Resolving Update Conflicts, Cache Issues, and Firewall Configurations for Seamless User Experience

Assignment Scenario - Ticket 1

You are a service desk employee for XYZ Corporation. You have received a ticket indicating that a recent update caused a conflict with an existing application. You are working on a fix for this problem, but in the meantime, you must undo the most recent update and temporarily pause automatic updates.

Task 1:

- Uninstall the most recent security updates for Microsoft Windows.
- Take a screenshot showing that a recent update is uninstalled, and that Windows is attempting to restart the computer to apply these changes.

Task 2:

- Temporarily pause automatic updates.
- Take a screenshot showing that updates have been temporarily paused on your device for up to 35 days.

Answer:

```
# Script: UndoRecentUpdateAndPauseUpdates.ps1
# Author: Service Desk, XYZ Corporation
# Description: This script uninstalls the most recent security updates for
Microsoft Windows
#              and temporarily pauses automatic updates.

# Ensure the script is run with administrative privileges
if (-not ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.
WindowsBuiltInRole] "Administrator")) {
    Write-Host "This script must be run as an administrator. Please restart the
script with elevated privileges." -ForegroundColor Red
    exit 1
}

Write-Host "Starting script to undo recent update and pause automatic updates..."
-ForegroundColor Cyan

# Task 1: Uninstall the most recent security updates for Microsoft Windows
```

```powershell
try {
    Write-Host "Retrieving a list of installed updates..." -ForegroundColor
Yellow
    $updates = Get-HotFix | Sort-Object InstalledOn -Descending | Where-Object {
$_.Description -eq "Security Update" }

    if ($updates.Count -eq 0) {
        Write-Host "No security updates found to uninstall." -ForegroundColor
Yellow
    } else {
        $mostRecentUpdate = $updates[0]
        Write-Host "Most recent security update found:
$($mostRecentUpdate.HotFixID) installed on $($mostRecentUpdate.InstalledOn)" -
ForegroundColor Green

        # Uninstall the most recent update
        Write-Host "Uninstalling the most recent security update:
$($mostRecentUpdate.HotFixID)..." -ForegroundColor Yellow
        wusa /uninstall /kb:$($mostRecentUpdate.HotFixID.Replace("KB", ""))
/quiet /norestart

        if ($LASTEXITCODE -eq 0) {
            Write-Host "Successfully uninstalled the most recent security
update." -ForegroundColor Green
        } else {
            Write-Host "Failed to uninstall the most recent security update. Exit
code: $LASTEXITCODE" -ForegroundColor Red
        }
    }
} catch {
    Write-Host "An error occurred while attempting to uninstall the most recent
security update: $_" -ForegroundColor Red
}

# Task 2: Temporarily pause automatic updates
try {
    Write-Host "Pausing automatic updates..." -ForegroundColor Yellow

    # Disable automatic updates via Windows Update settings
    $updateService = New-Object -ComObject Microsoft.Update.ServiceManager
    $updateSession = New-Object -ComObject Microsoft.Update.Session
    $updateSearcher = $updateSession.CreateUpdateSearcher()

    # Set the service to manual mode (paused)
    $updateService.Services | ForEach-Object {
```

```
        if ($_.IsDefaultAUService) {
            $_.IsDefaultAUService = $false
        }
    }

    # Configure Group Policy to pause updates (requires registry modification)
    $registryPath = "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU"
    if (-not (Test-Path $registryPath)) {
        New-Item -Path $registryPath -Force | Out-Null
    }

    # Set the registry values to pause updates
    Set-ItemProperty -Path $registryPath -Name "NoAutoUpdate" -Value 1 -Type
DWord
    Set-ItemProperty -Path $registryPath -Name "AUOptions" -Value 1 -Type DWord

    Write-Host "Automatic updates have been paused successfully." -
ForegroundColor Green
} catch {
    Write-Host "An error occurred while attempting to pause automatic updates:
$_" -ForegroundColor Red
}

Write-Host "Script execution completed." -ForegroundColor Cyan
```

Assignment Scenario - Ticket 2

A user is unable to access the most current version of a website. The issue may be related to the browser cache. Clear Chrome browsing data without losing the passwords and other sign-in data.

Task 3:

- Clear Chrome browsing history, cookies, site data, cached images, and files without losing the passwords and other sign-in data.
- Take a screenshot showing that you selected the appropriate checkboxes to clear the browsing history, cookies, other site data, cached images, and files. Your screenshot should show that you did not select the checkboxes to clear passwords and other sign-in data.

Answer:

```
import os
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.keys import Keys

def clear_chrome_browsing_data(chromedriver_path):
    try:
        # Set up Chrome options
        chrome_options = webdriver.ChromeOptions()
        chrome_options.add_argument("--start-maximized")  # Open Chrome in
maximized mode
        chrome_options.add_experimental_option("excludeSwitches", ["enable-
automation"])  # Disable automation message

        # Initialize Chrome WebDriver
        service = Service(chromedriver_path)
        driver = webdriver.Chrome(service=service, options=chrome_options)

        # Open Chrome's settings page
        driver.get("chrome://settings/clearBrowserData")
        time.sleep(3)  # Wait for the settings page to load

        # Use JavaScript to set the time range to "All time"
        driver.execute_script("document.querySelector('settings-
ui').shadowRoot.querySelector('settings-
main').shadowRoot.querySelector('settings-basic-
page').shadowRoot.querySelector('settings-section > settings-privacy-
page').shadowRoot.querySelector('settings-clear-browsing-data-
dialog').shadowRoot.querySelector('#clearFromBasic').value = 4;")

        # Uncheck "Passwords" and "Other sign-in data"
        driver.execute_script("""
            const checkboxes = document.querySelector('settings-
ui').shadowRoot.querySelector('settings-
main').shadowRoot.querySelector('settings-basic-
page').shadowRoot.querySelector('settings-section > settings-privacy-
page').shadowRoot.querySelector('settings-clear-browsing-data-
dialog').shadowRoot.querySelectorAll('input[type="checkbox"]');
            checkboxes.forEach(checkbox => {
                if (checkbox.parentElement.innerText.includes("Passwords") ||
checkbox.parentElement.innerText.includes("Other sign-in data")) {
                    checkbox.checked = false;
```

```
                } else {
                    checkbox.checked = true;
                }
            });
        """)

        # Click the "Clear data" button
        driver.execute_script("""
            document.querySelector('settings-
ui').shadowRoot.querySelector('settings-
main').shadowRoot.querySelector('settings-basic-
page').shadowRoot.querySelector('settings-section > settings-privacy-
page').shadowRoot.querySelector('settings-clear-browsing-data-
dialog').shadowRoot.querySelector('#clearBrowsingDataConfirm').click();
        """)
        time.sleep(5)  # Wait for the clearing process to complete

        print("Chrome browsing data cleared successfully without affecting
passwords and sign-in data.")

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        # Close the browser
        driver.quit()

if __name__ == "__main__":
    # Path to your ChromeDriver executable
    chromedriver_path = "/path/to/chromedriver"  # Replace with the actual path
to your ChromeDriver
    clear_chrome_browsing_data(chromedriver_path)
```

Assignment Scenario - Ticket 3

Because of personalized security controls, a user has requested only to permit the Chrome browser access to the internet. To accomplish this task, you will ensure that the workstation enables Windows Defender Firewall, and that Google Chrome can communicate through the firewall. You will also ensure that Firefox is blocked.

Task 4:

- Enable Windows Firewall.

- Take a screenshot showing that Windows Firewall is enabled.

Task 5:

- Verify that Google Chrome is allowed to communicate through Windows Defender Firewall.
- Take a screenshot showing that Google Chrome is allowed to communicate through Windows Firewall.

Task 6:

- Block Firefox from communicating through Windows Defender Firewall.
- Take a screenshot showing that Firefox cannot communicate through Windows Firewall.

Answer:

```
# Task 4: Enable Windows Defender Firewall
Write-Host "Enabling Windows Defender Firewall..." -ForegroundColor Yellow

# Enable the Windows Defender Firewall for all profiles (Domain, Private, Public)
Set-NetFirewallProfile -Profile Domain, Private, Public -Enabled True

Write-Host "Windows Defender Firewall has been enabled." -ForegroundColor Green

# Task 5: Verify that Google Chrome is allowed to communicate through Windows
Defender Firewall
Write-Host "Ensuring Google Chrome is allowed through Windows Defender
Firewall..." -ForegroundColor Yellow

# Define the path to Google Chrome executable
$chromePath = "C:\Program Files\Google\Chrome\Application\chrome.exe"

# Check if a rule for Google Chrome already exists
$chromeRule = Get-NetFirewallApplicationFilter | Where-Object { $_.Program -eq
$chromePath }

if (-not $chromeRule) {
    # If no rule exists, create a new rule to allow Google Chrome
    New-NetFirewallRule -DisplayName "Allow Google Chrome" -Direction Outbound -
Program $chromePath -Action Allow
    Write-Host "A new firewall rule has been created to allow Google Chrome." -
ForegroundColor Green
} else {
    # If a rule exists, ensure it is enabled
```

```
    Set-NetFirewallRule -DisplayName "Allow Google Chrome" -Enabled True
    Write-Host "The existing firewall rule for Google Chrome has been verified
and enabled." -ForegroundColor Green
}

# Task 6: Block Firefox from communicating through Windows Defender Firewall
Write-Host "Blocking Firefox from communicating through Windows Defender
Firewall..." -ForegroundColor Yellow

# Define the path to Firefox executable
$firefoxPath = "C:\Program Files\Mozilla Firefox\firefox.exe"

# Check if a rule for Firefox already exists
$firefoxRule = Get-NetFirewallApplicationFilter | Where-Object { $_.Program -eq
$firefoxPath }

if (-not $firefoxRule) {
    # If no rule exists, create a new rule to block Firefox
    New-NetFirewallRule -DisplayName "Block Firefox" -Direction Outbound -Program
$firefoxPath -Action Block
    Write-Host "A new firewall rule has been created to block Firefox." -
ForegroundColor Green
} else {
    # If a rule exists, ensure it is set to block
    Set-NetFirewallRule -DisplayName "Block Firefox" -Action Block -Enabled True
    Write-Host "The existing firewall rule for Firefox has been updated to block
communication." -ForegroundColor Green
}

Write-Host "All tasks have been completed successfully." -ForegroundColor Cyan
```