INTEGRITY LAB

NPGS3600CS

This Lab Activity will show users how to identify changes in files based on their hash value and will compare a production Site towards a golden Image to find out if changes were made.

Table of Contents

Definition of Integrity 2
What is PowerShell 2
Preparation 3
Activity 1: Listing items in a directory 4
Activity 2: Listing items in a directory ***recursively*** 5
Activity 3: Listing ***all*** items in a directory recursively 6
Activity 4: Getting the hash value of all items 7
Activity 5: Identify changes in a target directory 8
Script: dirComparsion.ps1 9
Activity 6: Identify possible malicious changes in a target directory 10
Appendix A: HTML Document – index.html 11
Appendix B: Cascading Style Sheet – style.css 12
Appendix C: JavaScript File – script.js 13

Integrity Lab

Definition of Integrity

"Data integrity is what the "I" in CIA Triad stands for. This is an essential component of the CIA Triad and designed to protect data from deletion or modification from any unauthorized party, and it ensures that when an authorized person makes a change that should not have been made the damage can be reversed." [1]

"In information security, data integrity means maintaining and assuring the accuracy and completeness of data over its entire lifecycle. This means that data cannot be modified in an unauthorized or undetected manner. This is not the same thing as referential integrity in databases, although it can be viewed as a special case of consistency as understood in the classic ACID model of transaction processing. Information security systems typically provide message integrity alongside confidentiality."[2]

What is PowerShell

"PowerShell is a task-based command-line shell and scripting language built on .NET. PowerShell helps system administrators and power-users rapidly automate tasks that manage operating systems (Linux, macOS, and Windows) and processes.

PowerShell commands let you manage computers from the command line. PowerShell providers let you access data stores, such as the registry and certificate store, as easily as you access the file system. PowerShell includes a rich expression parser and a fully developed scripting language." [3]

[1] https://www.forcepoint.com/cyber-edu/cia-triad
[2] https://en.wikipedia.org/wiki/Information_security#Integrity
[3] https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7

Preparation

Unzip the ***integrity_lab.zip*** archive into your Documents folder. If you don't have the zip file at the end of this document you will find the documents attached as appendixes.

The www_release folder will be used as your golden image, this concept will be explained later.

The www_release (folder) contains:

- css (folder)
  - **style.css**
- js (folder)
  - **script.js**
- **index.html**

Copy and Paste the contents of ***www_release*** folder into a new folder called ***wwwroot***.

The wwwroot (folder) should contain:

- css (folder)
  - **style.css**
- js (folder)
  - **script.js**
- **index.html**

Activity 1: Listing items in a directory

1. Open the command prompt. Use the ***Start*** button and type ***cmd***.
2. Type "powershell" and hit Enter.

Now that PowerShell is running:

1. Type "Set-Location $HOME\Documents\integrity_lab".

2. Type "Get-ChildItem", you should see both folders, ***wwwroot*** and ***www_release***.

3. List files in www_release, type "Get-ChildItem .\www_release\", what folders and/or files you get?
    a. name? __css_____, is a directory? __yes__
    b. name? __js _____, is a directory? __yes__
    c. name? __index.html_____, is a directory? __no__

4. List files in wwwroot, type "Get-ChildItem .\wwwroot\", what folders and/or files you get?
    a. name? __css_____, is a directory? __yes__
    b. name? __js _____, is a directory? __yes__
    c. name? __index.html_____, is a directory? __no__

5. Are the listed files ***all*** the files inside these folders? __no___

Activity 2: Listing items in a directory ***recursively***

If your answer to the last question of Activity 1 was ***No***, you are correct.

Not all files were included when the command Get-ChildItem was used.

Let's fix that:

1. Type "Get-ChildItem .\www_release\ -Recurse", what NEW folders and/or files you get?
    a. What is the name? __style.css_____, Is a directory? __no___, Where is it from? ___css_____

    b. What is the name? __script.js_____, Is a directory? __no___, Where is it from? ___js_____

2. Type "Get-ChildItem .\wwwroot\ -Recurse", what NEW folders and/or files you get?
    a. What is the name? __style.css_____, Is a directory? __no___, Where is it from? ___css_____

    b. What is the name? __script.js_____, Is a directory? __no___, Where is it from? ___js_____

3. Now, are the listed files ***all*** the files inside these folders? __no___

Activity 3: Listing ***all*** items in a directory recursively

If your answer to the last question of Activity 2 was ***Yes***, under normal circumstances, you would be correct, this is not the case.

1. Let's do a test, using the file explorer go to wwwroot.

2. Right-click, select **New**, select **Text Document**, type **newFile**.txt. You don't have to add content to the newly created file for the test.

3. Right-click newFile.txt, select **Properties**, in Attributes **mark Hidden**, click **Apply** and **Ok**.

4. Go back to the PowerShell prompt, type "Set-Location .\wwwroot\"

5. Let's check our directory, type "Get-ChildItem -Recurse", do you see the newly created and hidden file listed? __no__

6. Let's try it again, this time with a new option included. Type "Get-ChildItem -Recurse -Force". Was the command able to list the newly added and hidden file this time around? __yes____

7. Now, finally, are the listed files ***all*** the files inside these folders? __yes__

Activity 4: Getting the hash value of all items

If your answer to the last question of Activity 3 was ***Yes***, now, you are correct.

Let's make sure we are in the right place.

1. Type "Set-Location $HOME\Documents\integrity_lab".

It is possible to use the Get-ChildItem command using a specified path.

2. Type "Get-ChildItem -Force -Recurse -Path .\www_release\".

   Did you get the files? __yes__

3. Type "Get-ChildItem -Force -Recurse -Path .\wwwroot\".

   Did you get the files? __yes__

Get the hash value of all files per directory, the following command *ForEach-Object* will apply the functions enclosed in {} to all the objects returned by the pipe "|" command. In this case, all files recursively retrieved from the specified path are going to get their hash value calculated.

4. Type "Get-ChildItem -Force -Recurse -Path .\www_release\ | ForEach-Object { Get-FileHash -Path $_.FullName }"

   Did you get the filenames and file hashes? __yes___

5. Which was the default hashing algorithm used? __SHA256____

Activity 5: Identify changes in a target directory

We know hash values help us identify when changes have occurred, this is possible because a change will result in a completely different hash value regardless of changes being minimum.

We previously saw the term *golden Image*, this term is used to identify a known-good state of a directory. It is used to compare it towards a production directory that could have been modified.

Let's make sure we are in the right place.

1. Type "Set-Location $HOME\Documents\integrity_lab".

In order to compare two directories, we will employ the Compare-Object function.

1. Type "Compare-Object .\www_release\ .\wwwroot\ | Where-Object{$_.SideIndicator -eq '=>'}"

   Compare-Object (function)
   -ReferenceObject Location1 (golden Image)
   -DifferenceObject Location2 (target Directory)

   The resulting objects of this command are then piped "|" to the function Where-Object and the ***Side Indicator*** is verified to be "**=>**" meaning ***modified or new***.

   In the next page is the completed script that can be used to identify changes in a directory based in a golden Image comparison.

Script: dirComparsion.ps1

```
<#
.SYNOPSIS
Find new or changed files in a directory compared to a known-good image.

.DESCRIPTION
The script looks for file changes/additions between a production directory (target) with a known-good directory.

.PARAMETER knownGood
Path of the known-good directory.

.PARAMETER productionImage
Path of the production directory (target).

.INPUTS
System.String

.OUTPUTS
System.String

.EXAMPLE
.\dirComparsion.ps1 -knownGood <PATH> -productionImage <PATH>
.\dirComparsion.ps1 -knownGood .\knownGoodDir\ -productionImage .\targetDir\
.\dirComparsion.ps1 -knownGood "D:\release3.0" -productionImage "C:\inetpub\wwwroot"

-- Input --
.\dirComparsion.ps1 -knownGood "D:\Users\<user>\Documents\knownGoodDir" -
productionImage "C:\Users\<user>\Documents\targetDir"

-- Output --
File analysis started.
Any file listed below is a new or changed file.

C:\Users\<user>\Documents\targetDir\index.html
C:\Users\<user>\Documents\targetDir\research.docx
C:\Users\<user>\Documents\targetDir\inventory.csv
C:\Users\<user>\Documents\targetDir\contactus.js

File analysis completed.

#>

<#
#
# Execution begins.
#
#>
param (
    [Parameter(Mandatory = $TRUE)][ValidateScript( { Test-Path $_ -PathType 'Container' })][String] $knownGood,
    [Parameter(Mandatory = $TRUE)][ValidateScript( { Test-Path $_ -PathType 'Container' })][String] $productionImage
)

# Recursively get all files in both directories, for each file calculate hash.
$good = Get-ChildItem -Force -Recurse -Path $knownGood | ForEach-Object { Get-FileHash -Path $_.FullName }
$prod = Get-ChildItem -Force -Recurse -Path $productionImage | ForEach-Object { Get-FileHash -Path $_.FullName }

Write-Host "File analysis started." -ForegroundColor Green
Write-Host "Any file listed below is a new or changed file.`n" -ForegroundColor Green

# Compare files hashes, select new or changed files, and print the path+filename.
(Compare-Object $good $prod -Property hash -PassThru | Where-Object { $_.SideIndicator -eq '=>' }).Path

Write-Host "`nFile analysis completed." -ForegroundColor Green
```

Activity 6: Identify possible malicious changes in a target directory

Copy and Paste the script above into a new file and call the file "dirComparison.ps1" that is the extension of PowerShell scripts.

To execute the script and find changed or new files in our target directory do the following.

1. Type ".\dirComparison.ps1 -knownGood .\www_release\ -productionImage .\wwwroot\".

    a. Did you get the files that are changed or new based on our golden Image? __yes_____

    b. What files were identified?

    newFile.txt

Appendix A: HTML Document – index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>My App</title>

    <link rel="stylesheet" href="css/style.css">

    <script async type="text/javascript" src="js/script.js"></script>
</head>
<body>
    <h1 id="myText"></h1>
</body>
</html>
```

Appendix B: Cascading Style Sheet – style.css

```
body {
    background-color: black;
    color: green;
}
```

Appendix C: JavaScript File – script.js

```
var d = new Date();
var date = d.getMonth() + "/" + d.getDay() + "/" +d.getFullYear();
var todayString = "Today is " + date + "\n";
window.document.getElementById("myText").innerText = todayString.repeat(10);
```