Obfuscated Files or Information

What is Obfuscation?

Obfuscation is defined as the action of making something obscure, unclear, or unintelligible. [1] In computing, obfuscation is used to create code or sets of instructions that are difficult for a human to understand and therefore make the process of reverse engineering more difficult.

Malicious actors can employ such techniques in the hopes of avoiding detection or if they are detected, to delay the security team from discovering what they are doing. It may take some time to de-obfuscate code or commands and this extra time may help an attacker further their goals.

Obfuscated Files or Information Exploitation

According to the *Red Canary 2022 Threat Detection Report* [2], the technique of using *Obfuscated Files or Information* was ranked 8th, as one of the most exploited techniques observed in 2021. Red Canary observed this technique being used in **19.9%** of organizations.

What MITRE ATTACK [3] framework technique ID is applied to Obfuscated Files or Information Exploitation?

- The technique ID assigned to *Obfuscated Files or Information* is **T1027**.

What type of Tactic uses this technique?

Provide a name and a brief description of the Tactic that this technique falls under.

- Defense Evasion

> Defense Evasion consists of techniques that adversaries use to avoid detection throughout their compromise. Techniques used for defense evasion include uninstalling/disabling security software or obfuscating/encrypting data and scripts. Adversaries also leverage and abuse trusted processes to hide and masquerade their malware. Other tactics' techniques are cross-listed here when those techniques include the added benefit of subverting defenses.
>
> **- MITRE ATTACK Framework: Defense Evasion** [4]

- Execution

> Execution consists of techniques that result in adversary-controlled code running on a local or remote system. Techniques that run malicious code are often paired with techniques from all other tactics to achieve broader goals, like exploring a network or stealing data. For example, an adversary might use a remote access tool to run a PowerShell script that does Remote System Discovery.
>
> **- MITRE ATTACK Framework: Execution** [5]

- Initial Access

> Initial Access consists of techniques that use various entry vectors to gain their initial foothold within a network. Techniques used to gain a foothold include targeted spear phishing and exploiting weaknesses on public-facing web servers. Footholds gained through initial access may allow for continued access, like valid accounts and use of external remote services, or may be limited-use due to changing passwords.
>
> **- MITRE ATTACK Framework: Initial Access** [6]

Obfuscation is primarily used as part of the the *Defense Evasion* tactic. An adversary may encode commands to avoid detection. They may also use forms of *Obfuscation* to hide malicious code during the *Initial Access* stage of an attack, an example would be the deployment of malware via a 'Supply Chain Attack'.

Obfuscated Files or Information Techniques & Sub-Techniques

The *Obfuscated Files or Information* technique has 6 sub-techniques. They are listed as follows:

- T1027.001: Binary Padding
- T1027.002: Software Packing
- T1027.003: Steganography
- T1027.004: Compile After Delivery
- T1027.005: Indicator Removal from Tools
- T1027.006: HTML Smuggling

Due to the complexity of some of the techniques, we will focus on learning about *Obfuscation* and some basic *Obfuscation* techniques.

Why do malicious actors use Obfuscated Files or Information?

A malicious actor will employ *Obfuscation* techniques in order to evade detection by security tools and analysts. Obfuscation is also regularly used within an enterprise environment, so the use of *Obfuscation* may blend in with normal business activity. [7]

What can Malicious Actors use Obfuscated Files or Information for?

Malicious actors may employ many *Obfuscation* techniques during an attack, from the use of:

- Encryption to prevent their malicious code from being detected.
- Compression to hide a files true size.
- Encoding to hide plaintext instructions.

A common area where *Obfuscation* techniques may be visible is with command-line logging for *PowerShell* or *Command Prompt*. According to Red Canary [7-1] , some of the most common techniques visible at the command-line are:

Base64 Encoding

> Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation.
>
> **- MDN Web Docs** [8]

Base64 encoding was the most common form of obfuscation detected in 2021 by Red Canary. It is used most often in conjunction with the *T1059.001: PowerShell* technique to obfuscate commands.

Plaintext: `Institute of Technology, Carlow`

Base64: `SW5zdGl0dXRlIG9mIFRlY2hub2xvZ3ksIENhcmxvdw==`

String Concatenation

> Concatenation is the operation of joining two strings together. It is also known as string concatenation.
>
> **- Techopedia** [9]

You may be familiar with the term Concatenation from your programming modules and these techniques are employed to avoid detection, and to combine a series of Strings into one command.

Some common types that have been observed are:

- The `+` operator combining string values.
- The `-join` operator combining characters, strings, bytes, and other elements.
- PowerShell has access to .NET methods and it can use the `[System.String]::Join()` method to combine characters.
- String interpolation enables another form of evasion by allowing adversaries to set values such that `u\` can equal `util.exe`, thereby allowing `cert%u%` to execute `certutil.exe`

Substrings

> A string contained within a larger string; a portion of a string that is itself a string.
>
> **- Lexico** [10]

This form of *Obfuscation* is the 2nd most common form detected by Red Canary in 2021.

Their example demonstrates how an adversary may avoid detection and execute a command using *Substrings*.

Take the following String:

```
$ENV:pubLic[13]+$env:PublIc[5]+’x’
```

The environmental variable `public` refers to:

```
C:\Users\Public
```

So, to break it down, we are to *concatenate* the 13th character, with the 5th character and the character 'x'.

This returns `iex`, shorthand for the *PowerShell* cmdlet called `Invoke-Expression`.

> The `Invoke-Expression` cmdlet evaluates or runs a specified string as a command and returns the results of the expression or command. Without `Invoke-Expression`, a string submitted at the command line is returned (echoed) unchanged.
>
> **- Microsoft PowerShell Documentation** [11]

Escape Characters (Escape Sequences)

> Escape sequences represent non-printable and special characters in character and literal strings. As such, they allow users to communicate with a display device or printer by sending non-graphical control characters to specify actions like question marks and carriage returns.
>
> **- Techopedia** [12]

The command shell of an Operating System will have escape characters built in for when a user may want to pass the character to the command shell, and not have it interpreted.

PowerShell and the Windows Command Shell can escape characters with the following characters:

- Backtick (`` ` ``)
- Backslash (`\`)
- Caret (`^`)

A malicious actor may escape characters in order to avoid detection on signature matches for Strings.

Can you name any significant Groups that leverage Obfuscated Files or Information for malicious activity?

> Groups are sets of related intrusion activity that are tracked by a common name in the security community. Analysts track clusters of activities using various analytic methodologies and terms such as threat groups, activity groups, threat actors, intrusion sets, and campaigns. Some groups have multiple names associated with similar activities due to various organizations tracking similar activities by different names. Organizations' group definitions may partially overlap with groups designated by other organizations and may disagree on specific activity.
>
> **- MITRE ATTACK Framework: Groups** [13]

This technique has been leveraged by some large cybercrime organizations, state actors and in significant breaches over the past number of years.

Please provide the groups name, a brief description of the group and the exploit used.

Group	Description	Exploit Used
APT 37	APT37 is a North Korean state-sponsored cyber espionage group that has been active since at least 2012.	APT37 obfuscates strings and payloads.
Cobalt Group	Cobalt Group is a financially motivated threat group that has primarily targeted financial institutions since at least 2016.	Cobalt Group obfuscated several scriptlets and code used on the victim’s machine, including through use of XOR and RC4.
Leviathan	Leviathan is a Chinese state-sponsored cyber espionage group that has been attributed to the Ministry of State Security's (MSS) Hainan State Security Department and an affiliated front company.	Leviathan has obfuscated code using base64 and gzip compression.
QakBot	QakBot is a modular banking trojan that has been used primarily by financially-motivated actors since at least 2007.	QakBot can use obfuscated and encoded scripts; it has also hidden code within Excel spreadsheets by turning the font color to white and splitting it across multiple cells.

What can you do to mitigate against Obfuscated Files or Information exploitation?

Please research mitigations and provide the type and a short description of the mitigation techniques.

ID	Mitigation	Description
M1049	Antivirus/Antimalware	Consider utilizing the Antimalware Scan Interface (AMSI) on Windows 10 to analyze commands after being processed/interpreted.
M1040	Behavior Prevention on Endpoint	On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent execution of potentially obfuscated scripts.

How can this type of attack be detected?

Detecting *Obfuscation* can be a difficult task due to the large number of ways that a file or information can be obfuscated.

- Monitor command-line arguments that have suspicious syntax, such as the use of many escape characters.
- Using an IDS and email filtering to identify compressed and encrypted files.
- Detonate attachments in a sandbox environment, such as CrowdStrike to analyse the file.
- Monitor for the creation of files on a system, an attacker may create obfuscated files.

ID	Data Source	Data Component
DS0017	Command	Command Execution
DS0022	File	File Creation
		File Metadata
DS0009	Process	Process Creation

Performing regular compromise assessments within an environment is also very beneficial to the organization and can also help with detecting threats, both past and present.

> Compromise assessments are high-level investigations where skilled teams utilize advanced tools to dig more deeply into their environment to identify ongoing or past attacker activity in addition to identifying existing weaknesses in controls and practices.
>
> **- CrowdStrike** [14]

These tests are usually performed by vulnerability scanners, and will assess the company's infrastructure. The scans will usually incorporate searching for known *Indicators of Compromise* (IOC) from recently investigated attacks.

> An Indicator of Compromise (IOC) is a piece of digital forensics that suggests that an endpoint or network may have been breached. Just as with physical evidence, these digital clues help information security professionals identify malicious activity or security threats, such as data breaches, insider threats or malware attacks.
>
> **- CrowdStrike** [15]

Indicators of Compromise includes:

- Files Hashes
- IP Addresses
- Sign in Activity from unexpected countries.
- Large volumes of sign in requests.

Log Collection

Listed below are log events to track:

- Windows Security Event ID 4688: Process creation
- Sysmon Event ID 1: Process creation
- Windows Security Event ID 1101 Antimalware-Scan-Interface (AMSI)

Obfuscated Files or Information Demonstration

In this section, we will demonstrate some of the techniques that can be performed with *Obfuscation* and then to view the logs to get an idea for what you should look for.

To help with this section, please open the GitHub link for the *Atomic Red Team* atomics page for the *Obfuscated Files or Information*.

- https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1027/T1027.md

T1027

From the *Atomic Red Team* Github for the technique *T1027: Obfuscated Files or Information* shows that there are 8 automatic tests built into the Atomic Red Team toolset.

It may not be possible to run all the tests, however we will run a couple so that you can view any relevant log information.

Step 1: Open Client Machine

- Open the Windows 10 machine connected to the Detection Lab configuration.
- Open PowerShell

Step 2: Confirm that Invoke-AtomicTest is Installed

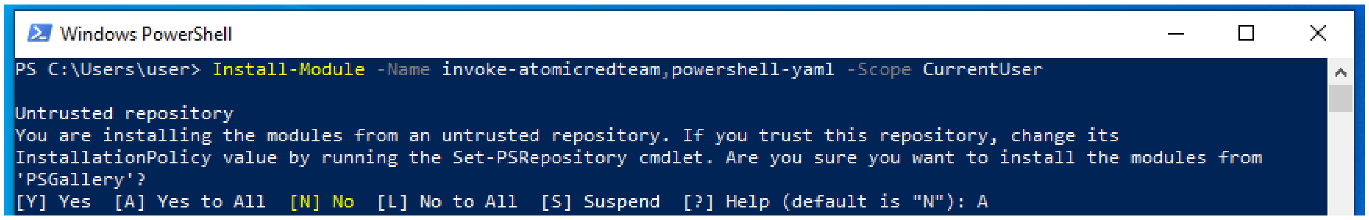
- Confirm that the `Invoke-AtomicTest` cmdlet is installed correctly. This command will install this module.

```
Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope CurrentUser
```

- Type `A` to confirm installing the Module.
- If the module is already installed, you will not be prompted to accept.

Further Reading about the installation process:

- https://github.com/redcanaryco/invoke-atomicredteam/wiki/Installing-Atomic-Red-Team



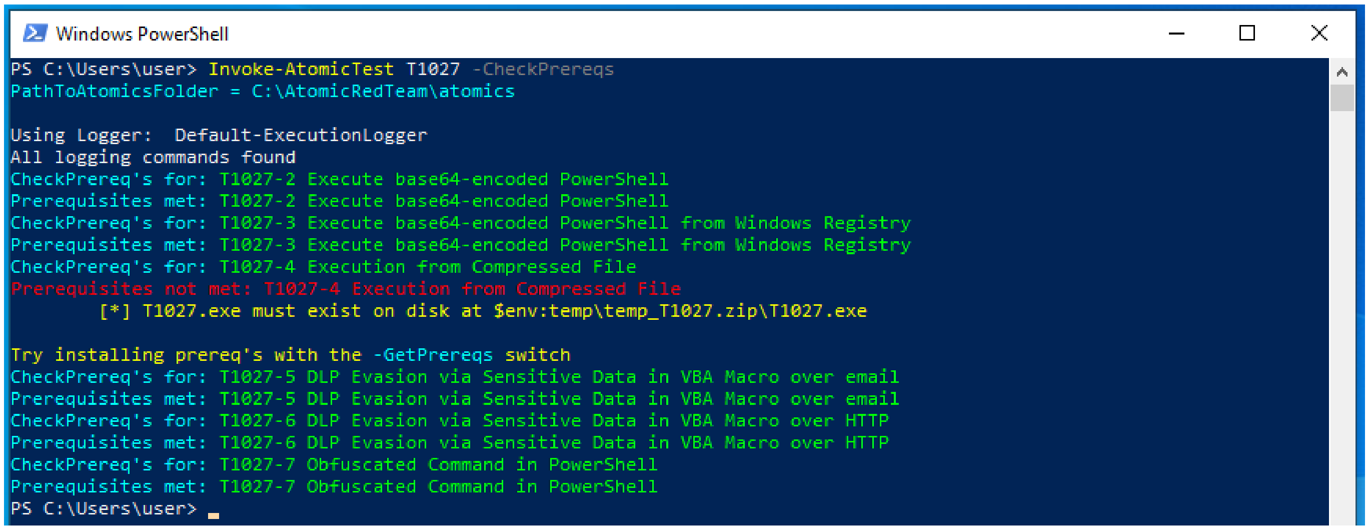
```
PS C:\Users\user> Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope CurrentUser

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): A
```

Step 3: Check the Prerequisites for T1027

- We need to confirm that all the prerequisites for the tests are available and installed correctly.

```
Invoke-AtomicTest T1027 -CheckPrereqs
```

- As we can see from the screenshot below, only one test does not have the required resources to complete.



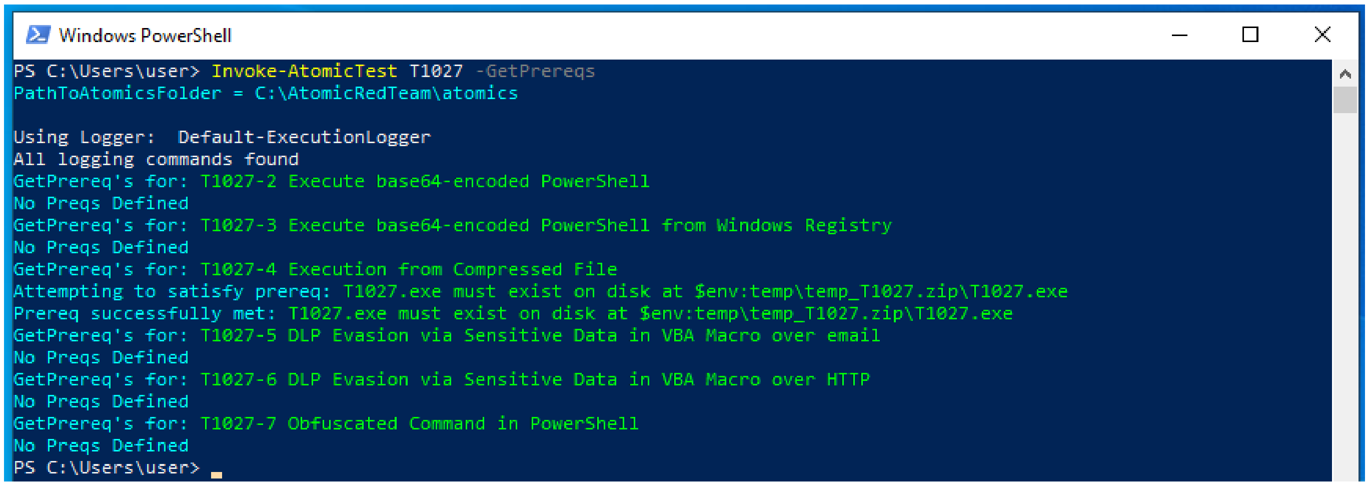
Windows PowerShell

```
PS C:\Users\user> Invoke-AtomicTest T1027 -CheckPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
CheckPrereq's for: T1027-2 Execute base64-encoded PowerShell
Prerequisites met: T1027-2 Execute base64-encoded PowerShell
CheckPrereq's for: T1027-3 Execute base64-encoded PowerShell from Windows Registry
Prerequisites met: T1027-3 Execute base64-encoded PowerShell from Windows Registry
CheckPrereq's for: T1027-4 Execution from Compressed File
Prerequisites not met: T1027-4 Execution from Compressed File
        [*] T1027.exe must exist on disk at $env:temp\temp_T1027.zip\T1027.exe

Try installing prereq's with the -GetPrereqs switch
CheckPrereq's for: T1027-5 DLP Evasion via Sensitive Data in VBA Macro over email
Prerequisites met: T1027-5 DLP Evasion via Sensitive Data in VBA Macro over email
CheckPrereq's for: T1027-6 DLP Evasion via Sensitive Data in VBA Macro over HTTP
Prerequisites met: T1027-6 DLP Evasion via Sensitive Data in VBA Macro over HTTP
CheckPrereq's for: T1027-7 Obfuscated Command in PowerShell
Prerequisites met: T1027-7 Obfuscated Command in PowerShell
PS C:\Users\user> _
```

Step 4: Get the Prerequisites for T1027

- Install the resources required to complete the relevant tests.

```
Invoke-AtomicTest T1027 -GetPrereqs
```



Windows PowerShell

```
PS C:\Users\user> Invoke-AtomicTest T1027 -GetPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
GetPrereq's for: T1027-2 Execute base64-encoded PowerShell
No Preqs Defined
GetPrereq's for: T1027-3 Execute base64-encoded PowerShell from Windows Registry
No Preqs Defined
GetPrereq's for: T1027-4 Execution from Compressed File
Attempting to satisfy prereq: T1027.exe must exist on disk at $env:temp\temp_T1027.zip\T1027.exe
Prereq successfully met: T1027.exe must exist on disk at $env:temp\temp_T1027.zip\T1027.exe
GetPrereq's for: T1027-5 DLP Evasion via Sensitive Data in VBA Macro over email
No Preqs Defined
GetPrereq's for: T1027-6 DLP Evasion via Sensitive Data in VBA Macro over HTTP
No Preqs Defined
GetPrereq's for: T1027-7 Obfuscated Command in PowerShell
No Preqs Defined
PS C:\Users\user> _
```

Step 5: Begin Testing

I will choose a select few tests to demonstrate the commands used to generate the logs. All the tests can be executed at once, however I prefer to do it test-by-test.

Some tests are designed for Linux or Mac. Ensure that you are attempting to demonstrate the Windows Tests.

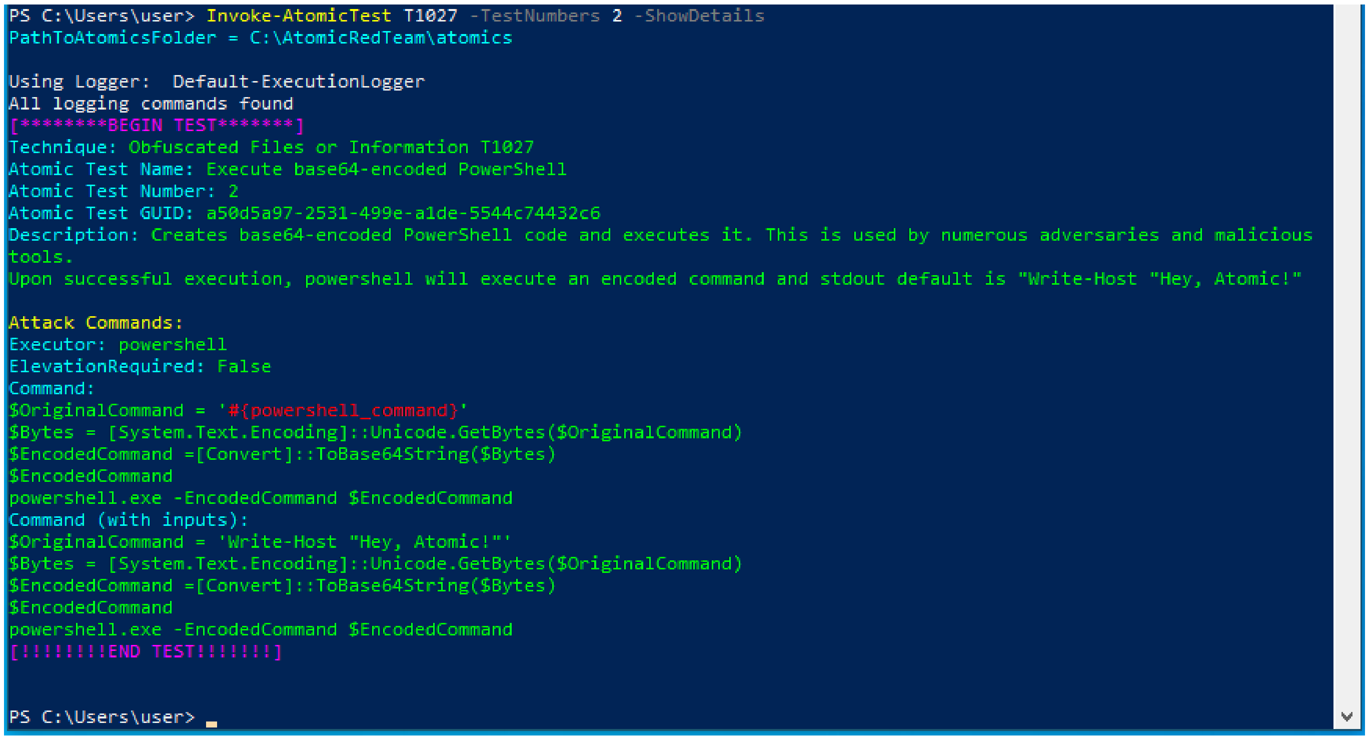
Test #2 - Execute base64-encoded PowerShell

This test shows how code may be encoded, in the hopes of avoiding detection. The code is then executed. Successful execution of this test should display 'Hey, Atomic!'.

Show Test Details

- Firstly, use the -ShowDetails switch to print the details of the specific test to the screen.

```
Invoke-AtomicTest T1027 -TestNumbers 2 -ShowDetails
```

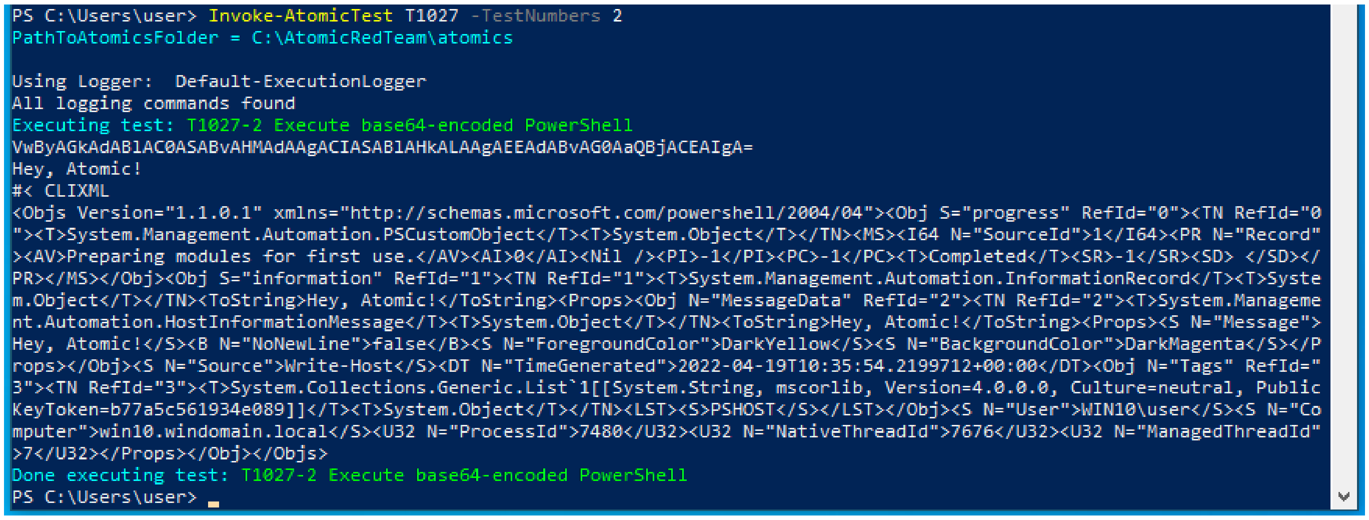


```
PS C:\Users\user> Invoke-AtomicTest T1027 -TestNumbers 2 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
[********BEGIN TEST*******]
Technique: Obfuscated Files or Information T1027
Atomic Test Name: Execute base64-encoded PowerShell
Atomic Test Number: 2
Atomic Test GUID: a50d5a97-2531-499e-a1de-5544c74432c6
Description: Creates base64-encoded PowerShell code and executes it. This is used by numerous adversaries and malicious
tools.
Upon successful execution, powershell will execute an encoded command and stdout default is "Write-Host "Hey, Atomic!"

Attack Commands:
Executor: powershell
ElevationRequired: False
Command:
$OriginalCommand = '#{powershell_command}'
$Bytes = [System.Text.Encoding]::Unicode.GetBytes($OriginalCommand)
$EncodedCommand =[Convert]::ToBase64String($Bytes)
$EncodedCommand
powershell.exe -EncodedCommand $EncodedCommand
Command (with inputs):
$OriginalCommand = 'Write-Host "Hey, Atomic!"'
$Bytes = [System.Text.Encoding]::Unicode.GetBytes($OriginalCommand)
$EncodedCommand =[Convert]::ToBase64String($Bytes)
$EncodedCommand
powershell.exe -EncodedCommand $EncodedCommand
[!!!!!!!!END TEST!!!!!!!]


PS C:\Users\user> _
```

Execute Test

- Next, we will run the test.

```
Invoke-AtomicTest T1027 -TestNumbers 2
```



```
PS C:\Users\user> Invoke-AtomicTest T1027 -TestNumbers 2
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
Executing test: T1027-2 Execute base64-encoded PowerShell
VwByAGkAdABlAC0ASABvAHMAdAAgACIASABlAHkALAAgAEEAdABvAG0AaQBjACEAIgA=
Hey, Atomic!
#< CLIXML
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><TN RefId="0
"><T>System.Management.Automation.PSCustomObject</T><T>System.Object</T></TN><MS><I64 N="SourceId">1</I64><PR N="Record"
><AV>Preparing modules for first use.</AV><AI>0</AI><Nil /><PI>-1</PI><PC>-1</PC><T>Completed</T><SR>-1</SR><SD> </SD></
PR></MS></Obj><Obj S="information" RefId="1"><TN RefId="1"><T>System.Management.Automation.InformationRecord</T><T>Syste
m.Object</T></TN><ToString>Hey, Atomic!</ToString><Props><Obj N="MessageData" RefId="2"><TN RefId="2"><T>System.Manageme
nt.Automation.HostInformationMessage</T><T>System.Object</T></TN><ToString>Hey, Atomic!</ToString><Props><S N="Message">
Hey, Atomic!</S><B N="NoNewLine">false</B><S N="ForegroundColor">DarkYellow</S><S N="BackgroundColor">DarkMagenta</S></P
rops></Obj><S N="Source">Write-Host</S><DT N="TimeGenerated">2022-04-19T10:35:54.2199712+00:00</DT><Obj N="Tags" RefId="
3"><TN RefId="3"><T>System.Collections.Generic.List`1[[System.String, mscorlib, Version=4.0.0.0, Culture=neutral, Public
KeyToken=b77a5c561934e089]]</T><T>System.Object</T></TN><LST><S>PSHOST</S></LST></Obj><S N="User">WIN10\user</S><S N="Co
mputer">win10.windomain.local</S><U32 N="ProcessId">7480</U32><U32 N="NativeThreadId">7676</U32><U32 N="ManagedThreadId"
>7</U32></Props></Obj></Objs>
Done executing test: T1027-2 Execute base64-encoded PowerShell
PS C:\Users\user> _
```

We can see from the testing, and the screenshot above, that testing was completed successfully.

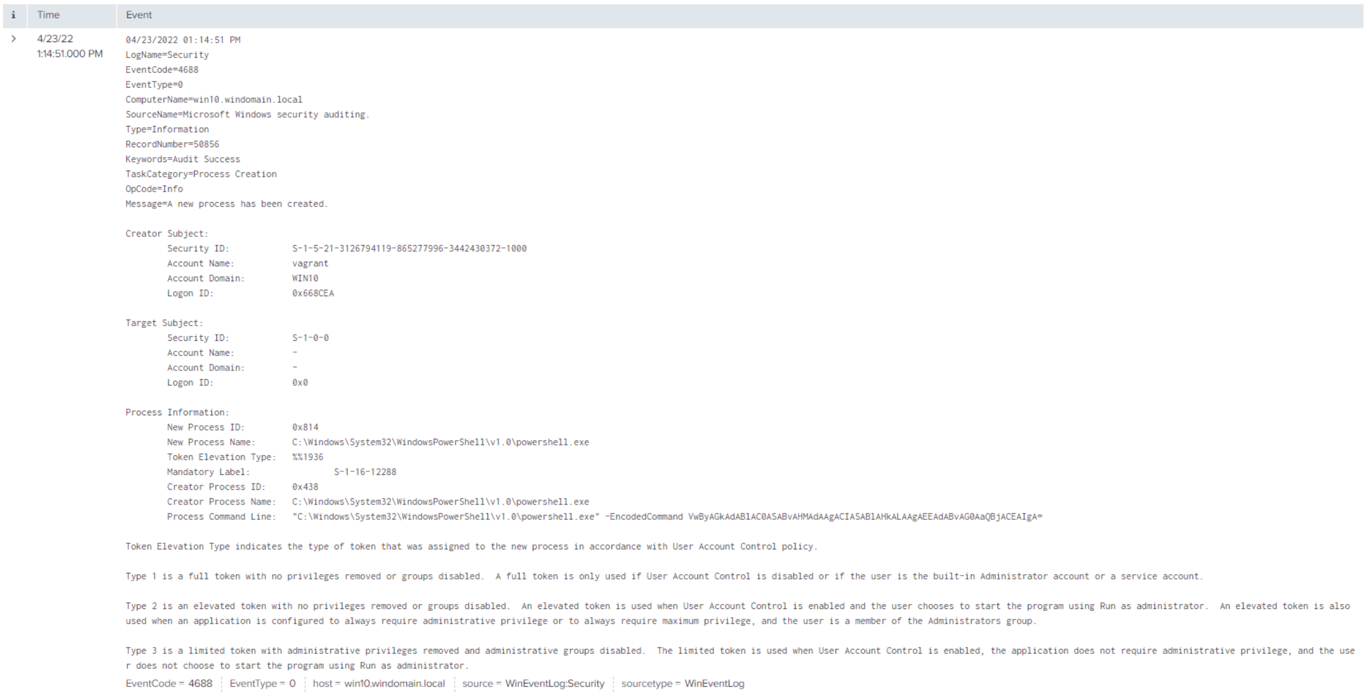
- We observe a base64 encoded string: `VwByAGkAdABlAC0ASABvAHMAdAAgACIASABlAHkALAAgAEEAdABvAG0AaQBjACEAIgA=`

- When decoded using CyberChef: `W.r.i.t.e.-.H.o.s.t. .".H.e.y.,. .A.t.o.m.i.c.!.".`
- The next line prints: `Hey, Atomic!`
    - The last line of the test calls the PowerShell executable, and is inputting the encoded message as a parameter.
    - The Encoded message instructs PowerShell to print `Hey, Atomic!`.

Logs

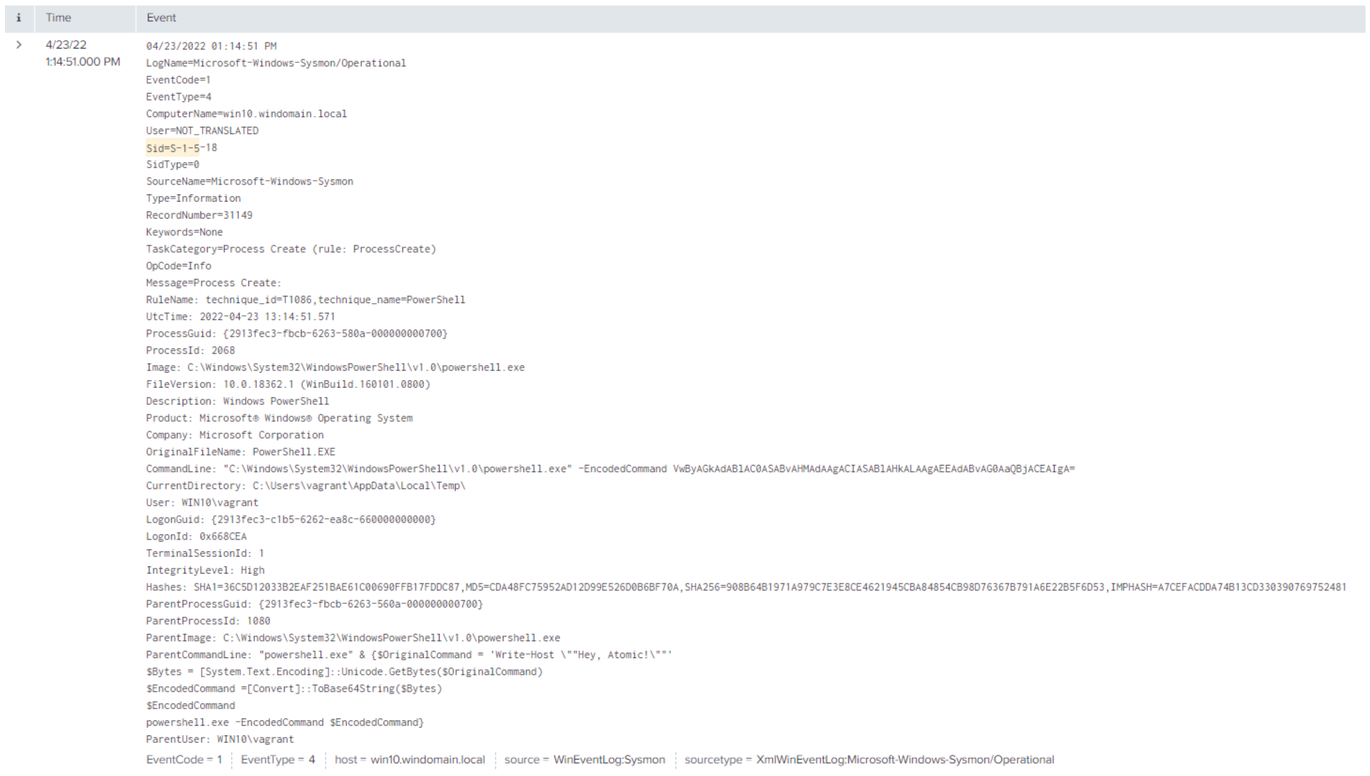
Next, open up the *Splunk - Search & Reporting* instance and begin searching for the log data surrounding the inputted commands.

- Windows Event Process Creation Event (4688): `index="wineventlog" process_command_line="\"C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe\" -EncodedCommand VwByAGkAdABlAC0ASABvAHMAdAAgACIASABlAHkALAAgAEEAdABvAG0AaQBjACEAIgA="`



```
i   Time               Event
>   4/23/22            04/23/2022 01:14:51 PM
    1:14:51.000 PM     LogName=Security
                       EventCode=4688
                       EventType=0
                       ComputerName=win10.windomain.local
                       SourceName=Microsoft Windows security auditing.
                       Type=Information
                       RecordNumber=50856
                       Keywords=Audit Success
                       TaskCategory=Process Creation
                       OpCode=Info
                       Message=A new process has been created.

                       Creator Subject:
                               Security ID:            S-1-5-21-3126794119-865277996-3442430372-1000
                               Account Name:           vagrant
                               Account Domain:         WIN10
                               Logon ID:               0x668CEA

                       Target Subject:
                               Security ID:            S-1-0-0
                               Account Name:           -
                               Account Domain:         -
                               Logon ID:               0x0

                       Process Information:
                               New Process ID:         0x814
                               New Process Name:       C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                               Token Elevation Type:   %%1936
                               Mandatory Label:                S-1-16-12288
                               Creator Process ID:     0x438
                               Creator Process Name:   C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                               Process Command Line:   "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -EncodedCommand VwByAGkAdABlAC0ASABvAHMAdAAgACIASABlAHkALAAgAEEAdABvAG0AaQBjACEAIgA=

                       Token Elevation Type indicates the type of token that was assigned to the new process in accordance with User Account Control policy.

                       Type 1 is a full token with no privileges removed or groups disabled.  A full token is only used if User Account Control is disabled or if the user is the built-in Administrator account or a service account.

                       Type 2 is an elevated token with no privileges removed or groups disabled.  An elevated token is used when User Account Control is enabled and the user chooses to start the program using Run as administrator.  An elevated token is also
                       used when an application is configured to always require administrative privilege or to always require maximum privilege, and the user is a member of the Administrators group.

                       Type 3 is a limited token with administrative privileges removed and administrative groups disabled.  The limited token is used when User Account Control is enabled, the application does not require administrative privilege, and the use
                       r does not choose to start the program using Run as administrator.
                       EventCode = 4688 | EventType = 0 | host = win10.windomain.local | source = WinEventLog:Security | sourcetype = WinEventLog
```

- Sysmon Process Creation Event: `index="sysmon" CommandLine="\"C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe\" -EncodedCommand VwByAGkAdABlAC0ASABvAHMAdAAgACIASABlAHkALAAgAEEAdABvAG0AaQBjACEAIgA="`



i	Time	Event

```
>  4/23/22           04/23/2022 01:14:51 PM
   1:14:51.000 PM    LogName=Microsoft-Windows-Sysmon/Operational
                     EventCode=1
                     EventType=4
                     ComputerName=win10.windomain.local
                     User=NOT_TRANSLATED
                     Sid=S-1-5-18
                     SidType=0
                     SourceName=Microsoft-Windows-Sysmon
                     Type=Information
                     RecordNumber=31149
                     Keywords=None
                     TaskCategory=Process Create (rule: ProcessCreate)
                     OpCode=Info
                     Message=Process Create:
                     RuleName: technique_id=T1086,technique_name=PowerShell
                     UtcTime: 2022-04-23 13:14:51.571
                     ProcessGuid: {2913fec3-fbcb-6263-580a-000000000700}
                     ProcessId: 2068
                     Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                     FileVersion: 10.0.18362.1 (WinBuild.160101.0800)
                     Description: Windows PowerShell
                     Product: Microsoft® Windows® Operating System
                     Company: Microsoft Corporation
                     OriginalFileName: PowerShell.EXE
                     CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -EncodedCommand VwByAGkAdABlAC0ASABvAHMAdAAgACIASABlAHkALAAgAEEAdABvAG0AaQBjACEAIgA=
                     CurrentDirectory: C:\Users\vagrant\AppData\Local\Temp\
                     User: WIN10\vagrant
                     LogonGuid: {2913fec3-c1b5-6262-ea8c-660000000000}
                     LogonId: 0x668CEA
                     TerminalSessionId: 1
                     IntegrityLevel: High
                     Hashes: SHA1=36C5D12033B2EAF251BAE61C00690FFB17FDDC87,MD5=CDA48FC75952AD12D99E526D0B6BF70A,SHA256=908B64B1971A979C7E3E8CE4621945CBA84854CB98D76367B791A6E22B5F6D53,IMPHASH=A7CEFACDDA74B13CD330390769752481
                     ParentProcessGuid: {2913fec3-fbcb-6263-560a-000000000700}
                     ParentProcessId: 1080
                     ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                     ParentCommandLine: "powershell.exe" & {$OriginalCommand = 'Write-Host \""Hey, Atomic!\""'
                     $Bytes = [System.Text.Encoding]::Unicode.GetBytes($OriginalCommand)
                     $EncodedCommand =[Convert]::ToBase64String($Bytes)
                     $EncodedCommand
                     powershell.exe -EncodedCommand $EncodedCommand}
                     ParentUser: WIN10\vagrant
                     EventCode = 1 | EventType = 4 | host = win10.windomain.local | source = WinEventLog:Sysmon | sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
```

What information do you think may be relevant to determine what occurred on the device?

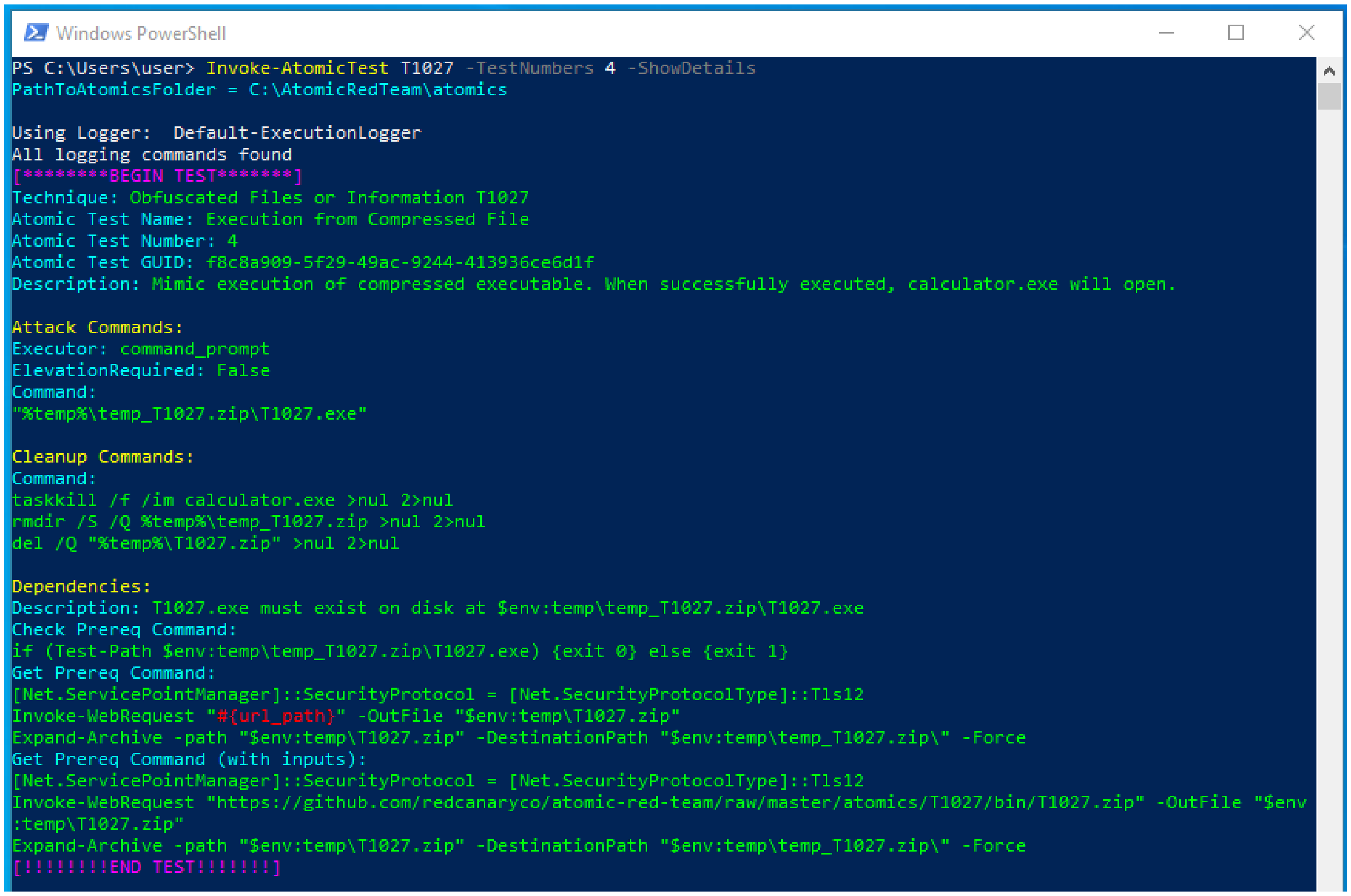
- Use the fields on the left of the Splunk Search to help filter the search results.
- Some of the relevant fields will be as follows:
  - Host
  - Command Line
  - Process Name
  - Parent Process
  - Process Paths
  - IP Addresses
  - File Hash

Test #4 - Execution from Compressed File

The purpose of this test is to demonstrate how an adversary may run an executable from a compressed folder. The test itself should launch `calc.exe`.

- Use the -ShowDetails switch to print the details of the specific test to the screen.

```
Invoke-AtomicTest T1027 -TestNumbers 4 -ShowDetails
```

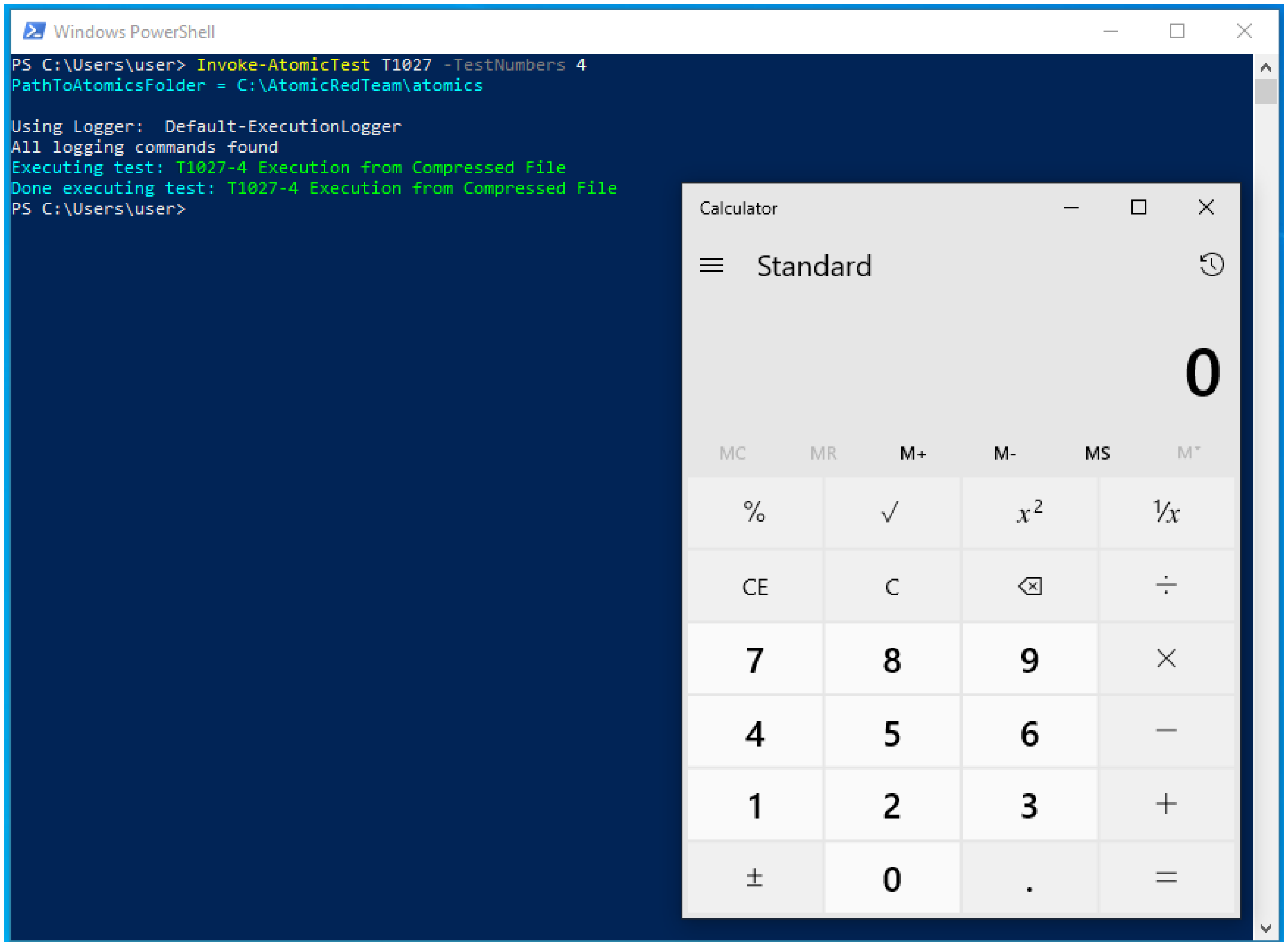


Windows PowerShell

```
PS C:\Users\user> Invoke-AtomicTest T1027 -TestNumbers 4 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
[********BEGIN TEST*******]
Technique: Obfuscated Files or Information T1027
Atomic Test Name: Execution from Compressed File
Atomic Test Number: 4
Atomic Test GUID: f8c8a909-5f29-49ac-9244-413936ce6d1f
Description: Mimic execution of compressed executable. When successfully executed, calculator.exe will open.

Attack Commands:
Executor: command_prompt
ElevationRequired: False
Command:
"%temp%\temp_T1027.zip\T1027.exe"

Cleanup Commands:
Command:
taskkill /f /im calculator.exe >nul 2>nul
rmdir /S /Q %temp%\temp_T1027.zip >nul 2>nul
del /Q "%temp%\T1027.zip" >nul 2>nul

Dependencies:
Description: T1027.exe must exist on disk at $env:temp\temp_T1027.zip\T1027.exe
Check Prereq Command:
if (Test-Path $env:temp\temp_T1027.zip\T1027.exe) {exit 0} else {exit 1}
Get Prereq Command:
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
Invoke-WebRequest "#{url_path}" -OutFile "$env:temp\T1027.zip"
Expand-Archive -path "$env:temp\T1027.zip" -DestinationPath "$env:temp\temp_T1027.zip\" -Force
Get Prereq Command (with inputs):
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
Invoke-WebRequest "https://github.com/redcanaryco/atomic-red-team/raw/master/atomics/T1027/bin/T1027.zip" -OutFile "$env
:temp\T1027.zip"
Expand-Archive -path "$env:temp\T1027.zip" -DestinationPath "$env:temp\temp_T1027.zip\" -Force
[!!!!!!!!END TEST!!!!!!!]
```

- Run the test

```
Invoke-AtomicTest T1027 -TestNumbers 4
```

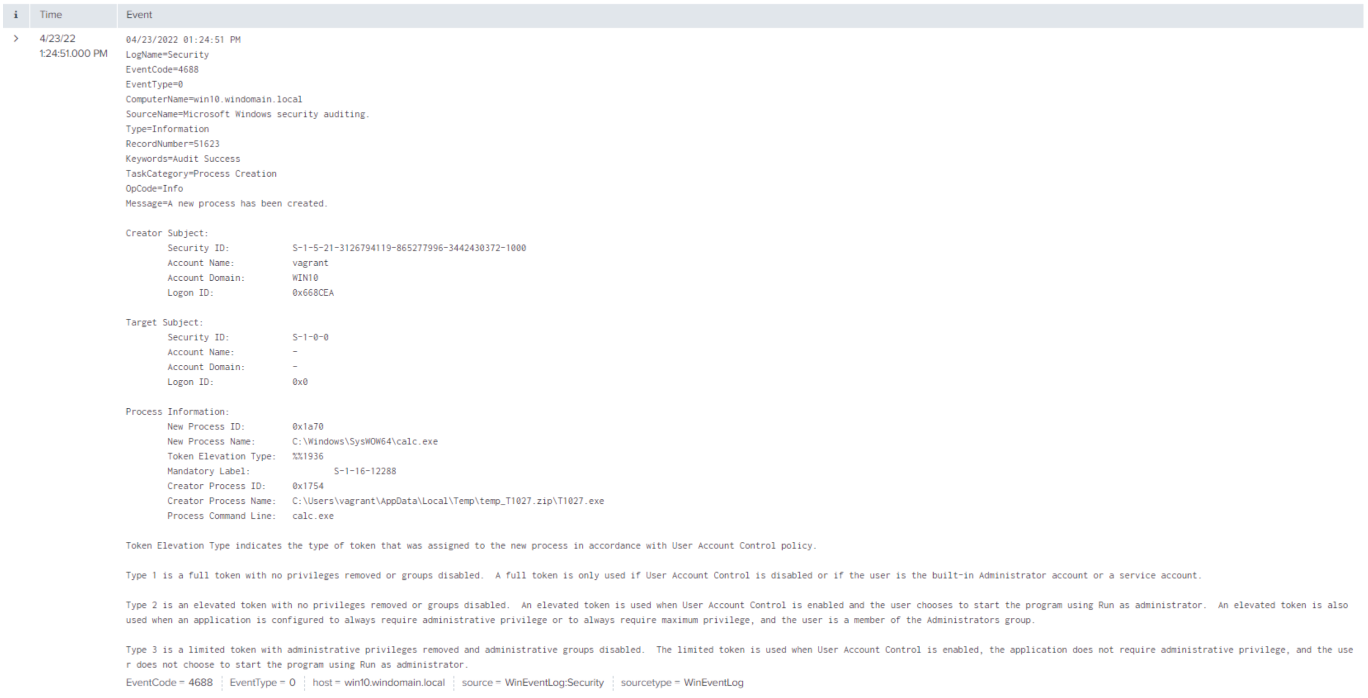


```
PS C:\Users\user> Invoke-AtomicTest T1027 -TestNumbers 4
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
Executing test: T1027-4 Execution from Compressed File
Done executing test: T1027-4 Execution from Compressed File
PS C:\Users\user>
```

- Based on the screenshot above, the test ran successfully.
    - Microsoft's Calculator program was launched successfully.

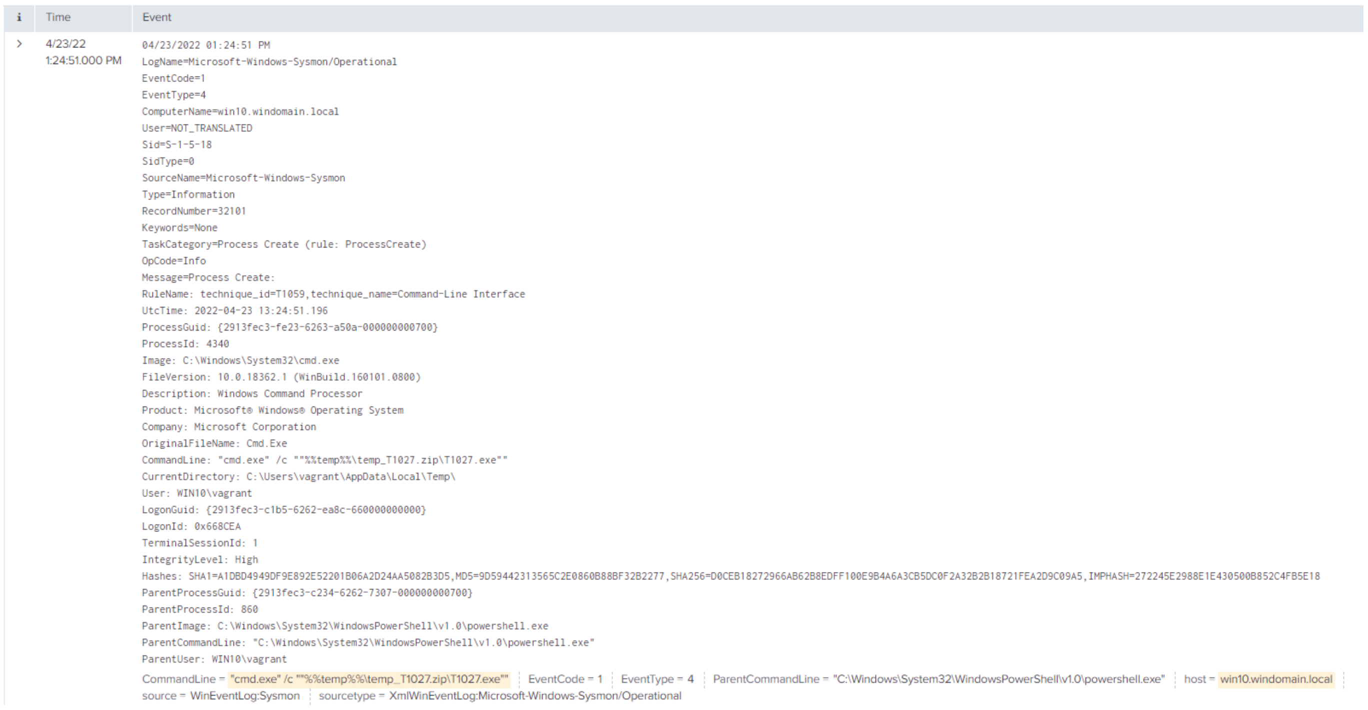
Next, open up the *Splunk - Search & Reporting* instance and begin searching for the log data surrounding the inputted commands.

- Windows Event Process Creation Event (4688): `index="wineventlog" Creator_Process_Name="C:\\Users\\vagrant\\AppData\\Local\\Temp\\temp_T1027.zip\\T1027.exe"`



```
i   Time              Event
>   4/23/22           04/23/2022 01:24:51 PM
    1:24:51.000 PM    LogName=Security
                      EventCode=4688
                      EventType=0
                      ComputerName=win10.windomain.local
                      SourceName=Microsoft Windows security auditing.
                      Type=Information
                      RecordNumber=51623
                      Keywords=Audit Success
                      TaskCategory=Process Creation
                      OpCode=Info
                      Message=A new process has been created.

                      Creator Subject:
                              Security ID:            S-1-5-21-3126794119-865277996-3442430372-1000
                              Account Name:           vagrant
                              Account Domain:         WIN10
                              Logon ID:               0x668CEA

                      Target Subject:
                              Security ID:            S-1-0-0
                              Account Name:           -
                              Account Domain:         -
                              Logon ID:               0x0

                      Process Information:
                              New Process ID:         0x1a70
                              New Process Name:       C:\Windows\SysWOW64\calc.exe
                              Token Elevation Type:   %%1936
                              Mandatory Label:                S-1-16-12288
                              Creator Process ID:     0x1754
                              Creator Process Name:   C:\Users\vagrant\AppData\Local\Temp\temp_T1027.zip\T1027.exe
                              Process Command Line:   calc.exe

                      Token Elevation Type indicates the type of token that was assigned to the new process in accordance with User Account Control policy.

                      Type 1 is a full token with no privileges removed or groups disabled.  A full token is only used if User Account Control is disabled or if the user is the built-in Administrator account or a service account.

                      Type 2 is an elevated token with no privileges removed or groups disabled.  An elevated token is used when User Account Control is enabled and the user chooses to start the program using Run as administrator.  An elevated token is also
                      used when an application is configured to always require administrative privilege or to always require maximum privilege, and the user is a member of the Administrators group.

                      Type 3 is a limited token with administrative privileges removed and administrative groups disabled.  The limited token is used when User Account Control is enabled, the application does not require administrative privilege, and the use
                      r does not choose to start the program using Run as administrator.
                      EventCode = 4688 | EventType = 0 | host = win10.windomain.local | source = WinEventLog:Security | sourcetype = WinEventLog
```

- Sysmon Process Creation Event: `index=sysmon host="win10.windomain.local" CommandLine="\"cmd.exe\" /c \"\"%%temp%%\\temp_T1027.zip\\T1027.exe\"\""`



```
i   Time              Event
>   4/23/22           04/23/2022 01:24:51 PM
    1:24:51.000 PM    LogName=Microsoft-Windows-Sysmon/Operational
                      EventCode=1
                      EventType=4
                      ComputerName=win10.windomain.local
                      User=NOT_TRANSLATED
                      Sid=S-1-5-18
                      SidType=0
                      SourceName=Microsoft-Windows-Sysmon
                      Type=Information
                      RecordNumber=32101
                      Keywords=None
                      TaskCategory=Process Create (rule: ProcessCreate)
                      OpCode=Info
                      Message=Process Create:
                      RuleName: technique_id=T1059,technique_name=Command-Line Interface
                      UtcTime: 2022-04-23 13:24:51.196
                      ProcessGuid: {2913fec3-fe23-6263-a50a-000000000700}
                      ProcessId: 4340
                      Image: C:\Windows\System32\cmd.exe
                      FileVersion: 10.0.18362.1 (WinBuild.160101.0800)
                      Description: Windows Command Processor
                      Product: Microsoft® Windows® Operating System
                      Company: Microsoft Corporation
                      OriginalFileName: Cmd.Exe
                      CommandLine: "cmd.exe" /c ""%%temp%%\temp_T1027.zip\T1027.exe""
                      CurrentDirectory: C:\Users\vagrant\AppData\Local\Temp\
                      User: WIN10\vagrant
                      LogonGuid: {2913fec3-c1b5-6262-ea8c-660000000000}
                      LogonId: 0x668CEA
                      TerminalSessionId: 1
                      IntegrityLevel: High
                      Hashes: SHA1=A1DBD4949DF9E892E52201B06A2D24AA5082B3D5,MD5=9D59442313565C2E0860B88BF32B2277,SHA256=D0CEB18272966AB62B8EDFF100E9B4A6A3CB5DC0F2A32B2B18721FEA2D9C09A5,IMPHASH=272245E2988E1E430500B852C4FB5E18
                      ParentProcessGuid: {2913fec3-c234-6262-7307-000000000700}
                      ParentProcessId: 860
                      ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                      ParentCommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
                      ParentUser: WIN10\vagrant
                      CommandLine = "cmd.exe" /c ""%%temp%%\temp_T1027.zip\T1027.exe"" | EventCode = 1 | EventType = 4 | ParentCommandLine = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" | host = win10.windomain.local |
                      source = WinEventLog:Sysmon | sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
```

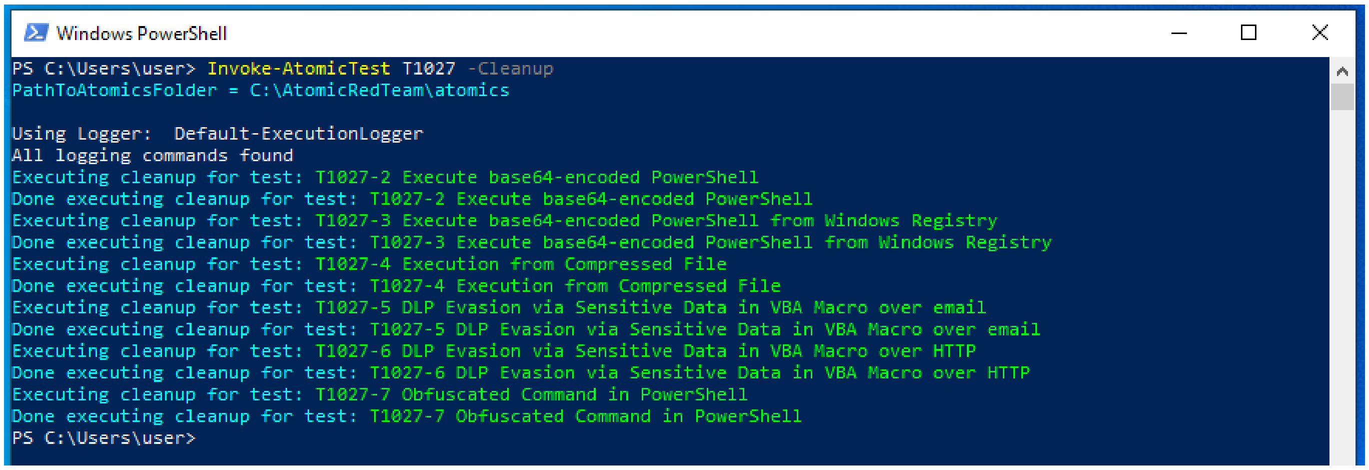
What information do you think may be relevant to determine what occurred on the device?

- Use the fields on the left of the Splunk Search to help filter the search results.
- Some of the relevant fields will be as follows:
    - Host
    - Command Line
    - Process Name
    - Parent Process
    - Process Paths
    - IP Addresses
    - File Hash

Step 6: Clean Up

- Some tests may change items within your environment.
- Run command the following command to clean up any changes made to the system while performing tests.

```
Invoke-AtomicTest T1027 -Cleanup
```



```
PS C:\Users\user> Invoke-AtomicTest T1027 -Cleanup
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger:  Default-ExecutionLogger
All logging commands found
Executing cleanup for test: T1027-2 Execute base64-encoded PowerShell
Done executing cleanup for test: T1027-2 Execute base64-encoded PowerShell
Executing cleanup for test: T1027-3 Execute base64-encoded PowerShell from Windows Registry
Done executing cleanup for test: T1027-3 Execute base64-encoded PowerShell from Windows Registry
Executing cleanup for test: T1027-4 Execution from Compressed File
Done executing cleanup for test: T1027-4 Execution from Compressed File
Executing cleanup for test: T1027-5 DLP Evasion via Sensitive Data in VBA Macro over email
Done executing cleanup for test: T1027-5 DLP Evasion via Sensitive Data in VBA Macro over email
Executing cleanup for test: T1027-6 DLP Evasion via Sensitive Data in VBA Macro over HTTP
Done executing cleanup for test: T1027-6 DLP Evasion via Sensitive Data in VBA Macro over HTTP
Executing cleanup for test: T1027-7 Obfuscated Command in PowerShell
Done executing cleanup for test: T1027-7 Obfuscated Command in PowerShell
PS C:\Users\user>
```

References

1. https://www.lexico.com/definition/obfuscate↩
2. https://resource.redcanary.com/rs/003-YRU-314/images/2022_ThreatDetectionReport_RedCanary.pdf↩
3. https://attack.mitre.org/↩
4. https://attack.mitre.org/tactics/TA0005/↩
5. https://attack.mitre.org/tactics/TA0002/↩
6. https://attack.mitre.org/tactics/TA0001/↩
7. https://redcanary.com/threat-detection-report/techniques/obfuscated-files-information/↩↩
8. https://developer.mozilla.org/en-US/docs/Glossary/Base64↩
9. https://www.techopedia.com/definition/3470/concatenation-programming↩

10. https://www.lexico.com/definition/substring↩
11. https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-expression?view=powershell-7.2↩
12. https://www.techopedia.com/definition/822/escape-sequence-c↩
13. https://attack.mitre.org/groups/↩
14. https://www.crowdstrike.com/cybersecurity-101/compromise-assessments/↩
15. https://www.crowdstrike.com/cybersecurity-101/indicators-of-compromise/↩