МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра комп’ютеризованих систем захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Ільєнко

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 р.

**МАГІСТЕРСЬКА АТЕСТАЦІЙНА РОБОТА**

**ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ**

**«МАГІСТР»**

Тема: «Засоби захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом»

Виконав: студент \_\_\_ групи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горовий І.М.

Керівник: д.т.н., професор\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оксіюк О.Г.

Нормоконтроль з ЄСКД (ЄСПД):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Єгоров С.В.

**Київ 2020**

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут комп’ютерних інформаційних технологій

Кафедра комп’ютеризованих систем захисту інформації

Освітній ступінь “Магістр”

Спеціальність 125 «Кібербезпека»

Освітньо-професійної програми 125-01 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

А.В. Ільєнко

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

“\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_2020 р.

**ЗАВДАННЯ**

**на виконання магістерської атестаційної роботи магістранта**

ГОРОВОГО ІГОРЯ МИКОЛАЙОВИЧА

1. **Тема роботи:** Засоби захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом

затверджена наказом ректора від 02.10.2019 р. № 2265/ст.

1. **Термін виконання роботи:** з 16.10.2019 р. по 10.02.2020 р.
2. **Вихідні дані роботи:**

* Операційні системи з відкритим кодом.
* Засоби захисту операційних систем з відкритим кодом.
* Засоби захисту мережі.
* Способи захисту від кіберзагроз.

1. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):**

* Дослідження клієнт-серверної архітектури
* Операційні системи з відкритим кодом
* Концепція безпеки Unix
* Особливості функціонування операційних систем з відкритим кодом.
* Концепція безпеки та проблеми безпеки Unix.
* Розробка рекомендацій до захисту клієнт-серверних мереж на базі операційних систем з відкритим кодом.

1. **Перелік обов'язкового графічного (ілюстративного) матеріалу:**

* Презентація Power Point.

1. **Календарний план-графік**

|  |  |  |  |
| --- | --- | --- | --- |
| № пор | Завдання | Термін виконання | Відмітка про виконання |
| 1 | Уточнення постановки задачі | 16.10.2019 | *Виконано* |
| 2 | Аналіз літературних джерел | 16.10.2019 - 22.10.2019 | *Виконано* |
| 3 | Обґрунтування вибору рішення | 22.10.2019 -24.10.2019 | *Виконано* |
| 4 | Збір інформації | 25.10.2019 - 15.11.2019 | *Виконано* |
| 5 | Аналіз даних та їх класифікація | 16.11.2019 - 27.11.2019 | *Виконано* |
| 6 | Дослідження клієнт-серверної архітектури | 28.11.2019 - 08.12.2019 | *Виконано* |
| 7 | Операційні системи з відкритим кодом | 09.12.2019 – 18.12.2019 | *Виконано* |
| 8 | Концепція безпеки Unix | 19.12.2019 – 27.12.2019 | *Виконано* |
| 9 | Дослідження вразливостей та загроз | 28.12.2019 – 22.01.2020 | *Виконано* |
| 10 | Розробка рекомендацій | 23.01.2020 – 25.01.2020 | *Виконано* |
| 11 | Оформлення і друк пояснювальної записки | 26.01.2020 – 27.01.2020 | *Виконано* |
| 12 | Оформлення презентації | 28.01.2020 | *Виконано* |
| 13 | Отримання рецензій від опонентів | 29.01.2020 | *Виконано* |
| 14 | Підготовка до захисту в ДЕК | 30.01.2020 | *Виконано* |

1. **Дата видачі завдання:** «02» жовтня 2019 р.

**Керівник дипломної роботи:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оксіюк О.Г.

(підпис)

**Завдання прийняв до виконання*:***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горовий І.М.

(підпис)

**РЕФЕРАТ**

Пояснювальна записка до дипломної роботи «Засоби захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом» складається зі списку скорочень, вступу, основної частини, що містить 3 розділи, висновків і списку літератури та джерел. Загальний обсяг роботи – 89 сторінок. Робота містить 19 рисунків, 3 таблиці та 6 додатків. Список використаних джерел включає 30 джерел.

КЛІЄНТ-СЕРВЕРНА СИСТЕМА, ОПЕРАЦІЙНІ СИСТЕМИ З ВІДКРИТИМ КОДОМ, МЕРЕЖЕВИЙ ЗАХИСТ, РЕКОМЕНДАЦІЇ ДО ЗАХИСТУ, ЗАСОБИ ЗАХИСТУ ОПЕРАЦІЙНИХ СИСТЕМ, ПРОБЛЕМИ БЕЗПЕКИ UNIX-ПОДІБНИХ СИСТЕМ.

Об’єкт дослідження – процес взаємодії елементів клієнт-серверної мережі з використанням механізмів захисту.

*Мета роботи* - застосування механізмів та засобів захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом.

*Предмет дослідження* – способи та засоби захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом.

*Метод дослідження* – аналіз засобів захисту даних, загроз та вразливостей в мережі.

В роботі проведено аналіз способів захисту оперційних систем та інформаційних ресурсів клієнт-серверної системи.

*Наукова новизна* дипломної роботи полягає в формуванні механізмів захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом та розробки рекомендації щодо захисту даної мережі.

*Практичне значення роботи* - сформовано та надано рекомендації до захисту клієнт-серверних мереж на базі операційних систем з відкритим кодом.

Результати здійснених у дипломній роботі досліджень можуть бути використані спеціалістами із захисту інформації та при проектуванні чи створенні клієнт-серверних мереж на базі операційних систем з відкритим кодом.

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ**

ОС – операцйійна система;

DDoS – Distributed Denial of Service;

DNS – Domain Name System;

DoS – Denial of Service;

FTP – File Transfer Protocol;

GRUB – GRand Unified Bootloader;

GUI – Graphical User Interface.

HTTP – Hypertext Transfer Protocol;

HTTPS – Hypertext Transfer Protocol Secure;

IP – Internet Protocol;

LAMP – Linux Apache MySQL PHP/Perl/Python;

LDAP – Lightweight Directory Access Protocol;

SCP – Secure Copy;

SSH – Secure SHell;

TCP – Transmission Control Protocol;

TELNET – Teletype Network;

USB – Universal Serial Bus.

ЗМІСТ

[ВСТУП 9](#_Toc10714549)

[РОЗДІЛ 1](#_Toc10714550) [ОСОБЛИВОСТІ РОБОТИ КЛІЄНТ-СЕРВЕРНИХ МЕРЕЖ ТА ОПЕРАЦІЙНИХ СИСТЕМ З ВІДКРИТИМ КОДОМ 11](#_Toc10714551)

[1.1 Загальний опис клієнт-серверної архітектури 11](#_Toc10714552)

[1.1.1 Клієнти та їх типи 11](#_Toc10714553)

[1.1.2 Сервери та їх типи 12](#_Toc10714554)

[1.1.3 Типи клієнт-серверних архітектур 14](#_Toc10714555)

[1.2 Особливості функціонування ОС з відкритим кодом на прикладі ОС UNIX 18](#_Toc10714556)

[1.2.1 Компоненти Unix 21](#_Toc10714557)

[1.2.2 Функції ядра та оболонки 22](#_Toc10714558)

[1.2.3 Файлова система та пристрої 25](#_Toc10714559)

[1.2.4 Процеси, диспетчеризація і таймер 26](#_Toc10714560)

[1.3 Особливості використання Linux в клієнт-серверних мережах 29](#_Toc10714561)

[1.4 Нормативно-правова база у сфері захисту інформації 32](#_Toc10714562)

[1.5 Постановка задачі 36](#_Toc10714563)

[1.6 Висновки за розділом 1 37](#_Toc10714564)

[РОЗДІЛ 2](#_Toc10714565) [ПРОБЛЕМИ БЕЗПЕКИ UNIX ТА СУПУТНЬОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 39](#_Toc10714566)

[2.1 Концепція безпеки Unix-подібних систем 39](#_Toc10714567)

[2.2 Переповнення буфера 45](#_Toc10714568)

[2.2.1 Загальний опис проблеми 45](#_Toc10714569)

[2.2.2 Вплив на властивості інформації 47](#_Toc10714570)

[2.3 Стан гонитви 48](#_Toc10714571)

[2.3.1 Загальний опис проблеми 48](#_Toc10714572)

[2.3.2 Вплив на властивості інформації 49](#_Toc10714573)

[2.4 Атака на відмову в обслуговуванні (DoS) та розподілена атака на відмову в обслуговуванні 51](#_Toc10714574)

[2.4.1 Загальний опис проблеми 51](#_Toc10714575)

[2.4.2 Вплив на властивості інформації 53](#_Toc10714576)

[2.5 Висновки за розділом 2 54](#_Toc10714577)

[РОЗДІЛ 3](#_Toc10714578) [ЗАСОБИ ЗАХИСТУ ТА РЕКОМЕНДАЦІЇ ДО ЗАХИСТУ КЛІЄНТ-СЕРВЕРНОЇ МЕРЕЖІ НА БАЗІ ОПЕРАЦІЙНИХ СИСТЕМ З ВІДКРИТИМ КОДОМ 55](#_Toc10714579)

[3.1 Принципи безпеки клієнт-сервених мереж 55](#_Toc10714580)

[3.2 Фізична безпека 57](#_Toc10714581)

[3.2.1 Забезпечення фізичної безпеки серверів, клієнтів та мережного обладнання 57](#_Toc10714582)

[3.2.2 Безпека завантаження 59](#_Toc10714583)

[3.3 Забезпечення захисту операційної системи 60](#_Toc10714584)

[3.3.1 Програмне забезпечення, драйвери та компоненти 60](#_Toc10714585)

[3.3.2 Керування графічним інтерфейсом 61](#_Toc10714586)

[3.4 Розробка механізмів захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом 62](#_Toc10714587)

[3.4.1 Розробка комплексу скриптів для перевірки та маніпуляцій критичних об’єктів сервера чи робочої станції 62](#_Toc10714588)

[3.4.2 Розробка скрипта для перевірки та створення логів про критичні об’єкти сервера чи робочої станції 66](#_Toc10714589)

[3.4.3 Налаштування фаєрволу для забезпечення безпеки мережі 67](#_Toc10714590)

[3.5 Висновки за розділом 3 68](#_Toc10714591)

[ВИСНОВКИ 70](#_Toc10714592)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72](#_Toc10714593)

[ДОДАТКИ 75](#_Toc10714594)

[ДОДАТОК А 75](#_Toc10714595)

[ДОДАТОК Б 76](#_Toc10714596)

[ДОДАТОК В 78](#_Toc10714597)

[ДОДАТОК Д 85](#_Toc10714598)

[ДОДАТОК Е 88](#_Toc10714599)

[ДОДАТОК Ж 90](#_Toc10714600)

# ВСТУП

На сьогоднішній день, завдяки стрімкому розвитку інформаційних технологій, змінюються способи ведення бізнесу, який стає все більше залежним від інформації, інформаційних систем та комунікаційних систем. Багато компаній використовують мережу Інтернет для проведення різних операцій, зокрема банківських, та для забезпечення своїх працівників можливістю обмінюватися різними даними(зокрема конфіденційними) в режимі реального часу, навіть якщо вони знаходяться далеко один від одного, що допомагає їх працювати ефективніше та економити час. Завдяки розвитку клієнт-серверної архітектури, все більше компаній надають різні послуги онлайн в мережі Інтернет або ж впроваджують різні сервери в локальних мережах для власних потреб. Так чи інакше кількість серверів майже завжди менша за кількість клієнтів, через що вони є важливішими в плані захисту, оскільки виведення з ладу серверу приведе до неможливості отримання певної послуги, яку надавав цей сервер, усіма клієнтами, що підключились чи планували підключитися до цього сервера під час його несправності.

За даними сайту W3Tech серед 10 мільйонів найпопулярніших сайтів 69.9% сайтів на яких вдається визначити операційну систему працюють на Unix[1], а 52,4% з тих, що працюють на Unix – працюють під управлінням операційних систем на основі ядра Linux[2]. Окрім веб-серверів Linux використовують на поштових серверах, LDAP-серверах, в хмарних обчисленнях та на мережевому обладнанні.

Захист мережі є дуже важливим для підтримки безперебійності надання послуг сервером, оскільки для їх надання потрібна можливість комунікації між сервером та клієнтом. Таким чином недоступність серверів може призвести до збитків і недоотримання доходів власником серверу. У випадку серверів що працюють у внутрішній мережі то несправність сервера може призвести до припинення внутрішніх для організації процесів, що в свою чергу може призвести до того, що працівники не зможуть виконувати свою роботу наслідком чого будуть збитки для організації.

Unix-подібні системи, до яких відносяь і Linux, мають свої переваги і недоліки над операційними системами сімейства Windows. В залежності від власних потреб і вимог до операційних систем компанії обирають операційні системи які мають найбільше переваг для задоволення їх потреб.

За таких умов забезпечення безпеки серверів, клієнтів та мережі в якій вони працюють може бути критично важливим для підприємств. Основними проблемами є те, що зловмисника важко відрізнити від звичайного користувача для публічних серверів та захист ресурсів і процесів для серверів в цілому.

Отже, виходячи з вище сказаного, та враховуючи тенденції сучасної організації захисту в клієнт-серверних мережах, актуальність цього захисту полягає у тому, що у будь-якій мережі з серверами, на яких працюють операційні системи сімейства Unix та мають доступ до мережі Інтернет потрібно організувати захист від зловмисників для забезпечення безперервності процесів всередині цієї мережі.

Метою роботи є застосування механізмів та засобів захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом.

Об’єкт дослідження – процес взаємодії елементів клієнт-серверної мережі з використанням механізмів захисту.

Предмет дослідження – способи та засоби захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом.

Практичне значення роботи - сформовано та надано рекомендації до захисту клієнт-серверних мереж на базі операційних систем з відкритим кодом.

# РОЗДІЛ 1

# ОСОБЛИВОСТІ РОБОТИ КЛІЄНТ-СЕРВЕРНИХ МЕРЕЖ ТА ОПЕРАЦІЙНИХ СИСТЕМ З ВІДКРИТИМ КОДОМ

## Загальний опис клієнт-серверної архітектури

Клієнт-серверна архітектура - це архітектура комп'ютерної мережі, в якій багато клієнтів (віддалених процесорів) запитують і отримують послуги від централізованого сервера (хост-комп'ютера). Клієнтські комп'ютери мають інтерфейс, який дозволяє користувачеві комп'ютера запитувати послуги сервера і відображати результати, які сервер повертає. Сервери чекають на запит, що надходить від клієнтів, а потім реагують на них [3]. В ідеалі, сервер надає клієнтам стандартизований і зрозумілий інтерфейс так, що клієнтам не потрібно знати специфіку системи (тобто апаратних засобів і програмного забезпечення), що надає послугу. Клієнти часто знаходяться на робочих станціях або на персональних комп'ютерах, а сервери знаходяться в інших місцях мережі, як правило, на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти та сервер мають різні завдання, які вони виконують. Багато клієнтів можуть одночасно отримувати доступ до інформації сервера, і в той же час клієнтський комп'ютер може виконувати інші завдання, наприклад, надсилання електронної пошти.

### Клієнти та їх типи

Клієнт - це частина комп'ютерного обладнання або програмного забезпечення, що надсилає запити до служби, яку надає сервер.

Товстий клієнт - це клієнт, який сам виконує основну частину будь-яких операцій обробки даних і не обов'язково покладається на сервер. Персональний комп'ютер є поширеним прикладом товстого клієнта через його відносно великий набір функцій і можливостей і низьку залежність від сервера.

Тонкий клієнт є клієнтом, який виконує мінімальну кількість операцій. Тонкі клієнти використовують ресурси сервера. Тонкий клієнт, як правило, показує оброблені дані, надані сервером прикладних програм, який виконує основну частину будь-якої необхідної обробки даних.

Гібридний клієнт - це суміш двох вищезазначених моделей клієнтів. Подібно до товстого клієнта, він обробляє дані локально, але покладається на сервер для зберігання постійних даних. Цей підхід передбачає функції та переваги як від товстого клієнта (мультимедійна підтримка, висока продуктивність), так і тонкого клієнта (висока керованість, гнучкість).

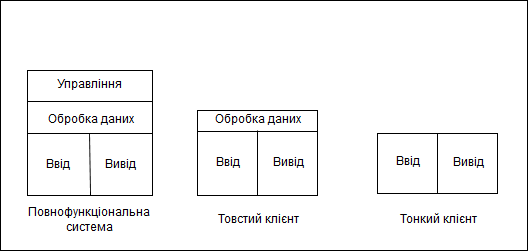


Рисунок 1.1 − Типи клієнтів

### Сервери та їх типи

Cервер являє собою комп'ютерну програму або пристрій, що забезпечує функціональність для інших програм або пристроїв, які називають клієнтами. Сервери можуть надавати різні функціональні можливості, які називають послугами або сервісами, такі як обмін даними або ресурсами між декількома клієнтами, або виконання обчислень для клієнта. Один сервер може обслуговувати декілька клієнтів, і один клієнт може використовувати кілька серверів. Процес клієнта може працювати на одному пристрої або може з'єднуватися через мережу з сервером на іншому пристрої. Типовими прикладами серверів є сервери баз даних, файлові сервери, поштові сервери, сервери друку, веб-сервери, ігрові сервери та сервери додатків[4].

Сервери поділяють на два типи за способом обробки запитів клієнтів:

* Ітеративні сервери
* Паралельні сервери

Ітеративний сервер - це найпростіша форма сервера, де серверний процес обслуговує одного клієнта, і після завершення запиту від першого клієнта він приймає запит від наступного клієнта. Тим часом інші клієнти продовжують чекати своєї черги на виконання запиту.

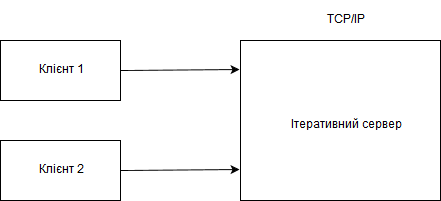


Рисунок 1.2 − Принцип роботи ітеративного сервера

Перевагою ітеративного серверу є простота побудови, а недоліками є можливість одночасної обробки лише одного запиту та велика затримка, що зростає зі збільшенням кількості клієнтів, що очікують на виконання свого запиту.

Паралельний сервер - це тип сервера, який виконує кілька одночасних процесів для одночасного обслуговування багатьох запитів. Оскільки один процес може тривати досить довго, а інший клієнт не може чекати на завершення попереднього запиту для отримання можливості обробки власного запиту – застосовують архітектуру з паралельним сервером.

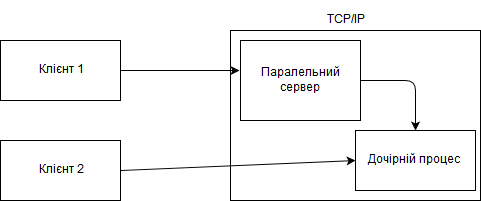


Рисунок 1.3 − Принцип роботи паралельного сервера

Перевагами паралельного сервера є можливість виконання декількох запитів одночасно, менший час очікування та ефективніше використання ресурсів. Недоліком такого типу серверів є більша склідність впровадження та експлуатації.

### Типи клієнт-серверних архітектур

Двохрівнева архітектура:

У цьому типі архітектури робоче навантаження поділяється між сервером (хостом системи) і клієнтом (на якому розміщений інтерфейс користувача). Здебільшого, фізично вони розташовані на окремих комп'ютерах, але не існує абсолютної вимоги до цього, тому логічно розділені рівні можуть бути розміщені на одному комп'ютері (наприклад, розробка та тестування).

Візуальна схема будови двохрівневої архітектури та взаємодії її елементів наведена на рисунку 1.4.

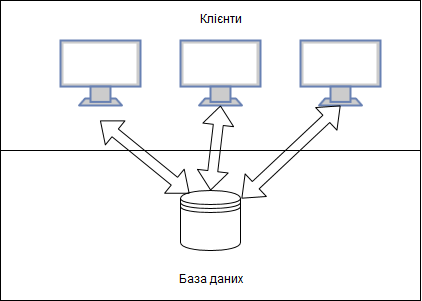


Рисунок 1.4 − Архітектура двох рівнів

Перевагами архітектури такого типу є:

* Зручність у розробці застосунків: програми легко розробляються завдяки простоті.
* Задоволеність користувачів: максимальна кількість користувачів з задоволеними запитами досягається завдяки точному та швидкому створенню прототипів додатків завдяки надійним інструментам.
* Застосовується для однорідного середовища: оскільки архітектура містить статичні правила обробки інформації, вона більш застосовна для однорідного середовища.
* Висока продуктивність: сервер баз даних і логіка обробки даних фізично близькі, що забезпечує більш високу продуктивність.

Дворівнева архітектура є гарним рішенням, коли робота з групою користувачів зазвичай невелика (до 100 одночасних користувачів), але для великої кількості користувачів вона має ряд обмежень:

* Продуктивність: продуктивність починає погіршуватися з ростом кількості користувачів - це пов'язано з тим, що кожен користувач має своє власне підключення, і сервер повинен підтримувати всі ці з'єднання, навіть якщо не виконується жодна робота.
* Безпека: Кожен користувач повинен мати свій індивідуальний доступ до бази даних, і йому повинні бути надані будь-які права, необхідні для запуску програми.
* Функціональність: Незалежно від типу клієнта, велика частина обробки даних повинна проводитись в базі даних, що робить клієнти повністю залежними від можливостей і реалізації бази даних. Це обмежує функціональність програм.
* Портативність: Оскільки дворівнева архітектура залежить від конкретної реалізації бази даних, перенесення існуючої програми на інші системи керування базами даних стає серйозною проблемою.

Ця архітектура є поширеною в мережі Інтернет, хоча вона може добре працювати у відключеному від мережі Інтернет локальному середовищі. Однак, багато в чому ця реалізація наслідує застарілу архітектуру мейнфреймів і дворівневі програми можуть страждати від багатьох втілень однієї й тієї ж проблеми.

Трирівнева архітектура:

Щоб подолати обмеження дворівневої архітектури, було введено додатковий рівень і створено стандартну трирівневу клієнт-серверну модель. Призначення додаткового рівня - керування виконанням програм і управління базами даних, як і у випадку дворівневої моделі, рівні можуть бути реалізовані на різних фізичних машинах, або кілька спільно розміщені на одній машині.

У трирівневій архітектурі логіка функціонального процесу, доступ до даних, зберігання комп'ютерних даних і інтерфейс користувача розробляються і підтримуються як самостійні модулі на окремій платформі.

Візуальна схема будови трирівневої архітектури та взаємодії її елементів наведена на рисунку 1.5.

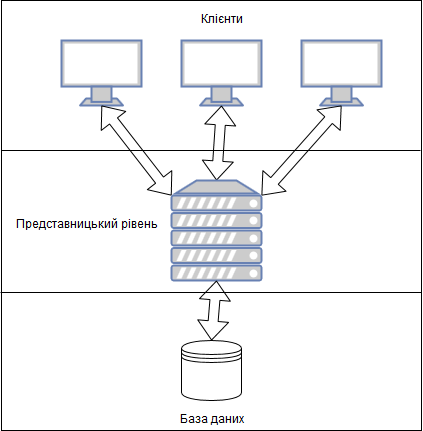


Рисунок 1.5 − Схема функціонування трирівневої моделі

Три рівні в трирівневій архітектурі:

* Прикладний рівень: займає найвищий рівень і відображає інформацію, пов'язану з доступною послугою. Цей рівень зв'язується з іншими рівнями, посилаючи результати у веб-переглядач та інші рівні мережі.
* Представницький рівень: також називається середній рівень або логічний рівень, цей рівень знаходиться між представницьким рівнем та рівнем даних. Він контролює функціональність програми, виконуючи детальну обробку даних.
* Рівень даних: представлений серверами баз даних, де зберігається та видається на запити інформація. Дані в цьому рівні зберігаються незалежно від серверів додатків або логіки обробки даних.

Перевагами трирівневої моделі є:

* Покращена цілісність даних: Пошкодження даних через клієнтський додаток може бути усунуто, оскільки дані проходять через середній рівень для оновлення бази даних, що гарантує її цілісність.
* Розширена безпека: Розміщення логіки обробки даних на централізованому сервері робить дані більш захищеними. Безпека даних покращується, оскільки клієнт не взаємодіє з базою даних безпосередньо.
* Прихована структура баз даних: Структура бази даних часто залишається прихованою від клієнтів, що дозволяє приховати будь-які зміни в базі даних.

Негативною особливістю трирівневої структури є складність побудови комунікації - зазвичай потрібно докладати більше зусиль при створенні трирівневих додатків, оскільки кількість точок зв'язку збільшується ( зв'язок між представницьким рівнем, прикладним рівнем та рівнем даних).

## Особливості функціонування ОС з відкритим кодом на прикладі ОС UNIX

UNIX був розроблений як операційна система спільного використання часу. Cтандартний інтерфейс (оболонка) простий і може бути легко налаштований або замінений іншим інтерфейсом. Файлова система реалізована як багаторівневе дерево і таким чином підтримує концепцію каталогів і підкаталогів. Файли користувачів розглядаються як потік байтів. Дискові файли та пристрої введення/виведення опрацьовуються аналогічно, що робить використання системного пристрою незалежним. Unix є багатозадачною і багатокористувацькою операційною системою. Процеси можуть легко створювати та припиняти інші процеси. У Unix зазвичай икористовується простий алгоритм планування процесора, наприклад, пріоритетний або кільцевий зворотний зв'язок. В основному, в якості схеми управління пам'яттю використовується пошуковий виклик. UNIX широко використовується для таких завдань, як обробка графіки, операції в реальному часі надання серверних чи мережних послуг [5].

Таким чином, основні можливості UNIX можуть бути перелічені наступним чином:

* багатокористувацька ОС
* багатозадачна ОС
* переносимість
* модульність
* безпека системи
* структурованість файлів і забезпечення їх базової безпеки
* незалежність від апаратного забезпеення
* комунікації

Багатокористувальницька ОС. UNIX дозволяє декільком користувачам (до 500) одночасно використовувати один і той же комп'ютер, приєднуючи термінали (клавіатуру і дисплей) так, що користувачі цих терміналів можуть запускати програми, отримувати доступ до файлів чи друкувати документи усі одночасно. Це забезпечує обмін ресурсів між користувачами. Алгоритми планування використовуються для забезпечення швидкої і неперервної роботи комп'ютера.

Багатозадачна ОС. У UNIX можна виконувати кілька завдань одночасно, тобто під час друку документа можна почати редагування нового документа, або запустити сканування списку розсилки електронної пошти для сортування, або обробити інший документ у редакторі. Багатозадачність підтримується шляхом розміщення декількох завдань у фоновому режимі під час роботи над завданням на "передньому плані".

Переносимість. Сама система UNIX надзвичайно портативна. За допомогою декількох модифікацій система UNIX може бути встановлена на різних пристроях.

Модульність. UNIX поставляється з декількома сотнями стандартних програм які є утилітами та інструментами, які забезпечують багате і продуктивне середовище для створення великих модулів з менших.

Безпека системи. Захист комп'ютерної системи означає захист комп'ютерного обладнання та інформації, що містяться в системі. Загрози системі можуть бути як внутрішніми, так і зовнішніми. Зовнішні загрози включають несанкціонований доступ до комп'ютерної системи, несанкціоноване прослуховування даних, що передаються по мережі, пожежі, несправності електромережі, стихійні лиха тощо. Внутрішні загрози мають більш серйозний характер, наприклад, помилкове або ціленаправлене руйнування даних програмного забезпечення, перевірка або доступ до конфіденційних даних користувачем, що не має права користування, зміна цих даних.

UNIX надає ряд гарантій з точки зору безпеки:

* + захист паролем для системного доступу
  + контроль доступу до окремих файлів
  + шифрування файлів даних
  + шифрування паролів
  + функції обліку системи

Структура файлів та їх безпека. Для легкого обслуговування UNIX використовує ієрархічну систему. Він має послідовний формат для файлів, що полегшує запис для прикладних програм.

UNIX дозволяє більш ніж одному користувачеві оновлювати той самий файл практично в один і той же час. Але якщо два користувачі оновлюють один і той же файл точно в той же час - можуть виникнути проблеми, які впливають на цілісність даних. Наприклад, операція запису одного користувача може скасувати те, що виконується іншим користувачем. Щоб звести до мінімуму ці проблеми, використовуються блокування записування і явні дозволи.

Незалежність від апаратного забезпечення. У UNIX, один з принципів дизайну полягає в тому, щоб обробляти файли дисків і пристрої введення-виведення як файли з точки зору користувача, таким чином, захищаючи користувача від особливостей пристрою. Це забезпечує незалежність від апаратного забезпечення для користувача. Апаратні пристрої отримують імена у файлових системах. Ці спеціальні файли пристроїв відомі як інтерфейси пристроїв, але доступ до них здійснюється користувачем так само, як і до інших файлів.

Комунікації. Комунікації є однією з сильних сторін Unix. Для Unix та Unix-подібних систем створено велику кількість програмного забезпечення для реалізації мережевих послуг. В Unix процеси користувача обмінюються інформацією з мережевими протоколами через об'єкт сокета.

### Компоненти Unix

Будь-яка операційна система Unix складається тільки з ядра. Однак командна оболонка і файлова система також вважаються невід'ємною частиною будь-якої операційної системи Unix. Таким чином, система Unix складається з наступних трьох основних компонентів:

* Ядро
* Оболонка
* Файлова система

Таким чином усі базові компоненти Unix взаємодіють між собою, а ядро взаємодіє з апаратним забезпеченням, прибираючи необхідність інших компонентів взаємодіяти з апаратним забезпеченням безпосередньо. Схема Unix, яка показує взаємодію між процесами, оболонками, ядром та апаратним забезпеченням наведена на наступному рисунку.

Візуальна схема будови Unix-подібних операційних систем наведена на рисунку 1.6.

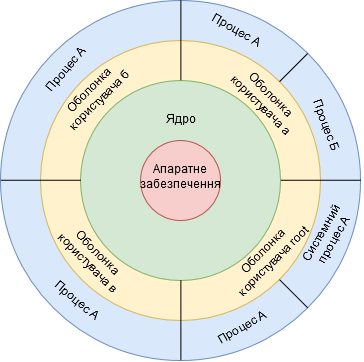


Рисунок 1.6 − Схема будови Unix-подібних операційних систем

### Функції ядра та оболонки

Ядро - це серце будь-якої операційної системи Unix. Воно є відносно невеликим фрагментом коду, який вбудований в апаратне забезпечення. Насправді, це набір програм, в основному написаних на C. Кожна система Unix має ядро (тільки одне), яке автоматично завантажується в пам'ять, як тільки система завантажується. Оскільки ядро знаходиться на апаратному забезпеченні, воно може безпосередньо зв'язуватися з апаратним забезпеченням.

Візуальна схема взаємозв'язку між ядром Unix та апаратним забезпеченням наведена на рисунку 1.7.

Ядро є єдиним компонентом, який може безпосередньо зв'язуватися з апаратним забезпеченням. Ядро керує всіма системними ресурсами, такими як пам'ять та пристрої вводу-виводу.

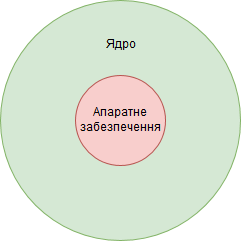


Рисунок 1.7 − Зв’язок ядра та апаратного забезпечення

Ядро розподіляє час між користувачами та процесами у випадку багатокористувацького середовища, вирішує пріоритети процесу, керує міжпроцесним зв'язком (IPC) і виконує багато інших завдань. Раніше всі програми, що входили до складу ядра, об'єднувалися і переміщувалися до пам'яті під час завантаження. Такі інтегровані ядра називаються монолітними ядрами. Проте в ці дні вони згруповані в різні модулі і тільки необхідний модуль переміщується в пам'ять під час завантаження. Цей просто необхідний і достатній модуль, що складається з невеликого набору програм ядра, називається мікроядро. Інші модулі переміщуються і виводяться з пам'яті залежно від вимог.

Кожна система Unix має, принаймні, одну оболонку. Оболонка - це програма, яка знаходиться на ядрі і діє як агент або інтерфейс між користувачами і ядром і, отже, апаратним забезпеченням.

Оболонка - це інтерпретатор або процесор команд. Як тільки система буде завантажена успішно, оболонка надає командний рядок (зазвичай символ $ або%), в якому користувач може вводити будь-яку команду Unix. Після прийняття команди, оболонка генерує легко виконуваний простий командний рядок, розбираючи її, оцінюючи змінні (якщо такі змінні присутні), виконує заміну команд, інтерпретує метасимволи типу \* і ? (якщо є) і ідентифікує шлях. Цей простий командний рядок потім передається на ядро для виконання [6].

Окрім командного інтерпретатора і інтерфейсу між користувачами і ядром, оболонка має певні можливості програмування. Використовуючи цю можливість, можна написати програми, які називаються програмами оболонки. Зазвичай програми оболонки називаються скриптами оболонки.

Оболонка Борна (The Bourne shell - sh) Це найпоширеніша оболонка, доступна в системах Unix, і перша велика оболонка, яка буде розроблена. Ця оболонка широко використовується в багатьох Unix-подібних системах. Оболонка названа на честь свого автора, Стівена Борна, що працював в лабораторії AT&T Bell. Ця оболонка розповсюджується як стандартна оболонка майже на всіх системах Unix.

Оболонка C (The C shell - csh) Біл Джой розробив цю оболонку як частину операційної системи BSD. Вона називається оболонкою C, оскільки її синтаксис і використання дуже схожі на мову програмування C. На жаль, ця оболонка не доступна на всіх машинах. Сценарії оболонки, написані в оболонці C, не сумісні з оболонкою Борна. Однією з головних переваг оболонки C над оболонкою Борна є її здатність виконувати процеси у фоновому режимі. Версія цієї оболонки з назвою tcsh доступна безкоштовно під Linux.

Оболонка Корна (The Korn shell - ksh) Ця оболонка була розроблена Девідом Корном у лабораторії AT&T Bell. В основному вона побудована на оболонці Борна. Вона також включає деякі особливості оболонки С. В даний час це одна з широко використовуваних оболонок. Вона може запускати сценарії оболонки Борна без будь-яких змін. Одна з її версій, оболонка Корна публічного домену (public-domain Korn shell - pdksh), постачається з Linux безкоштовно.

Знову оболонка Борна (The Bourne-Again shell - bash) Ця оболонка була розроблена Б Фоксом і С Ремі у Free Software Foundation. Деякі варіанти операційних систем Linux поставляються з цією оболонкою як стандартною оболонкою. Ця оболонка є вільним програмним забезпеченням.

Оболонка як командний процесор. Як вже було сказано, оболонка виступає як командний процесор, так і невелика мова програмування. Далі наведений короткий опис поведінки оболонки як командного процесора. Під час інтерпретації командного рядка, вказаного в його терміналі, оболонка виконує одну або кілька або всі наступні кроки, залежно від вмісту командного рядка, наданого йому.

1. Вона розбирає командний рядок і ідентифікує кожне слово в ньому і видаляє наявні додаткові пробіли або табуляцію, якщо такі є.
2. Оцінює всі наявні змінні, які можуть бути з префіксом $.
3. Якщо команди знаходяться в подвійних лапках, вони виконуються і їх результат вставляється в командний рядок. Іншими словами, відбувається заміщення команди.
4. Потім оболонка перевіряє будь-яке перенаправлення вводу та/або виводу і встановлює відповідність між файлами.
5. Після цього перевіряється наявність символів підстановки, наприклад \*, ? і [,]. Якщо будь-який з цих символів присутній, генерується ім'я файлу та підстановка.
6. Потім оболонка шукає необхідні команди, а також файли, витягує їх і передає їх до ядра для виконання. Маршрут або шлях для пошуку необхідних команд буде містити змінну оболонки PATH. Оболонка також бере до уваги крапку з комою і логічні оператори, що дозволяє виконувати декілька команд.

### Файлова система та пристрої

Файлова система - це компонент Unix, який дозволяє користувачеві переглядати, організовувати, захищати та взаємодіяти з файлами та каталогами, які розташовані на пристроях зберігання даних. У Unix існують різні типи файлових систем: дискові, мережеві, спеціальні або віртуальні [7].

Отже, файли у Unix характеризуються:

* Ієрархічною структурою,
* Узгодженою обробкою масивів даних,
* Можливістю створення і видалення файлів,
* Динамічним розширенням файлів,
* Захистом інформації в файлах,
* Трактуванням периферійних пристроїв як файлів.

Файлова система являє собою дерево, з єдиною основою - коренем (root) та позначається як "/".

Кожна вершина в структурі дерева файлової системи, крім листків, є каталогом файлів, а файли, що відповідають дочірнім вершинам, є або каталогами, або звичайними файлами, або файлами пристроїв.

Каталоги схожі на звичайні файли в тому сенсі, що система представляє інформацію в каталозі набором байтів, але ця інформація включає в себе імена файлів в каталозі в оголошеному форматі.

Для користувача система UNIX трактує пристрої так, ніби вони є файлами. Пристрої, для яких призначені спеціальні файли пристроїв, стають вершинами в структурі файлової системи. Зверення програмного забезпечення до пристрою має той самий синтаксис, що і звернення до звичайних файлів; семантика операцій читання й запису по відношенню до пристроїв в більшій мірі збігається з семантичними операціями читання і запису звичайних файлів. Спосіб захисту пристроїв збігається з способом захисту звичайних файлів [8].

### Процеси, диспетчеризація і таймер

Програмою називається виконуваний файл, а процесом називається послідовність операцій програми або частина програми при її виконанні. В системі UNIX може одночасно виконуватися безліч процесів (цю особливість іноді називають мультипрограмування або багатозадачним режимом), при чому їх число логічно не обмежується, і безліч частин програми можуть одночасно перебувати в системі. Різні системні операції дозволяють процесам породжувати нові процеси, завершують процеси, синхронізують виконання етапів процесу і керують реакцією на різні події. Завдяки різним зверненнями до операційної системи, процеси виконуються незалежно один від одного [9].

Можливий набір станів процесу міститься в наступному переліку:

1. Процес виконується в режимі завдання.
2. Процес виконується в режимі ядра.
3. Процес не виконується, але готовий до запуску під управлінням ядра.
4. Процес призупинено і знаходиться в оперативній пам'яті.
5. Процес готовий до запуску, але програма підкачки (нульовий процес) повинна ще завантажити процес в оперативну пам'ять, перш ніж він буде запущений під керуванням ядра.
6. Процес призупинено і програма підкачки вивантажила його в зовнішню пам'ять, щоб в оперативній пам'яті звільнити місце для інших процесів.
7. Процес повернутий з привілейованого режиму (режиму ядра) в непривілейований (режим завдання), ядро резервує його і перемикає контекст на інший процес.
8. Процес знову створений і знаходиться в перехідному стані; процес існує, але не готовий до виконання, хоча і не припинений. Цей стан є початковим станом всіх процесів, крім нульового.
9. Процес викликає системну функцію exit і припиняє існування. Однак, після нього залишився запис, що містить код виходу, і деяка хронометрична статистика, що збирається батьківським процесом. Цей стан є останнім станом процесу.

В системі розподілу часу ядро надає процесу ресурси центрального процесора (ЦП) на інтервал часу, званий квантом, після закінчення якого вивантажує цей процес і запускає інший, періодично перевпорядковуючи чергу процесів. Алгоритм планування процесів в системі UNIX використовує час виконання в якості параметра. Кожен активний процес має пріоритет планування; ядро перемикає контекст на процес з найвищим пріоритетом. При переході виконуваного процесу з режиму ядра в режим завдання ядро перераховує його пріоритет, періодично і в режимі завдання перевстановлює пріоритет кожного процесу, готового до виконання [10].

Тобто, відповідно до цього принципу ядро надає процесу ресурси ЦП на квант часу, після закінчення якого вивантажує цей процес і повертає його в одну з декількох черг, регульованих пріоритетами. Перш ніж процес завершиться, йому може знадобитися безліч разів пройти через цикл зі зворотним зв'язком. Коли ядро виконує перемикання контексту і відновлює контекст процесу, процес відновлює виконання з точки припинення.

Зразу після перемикання контексту ядро запускає алгоритм планування виконання процесів, вибираючи на виконання процес з найвищим пріоритетом серед процесів, що знаходяться в станах "резервування" і "готовності до виконання, будучи завантаженим в пам'ять". Розглядати процеси, не завантажені в пам'ять, не має сенсу, оскільки не будучи завантаженим, процес не може виконуватися. Якщо найвищий пріоритет мають відразу кілька процесів, ядро, використовуючи принцип кільцевого списку (каруселі), вибирає серед них той процес, який знаходиться в стані "готовності до виконання" довше за інших. Якщо жоден з процесів не може бути обраний для виконання, ЦП простоює до моменту отримання наступного переривання, яке відбудеться не пізніше ніж через один таймерний тік; після обробки цього переривання ядро знову запустить алгоритм планування.

У функції програми обробки переривань за таймером входить:

* Перезапуск годинника
* Виклик на виконання функцій ядра, що використовують вбудований годинник
* Підтримка можливості профілювання виконання процесів в режимах ядра і завдання
* Збір статистики про систему і процеси, що протікають в ній
* Стеження за часом
* Посилка процесам сигналів "будильника" по запиту
* Періодичне поновлення процесу підкачки
* Управління диспетчеризацією процесів

Деякі з функцій реалізуються при кожному перериванні по таймеру, інші - після декількох таймерних тіків.

Програма обробки переривань за таймером запускається з високим пріоритетом звернення до процесора, не допускаючи під час роботи виникнення інших зовнішніх подій (таких як переривання від периферійних пристроїв). Тому програма обробки переривань за таймером працює дуже швидко, за максимально короткий час пробігаючи свої критичні відрізки, які повинні виконуватися без переривань з боку інших процесів.

## Особливості використання Linux в клієнт-серверних мережах

Linux - це сімейство Unix-подібних операційних систем з відкритим вихідним кодом, заснованих на ядрі Linux, ядро операційної системи випущено вперше 17 вересня, 1991 - Лінусом Торвальдсом. Linux, як правило, упакований у дистрибутив Linux [11].

Дистрибутив включає ядро Linux і додаткове системне програмне забезпечення та бібліотеки, багато з яких надаються проектом GNU. Багато дистрибутивів Linux використовують слово "Linux" в своїй назві, але Фонд вільного програмного забезпечення використовує назву GNU/Linux, щоб підкреслити важливість програмного забезпечення GNU, викликаючи певні суперечки.

Популярні дистрибутиви Linux включають Debian, Fedora і Ubuntu. Комерційні дистрибутиви включають Red Hat Enterprise Linux і SUSE Linux Enterprise Server. Дистрибутиви для настільних комп'ютерів Linux включають віконну систему, таку як X11 або Wayland, а також робоче середовище, наприклад GNOME або KDE Plasma. Дистрибутиви, призначені для серверів, можуть взагалі не мати графічного середовища та включати стек рішення, наприклад LAMP. Оскільки Linux вільно розповсюджується, кожен може створити дистрибутив для будь-яких цілей [12].

Спочатку Linux був розроблений для персональних комп'ютерів на базі архітектури Intel x86, але з тих пір був перенесений на більше платформ, ніж будь-яка інша операційна система. Linux є провідною операційною системою на серверах та інших великих системах, таких як мейнфрейми, і єдиною ОС, що використовується на ТОП500 суперкомп'ютерах (з листопада 2017 року, поступово усунувши всіх конкурентів). Його використовують близько 2,3% настільних комп'ютерів.

Linux також працює на вбудованих системах, тобто на пристроях, чия операційна система, як правило, вбудована в прошивку і дуже адаптована до системи. Серед таких систем є маршрутизатори, засоби автоматизації, телевізори, цифрові відеомагнітофони, консолі для відеоігор і смарт-годинники. Багато смартфонів і планшетних комп'ютерів запускають Android та інші похідні Linux. Через домінування Android на смартфонах, Linux має найбільшу встановлену базу всіх операційних систем загального призначення.

Linux є одним з найвідоміших прикладів співпраці вільного та відкритого програмного забезпечення. Вихідний код може використовуватися, змінюватися та розповсюджуватися - комерційно або некомерційно - будь-яким користувачем згідно з його відповідними ліцензіями, такими як GNU General Public License.

Linux став популярним на ринку інтернет-серверів, особливо завдяки комплекту програмного забезпечення LAMP.

Стек Linux-рішень для веб-серверів, що базуються на Linux, мають всі загальні переваги та переваги вільного та відкритого програмного забезпечення. Деякі більш відомі приклади:

* LAMP-стек
* MEAN-стек

Існують різні вільно доступні реалізації серверів LDAP. Крім того, Univention Corporate Server, як інтегрована система управління на базі Debian, підтримує функції, що надаються Microsoft Active Directory для адміністрування комп'ютерів під керуванням Microsoft Windows.

Безкоштовне програмне забезпечення маршрутизації, доступне для Linux, включає BIRD, B.A.T.M.A.N., Quagga і XORP. Будь-яке обладнання, будь то персональний комп'ютер чи корпоративний сервер, немодифіковане ядро Linux або адаптоване і оптимізоване ядро Linux, здатне здійснювати маршрутизацію зі швидкістю, обмеженою пропускною здатністю апаратних шин.

Згідно з останніми досліжденням W3Tech серед публічних Інтернет серверів(веб-сервери, DNS-сервери та поштові сервери) 69.8% працювали на Unix-подібних системах, і 52.5% з цих серверів працювали під управлінням систем на основі ядра Linux.

Найпоширенішим використанням операційних систем на основі ядра Linux є веб-сервери на основі LAMP-стеку.

Веб-сервери на базі ядра Linux складаються з чотирьох програмних компонентів. Ці компоненти є шарами, що підтримують один одного та складають стек програмного забезпечення. Веб-сайти та веб-програми виконуються на цьому базовому стеці. Загальноприйнятими компонентами програмного забезпечення, що складають традиційний стек LAMP, є:

* Linux: операційна система (ОС) - це перший шар. Linux є фундаментом для моделі стека. Всі інші шари виконуються поверх цього шару.
* Apache: Другий шар складається з програмного забезпечення веб-сервера, як правило, веб-сервера Apache. Цей шар розташований поверх шару Linux. Веб-сервер Apache обробляє запити і обслуговує веб-ресурси через HTTP, так що програма доступна для всіх, хто має доступ до публічного домену, через просту URL-адресу.
* MySQL: Третій рівень - це бази даних. MySQL зберігає дані, які можна запитувати за допомогою скриптів для створення веб-сайту. MySQL зазвичай стоїть на вершині шару Linux поряд з Apache (2-м шаром). У конфігураціях з багатьма серверами, може бути виділено цілий окремий сервер лише для MySQL (окремо від Apache).
* PHP: Це четвертий і останній шар, на вершині стеку. Шар скриптів складається з PHP та/або інших подібних мов веб-програмування. Веб-сайти та веб-програми виконуються в цьому шарі [13].

Стек LAMP можна візуалізувати наступним чином:

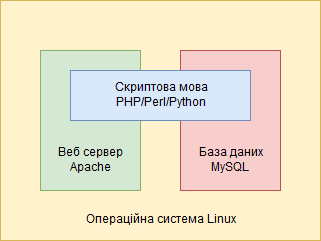


Рисунок 1.8 − Будова LAMP-стека

## Нормативно-правова база у сфері захисту інформації

В даній дипломній роботі було розглянуто такі нормативно-правові документи в сфері захисту інформації:

1. Закон України "Про iнформацiю".

Стаття 1. Визначення термінів

1. У цьому Законі наведені нижче терміни вживаються в такому значенні: документ - матеріальний носій, що містить інформацію, основними функціями якого є її збереження та передавання у часі та просторі;

захист інформації - сукупність правових, адміністративних, організаційних, технічних та інших заходів, що забезпечують збереження, цілісність інформації та належний порядок доступу до неї;

інформація - будь-які відомості та/або дані, які можуть бути збережені на матеріальних носіях або відображені в електронному вигляді;

суб'єкт владних повноважень - орган державної влади, орган місцевого самоврядування, інший суб'єкт, що здійснює владні управлінські функції відповідно до законодавства, у тому числі на виконання делегованих повноважень.

Стаття 21. Інформація з обмеженим доступом

1. Інформацією з обмеженим доступом є конфіденційна, таємна та службова інформація.

2. Конфіденційною є інформація про фізичну особу, а також інформація, доступ до якої обмежено фізичною або юридичною особою, крім суб'єктів владних повноважень. Конфіденційна інформація може поширюватися за бажанням (згодою) відповідної особи у визначеному нею порядку відповідно до передбачених нею умов, а також в інших випадках, визначених законом.

Відносини, пов'язані з правовим режимом конфіденційної інформації, регулюються законом [14].

2. Положення про технічний захист інформації в Україні.

1. Це Положення визначає правові та організаційні засади технічного захисту важливої для держави, суспільства і особи інформації, охорона якої забезпечується державою відповідно до законодавства.

Технічний захист інформації здійснюється щодо органів державної влади, органів місцевого самоврядування, органів управління Збройних Сил України та інших військових формувань, утворених згідно із законодавством України, відповідних підприємств, установ, організацій (далі - органи, щодо яких здійснюється ТЗІ).

2. Ужиті в цьому Положенні терміни мають таке значення:

конфіденційність - властивість інформації бути захищеною від несанкціонованого ознайомлення;

цілісність - властивість інформації бути захищеною від несанкціонованого спотворення, руйнування або знищення;

доступність - властивість інформації бути захищеною від несанкціонованого блокування;

технічний захист інформації (ТЗІ) - діяльність, спрямована на забезпечення інженерно-технічними заходами конфіденційності, цілісності та доступності інформації;

інформаційна система - автоматизована система, комп'ютерна мережа або система зв'язку;

дозвіл - документ, що надає право на виконання робіт з технічного захисту інформації для власних потреб;

комплекс технічного захисту інформації - сукупність заходів та засобів, призначених для реалізації технічного захисту інформації в інформаційній системі або на об'єкті[15].

3. Закон України "Про основні засади забезпечення кібербезпеки України".

Стаття 4. Об’єкти кібербезпеки та кіберзахисту

1. Об’єктами кібербезпеки є:

1) конституційні права і свободи людини і громадянина;

2) суспільство, сталий розвиток інформаційного суспільства та цифрового комунікативного середовища;

3) держава, її конституційний лад, суверенітет, територіальна цілісність і недоторканність;

4) національні інтереси в усіх сферах життєдіяльності особи, суспільства та держави;

5) об’єкти критичної інфраструктури.

2. Об’єктами кіберзахисту є:

1) комунікаційні системи всіх форм власності, в яких обробляються національні інформаційні ресурси та/або які використовуються в інтересах органів державної влади, органів місцевого самоврядування, правоохоронних органів та військових формувань, утворених відповідно до закону;

2) об’єкти критичної інформаційної інфраструктури;

3) комунікаційні системи, які використовуються для задоволення суспільних потреб та/або реалізації правовідносин у сферах електронного урядування, електронних державних послуг, електронної комерції, електронного документообігу[16].

4. Конвенція про кіберзлочинність

В Конвенції про кіберзлочинність дані визначення таким термінам, як: комп'ютерна система, комп'ютерні дані, постачальник послуг, дані про рух інформації. Також в конвенції описані такі загрози, як: незаконний доступ, нелегальне перехоплення, втручання у дані та втручання у систему.

Стаття 2 - Незаконний доступ

Кожна Сторона вживає такі законодавчі та інші заходи, які можуть бути необхідними для встановлення кримінальної відповідальності відповідно до її внутрішнього законодавства за навмисний доступ до цілої комп'ютерної системи або її частини без права на це. Сторона може вимагати, щоб таке правопорушення було вчинене шляхом порушення заходів безпеки з метою отримання комп'ютерних даних або з іншою недобросовісною метою, або по відношенню до комп'ютерної системи, поєднаної з іншою комп'ютерною системою.

Стаття 3 - Нелегальне перехоплення

Кожна Сторона вживає такі законодавчі та інші заходи, які можуть бути необхідними для встановлення кримінальної відповідальності відповідно до її внутрішнього законодавства за навмисне перехоплення технічними засобами, без права на це, передач комп'ютерних даних, які не є призначеними для публічного користування, які проводяться з, на або всередині комп'ютерної системи, включаючи електромагнітні випромінювання комп'ютерної системи, яка містить в собі такі комп'ютерні дані. Сторона може вимагати, щоб таке правопорушення було вчинене з недобросовісною метою або по відношенню до комп'ютерної системи, поєднаної з іншою комп'ютерною системою.

Стаття 4 - Втручання у дані

1. Кожна Сторона вживає такі законодавчі та інші заходи, які можуть бути необхідними для встановлення кримінальної відповідальності відповідно до її внутрішнього законодавства за навмисне пошкодження, знищення, погіршення, зміну або приховування комп'ютерної інформації без права на це.

Стаття 5 - Втручання у систему

Кожна Сторона вживає такі законодавчі та інші заходи, які можуть бути необхідними для встановлення кримінальної відповідальності відповідно до її внутрішнього законодавства за навмисне серйозне перешкоджання функціонуванню комп'ютерної системи шляхом введення, передачі, пошкодження, знищення, погіршення, заміни або приховування комп'ютерних даних без права на це[17].

## Постановка задачі

Для виконання мети цієї дипломної роботи необхідно виконати наступні завдання:

* Розглянути клієнт-серверну архітектуру, її елементи.
* Розглянути Unix системи та їх функціонування.
* Розглянути концепцію безпеки Unix-подібних систем та її проблеми.
* Визначити атаки та ризики для ОС Unix та супутнього ПЗ.
* Розробити рекомендації до захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом
* Розробити механізми захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом

## Висновки за розділом 1

В даному розділі було проаналізовано роботу, класифікацію та переваги клієнт-серверних мереж. Надано опис двохрівневої та трирівневої клієнт-серверної архітектури. Надано опис тонких, товстих та гібридних клієнтів та особливості їх функціонування.

Також в даному розділі було описано та проаналізовано роботу операційних систем сімейства Unix. Описано такі компонетни Unix як ядро, оболонка, файлова система, а також особливості функціонування, процеси, права доступу та пристрої.Також надано коротку характеристику та область використання для операційних систем на основі ядра Linux, що належать до сімейства Unix, зокрема найпоширенішого використання – LAMP-стеку.

В цьому розділі наведено основні закони України в сфері захисту інформації, серед яких: Закон України "Про iнформацiю", Положення про технічний захист інформації в Україні, Закон України "Про основні засади забезпечення кібербезпеки України" та Конвенція про кіберзлочинність.

В кінці цього розділу визначено основні завдання, які необхідно виконати для досягнення мети даної дипломної роботи

Таким чином в даному розділі було виконано перший етап розробки рекомендацій захисту клієнт-серверних мереж на базі операційних систем з відкритим кодом.

# РОЗДІЛ 2

# ПРОБЛЕМИ БЕЗПЕКИ UNIX ТА СУПУТНЬОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Не зважаючи на те, що вразливості в системах безпеки відомі ще з часів черв’яка Морріса, все ті ж вразливості знаходять в програмному запезпеченні важливому для функціонування інформаційних систем. Цей розділ описує вразливості які можуть зашкодити інформаційним процесам в мережевих системах в яких використовуються операційні системи Unix/Linux та пов’язане з ними програмне забезпечення.

## 2.1 Концепція безпеки Unix-подібних систем

В UNIX роль номінального суб'єкта безпеки грає користувач. Кожному користувачеві видається (зазвичай - одне) вхідне ім'я (login). Кожному вхідному імені відповідає єдине число, ідентифікатор користувача (User IDentifier, UID). Це число і є ярлик суб'єкта, яким система користується для визначення прав доступу [18].

Кожен користувач входить в одну або більше груп. Група - це утворення, яке має власний ідентифікатор групи (Group IDentifier, GID), об'єднує декількох користувачів системи, а отже, відповідає поняттю множинний суб'єкт. Значить, GID - це ярлик множинного суб'єкта, яких у дійсного суб'єкта може бути більше одного. Таким чином, одному UID відповідає список GID.

Роль дійсного (працюючого з об'єктами) суб'єкта грає процес. Кожен процес забезпечений єдиним UID - це ідентифікатор користувача, що запустив процес. Будь-який процес, породжений деяким процесом, успадковує його UID. Таким чином, всі процеси, що запускаються за бажанням користувача, матимуть його ідентифікатор. UID враховуються, наприклад, коли один процес посилає іншому сигнал. У загальному випадку дозволяється посилати сигнали «своїм» процесам (тим, що мають такий же UID).

Права доступу. Роль об'єктів в UNIX грають багато реальних об'єктів, зокрема представлені в файлової системі: файли, каталоги, пристрої та канали. Кожен файл забезпечений UID - ідентифікатором користувача-власника. До того ж у файлу є єдиний GID, що визначає групу, якій він належить.

На рівні файлової системи в UNIX визначається три види доступу: читання (read, r), запис (write, w) і виконання (execution, x). Право на читання з файлу дає доступ до інформації, що міститься в ньому, а право запису - можливість її змінювати. У кожного файлу є список того, що з ним може робити власник (якщо збігається UID процесу і файлу), член групи власників (якщо збігається GID) і хто завгодно (якщо нічого не збігається). Такий список для кожного об'єкта системи займає всього кілька байт.

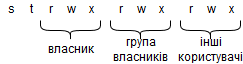


Рисунок 2.1 − Базові права доступу в UNIX

Прапор виконання трактується по-різному в залежності від типу файлу. У випадку простого файлу він задає можливість виконання файлу, тобто запуску програми, що міститься в цьому файлі. Для каталогу - це можливість доступу до файлів в цьому каталозі (точніше кажучи, до атрибутів цих файлів - імені, прав доступу і т.п.).

Розглянемо послідовність перевірки прав на прикладі.

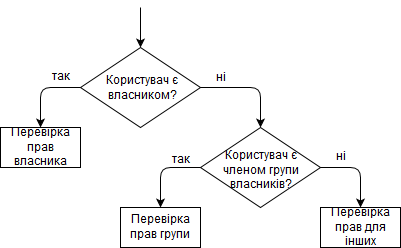


Рисунок 2.2 − Послідовність перевірки прав доступу в UNIX

Нехай файл має такі атрибути:

file.txt alice: users rw- r-- ---

Тобто файл належить користувачеві «alice», групі «users» і має права на читання і запис для власника і тільки читання для групи.

Нехай файл намагається прочитати користувач «bob». Він не є власником, проте він є членом групи «users». Отже, він має права на читання цього файлу.

Спільні каталоги. Право записувати в католог трактується як можливість створення і видалення файлів, а також зміна атрибутів файлів (наприклад, перейменування). При цьому суб'єкту не обов'язково мати права на запис в ці файли, що видаляються.

Таким чином, зі свого каталогу користувач може видалити будь-який файл. А якщо записувати в каталог дозволено всім, то будь-який користувач зможе видалити в ньому будь-який файл. Для уникнення цієї проблеми було додано ще один біт в права доступу каталогу: біт нав'язливості (sticky, t-біт). При його установці користувач, який має доступ на запис в цей каталог, може змінювати тільки власні файли.

Підміна ідентифікатора суб'єкта. В UNIX існує механізм, що дозволяє користувачам запускати процеси від імені інших користувачів. Це може бути корисним, якщо одному користувачеві необхідно на час надавати права іншого (наприклад, суперкористувача).

Для вирішення підміни ідентифікатора користувача застосовується біт підміни ідентифікатора користувача (set user id, suid-біт, s). Цей біт застосовується спільно з бітом виконання (x) для звичайних файлів. При установці цього біта на виконуваний файл процес запускається від імені власника, а не від імені користувача котрий запускає.

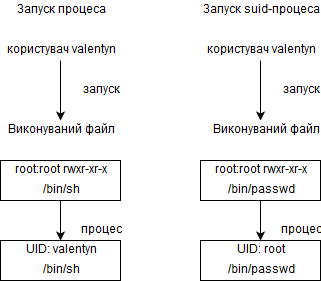


Рисунок 2.3 − Підміна ідентифікатора користувача

Підміна ідентифікатора користувача є потенційною загрозою безпеці системи і повинна використовуватися обережно.

Недоліки базової моделі доступу і її розширення. Обмеженість системи прав UNIX призводить до того, що, наприклад, неможливо створити такий стан речей, коли одна група користувачів могла б тільки читати з файлу, інша - тільки запускати його, а всім іншим файл взагалі не був би доступний. Інша справа, що такий стан речей зустрічається нечасто.

Згодом в різних версіях UNIX стали з'являтися розширення прав доступу, що дозволяють встановлювати права на окремі об'єкти системи. Спочатку це були так звані прапори - додаткові атрибути файлу, що не дозволяють, наприклад, перейменовувати його або видаляти з нього інформацію під час запису (можна тільки дописувати). Прапори не усувають головного недоліку, зате їх легко організувати без зміни файлової системи: кожен прапор займає рівно один біт.

Багато сучасних файлових систем UNIX підтримують також списки доступу (ACL), за допомогою яких можна для кожного об'єкта задавати права всіх суб'єктів на доступ до нього.

На практиці прапори або управління доступом використовувати доводиться нечасто. У більшості випадків така необхідність виникає як виняток - наприклад, для тимчасового зменшення прав або для тимчасового надання доступу (легко зробити за допомогою ACL), а також при роботі з дуже важливими файлами.

Суперкористувач. Користувач root (він же привілейований користувач) має нульові UID і GID і грає роль довіреного суб'єкта UNIX. Це означає, що він не підкоряється законам, які керують правами доступу, і може на свій розсуд ці права змінювати. Більшість налаштувань системи доступні для запису тільки суперкористувачеві.

В UNIX існує рівень доступу ядра і рівень доступу системи. Суперкористувач працює на рівні доступу ядра, так що є по суті продовженням самої системи.

Багато команд повинні виконуватися тільки від імені суперкористувача, так як в них відбувається взаємодія з частинами ядра, що відповідають за взаємодію з апаратурою, права доступу і т.п.. Якщо ж такі команди дозволяється запускати простим користувачам, застосовується розглянутий вище механізм підміни ідентифікатора користувача.

Системне адміністрування в UNIX проводиться від імені користувача root. При роботі від цього імені слід бути дуже обережним: виконання невірної команди може привести до краху системи і знищення інформації. Тому навіть адміністратори ніколи не працюють в сеансі суперкористувача весь час, а переходять в режим суперкористувача тільки тоді, коли це дійсно необхідно (наприклад, за допомогою команди su).

Аутентифікація користувачів. В UNIX сеанс роботи користувача починається з його аутентифікації і закінчується його виходом із системи. При вході в систему виконується наступна послідовність дій:

* процес getty очікує реакції користувача на одній з термінальних ліній, в разі активності користувача виводить запрошення;
* після введення імені користувача запускається програма login, яка перевіряє справжність користувача. Стандартним механізмом є перевірка пароля, заданого для даного користувача;
* переконавшись, що пароль введений правильно, login запускає командний інтерпретатор з встановленими UID і GID даного користувача. Таким чином, права доступу будь-якої програми (дійсного суб'єкта), запущеної користувачем в цьому сеансі роботи, будуть визначатися правами номінального суб'єкта UID + GID.

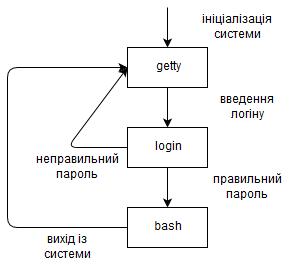


Рисунок 2.4 − Процес входу в систему.

При роботі у мережі роль getty виконує мережевий демон, наприклад ssh. У деяких сучасних UNIX-системах існують розширення систем авторизації і аутентифікації. Наприклад, в Linux-системах цей механізм називається змінні модулі розпізнавання (Pluggable Authentication Modules, PAM).

## Переповнення буфера

### Загальний опис проблеми

Зловмисники, як правило, використовують переповнення буфера, щоб пошкодити стек виконання веб-додатку. Надіславши ретельно створені вхідні дані в додаток, зловмисник може змусити додаток виконати довільний код, можливо, взявши керування сисеми на себе. Зловмисникам вдалося виявити переповнення буфера в приголомшливій кількості продуктів і компонентів.

Недоліки переповнення буфера можуть бути присутніми як у веб-сервері, так і у серверних продуктах, які обслуговують статичні та динамічні частини сайту або в самих веб-застосунках чи інших застосунках. Якщо переповнення буфера буде знайдене у часто використовуваних серверних продуктах то, ймовірно, стане широко відомим і може становити значний ризик для користувачів цих продуктів. Коли веб-програми використовують бібліотеки, такі як графічна бібліотека для створення зображень або бібліотеки зв'язку для відправки електронної пошти, вони відкриваються для потенційних атак переповнення буфера. Література, що описує атаки переповнення буфера на найпоширеніші продукти, легко доступна, і про нововиявлені вразливості повідомляється майже щодня [19].

Переповнення буфера може також бути знайдене в спеціальному коді веб-додатків, і навіть може бути більш ймовірним, враховуючи відсутність ретельної перевірки веб-додатків. Атаки переповнення буфера на спеціальні веб-програми іноді можуть привести до цікавих результатів. У деяких випадках надсилання великих входних даних може призвести до збоїв у роботі веб-програми або бази даних. Можна викликати атаку на відмову в обслуговуванні веб-сайту, залежно від серйозності та специфіки недоліку. Надмірно великі вхідні дані можуть призвести до того, що програма відобразить детальне повідомлення про помилку, що може призвести до успішної атаки на систему.

Атаки переповнення буфера зазвичай покладаються на дві технології (або їх комбінацію):

* Запис даних на певні адреси пам'яті
* Неправильна інтерпретація даних операційною системою

Це означає, що сильно типізовані мови програмування (та середовища), які забороняють прямий доступ до пам'яті, зазвичай перешкоджають переповненню буфера.

*Таблиця 2.1*

Порівняння типізації та функцій мов програмування

|  |  |  |  |
| --- | --- | --- | --- |
| Мова/Середовище | Компільована чи інтерпретована | Сильна типізація | Прямий доступ до пам’яті |
| Java, віртуальна машина Java (JVM) | Обидва варіанти | Так | Ні |
| .NET | Обидва варіанти | Так | Ні |
| Perl | Обидва варіанти | Так | Ні |
| Інтерпретатор Python | Інтерпретована | Так | Ні |
| Ruby | Інтерпретована | Так | Ні |
| C/C++ | Компільована | Ні | Так |
| Асемблер | Компільована | Ні | Так |

Переповнення стеку є найбільш зрозумілою і найпоширенішою формою переповнення буфера. Основи переповнення стека прості:

* Існують два буфери, вихідний буфер, що містить довільні вхідні дані (імовірно, від зловмисника), і буфер призначення, який є занадто малим для вхідних даних атаки. Другий буфер розташований на стеку і частково суміжний з поверненням функції на стеку.
* Недосконалий код не перевіряє, що вихідний буфер занадто великий для розміщення в буфері призначення. Він копіює вхідні дані атаки в буфер призначення, перезаписуючи додаткову інформацію в стеку (наприклад, адресу повернення функції).
* Коли функція повертається, центральний процесор виконує кадр стека і виводить (тепер змінену) адресу повернення зі стека. Керування не повертається до функції, як повинно. Замість цього виконується довільний код (обраний зловмисником при створенні початкового входу).

Переповнення буфера через рядки форматування (що зазвичай називаються "вразливостями форматування рядка") є вузькоспеціалізованими переповненнями буферів, які можуть мати ті ж ефекти, що й інші атаки переповнення буфера. В основному, вразливості форматування рядка використовують переваги змішування даних і керуючої інформації в певних функціях, таких як printf у C/C++.

Якщо програма приймає два числа фіксованого розміру та виконує операцію з ними, результат може не відповідати одному і тому ж бітовому розміру. Наприклад, якщо два 8-розрядні числа 168 і 192 додаються разом і зберігаються в іншому 8-бітовому записі, результат не поміститься в 8-бітовий запис для результату.

### Вплив на властивості інформації

Атаки які реалізують вразливість переповнення буферу в основному впрливають на доступність інформації, руйнуючи стек виконня. Рідше ніж руйнування стеку відбувається виконання коду вставленого зловмисником, оскільки це вимагає більшої підготовки та знань[20];[21];[22].

*Таблиця 2.2*

Вплив переповнення буфера на властивості інформації

|  |  |  |
| --- | --- | --- |
| Властивість | Наслідки | Вплив |
| Доступність | Нестабільна робота чи припинення програми | Переповнення буфера зазвичай призводить до збоїв. Можливі й інші атаки, що призводять до відсутності доступності, включаючи переведення виконання програми в нескінченний цикл. |
| Цілісність,  Конфіденційність,  Доступність | Несанкціоноване виконання коду | Переповнення буфера часто може бути використано для виконання довільного коду, який зазвичай виходить за рамки дозволеної політикою безпеки роботи програми. |

## Стан гонитви

### Загальний опис проблеми

Стан гонитви виникає, коли програма не працює так, як передбачається, через непередбачене упорядкування подій, яке викликає конфлікт над одним і тим же ресурсом. Багато проблем з безпекою виникають, якщо зловмисник ззовні може несподівано втрутитися в програму. Десятиліття тому стан гонитви був меншою проблемою. Раніше комп'ютерні системи виконували одну просту послідовну програму одночасно, і ніщо не могло перервати її або змагатися з нею. Але сьогоднішні комп'ютери, як правило, мають велику кількість процесів і потоків, що працюють одночасно, і часто мають кілька процесорів, що виконують різні програми одночасно. Це більш гнучко, але існує небезпека: якщо ці процеси і потоки поділяють будь-які ресурси, вони можуть втручатися в роботу один одного[23].

Уразливість до стану гонитви є однією з найбільш поширених вразливостей у програмному забезпеченні, а на UNIX-подібних системах каталоги /tmp та /var/tmp часто використовуються таким чином, що створюється потенційна вразливість типу стан гонитви.

Стан гонитви може мати негативні наслідки для безпеки, коли очікувана синхронізація знаходиться в критично важливому коді безпеки, наприклад, записує чи автентифікований користувач, або змінює важливу інформацію про стан, на яку не повинен впливати сторонній.

### Вплив на властивості інформації

Стан перегонів відбувається в середовищах, що перетинаються і фактично є властивістю кодової послідовності. Залежно від контексту, кодова послідовність може бути у вигляді виклику функції, невеликої кількості інструкцій, серії викликів програм і т.д.[24].

Стан гонитви порушує властивості, які тісно пов'язані з:

* Ексклюзивністю - кодовій послідовності надається ексклюзивний доступ до спільного ресурсу, тобто ніяка інша кодова послідовність не може змінювати властивості спільного ресурсу до того, як початкова послідовність завершить виконання.
* Атомність - кодова послідовність є поведінково-атомною, тобто жоден інший потік або процес не може одночасно виконувати ту ж послідовність інструкцій над одним і тим же ресурсом.

Стан гонитви існує, коли "заважаюча кодова послідовність" все ще може отримати доступ до спільного ресурсу, порушуючи ексклюзивність. Програмісти можуть припустити, що певні кодові послідовності виконуються занадто швидко, щоб бути під впливом заважаючої послідовності коду, впротилежному випадку - це порушує атомічність. Наприклад, одна операція "x ++" може виглядати атомною на шарі коду, але насправді не є атомною на шарі інструкцій, оскільки вона включає читання (початкове значення x), а потім обчислення (x + 1). ), за яким йде запис (збережіть результат до x)[25].

Послідовність коду, що заважає, може бути "довіреною" або "недовіреною". Довірена інтерферуюча кодова послідовність виконується в межах програми, вона не може бути змінена зловмисником, і її можна викликати лише опосередковано. Ненадійна послідовність коду, що заважає, може бути створена безпосередньо зловмисником, і, як правило, вона є зовнішньою для вразливої програми.

У таблиці нижче вказані різні індивідуальні наслідки, пов'язані зі слабкістю. Властивість ідентифікує область безпеки програми, яка порушується, в той час як вплив описує негативний технічний вплив, який виникає, якщо зловмисник успішно використовує цю слабкість.

*Таблиця 2.3*

Вплив на властивості безпеки програм та наслідки вразливостей типу стан гонитви

|  |  |  |
| --- | --- | --- |
| Властивість | Наслідки | Вплив |
| Доступність | Виснаження ресурсів | Коли стан гонитви дозволяє обходити процедуру очищення ресурсів або викликати кілька процедур ініціалізації, це може призвести до вичерпання ресурсу |

*Продовження таблиці 2.3*

|  |  |  |
| --- | --- | --- |
| Доступність | Нестабільна робота чи припинення програми | Коли стан гонитви дозволяє багатьом потокам отримувати доступ до ресурсу одночасно, це може призвести до несподіваних наслідків для програми, що може стати причиною нестабільної роботи чи припинення її виконання. |
| Цілісність/  Конфіденційність | Запис/зчитування даних програм | Коли стан гонитви поєднується з передбачуваними іменами ресурсів і вільними дозволами, зловмисник може перезаписати або отримати доступ до конфіденційних даних. |

## Атака на відмову в обслуговуванні (DoS) та розподілена атака на відмову в обслуговуванні

### Загальний опис проблеми

Атака "відмова в обслуговуванні" (DoS) відбувається тоді, коли законні користувачі не можуть отримати доступ до інформаційних систем, пристроїв або інших мережевих ресурсів через дії зловмисника. Послугами можуть бути електронна пошта, веб-сайти, облікові записи в Інтернеті (наприклад, банківські) або інші послуги, які залежать від комп'ютера або мережі, якої це стосується. Відмова в обслуговуванні досягається шляхом завантаження цільового хоста або мережі трафіком до тих пір, поки цільовий об'єкт не матиме змоги відповісти або просто припинить функціонувати, запобігаючи можливості доступу для легітимних користувачів. Атаки DoS можуть коштувати організації як часу, так і грошей, поки їх ресурси та послуги недоступні[26].

Розподілена атака відмови в обслуговуванні (DDoS) відбувається, коли кілька машин працюють разом, щоб атакувати одну ціль. DDoS дозволяє створювати значно більше запитів, що надсилаються до цілі, таким чином збільшуючи потужність атаки. Це також збільшує труднощі відстеження, оскільки справжнє джерело нападу важче виявити.

Для DDoS зловмисники часто використовують ботнет - групу захоплених пристроїв, підключених до Інтернету, для здійснення масштабних атак. Зловмисники використовують уразливості системи безпеки або слабкі місця пристроїв для керування численними пристроями за допомогою програмного забезпечення для керування та контролю. Під час керування зловмисник може командувати своїм ботнетом та проводити DDoS атаки на цілі. У цьому випадку заражені пристрої також є жертвами атаки.

Після встановлення, ботнет, що складається з скомпрометованих пристроїв, також може бути переданий в оренду іншим потенційним зловмисникам. Часто ботнет доступний для послуг "атаки за наймом", які дозволяють навіть самим некваліфікованим користувачам запускати DDoS-атаки.

Кількість атак DDoS збільшується в міру того, як все більше пристроїв приходять в Інтернет через Інтернет речей (IoT). Прилади IoT часто використовують паролі за замовчуванням і не мають надійних механізмів безпеки, що робить їх уразливими для компрометації та експлуатації. Інфікування пристроїв IoT часто залишається непоміченим користувачами, і зловмисник може легко компрометувати сотні тисяч цих пристроїв для проведення масштабної атаки не знаючи власників пристроїв.

Існує багато різних методів для виконання DoS-атаки. Найбільш поширений метод атаки полягає у тому, що зловмисник заповнює мережевий сервер трафіком. У цьому типі DoS-атаки зловмисник посилає декілька запитів на цільовий сервер, перевантажуючи його трафіком. Ці сервісні запити є нелегітимними і мають сфабриковані адреси повернення, які вводять в оману сервер, коли він намагається аутентифікувати запитувача. Оскільки запити зловмисника постійно обробляються, сервер перевантажений, що викликає умову відмови у доступі(DoS) для легітимних відправників запитів.

* У атаці типу Smurf, зловмисник посилає пакети міжмережевого протоколу керуючих повідомлень(ICMP) до декількох хостів з підробленим джерелом IP-адреси, що належить цільовій машині. Одержувачі цих підроблених пакетів відповідатимуть, а цільові хости будуть завантажені цими відповідями.
* Атака SYN flood виникає, коли зловмисник посилає запит на підключення до цільового сервера, але ніколи не завершує з'єднання через так зване "тристороннє рукостискання" - метод, який використовується в мережі TCP/IP для створення з'єднання між локальним хостом/клієнтом і сервером. Неповне рукостискання залишає підключений порт у зайнятому стані і недоступним для подальших запитів. Зловмисник продовжуватиме надсилати запити, завантажуючи всі відкриті порти, щоб легітимні користувачі не могли з'єднатися.

На окремі мережі можуть впливати атаки DoS, безпосередньо не націлені на них. Якщо мережевий постачальник послуг Інтернету (ISP) або постачальник хмарних послуг був націлений і атакований, мережа також втрачає можливість надавати послуги.

### Вплив на властивості інформації

Відмова в обслуговуванні(DoS) є найбільш поширеним результатом вичерпання ресурсів. Програмне забезпечення може сповільнитися, припинити функціонувати повністю чи частково через невиправні помилки або заблокувати легітимних користувачів. Таким чином ця атака порушує властивість доступності до ресурсів та послуг цільової системи [27];[28].

## 2.5 Висновки за розділом 2

В даному розділі було виконано опис концепції безпеки Unix та її основні недоліки та проблеми.

Проаналізовано основні проблеми безпеки для клієнт-серверних мереж на базі операційних систем з відкритим кодом, а саме:

* вразливість переповнення буфера
* вразливість стану гонитви
* атака на відмову в обслуговуванні

Також в цьому розділі проаналізовано властивості інформації, які порушуються при виникненні розглянутих проблем.

# РОЗДІЛ 3

# ЗАСОБИ ЗАХИСТУ ТА РЕКОМЕНДАЦІЇ ДО ЗАХИСТУ КЛІЄНТ-СЕРВЕРНОЇ МЕРЕЖІ НА БАЗІ ОПЕРАЦІЙНИХ СИСТЕМ З ВІДКРИТИМ КОДОМ

## Принципи безпеки клієнт-сервених мереж

Деякі принципи ІТ-безпеки є універсальними незалежно від того, яка операційна система розглядається. Розробники, адміністратори та працівники відділу безпеки завжди повинні мати на увазі наступне:

• Уникання безпеки через невідомість. Програма або система не повинні залежати від відсутності знань користувача про її внутрішню роботу для забезпечення її безпеки. Наприклад розробники програми можуть вважати, що критичні конфігураційні файли, що містять конфіденційну інформацію в тексті ніхто не побачить, якщо ніхто не знає про їх існування. Або, це може бути використання стандартного пароля імені користувача та пароля для аутентифікації, коли системний адміністратор припускає, що ніхто не спробує вгадати.

• Поглиблення побудови безпеки. Інший принцип, якому система повинна відповідати - це поглиблена безпека. Вона не повинна залежати від будь-якого одного шару безпеки для захисту критичних даних. Хороша інфраструктура безпеки завжди створює декілька шарів безпеки для захисту від тих самих вразливостей.

• Створення безпеки з етапу проектування. Обрані топологія мережі та архітектура проектування повинні вирішувати проблеми безпеки безпосередньо на стадії проектування. Необхідно забезпечити баланс між безпекою та функціональністю під час розробки архітектури. Накладання безпеки на вже існуючу інфраструктуру може мати небажані або навіть катастрофічні результати.

• Використання відкритих стандартів замість власницьких протоколів та алгоритмів. При розробці архітектури або програми, як кінцевому користувачу, так і розробнику буде корисним використання відкритих стандартів, на відміну від власницьких алгоритмів і протоколів. Особливо це стосується шифрування та безпечних комунікацій.

• Дотримання принципу найменших привілеїв. Система повинна надавати користувачам тільки ті привілеї, які є абсолютно необхідними для виконання ними своїх функцій. Це ключ до авторизації. Крім того, це єдина сфера, де можна легко керувати простотою функціональності за рахунок безпеки. Функції та критичні файли в системі повинні бути чітко визначені, а також повинні бути встановлені відповідні привілеї операційної системи.

• Заборонити все, дозволити лише потрібне. Кожен раз, коли існує можливість вибору або скасування вибору певних опцій, привілеїв або входів користувача, адміністратори повинні завжди дотримуватися принципу заборони всіх опцій і дозволяти лише ті, які потрібні. Це автоматично випливає з принципу найменших привілеї. У випадку прав доступу користувачів, всі можливі привілеї спершу повинні бути відхилені, а потім - повинні бути надані лише необхідні привілеї.

• Безпека як процес, а не одноразове рішення. Програма повинна мати вбудовані модулі для постійного контролю своєї безпеки. Ніколи не слід вважати, що програма захищена, якщо аутентифікація, авторизація та перевірка входу були належним чином реалізовані. Правильне використання інформаційної панелі та огляду можуть дозволити організації відслідковувати порушення безпеки та несправності прикладних програм. Організація повинна чітко делегувати відповідальність окремій особі або групі осіб для моніторингу та аналізу даних про функціонування через регулярні проміжки часу.

• Розподіл обов'язків - це важливий принцип безпеки, який можна знайти в міжнародних стандартах в сфері безпеки (наприклад в COBIT). Організація повинна мати структуру, в якій системний адміністратор і адміністратор безпеки є двома окремими людьми. У більшості випадків системний адміністратор має повні права на систему. Найбільш важливим способом отримання противаги є передача аудиторських повноважень до іншої системи, над якою цей адміністратор не має повноважень. Старший адміністратор безпеки повинен стежити за цим.

## Фізична безпека

Фізична безпека є одним з найважливіших аспектів безпеки в будь-якій організації. Вона повинна включати не тільки сервери і робочі станції, але й автономні матеріали, такі як діаграми топології, конфігураційні листи, інсталяційні носії та запасні частини. Захист цих носіїв інформації так само важливий, як і захист самих серверів. Фізичні перевірки безпеки повинні включати перевірки вразливостей "відкритого перегляду". Ці уразливості виникають, коли конфіденційні дані є видимими для неавторизованих осіб, оскільки вони не зберігаються належним чином. Одним з прикладів уразливості відкритого перегляду є розширена діаграма топології, закріплена на стіні в кімнаті сервера, яку видно всім, хто проходить повз відкрите приміщення.

Фізичні засоби безпеки, як правило, однакові, незалежно від використовуваних систем. Однак, існують певні деталі, пов'язані з Linux, які необхідно мати на увазі. Крім того, фізичні елементи керування, які можуть бути реалізовані у випадку робочих станцій, дуже відрізняються від тих, що стосуються серверів.

### Забезпечення фізичної безпеки серверів, клієнтів та мережного обладнання

Розташування сервера є критичним. Важливо, щоб всі сервери були фізично захищені в серверній кімнаті. Випадковий зовнішній користувач не повинен бути в змозі зазирнути і побачити всю установку серверної кімнати, або зі скла або з відкритих дверей. Монітори повинні бути добре приховані від "відкритого перегляду".

Тільки обмежений набір персоналу повинен мати доступ до серверної кімнати. Весь такий доступ повинен контролюватися. Цей моніторинг можна здійснити, встановивши систему контролю доступу (наприклад, систему інфрачервоних карток), яка здійснює запис кожного разу, коли сканер сканує інфрачервону карту. Журнали повинні зберігатися надійно. Організація повинна періодично переглядати ці журнали доступу, щоб визначити будь-які надзвичайні записи, такі як певний адміністратор, що входить до кімнати до або після призначеної йому зміни, або адміністратора, що входить до кімнати двічі без відповідної кількості виходів, і так далі. Перелік уповноважених працівників має бути опублікований, і тільки такі працівники повинні мати право доступу до ключів доступу.

Компоненти серверів повинні бути захищені, щоб запобігти жорсткому перезавантаженню, тобто перезавантаженню сервера, натиснувши кнопку перезапуску. Захист компонентів також запобігає завантаженню сервера за допомогою альтернативних носіїв. Часто можна завантажувати сервер за допомогою завантажувального компакт-диска або USB-носія. Це може дозволити зловмисникам завантажувати на сервері операційну систему за своїм вибором і обходити механізми безпеки, які були реалізовані в існуючій системі.

Сервери також повинні бути захищені від природних небезпек, таких як вогонь, вода, природні або техногенні катастрофи. Зазвичай це робиться шляхом встановлення вогнегасників, детекторів диму, детекторів води тощо.

У сучасних мережевих умовах фізична безпека робочих станцій стала майже такою ж важливою, як і для серверів. Розміщення великих механізмів фізичного контролю доступу для серверної кімнати є марним, коли все, що зловмисник повинен зробити - це підійти до робочої станції і увійти в сервер. Необхідно забезпечити щоб всі робочі станції мали достатній час блокування екрана. Користувачі повинні бути навчені не залишати свої термінали, що входять до системи, і виходити з системи, коли вони залишають свою робочу зону протягом періоду, що перевищує 10 хвилин. Однією з поширених помилок, яку роблять більшість користувачів, є записування критичної інформації на листах паперу. Потім вони залишаються на робочому столі. Адекватне навчання користувачів і періодичні перевірки гарантують, що користувачі не залишають свої системи з відкритим доступом до критичної інформації.

Більшість критичних мережевих компонентів, таких як маршрутизатори та брандмауери, повинні бути фізично захищені в самому приміщенні сервера. Вільно доступні інструменти, такі як наприклад ettercap дозволяють зловмисникам перехоплювати трафік навіть у комутованій мережі. Всі засоби передачі повинні бути закріплені, щоб не було встановлено прослуховування на кабелях. Інфраструктура волоконно-оптичного кабелю, хоча й дещо зменшує фізичну небезпеку прослуховування, є досить дорогою.

### Безпека завантаження

Крім захисту компонентів серверів, необхідно також захистити послідовність завантаження. Це стосується як робочих станцій, так і серверів. У більшості дистрибутивів Linux послідовність завантаження може бути забезпечена шляхом налаштування базової системи вводу/виводу (BIOS). Послідовність завантаження повинна гарантувати, що основним пристроєм завантаження завжди є жорсткий диск. В ній не повинно бути компакт-дисків або портів USB. Також необхідно встановити пароль BIOS.

Програмне забезпечення завантажувача Linux дозволяє завантажувати декілька операційних систем. В середовищі серверного рівня на сервер потрібно завантажувати лише одну операційну систему. Зловмисник може спробувати завантажитися в альтернативну операційну систему, обходячи таким чином механізми безпеки, які могли б бути застосовані для дійсної операційної системи. Два найпоширеніших завантажувача Linux - це LILO (LInux LOader) і GRUB (GRand Unified Bootloader).

## Забезпечення захисту операційної системи

### Програмне забезпечення, драйвери та компоненти

Наполеливо рекомендується не встановлювати драйвери та компоненти, які не є абсолютно необхідними для функціональності сервера. Необхідно мати повний список апаратних компонентів сервера, щоб скласти точний список компонентів ядра. Наприклад, не потрібно встановлювати драйвери для підтримки USB, якщо апаратне забезпечення сервера не містить жодних портів USB. Аналогічно, підтримку для різних файлових систем можна скасувати, якщо вони не обслуговуються. Важливо відзначити, що, незважаючи на те, що потреба у одній з таких функціональностей може виникнути пізніше, ці драйвери та компоненти завжди можна додати.

Потрібно отримати список програмного забезпечення, встановленого в системі. Непотрібні пакети повинні бути видалені, оскільки вони споживають ресурси.

Деякі непотрібні пакунки, які можна знайти в установці Linux за замовчуванням і видалити якщо їз використання не потрібне для роботи сервера чи робочої станції :

* Компілятори, бібліотеки та інші засоби розробки
* Пакунки, що надають мережні послуги , такі як Apache чи FTP
* Утиліти для текстової та графічної обробки
* Драйвери для різних типів апаратних пристроїв і файлових систем
* Графічні середовища, такі як Gnome чи KDE

Уразливість системи безпеки в системі виникає з двох основних причин неправильної конфігурації і неоновлених систем. Щоб захистити систему від другої категорії вразливостей, дуже важливо, щоб всі патчі, рекомендовані виробниками, застосовувалися до системи, як тільки вони будуть випущені. Останній список патчів, рекомендованих постачальниками, доступний на веб-сайті постачальника. Більшість дистрибутивів Linux мають утиліти, які автоматично перевіряють наявність останніх оновлень і завантажують їх або повідомляють системному адміністратору.

### Керування графічним інтерфейсом

Як і в інших варіантах UNIX, майже всі завдання системного адміністрування можуть виконуватися з командного рядка. Таким чином, існує дуже мало обставин, за яких необхідно встановити графічний інтерфейс користувача (GUI). Насправді для більшості серверів рекомендується дотримуватися політики мінімальної інсталяції і не встановлювати непотрібні компоненти, такі як графічний інтерфейс користувача. Крім того, графічні інтерфейси Linux використовує здебільшого систему X Windows. Ця система має багато відомих атак проти неї, і зловмисник може отримати доступ до GUI сервера за допомогою системи X Windows. Це дозволить йому/їй виконувати команди, навіть якщо засоби віддаленого входу, такі як Telnet або rlogin відключені. Якщо все ж GUI необхідно встановити то варто використовувати графічне середовище на основі Wayland а не X Wndows.

Якщо графічний інтерфейс встановлено, він повинен бути налаштований на блокування після заданого інтервалу часу. Цей часовий інтервал змінюється в залежності від інших механізмів фізичної безпеки. Наприклад, для серверів, які фізично захищені, час блокування екрана може становити до 15 хвилин. Для робочих станцій Linux час блокування не повинен перевищувати декількох хвилин. Крім того потрібно налаштувати блокування так, щоб для розблокування необхідно було ввести пароль.

## Розробка механізмів захисту клієнт-серверної мережі на базі операційних систем з відкритим кодом

### **Розробка комплексу скриптів для перевірки та маніпуляцій критичних об’єктів сервера чи робочої** станції

В даній дипломній роботі було втілено shell-скрипт для bash, що спрощуватиме роботу адміністратора, який відповідатиме за сервери та/чи робочі станції, за рахунок зручного інтуїтивного інтерфейсу. До його функцій входить:

* перевірка на наявність у системі небажаних пакетів;
* перевірка на те, чи запущені критичні процеси у системі;
* логування результатів у файл /var/log/adminsh/adminsh-*дата*;
* перевірка та можливість змінити власність і права читання-записування-виконання критичних директорій і файлів;
* запуск та зупинка сервісів системи.

Теоретично, усі ці можливості надає Linux-система за замовчуванням. Проте вищеназваний bash-скрипт покликаний для полегшення адміністратором швидкої перевірки критичних об’єктів робочих станцій та серверів.

Скрипт розділено на модулі відповідно до призначень:

* admin.sh (відповідає за логування)
* admin-core.sh (відповідає за перевірки певних процесів та програм)
* admin-pac.sh (відповідає за маніпуляції з пакунками та процесами)
* admin-chmod.sh (відповідає за власність критичних директорій)
* admin-user.sh (меню для подальших маніпуляцій)

Скрипт admin.sh відповідає за логування і є першим скриптом який запускається адміністратором для інфціалізації перевірки. Він зберігає лог у вигляді текстового файлу у папці /var/log/adminsh/ з надписом adminsh та датою створення. Лістінг коду цього скрипта наведено в додатку А. Результат запуску даного скрипта наведено на наступному малюнку.

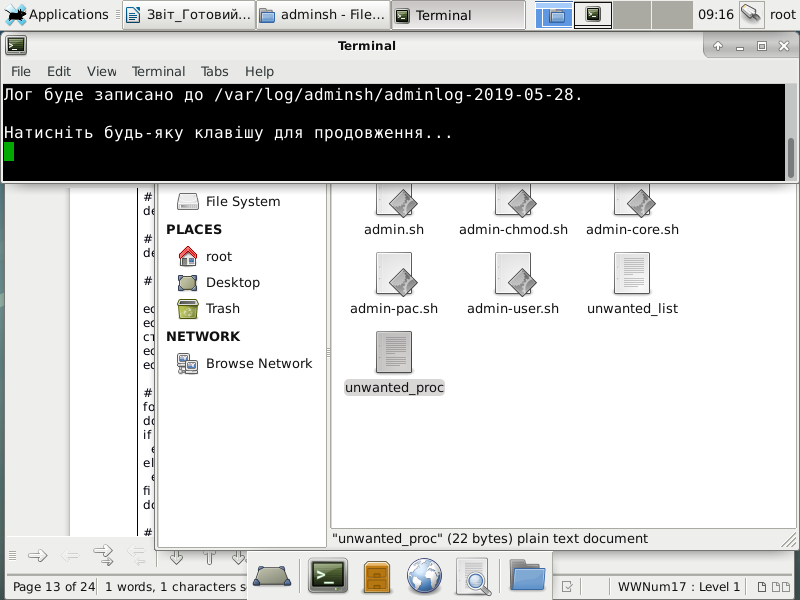


Рисунок 3.1 − Запуск скрипта admin.sh

Скрипт admin-core.sh відповідає за перевірки певних процесів та програм. Він виконує перевірку на наявність певних пакунків в системі, щоб адміністратор міг переконатися в їх відсутності або ж у випадку їх присутності виконав певні дії, такі як видаленння небажаного пакунку з цільової системи. Також в функції цього скрипта входить перевірка запущених процесів, що дозволяє адміністратору бачити чи критично важливі для системи процеси є запущеними. Лістінг коду цього скрипта наведено в додатку Б. Результат роботи даного скрипта наведено на наступному малюнку.

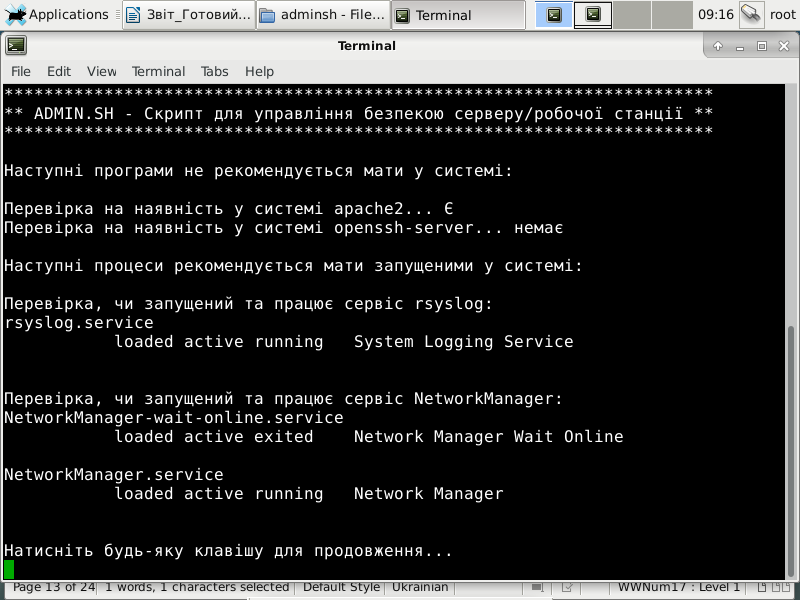


Рисунок 3.2 − Запуск скрипта admin-core.sh

Скрипт admin-pac.sh відповідає за маніпуляції з пакунками та процесами. Він дозволяє змінювати списки процесів та пакунків які потрібно перевірити скрипту admin-core.sh, а також зупиняти процеси якщо це необхідно. Лістінг коду цього скрипта наведено в додатку В. Результат роботи даного скрипта наведено на наступному малюнку.

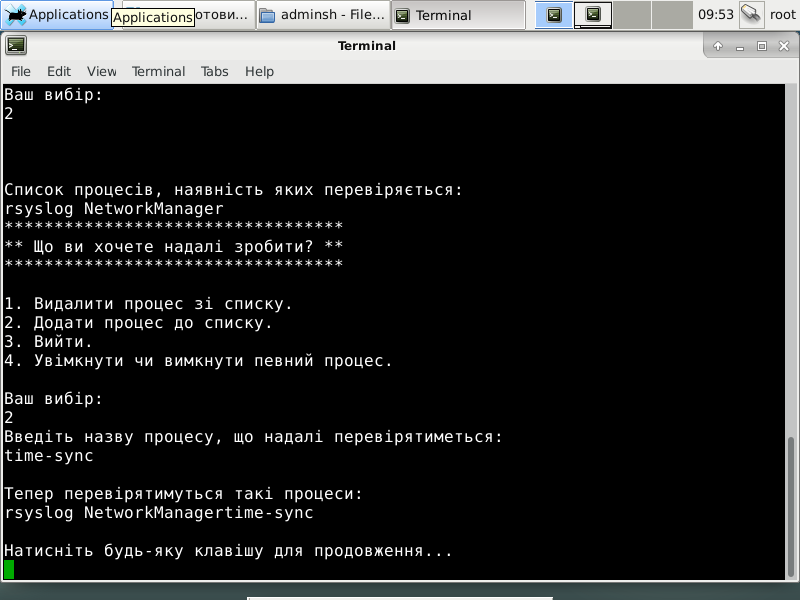


Рисунок 3.3 − Запуск скрипта admin-pac.sh

Скрипт admin-chmod.sh відповідає за власність критичних директорій. Він дозволяє перевірити права та змінити їх для каталогів, визначених адміністратором. Лістінг коду цього скрипта наведено в додатку Д. Результат роботи даного скрипта наведено на наступному малюнку.

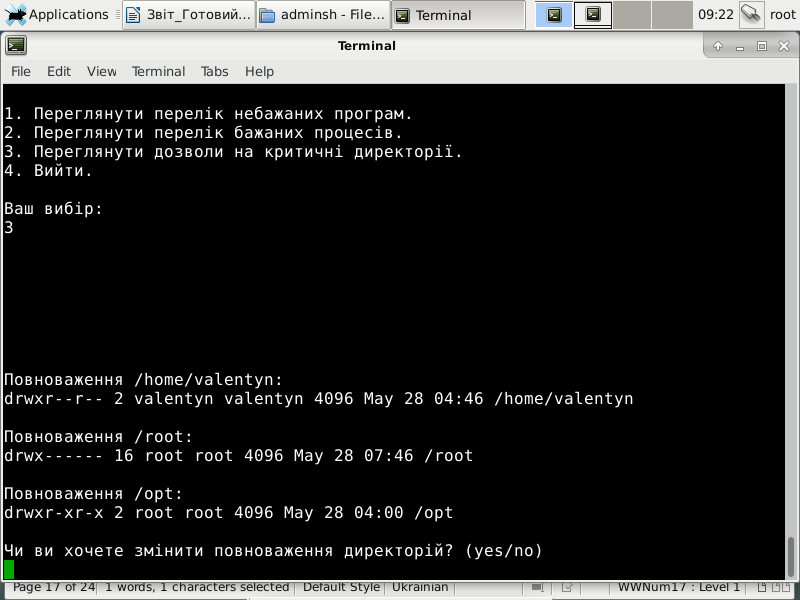


Рисунок 3.4 − Запуск скрипта admin-chmod.sh

Скрипт admin-user.sh відповідає за меню для подальших маніпуляцій зі зручним інтуїтивним інтерфейсом. Лістінг коду цього скрипта наведено в додатку Е. Результат роботи даного скрипта наведено на наступному малюнку.

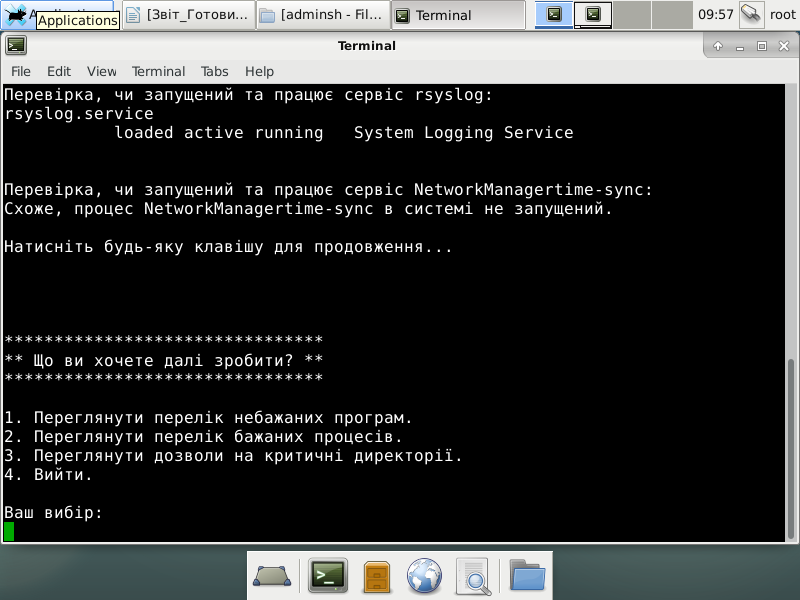


Рисунок 3.5 − Запуск скрипта admin-user.sh

Даний комплекс скриптів рекомендується запускати адміністратору безпочередньо на робочій станції чи сервері, або ж під’єднуватись за допомогою ssh і виконувати його. Для створення логів рекомендується використовувати автоматичний скрипт наведений далі.

### **3.4.2 Розробка скрипта для перевірки та створення логів про критичні об’єкти сервера чи робочої** станції

Наступний скрипт дозволяє автоматизовано збирати інформацію з клієнтів чи серверів якщо вони мають доступ до мережі. Він виконує частину функцій попередніх скриптів(логування, перевірка процесів та перевірка прав) але не потребує участі адміністратора в процесі і відправляє зібрану інформацію на сервер. Для його налаштування потрібно додати “\* \* 1-31 \* /usr/bin/adminsh/admin-core2.sh | tee –a /var/log/adminsh/adminlog-`date +%F-%R`” до файлу cron аби він запускався кожен день, або ж змінити “\* \* 1-31 \*” на інше значення аби він запускався через інший проміжок часу. Також ротрібно згенерувати ключі ssh за допомогою команди “ssh-keygen -t rsa” і додати відкритий ключ на сервер для забезпечення доступу до нього. Таким чином даний скрипт дозволить відправляти логи на сервер та копіювати з нього новий файл конфігурації. Лістінг коду цього скрипта наведено в додатку Ж. Результат роботи даного скрипта наведено на наступному рисунку.

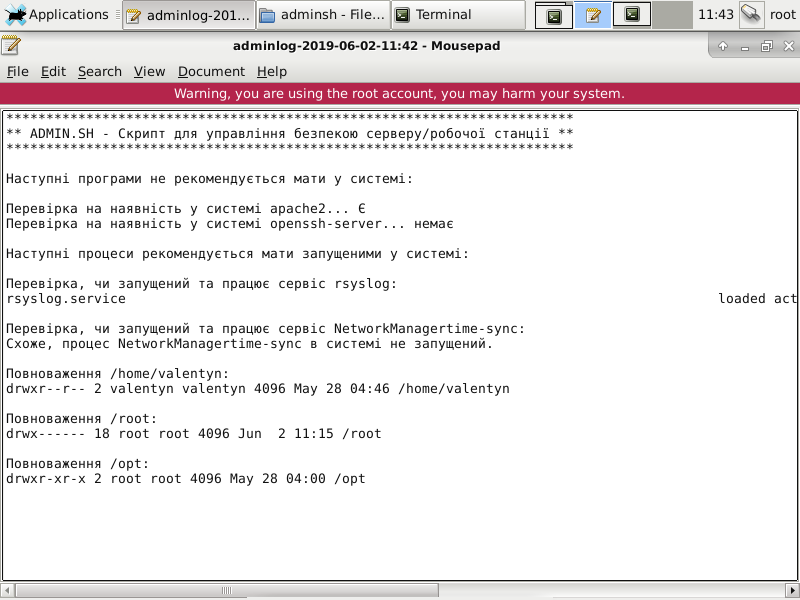


Рисунок 3.6 − Лог створений скриптом admin-core2.sh

### 3.4.3 Налаштування фаєрволу для забезпечення безпеки мережі

Для забезпечення бкзпеки мережі необхідно коректно налаштувати фаєрвол. Сервіси до яких можна отримати доступ ззовні можуть становити небезпеку для мережі і її компонентів, тому мінімізація сервісів є найвищим пріоритетом. Для доступу до веб сайтів та завантаження інформації з них рекомендується дозволити трафік з 80(HTTP) та 443(HTTPS) портів, також для роботи DNS серверів потрібно дозволити роботу на 53-у порту. Для забезпечення доступу по SSH ззовні необхідно дозволити роботу на порті 22. Отже, враховуючи всі ці рекомендації, мною було налаштовано фаєрвол, його налаштування показане на наступному рисунку.

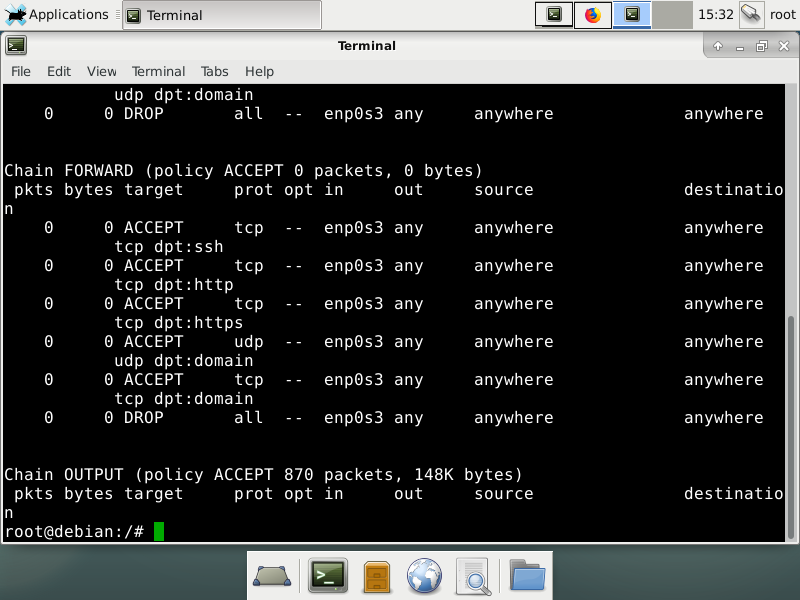


Рисунок 3.7 − Налаштування фаєрволу

## 3.5 Висновки за розділом 3

В даному розділі було розроблено основні рекомендаці для забезпечення захисту клієнт-серверних мереж на базі операційних систем з відкритим кодом.

Створено список принципів які рекомендується дотримуватись для забезпечення безпеки в мережі, на клієнтах, серверах та фізичному просторі навколо цих компонентів.

Також розроблено механізми захисту клієнт серверних пристроїв у вигляді скриптів, які дозволяють перевіряти права доступу на певні каталоги, запущені процеси та встановлені додатки, змінювати права доступу та запускати чи вимикати певні процеси, що допомагає дотримуватись певних рекомендацій наведених в даному розділі.

Також розроблено рекомендації щодо захисту мережі і налаштовано фаєрвол як механізм захисту мережі у відповідності до розроблених рекомендацій та списку принципів яких варто дотримуватись.

# ВИСНОВКИ

В даній дипломній роботі виконано наступні завдання:

* проаналізовано роботу, класифікацію та переваги клієнт-серверних мереж, надано опис двохрівневої та трирівневої клієнт-серверної архітектури, надано опис тонких, товстих та гібридних клієнтів та особливості їх функціонування
* описано та проаналізовано роботу операційних систем сімейства Unix, описано такі компоненти Unix як ядро, оболонка, файлова система, а також особливості функціонування, процеси та пристрої, також надано коротку характеристику та область використання для операційних систем на основі ядра Linux, що належать до сімейства Unix, зокрема найпоширенішого використання – LAMP-стеку
* виконано опис концепції безпеки Unix-подібних систем та її основних недоліків та проблем
* проаналізовано основні проблеми безпеки для клієнт-серверних мереж на базі операційних систем з відкритим кодом та властивості інформації, які порушуються при виникненні розглянутих проблем
* розроблено основні рекомендаці для забезпечення захисту клієнт-серверних мереж на базі операційних систем з відкритим кодом та створено список принципів, які рекомендується дотримуватись для забезпечення безпеки в мережі, на клієнтах, серверах та фізичному просторі навколо цих компонентів
* розроблено механізми захисту клієнт серверних пристроїв у вигляді скриптів, які дозволяють перевіряти права доступу на певні каталоги, запущені процеси та встановлені додатки, змінювати права доступу та запускати чи вимикати певні процеси, що допомагає дотримуватись певних рекомендацій розроблених в цій дипломній роботі
* розроблено рекомендації щодо захисту мережі і налаштовано фаєрвол як механізм захисту мережі у відповідності до розроблених рекомендацій та списку принципів яких варто дотримуватись.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Usage of operating systems for websites[Electronic resource]. – Access: https://w3techs.com/technologies/overview/operating\_system/all
2. Usage statistics and market share of Unix for websites [Electronic resource]. – Access: https://w3techs.com/technologies/details/os-unix/all/all
3. Client-server architecture [Electronic resource]. – Access: https://www.britannica.com/technology/client-server-architecture
4. IT Explained: Server [Electronic resource]. – Access: https://www.paessler.com/it-explained/server
5. UNIX and Linux System Administration Handbook, 5th Edition / Dan Mackin, Ben Whaley, Trent R. Hein, Garth Snyder, Evi Nemeth. – Addison-Wesley Professional, 2017. – P. 1500
6. Introduction to Unix and Shell Programming / M. G. Venkateshmurthy. – Pearson Education, 2009. – P. 392
7. A Fast File System for UNIX\* [Electronic resource]. – Access: https://people.eecs.berkeley.edu/~brewer/cs262/FFS.pdf
8. UNIX concepts and applications / Sumitabha Das. – Tata McGraw-Hill, 2006. – P. 671
9. Unix Systems Programming: Communication, Concurrency, and Threads / Steven Robbins, Kay A. Robbins. – Prentice Hall, 2003. – P. 893
10. UNIX Operating System: The Development Tutorial via UNIX Kernel Services / Yukun Liu, Yong Yue, Liwei Guo. – Springer Science & Business Media, 2011. – P. 368
11. How Linux Works, 2nd Edition / Brian Ward . – No Starch Press, 2014. – P. 392
12. 10 Top Most Popular Linux Distributions of 2016 [Electronic resource]. – Access: https://www.tecmint.com/top-best-linux-distributions-2016/
13. What is a LAMP stack? [Electronic resource]. – Access: https://www.liquidweb.com/kb/what-is-a-lamp-stack/
14. Закон України "Про iнформацiю" [Електронне джерело]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2657-12
15. Положення про технічний захист інформації в Україні [Електронне джерело]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1229/99
16. Закон України "Про основні засади забезпечення кібербезпеки України". [Електронне джерело]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2163-19
17. Конвенція про кіберзлочинність [Електронне джерело]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/994\_575
18. Practical UNIX and Internet Security, 3rd Edition. / Alan Schwartz, Gene Spafford, Simson Garfinkel. – O'Reilly Media, Inc., 2003. – P. 988
19. Buffer Overflows [Electronic resource]. – Access: https://www.owasp.org/index.php/Buffer\_Overflows
20. CWE-121: Stack-based Buffer Overflow [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/121.html
21. CWE-122: Heap-based Buffer Overflow [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/122.html
22. CWE-680: Integer Overflow to Buffer Overflow [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/680.html
23. Avoid Race Conditions [Electronic resource]. – Access: https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/avoid-race.html
24. CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/366.html
25. CWE-366: Race Condition within a Thread [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/362.html
26. Understanding Denial-of-Service Attacks [Electronic resource]. – Access: https://www.us-cert.gov/ncas/tips/ST04-015
27. CWE-400: Uncontrolled Resource Consumption [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/400.html
28. CWE-406: Insufficient Control of Network Message Volume (Network Amplification) [Electronic resource]. – Access: https://cwe.mitre.org/data/definitions/406.html
29. Захист ресурсів в корпоративних розподілених мережах / В.Є.Клочко, В.В.Романюк, І.І.Пархоменко. – 2 міжнародна науково-практична конференція “проблеми кібербезпеки інформаційно-телекомунікаційних систем” (PCSIТS) (11-12 квітня 2019)
30. Аналіз можливих загроз безпеці інформації в корпоративних мережах в середовищі операційних систем на основі ядра Linux / В.В.Романюк, В.Є.Клочко, І.І.Пархоменко – V Міжнародна науково-практична конференція «Інформаційні технології та взаємодії»(20-21 листопада 2018)

# ДОДАТКИ

# ДОДАТОК А

Лістінг коду скрипта admin.sh, що відповідає за логування

#!/bin/bash

clear

function keytocontinue {

echo -e "\nНатисніть будь-яку клавішу для продовження..."

read -n 1 -s

}

echo Лог буде записано до /var/log/adminsh/adminlog-`date +%F-%R`.

keytocontinue

echo Лог за `date +%c` > /var/log/adminsh/adminlog-`date +%F-%R`

echo -e "\n" > /var/log/adminsh/adminlog-`date +%F-%R`

clear

#Запуск функції

source /usr/bin/adminsh/admin-core.sh 2>&1 | tee -a /var/log/adminsh/adminlog-`date +%F-%R`

# ДОДАТОК Б

Лістінг коду скрипта admin-core.sh, що відповідає за перевірки певних процесів та програм.

#!/bin/bash

#Список небажаних програм

declare -a UNWANTED=(`cat /usr/bin/adminsh/unwanted\_list`)

#Список системних процесів на перевірку

declare -a PROC\_LIST=(`cat /usr/bin/adminsh/unwanted\_proc`)

#Початок роботи

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "\*\* ADMIN.SH - Скрипт для управління безпекою серверу/робочої станції \*\*"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo -e "\nНаступні програми не рекомендується мати у системі:\n"

#Перевірка небажаних програм

for ITER in "${UNWANTED[@]}"

do

if dpkg -s $ITER &> /dev/null;then

echo "Перевірка на наявність у системі" "$ITER"... Є

else

echo "Перевірка на наявність у системі" "$ITER"... немає

fi

done

#Перевірка бажаних процесів

echo -e "\nНаступні процеси рекомендується мати запущеними у системі:"

for ITER2 in "${PROC\_LIST[@]}"

do

echo -e "\nПеревірка, чи запущений та працює сервіс" "$ITER2"':'

if systemctl list-units | grep "$ITER2";then

echo a > /dev/null

else

echo "Схоже, процес $ITER2 в системі не запущений."

fi

done

keytocontinue

source /usr/bin/adminsh/admin-user.sh

# ДОДАТОК В

Лістінг коду скрипта admin-pac.sh, що відповідає за маніпуляції з пакунками та процесами

#!/bin/bash

#Вибір з чим працювати. Це меню фактично однакове як для пакунків, так і для процесів.

EXITMENUU=0

if [ $1 -eq 1 ];then

MENU1="програм"

MENU2="програму"

MENU3="програми"

MENU4="програми"

MENU5="3. Вийти.\n\nВаш вибір:"

PLIST="/usr/bin/adminsh/unwanted\_list"

elif [ $1 -eq 2 ];then

MENU1="процесів"

MENU2="процес"

MENU3="процесу"

MENU4="процеси"

MENU5="3. Вийти.\n4. Увімкнути чи вимкнути певний процес.\n\nВаш вибір:"

PLIST="/usr/bin/adminsh/unwanted\_proc"

else

#Означає, що скрипт був викликаний не з меню (або аргументи неправильні).

echo -e "Щось пішло не так! Аргумент неправильний.\nВийдіть за допомогою Ctrl-C."

fi

#Основне меню. Вилучення та додавання пунктів.

while [ $EXITMENUU -eq 0 ]

do

echo -e "\n\n\nСписок $MENU1, наявність яких перевіряється:"

cat $PLIST

echo -e "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "\*\* Що ви хочете надалі зробити? \*\*"

echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

echo "1. Видалити $MENU2 зі списку."

echo "2. Додати $MENU2 до списку."

echo -e $MENU5

read YOURCHOICE

case $YOURCHOICE in

'1')

cp $PLIST /usr/bin/adminsh/sed\_file

echo "Введіть назву $MENU3, що не треба більше перевіряти:"

read UNW

if grep -q $UNW /usr/bin/adminsh/sed\_file;then

echo -e "\nНадалі перевірятимуться лише:"

sed "s/$UNW//g" /usr/bin/adminsh/sed\_file > $PLIST

cat $PLIST

echo " "

echo "$UNW більше не перевірятиметься."

else

echo -e "\n$UNW не перевірявся."

fi

rm /usr/bin/adminsh/sed\_file

keytocontinue

;;

'2')

echo "Введіть назву $MENU3, що надалі перевірятиметься:"

read NEWPR

echo `cat $PLIST`$NEWPR" " > $PLIST

echo -e "\nТепер перевірятимуться такі $MENU4:"

cat $PLIST

keytocontinue

;;

'3')

EXITMENUU=1

;;

'4')

#Якщо користувач вибрав процеси, виникає меню, де він може зупиняти та запускати процеси.

if [ $1 -eq 1 ];then

echo -e "\nНа жаль, даний вибір недоступний."

keytocontinue

elif [ $1 -eq 2 ];then

EXITTHISMENU=0

#Меню для управління процесами

while [ $EXITTHISMENU -eq 0 ]

do

echo " "

declare -a PLIST2=(`cat $PLIST`)

for PROCX in "${PLIST2[@]}"

do

if systemctl list-units | grep $PROCX &> /dev/null;then

echo Процес $PROCX... запущений.

else

echo Процес $PROCX... не запущений.

fi

done

echo -e "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "\*\* Що ви хочете зробити? \*\*"

echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

echo "1. Запустити процес."

echo "2. Зупинити процес."

echo "3. Увімкнути процес."

echo "4. Вимкнути процес."

echo -e "5. Вийти.\n\nВаш вибір:"

read PROCCHOICE

case $PROCCHOICE in

'1')

echo "Введіть назву процесу:"

read PROCCHOICE

systemctl start $PROCCHOICE

;;

'2')

echo "Введіть назву процесу:"

read PROCCHOICE

systemctl stop $PROCCHOICE

;;

'3')

echo "Введіть назву процесу:"

read PROCCHOICE

systemctl enable $PROCCHOICE

;;

'4')

echo "Введіть назву процесу:"

read PROCCHOICE

systemctl disable $PROCCHOICE

;;

'5')

EXITTHISMENU=1

;;

\*)

echo "\nНа жаль, даний вибір недоступний."

keytocontinue

esac

done

else

echo -e "Помилка!"

keytocontinue

fi

;;

\*)

echo -e "\nНа жаль, даний вибір недоступний."

keytocontinue

;;

esac

done

# ДОДАТОК Д

Лістінг коду скрипта admin-chmod.sh, що відповідає за власність критичних директорій.

#!/bin/bash

echo -e "\n\n\n\n\n"

declare -a CHMODLIST=("/home/valentyn" "/root" "/opt")

for CHMODFOLDER in "${CHMODLIST[@]}"

do

echo -e "\nПовноваження $CHMODFOLDER:"

ls -ld $CHMODFOLDER

done

echo -e "\nЧи ви хочете змінити повноваження директорій? (yes/no)"

read CHMODCHOICE

case $CHMODCHOICE in

'yes'|'y'|'Y'|'так'|'т'|'Т')

echo -e "\nБудь ласка, введіть 3-значні коди для chmod.\nEnter для пропускання внесення змін.\n"

for CHMODFOLDER in "${CHMODLIST[@]}"

do

echo "Директорія $CHMODFOLDER:"

read CHMODREQUEST

case $CHMODREQUEST in

"")

echo -e "Зміни до $CHMODFOLDER не вноситимуться.\n"

;;

\*)

chmod -R $CHMODREQUEST $CHMODFOLDER

echo " "

;;

esac

done

;;

'no'|'n'|'N'|'ні'|'н'|'Н')

echo Зміни не будуть внесені.

;;

\*)

echo "OK."

;;

esac

echo -e "\nЧи ви хочете змінити власників директорій? (yes/no)"

read CHMODCHOICE

case $CHMODCHOICE in

'yes'|'y'|'Y'|'так'|'т'|'Т')

echo -e '\nБудь ласка, введіть користувача:групу, які володітимуть директоріями.\nEnter для невнесення змін.\n'

for CHOWNFOLDER in "${CHMODLIST[@]}"

do

echo "Директорія $CHOWNFOLDER:"

read CHOWNREQUEST

case $CHOWNREQUEST in

"")

echo -e "Зміни до $CHOWNFOLDER не вноситимуться.\n"

;;

\*)

chown -R $CHOWNREQUEST $CHOWNFOLDER

;;

esac

done

;;

'no'|'n'|'N'|'ні'|'н'|'Н')

echo Зміни не будуть внесені.

;;

\*)

echo OK.

;;

esac

# ДОДАТОК Е

Лістінг коду скрипта admin-user.sh, що відповідає за меню для подальших маніпуляцій зі зручним інтуїтивним інтерфейсом.

#!/bin/bash

EXITMENU=1

#Петля (loop), з якої можна вийти, лише якщо ввести 4. Для користувача - меню.

while [ $EXITMENU -eq 1 ]

do

echo -e "\n\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "\*\* Що ви хочете далі зробити? \*\*"

echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

echo "1. Переглянути перелік небажаних програм."

echo "2. Переглянути перелік бажаних процесів."

echo "3. Переглянути дозволи на критичні директорії."

echo "4. Вийти."

echo -e "\nВаш вибір:"

read CHOICEMENU

#У залежності від вибору виконується певний модуль.

case $CHOICEMENU in

'1'|'2')

source /usr/bin/adminsh/admin-pac.sh $CHOICEMENU

;;

'3')

source /usr/bin/adminsh/admin-chmod.sh

;;

'4')

EXITMENU=0

;;

\*)

echo -e "\nНа жаль, даний вибір недоступний"

keytocontinue

;;

esac

done

# ДОДАТОК Ж

Лістінг коду скрипта admin-core2.sh, що дозволяє автоматизовано збирати інформацію з клієнтів чи серверів якщо вони мають доступ до мережі.

#!/bin/bash

#Копіювання файлів конфігурації

scp -i /usr/bin/adminsh/id\_rsa loguser@192.168.0.3:/usr/bin/adminsh/\{ unwanted\_list, unwanted\_proc\}

#Список небажаних програм (беріть у лапки, розділяйте пробілами)

declare -a UNWANTED=(`cat /usr/bin/adminsh/unwanted\_list`)

#Список системних процесів на перевірку

declare -a PROC\_LIST=(`cat /usr/bin/adminsh/unwanted\_proc`)

#Початок, власне, роботи

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "\*\* ADMIN.SH - Скрипт для управління безпекою серверу/робочої станції \*\*"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo -e "\nНаступні програми не рекомендується мати у системі:\n"

#Перевірка небажаних програм

for ITER in "${UNWANTED[@]}"

do

if dpkg -s $ITER &> /dev/null;then

echo "Перевірка на наявність у системі" "$ITER"... Є

else

echo "Перевірка на наявність у системі" "$ITER"... немає

fi

done

#Перевірка бажаних процесів

echo -e "\nНаступні процеси рекомендується мати запущеними у системі:"

for ITER2 in "${PROC\_LIST[@]}"

do

echo -e "\nПеревірка, чи запущений та працює сервіс" "$ITER2"':'

if systemctl list-units | grep "$ITER2";then

echo a > /dev/null

else

echo "Схоже, процес $ITER2 в системі не запущений."

fi

done

declare -a CHMODLIST=("/home/valentyn" "/root" "/opt")

for CHMODFOLDER in "${CHMODLIST[@]}"

do

echo -e "\nПовноваження $CHMODFOLDER:"

ls -ld $CHMODFOLDER

done

scp -i /usr/bin/adminsh/id\_rsa /var/log/adminsh/adminlog-`date +%F` loguser@192.168.0.3:/logs/adminsh/host1-`date +%F`