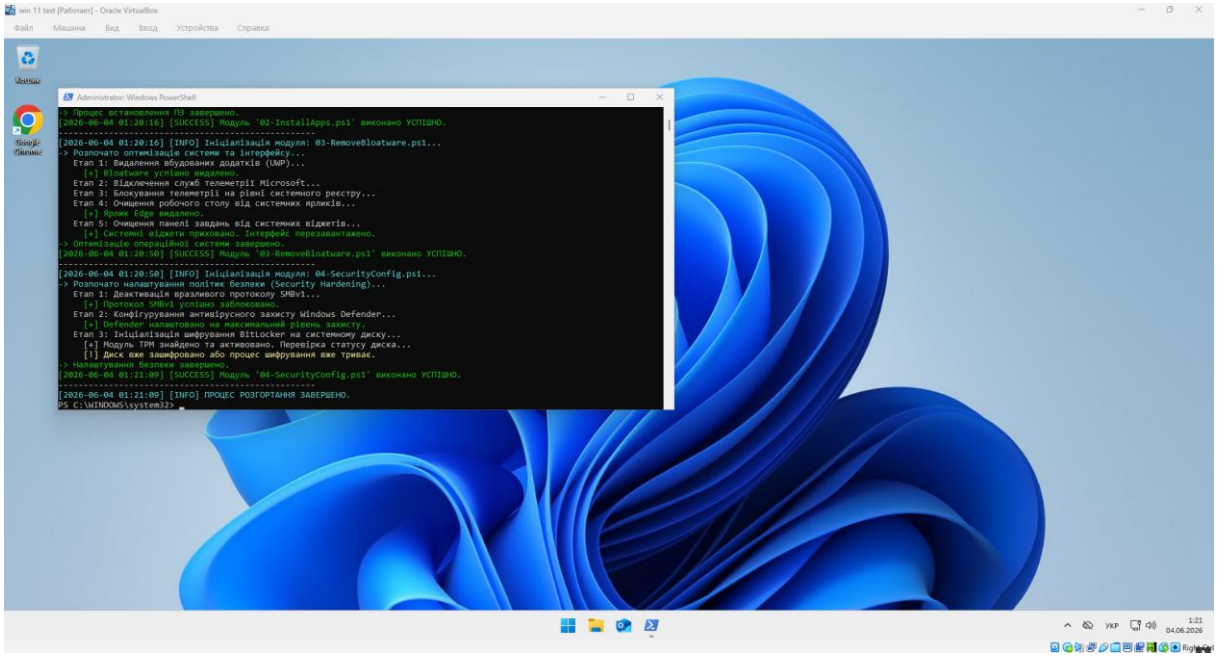


Рисунок 3.10 - Успішне завершення роботи комплексу скриптів та візуальні зміни інтерфейсу ОС

Останнім етапом тестування стала перевірка роботи підсистеми аудиту. Як і було спроєктовано в архітектурі, Оркестратор автоматично згенерував структурований текстовий лог-файл у директорії C:\DeploymentLogs\. Відкривши цей файл (рис. 3.11), можна переконатися, що система коректно зафіксувала час початку та успішного завершення ([SUCCESS]) кожного з чотирьох конфігураційних модулів. Кодування файлу UTF-8 with BOM збереглося коректно, що підтверджує готовність логів до подальшого машинного аналізу.

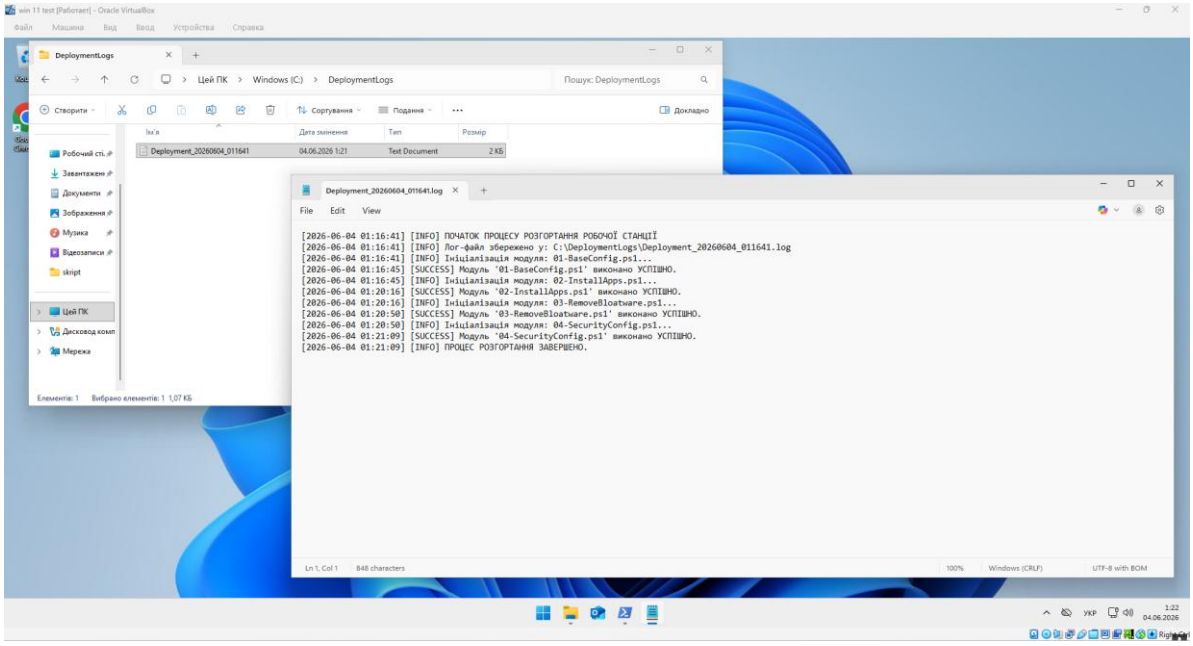


Рисунок 3.11 - Згенерований файл системного журналу (логування) з результатами розгортання

Проведене тестування підтвердило абсолютну працездатність розробленої системи. Скрипти відпрацювали без жодних критичних збоїв, успішно інтегрувавши необхідне ПЗ та застосувавши жорсткі політики безпеки. Загальний час, витрачений на виконання всього циклу налаштування віртуальної машини, склав менше двох хвилин (не враховуючи час завантаження пакетів з мережі Інтернет), що повністю підтверджує економічну та часову ефективність впровадження концепції «Інфраструктура як код» у порівнянні з ручним системним адмініструванням.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи реалізовано комплексну систему автоматизації розгортання робочих станцій у корпоративному середовищі за допомогою скриптів PowerShell. Проєкт довів, що перехід від ручного конфігурування до парадигми “Інфраструктура як код” є необхідним кроком для посилення безпеки, стандартизації та оптимізації бізнес-процесів компаній будь-якого масштабу.

Теоретичний аналіз підтвердив, що традиційне ручне адміністрування є економічно неефективним. Воно не лише забирає багато часу IT-спеціалістів, але й провокує зміщення конфігурацій, що створює критичні вразливості в інфраструктурі. Водночас комерційні Enterprise-рішення, попри свій потужний функціонал, часто є занадто дорогими або складними у розгортанні для багатьох організацій.

Саме тому як основний інструмент було обрано PowerShell. Його об'єктно-орієнтована архітектура (.NET), глибока інтеграція з API Windows, системним реєстром та WMI надають фундаментальні переваги. Головний плюс полягає в тому, що PowerShell вбудовано в систему за замовчуванням: це дозволяє налаштовувати ПК одразу після встановлення ОС, без розгортання додаткових агентів.

В основу архітектури покладено гібридну стратегію розгортання: встановлюється чистий “тонкий образ” системи, після чого всі кастомізації виконуються післяінсталяційними (post-install) скриптами. Для керування цим процесом розроблено гнучку модульну систему. Вона складається з головного оркестратора та окремих модулів для базової конфігурації, інсталяції ПЗ і налаштування безпеки. Використання конструкцій Try-Catch зробило процес відмовостійким, а структурована система логування фіксує події у форматі JSON та в системних журналах Windows Event Log.

Значним досягненням стала автоматизація встановлення програм через нативний пакетний менеджер Winget. Це дозволило відмовитися від ручного завантаження інсталяторів: скрипти динамічно зчитують конфігураційні файли, перевіряють наявність програм та завантажують актуальні версії, що кардинально зменшує навантаження на відділ підтримки.

Особливу увагу приділено кібербезпеці, спеціальний модуль автоматизує вимкнення вразливих протоколів (SMBv1, LLMNR, NetBIOS) та системної телеметрії. Забезпечено автоматичне шифрування диска BitLocker із перевіркою чипа TPM, а також налаштування Microsoft Defender і правил Attack Surface Reduction. Це створює надійний захист комп'ютера ще до застосування доменних політик.

Працездатність коду перевірено на тестовому стенді VirtualBox. Система успішно пройшла стрес-тести з імітацією розривів мережі, підтвердивши здатність продовжувати розгортання після перезавантажень завдяки маркерам стану в реєстрі.

Практичне впровадження комплексу скоротило час налаштування однієї робочої станції з 90–120 до 15–20 хвилин (приріст швидкості понад 80%). Оскільки рішення базується на вбудованих інструментах ОС, воно є абсолютно безкоштовним, усуває ризики “людського фактору” та повністю готове до використання в реальних IT-інфраструктурах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гавриленко О. В., Мельник А. О. Сучасні методи автоматизації розгортання IT-інфраструктури підприємства // Вісник Національного університету “Львівська політехніка”. Серія: Інформаційні системи та мережі. 2021. Вип. 9. С. 45–52 (дата звернення: 06.11.2025).

2. Morris K. Infrastructure as Code: Dynamic Systems for the Cloud Age. 2nd ed. O'Reilly Media, 2020 (дата звернення: 06.11.2025).

3. Darden A. PowerShell for Sysadmins: Workflow Automation Made Easy. San Francisco: No Starch Press, 2020 (дата звернення: 06.11.2025).

4. Петренко В. С., Сидоренко М. В. Порівняльний аналіз систем керування конфігураціями (MECM, Ansible, Puppet) для гетерогенних мереж // Кібербезпека: освіта, наука, техніка. 2020. № 3(11). С. 88–97 (дата звернення: 10.11.2025).

5. Левченко О. М. Технології адміністрування корпоративних мереж: навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2022 (дата звернення: 01.12.2025).

6. Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security (SP 800-46 Rev. 2) // National Institute of Standards and Technology (NIST). 2021 (дата звернення: 01.12.2025).

7. Рекомендації щодо захисту робочих станцій в державних та корпоративних мережах // Державна служба спеціального зв’язку та захисту інформації України (ДССЗЗІ) 2022. (дата звернення: 01.12.2025).

8. Jones D., Hicks J. Learn PowerShell in a Month of Lunches. 3rd ed. Manning Publications, 2020 (дата звернення: 01.12.2025).

9. Fultz S. Practical Chocolatey: The Package Manager for Windows. Apress, 2021 (дата звернення: 17.12.2025).

10. Lim T. Windows PowerShell 5.1 and 7.2: IT Pro Crash Course. Packt Publishing, 2022 (дата звернення: 17.12.2025).

11. Ruest N., Ruest A. Mastering Microsoft Endpoint Manager: Deploy and manage Windows 10, Windows 11, and iOS/Android devices. Sybex, 2021 (дата звернення: 17.12.2025).

12. PowerShell Documentation // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/powershell/ (дата звернення: 17.12.2025).

13. Windows Remote Management (WinRM) Overview // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/winrm/portal (дата звернення: 22.02.2026).

14. Use the winget tool to install and manage applications // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/package-manager/winget/ (дата звернення: 22.02.2026).

15. Just Enough Administration (JEA) Overview // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/powershell/scripting/learn/remoting/jea/overview (дата звернення: 22.02.2026).

16. Windows PowerShell Desired State Configuration (DSC) // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/powershell/dsc/overview (дата звернення: 22.02.2026).

17. How AMSI helps you defend against malware // Microsoft Security. – URL: https://learn.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps (дата звернення: 22.02.2026).

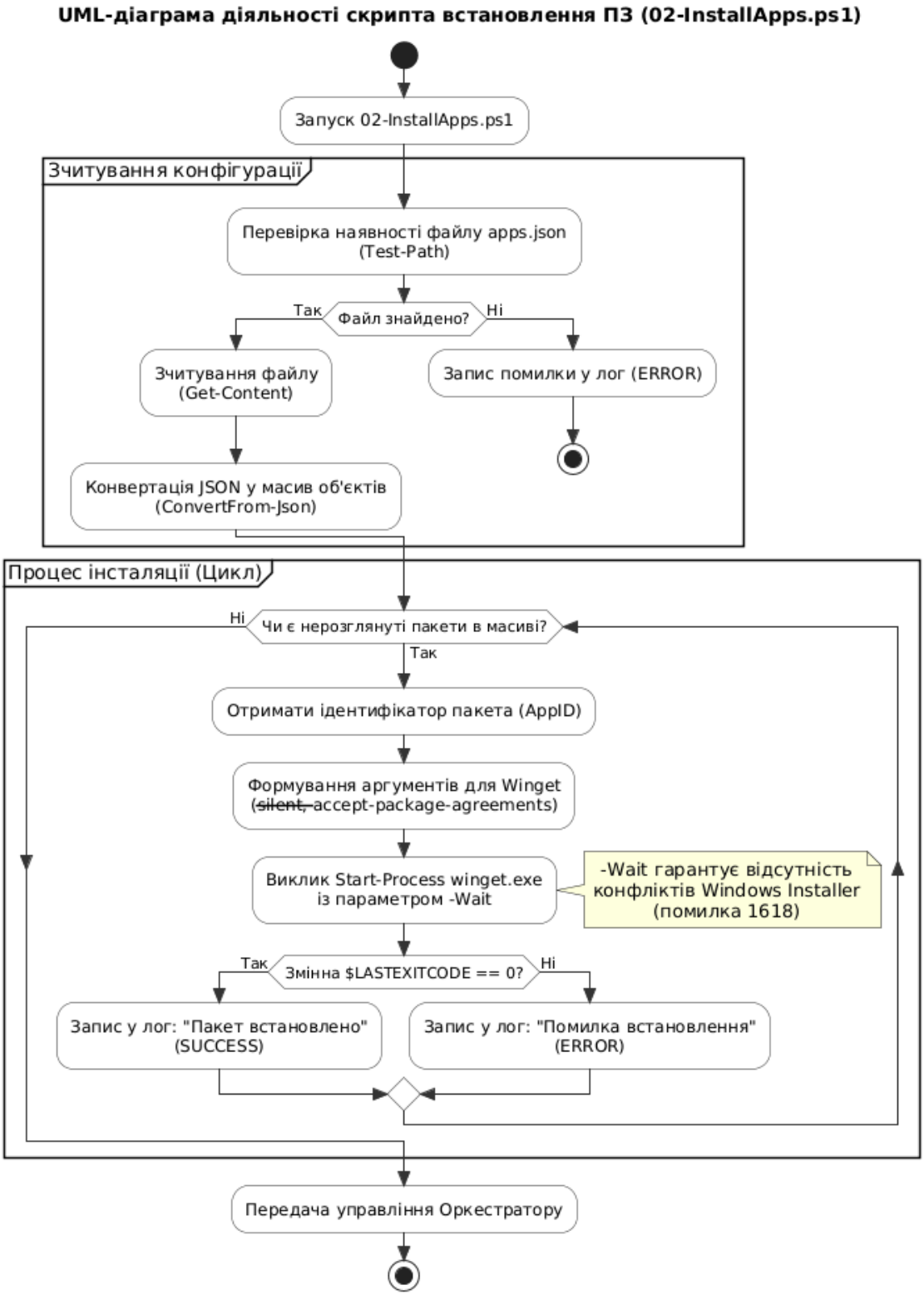
18. CIS Microsoft Windows Desktop Benchmarks // Center for Internet Security (CIS). – URL: https://www.cisecurity.org/benchmark/microsoft_windows_desktop (дата звернення: 22.02.2026) (дата звернення: 22.02.2026).

19. Коваленко І. І. Забезпечення надійності та аудиту в скриптах системного адміністрування на базі ОС Windows // Сучасний захист інформації. 2023. № 1(45). С. 22–29 (дата звернення: 22.02.2026).

20. Положення про організацію заходів із забезпечення інформаційної безпеки (Затверджено Постановою Правління НБУ) // Національний банк України. 2021 (дата звернення: 22.02.2026).

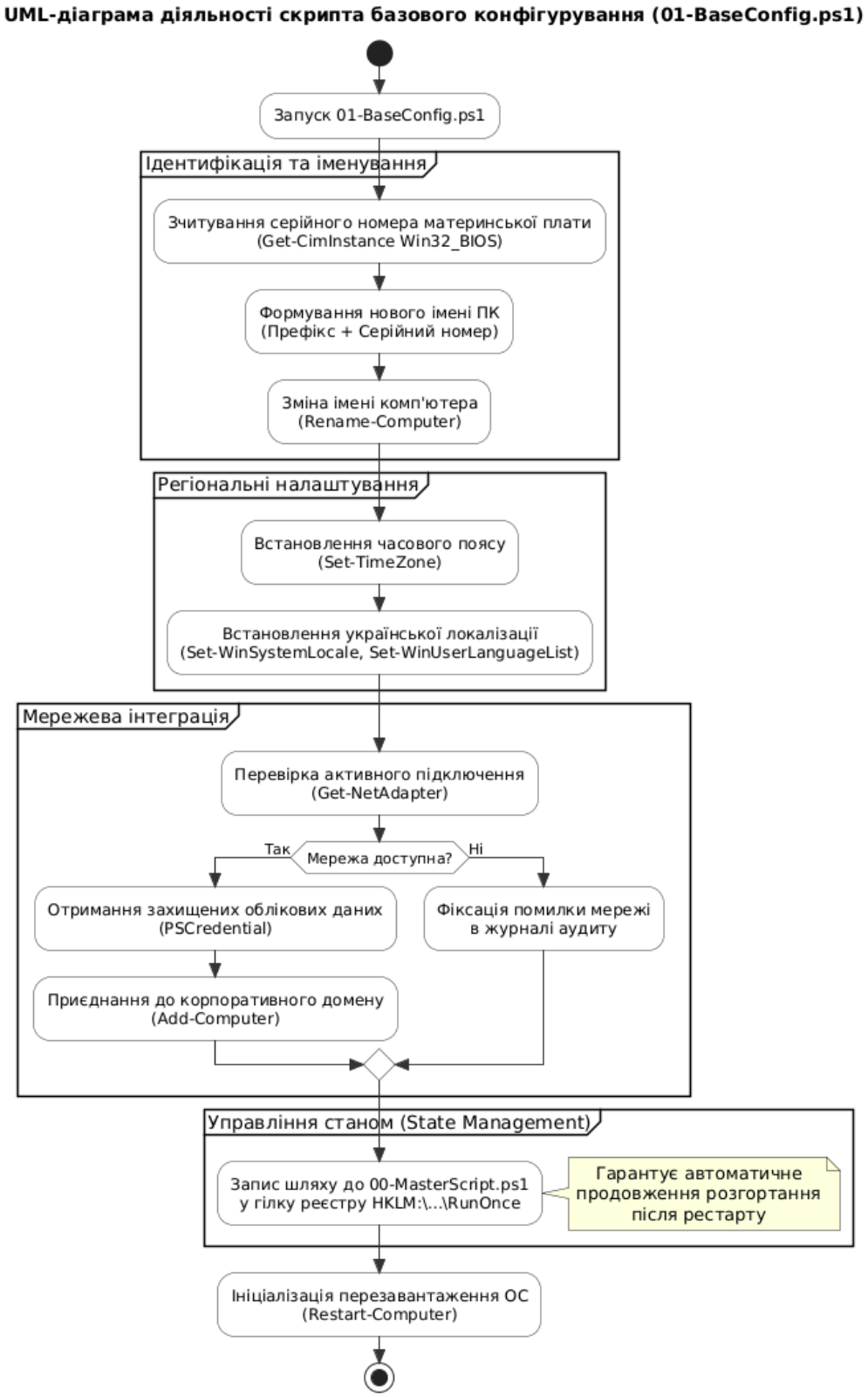
ДОДАТКИ

Додаток А



UML-діаграма діяльності скрипта базового конфігурування

Додаток Б



UML-діаграма діяльності скрипта автоматизованого встановлення ПЗ

Додаток С - ЛІСТИНГ ПРОГРАМНОГО КОДУ

00-MasterScript

```
<#
.SYNOPSIS
    Головний скрипт (Оркестратор) для автоматизації розгортання робочої
станції.
.DESCRIPTION
    Цей скрипт послідовно запускає 4 модулі налаштування, фіксує результати
    їх виконання та генерує загальний лог-файл.
#>
if
(!([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]:
:GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrato
r)) {
    Write-Warning "Будь ласка, запустіть PowerShell від імені
Адміністратора!"
    Break
}
$ScriptDirectory = $PSScriptRoot
$LogFolder = "C:\DeploymentLogs"
$LogFile = Join-Path $LogFolder "Deployment_$(Get-Date -Format
'yyyyMMdd_HHmmss').log"
if (!(Test-Path $LogFolder)) {
    New-Item -ItemType Directory -Force -Path $LogFolder | Out-Null
}
$ScriptsToRun = @(
    "01-BaseConfig.ps1",
    "02-InstallApps.ps1",
    "03-RemoveBloatware.ps1",
    "04-SecurityConfig.ps1"
)
Function Write-Log {
    Param(
        [Parameter(Mandatory=$true)][string]$Message,
        [ValidateSet("INFO", "SUCCESS", "WARNING", "ERROR")][string]$Level =
"INFO"
    )
    $Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
```

Продовження додатку С

```
  $LogEntry = "[$Timestamp] [$Level] $Message"
```

```
    # ТУТ ВИПРАВЛЕНО: Додано примусове кодування UTF8 для збереження
українських літер
    Add-Content -Path $LogFile -Value $LogEntry -Encoding UTF8
    switch ($Level) {
        "SUCCESS" { Write-Host $LogEntry -ForegroundColor Green }
        "ERROR"   { Write-Host $LogEntry -ForegroundColor Red }
        "WARNING" { Write-Host $LogEntry -ForegroundColor Yellow }
        "INFO"    { Write-Host $LogEntry -ForegroundColor Cyan }
    }
}
Clear-Host
Write-Log "ПОЧАТОК ПРОЦЕСУ РОЗГОРТАННЯ РОБОЧОЇ СТАНЦІЇ" "INFO"
Write-Log "Лог-файл збережено у: $LogFile" "INFO"
Write-Host "------------------------------------------------"
foreach ($ScriptName in $ScriptsToRun) {
    $FullPath = Join-Path $ScriptDirectory $ScriptName
    if (Test-Path $FullPath) {
        Write-Log "Ініціалізація модуля: $ScriptName..." "INFO"
        try {
            $execution = & $FullPath -ErrorAction Stop
            Write-Log "Модуль '$ScriptName' виконано УСПІШНО." "SUCCESS"
        }
        catch {
            $ErrorMessage = $_.Exception.Message
            Write-Log "ПОМИЛКА під час виконання '$ScriptName':
$ErrorMessage" "ERROR"
            Write-Log "Перехід до наступного модуля..." "WARNING"
        }
    }
    else {
        Write-Log "Файл не знайдено: $ScriptName. Перевірте наявність файлу в
папці." "ERROR"
    }
    Write-Host "------------------------------------------------"
}
```

Продовження додатку С

```
Write-Log "ПРОЦЕС РОЗГОРТАННЯ ЗАВЕРШЕНО." "INFO"
```

01-BaseConfig

```
<#
.SYNOPSIS
    Модуль 1: Базове налаштування ОС
```

```
.DESCRIPTION
    Скрипт зчитує серійний номер материнської плати, перейменовує ПК,
    встановлює українську мову системи та київський часовий пояс.
#>

Write-Host "-> Розпочато базове налаштування ОС..." -ForegroundColor Cyan

# 1. Налаштування часового поясу (Київ / FLE Standard Time)
try {
    Write-Host "   Встановлення київського часового поясу..."
    Set-TimeZone -Id "FLE Standard Time" -ErrorAction Stop
} catch {
    Write-Warning "   Не вдалося встановити часовий пояс. Можливо, він вже
встановлений."
}

# 2. Налаштування української локалізації (uk-UA)
try {
    Write-Host "   Застосування українських регіональних стандартів (uk-
UA)..."
    Set-WinSystemLocale -SystemLocale uk-UA -ErrorAction Stop
    Set-Culture -CultureInfo uk-UA -ErrorAction Stop
    # Встановлюємо українську та англійську мови введення
    Set-WinUserLanguageList -LanguageList uk-UA, en-US -Force -ErrorAction
Stop
} catch {
    Write-Warning "   Виникла помилка під час зміни мовних налаштувань."
}

# 3. Отримання серійного номера та перейменування ПК
try {
```

Продовження додатку С

```
    Write-Host "   Зчитування серійного номера з BIOS..."
    $Bios = Get-CimInstance -ClassName Win32_BIOS -ErrorAction Stop
    $SerialNumber = $Bios.SerialNumber

    # Перевірка для віртуальних машин (VirtualBox/Hyper-V часто віддають
порожній або стандартний номер)
    if ([string]::IsNullOrWhiteSpace($SerialNumber) -or $SerialNumber -eq
"0") {
        Write-Host "   [!] Виявлено віртуальне середовище. Генеруємо
випадковий серійний номер..." -ForegroundColor Yellow
```

```
        $SerialNumber = "VM-" + (Get-Random -Minimum 1000 -Maximum 9999)
    }

    # Формуємо нове ім'я (префікс CORP- та перші 10 символів серійника, щоб
не перевищити ліміт Windows у 15 символів)
    $SafeSerial = $SerialNumber.Substring(0,
[math]::Min($SerialNumber.Length, 10))
    $NewComputerName = "CORP-$SafeSerial"
    $CurrentName = $env:COMPUTERNAME

    if ($CurrentName -ne $NewComputerName) {
        Write-Host "   Зміна імені комп'ютера з '$CurrentName' на
'$NewComputerName'..."
        Rename-Computer -NewName $NewComputerName -Force -ErrorAction Stop
        Write-Host "   [!] Ім'я успішно змінено. Зміни набудуть чинності
після перезавантаження." -ForegroundColor Yellow
    } else {
        Write-Host "   Ім'я комп'ютера вже відповідає корпоративному
стандарту ($CurrentName)."
    }
} catch {
    Throw "Критична помилка під час перейменування ПК:
$($_.Exception.Message)"
}

Write-Host "-> Базове налаштування завершено успішно." -ForegroundColor Green
```

Продовження додатку С

02-InstallApps

```
<#
.SYNOPSIS
    Модуль 2: Автоматизоване встановлення програмного забезпечення.
.DESCRIPTION
    Скрипт зчитує конфігураційний файл apps.json та встановлює
    кожен пакет за допомогою Windows Package Manager (Winget) у фоновому
режимі.
#>
Write-Host "-> Розпочато встановлення програмного забезпечення..." -
ForegroundColor Cyan
# Визначаємо шлях до JSON-файлу в тій самій папці, де лежить скрипт
```

```
$JsonPath = Join-Path $PSScriptRoot "apps.json"
# Перевірка наявності файлу конфігурації
if (-not (Test-Path $JsonPath)) {
    Throw "Файл конфігурації $JsonPath не знайдено!"
}
Write-Host "   Зчитування списку програм з apps.json..."
try {
    $JsonContent = Get-Content -Path $JsonPath -Raw | ConvertFrom-Json
    $AppList = $JsonContent.Apps
} catch {
    Throw "Помилка читання JSON файлу. Перевірте синтаксис файлу apps.json."
}
# Цикл встановлення кожної програми зі списку
foreach ($App in $AppList) {
    Write-Host "   Встановлення пакету: $($App.Name) (ID: $($App.ID))..."
    try {
        # Формуємо аргументи для тихої інсталяції без участі користувача
        $InstallArgs = "install --id $($App.ID) --exact --silent --accept-
package-agreements --accept-source-agreements"

        # Використовуємо Start-Process з параметром -Wait, як ми й описували
в дипломі
        # Це запобігає конфліктам служби Windows Installer
```

Продовження додатку С

```
        $Process = Start-Process -FilePath "winget" -ArgumentList
$InstallArgs -Wait -NoNewWindow -PassThru
        # Перевірка коду завершення процесу
        if ($Process.ExitCode -eq 0) {
            Write-Host "     [+] $($App.Name) встановлено успішно." -
ForegroundColor Green
        } else {
            Write-Warning "     [-] Помилка встановлення $($App.Name). Код
завершення: $($Process.ExitCode)"
        }
    } catch {
        Write-Warning "     [!] Не вдалося ініціювати встановлення
$($App.Name). Деталі: $($_.Exception.Message)"
    }
}

Write-Host "-> Процес встановлення ПЗ завершено." -ForegroundColor Green
```

03-RemoveBloatware

```
<#
.SYNOPSIS
    Модуль 3: Оптимізація системи (Bloatware, Телеметрія, UI).
.DESCRIPTION
    Скрипт видаляє небажані UWP-додатки, вимикає служби телеметрії,
    прибирає ярлик Edge з робочого столу та жорстко вимикає віджети через політики.
#>
Write-Host "-> Розпочато оптимізацію системи та інтерфейсу..." -ForegroundColor Cyan
# 1. Список небажаних UWP-додатків
$BloatwareList = @(
    "*BingWeather*",
    "*Microsoft3DViewer*",
    "*MicrosoftSolitaireCollection*",
    "*WindowsFeedbackHub*",
    "*XboxApp*",
    "*XboxGamingOverlay*",
```

Продовження додатку С

```
    "*ZuneMusic*",
    "*ZuneVideo*",
    "*YourPhone*",
    "*GetHelp*",
    "*SkypeApp*"
)
Write-Host "   Етап 1: Видалення вбудованих додатків (UWP)..."
foreach ($App in $BloatwareList) {
    Get-AppxPackage -Name $App -AllUsers -ErrorAction SilentlyContinue | Remove-AppxPackage -ErrorAction SilentlyContinue
    Get-AppxProvisionedPackage -Online -ErrorAction SilentlyContinue | Where-Object { $_.DisplayName -like $App } | Remove-AppxProvisionedPackage -Online -ErrorAction SilentlyContinue
}
Write-Host "     [+] Bloatware успішно видалено." -ForegroundColor Green
# 2. Відключення служб телеметрії
$ServicesToDisable = @(
    "DiagTrack",
    "dmwappushservice",
    "RetailDemo"
```

```
)
Write-Host "   Етап 2: Відключення служб телеметрії Microsoft..."
foreach ($Service in $ServicesToDisable) {
    try {
        $SvcStatus = Get-Service -Name $Service -ErrorAction SilentlyContinue
        if ($SvcStatus) {
            if ($SvcStatus.Status -eq 'Running') { Stop-Service -Name
$Service -Force -ErrorAction SilentlyContinue }
            Set-Service -Name $Service -StartupType Disabled -ErrorAction
Stop
        }
    } catch { }
}

# 3. Вимкнення збору даних через реєстр
try {
```

Продовження додатку С

```
    Write-Host "   Етап 3: Блокування телеметрії на рівні системного
реєстру..."
    $RegistryPath =
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\DataCollection"
    if (-not (Test-Path $RegistryPath)) { New-Item -Path $RegistryPath -Force
| Out-Null }
    Set-ItemProperty -Path $RegistryPath -Name "AllowTelemetry" -Value 0 -
Type DWord -Force -ErrorAction Stop
} catch { }
# 4. Видалення ярлика Microsoft Edge з робочого столу
Write-Host "   Етап 4: Очищення робочого столу від системних ярликів..."
$PublicDesktop = [Environment]::GetFolderPath("CommonDesktopDirectory")
$UserDesktop = [Environment]::GetFolderPath("Desktop")
Remove-Item -Path "$PublicDesktop\Microsoft Edge.lnk" -Force -ErrorAction
SilentlyContinue
Remove-Item -Path "$UserDesktop\Microsoft Edge.lnk" -Force -ErrorAction
SilentlyContinue
$EdgeRegPath = "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer"
Set-ItemProperty -Path $EdgeRegPath -Name
"DisableEdgeDesktopShortcutCreation" -Value 1 -Type DWORD -Force -ErrorAction
SilentlyContinue
Write-Host "     [+] Ярлик Edge видалено." -ForegroundColor Green
# 5. Очищення панелі завдань через Корпоративні Політики (GPO)
Write-Host "   Етап 5: Очищення панелі завдань від системних віджетів..."
```

```
# Жорстке вимкнення Віджетів (Погоди)
$DshPolicy = "HKLM:\SOFTWARE\Policies\Microsoft\Dsh"
if (!(Test-Path $DshPolicy)) { New-Item -Path $DshPolicy -Force | Out-Null }
Set-ItemProperty -Path $DshPolicy -Name "AllowNewsAndInterests" -Value 0 -
Type DWord -Force -ErrorAction SilentlyContinue
# Жорстке вимкнення Чату (Teams)
$ChatPolicy = "HKLM:\SOFTWARE\Policies\Microsoft\Windows\Windows Chat"
if (!(Test-Path $ChatPolicy)) { New-Item -Path $ChatPolicy -Force | Out-Null
}
Set-ItemProperty -Path $ChatPolicy -Name "ChatIcon" -Value 3 -Type DWord -
Force -ErrorAction SilentlyContinue
# Вимкнення іконки "Перегляд завдань"
$AdvancedKey =
"HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced"
Set-ItemProperty -Path $AdvancedKey -Name "ShowTaskViewButton" -Value 0 -
Force -ErrorAction SilentlyContinue
```



```
# Вимкнення рядка "Пошук"
$SearchKey = "HKCU:\Software\Microsoft\Windows\CurrentVersion\Search"
if (!(Test-Path $SearchKey)) { New-Item -Path $SearchKey -Force | Out-Null }
Set-ItemProperty -Path $SearchKey -Name "SearchboxTaskbarMode" -Value 0 -
Force -ErrorAction SilentlyContinue
# Примусове перезавантаження інтерфейсу
Stop-Process -Name explorer -Force
Write-Host "     [+] Системні віджети приховано. Інтерфейс перезавантажено."
-ForegroundColor Green
Write-Host "-> Оптимізацію операційної системи завершено." -ForegroundColor
Green
```

04-SecurityConfig

```
<#
.SYNOPSIS
    Модуль 4: Налаштування політик кібербезпеки (Security Hardening).
.DESCRIPTION
    Скрипт вимикає вразливий мережевий протокол SMBv1, конфігурує
    Windows Defender та активує апаратне шифрування диска BitLocker (за
наявності TPM).
#>
Write-Host "-> Розпочато налаштування політик безпеки (Security
Hardening)..." -ForegroundColor Cyan
# 1. Відключення застарілого та небезпечного протоколу SMBv1
```

```
try {
    Write-Host "   Етап 1: Деактивація вразливого протоколу SMBv1..."
    # Вимикаємо серверну частину через реєстр
    Set-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" -Name SMB1
-Type DWORD -Value 0 -Force -ErrorAction SilentlyContinue
    # Вимикаємо клієнтську частину через нативний командлет
    Set-SmbServerConfiguration -EnableSMB1Protocol $false -Force -
Confirm:$false -ErrorAction SilentlyContinue
    Write-Host "     [+] Протокол SMBv1 успішно заблоковано." -
ForegroundColor Green
} catch {
    Write-Warning "     [-] Не вдалося змінити налаштування SMBv1. Подробиці:
$($_.Exception.Message)"
```

Продовження додатку С

```
}
# 2. Примусове налаштування Windows Defender
try {
    Write-Host "   Етап 2: Конфігурування антивірусного захисту Windows
Defender..."
    # Гарантуємо, що захист у реальному часі увімкнено
    Set-MpPreference -DisableRealtimeMonitoring $false -ErrorAction Stop
    # Вмикаємо автоматичне надсилання підозрілих зразків до Microsoft для
аналізу
    Set-MpPreference -SubmitSamplesConsent SendAllSamples -ErrorAction Stop
    Write-Host "     [+] Defender налаштовано на максимальний рівень
захисту." -ForegroundColor Green
} catch {
    Write-Warning "     [-] Не вдалося застосувати політики Defender."
}
# 3. Активація повнодискового шифрування BitLocker
try {
    Write-Host "   Етап 3: Ініціалізація шифрування BitLocker на системному
диску..."
    # Перевіряємо статус модуля TPM (Trusted Platform Module)
    $Tpm = Get-Tpm
    if ($Tpm.TpmPresent -and $Tpm.TpmReady) {
        Write-Host "     [+] Модуль TPM знайдено та активовано. Перевірка
статусу диска..."
        $Drive = $env:SystemDrive
        $BitLockerStatus = Get-BitLockerVolume -MountPoint $Drive
        if ($BitLockerStatus.VolumeStatus -eq 'FullyDecrypted') {
```

```
            # Папка для збереження ключа відновлення (для демонстрації на
захисті)
            $KeyPath = "C:\DeploymentLogs"
            if (!(Test-Path $KeyPath)) { New-Item -ItemType Directory -Force
-Path $KeyPath | Out-Null }
            Write-Host "     [*] Розпочато шифрування диска $Drive. Це може
зайняти кілька хвилин..."
            # Вмикаємо шифрування (тільки зайнятий простір для швидкості) та
зберігаємо ключ
            Enable-BitLocker -MountPoint $Drive -EncryptionMethod XtsAes256 -
UsedSpaceOnly -SkipHardwareTest -RecoveryKeyPath $KeyPath -
RecoveryKeyProtector -ErrorAction Stop
```

Продовження додатку С

```
            Write-Host "     [+] BitLocker успішно активовано!" -
ForegroundColor Green
            Write-Host "     [!] Ключ відновлення збережено у папку:
$KeyPath" -ForegroundColor Yellow
        } else {
            Write-Host "     [!] Диск вже зашифровано або процес шифрування
вже триває." -ForegroundColor Yellow
        }
    } else {
        Write-Warning "     [-] Криптографічний модуль TPM не знайдено або
він не готовий."
        Write-Warning "         Переконайтеся, що в налаштуваннях VirtualBox
увімкнено vTPM 2.0."
    }
} catch {
    Write-Warning "     [-] Помилка під час налаштування BitLocker:
$($_.Exception.Message)"
}
Write-Host "-> Налаштування безпеки завершено." -ForegroundColor Green
```

apps.json

```
{
  "Apps": [
    {
      "Name": "Google Chrome",
      "ID": "Google.Chrome"
    },
    {
```

```
      "Name": "7-Zip",
      "ID": "7zip.7zip"
    },
    {
      "Name": "Notepad++",
      "ID": "Notepad++.Notepad++"
    }
  ]
}
```