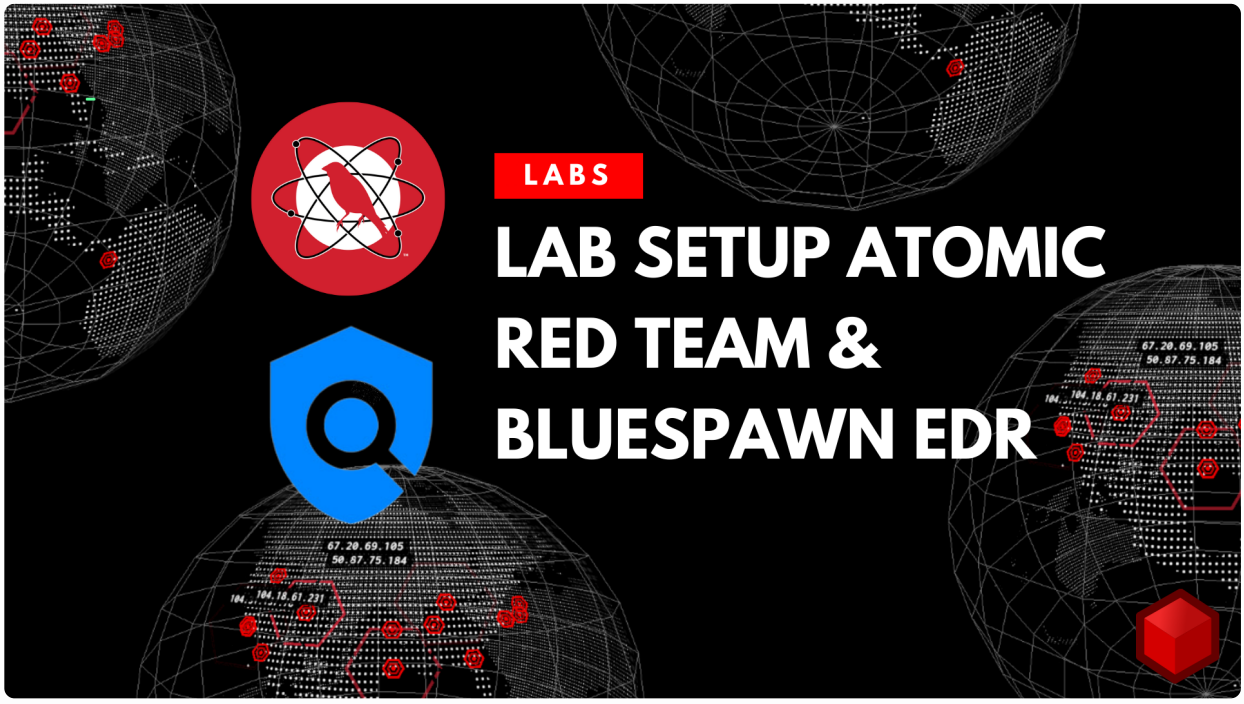
Lab Setup Atomic Red Team & Bluespawn EDR



Part 1: Install Atomic Red Team

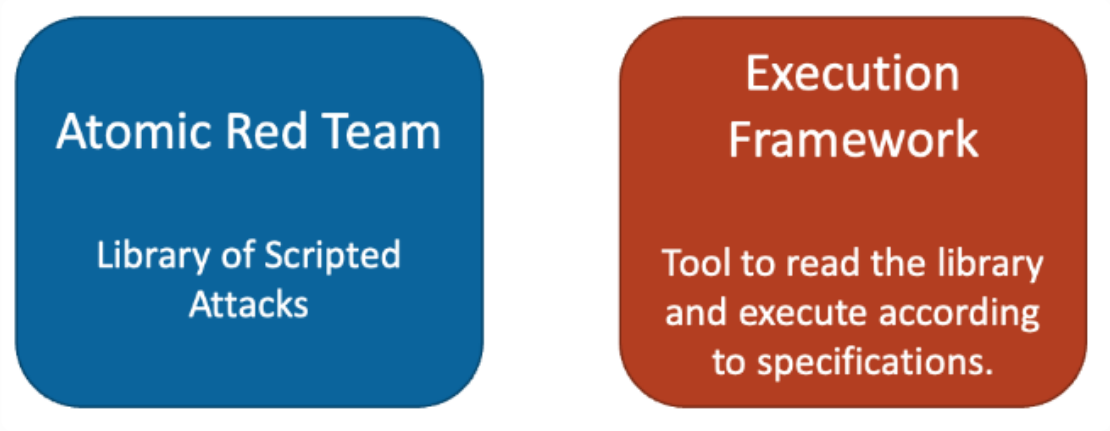
Objective: Install and Configure the Atomic Red Team library of scripted attacks and the PowerShell Execution framework to simplify the execution of atomic tests.

Lab VMs Needed: Windows PowerShell

Instructions:

An “Execution Framework” is a tool to aid in the execution of atomic tests, making it so we don’t have to copy and paste commands into the specified executors (e.g. PowerShell or cmd.exe).

Remember that the Atomic Red Team project itself is just a library of commands used to run specific cyber-attacks. You can run these commands manually or use an “Execution Framework” to automate the execution.

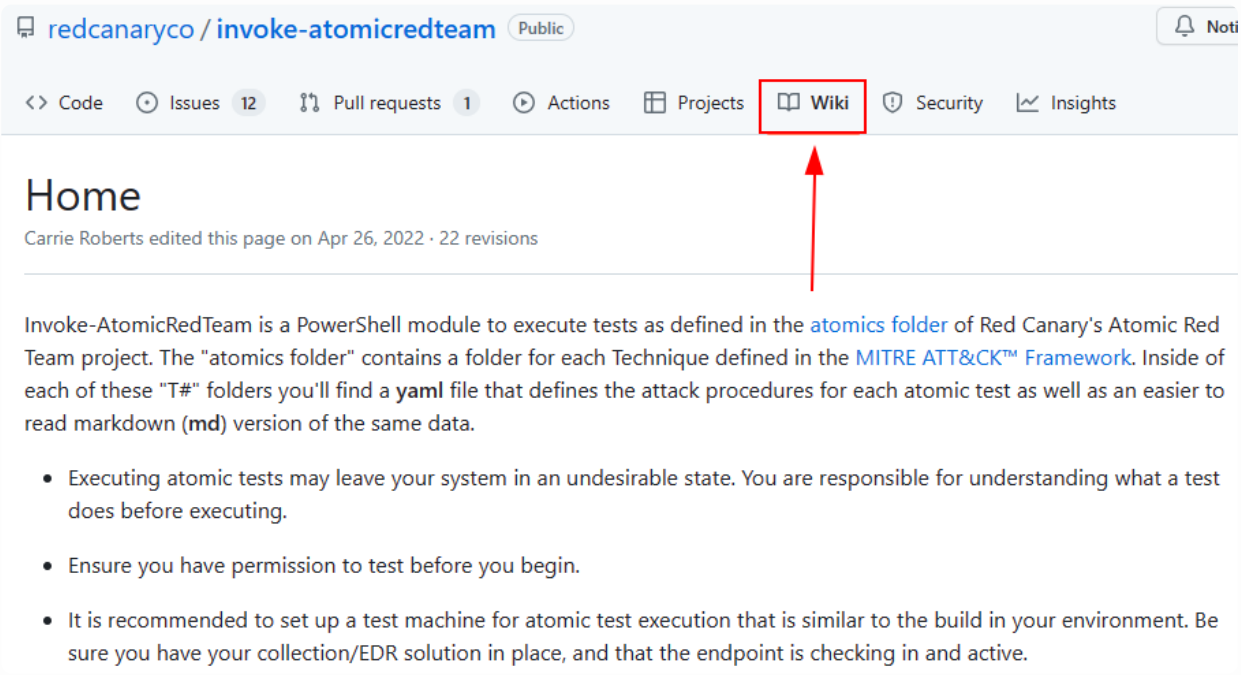


There are a variety of execution frameworks that can be used to read the Atomic Red Team library of scripted attacks and execute them. The names of some of these frameworks are listed below:

- Invoke-AtomicRedTeam: Cross-Platform PowerShell Execution Framework 1
- Prelude Operator: Cross-Platform, multi-language Command and Control (server/client) style Execution Framework (free community edition)
- CALDERA: Similar to Prelude Operator, Develop by the MITRE organization
- Atomic Operator: Cross-Platform Python Execution Framework

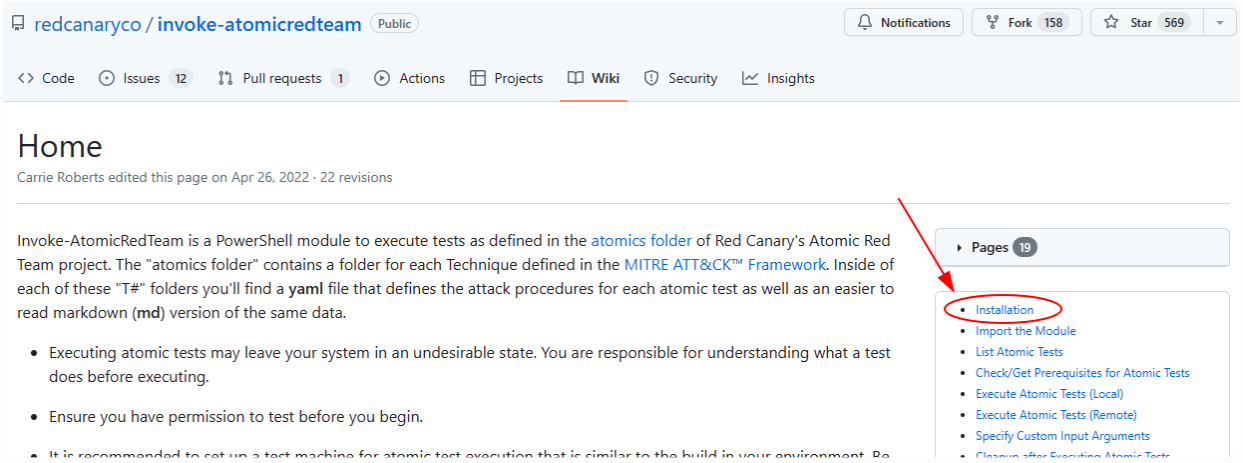
The PowerShell Execution Framework works cross-platform when PowerShell core is installed on macOS and Linux. For more details do check out below wiki :

- https://github.com/redcanaryco/invoke-atomicredteam/wiki



The Wiki is full of helpful information about how to install, configure and use Atomic Red Team and the execution framework.

Click on the "Installation" link on the right

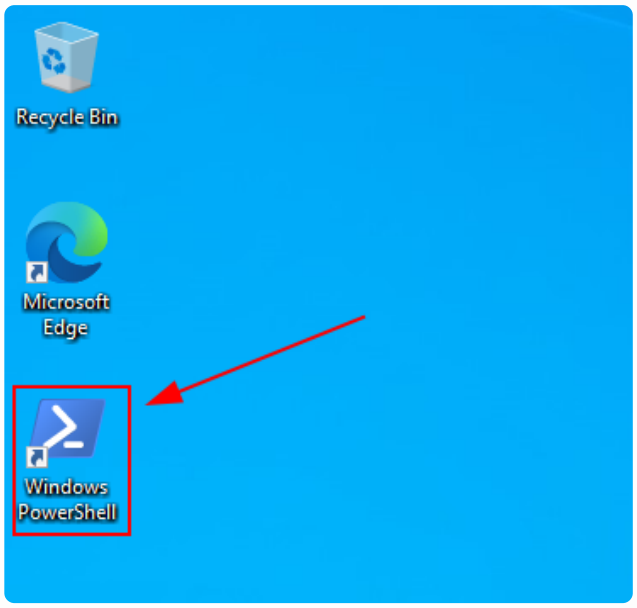


Please read through the installation instructions on the Wiki. You will notice there are three installation options. The first installs only the execution framework, without the atomic test definitions (the library of attacks). This is helpful if we are in an environment where we don't want to download the entire atomics folder full of simulated malware. We may wish to avoid this to avoid setting off alerts or have many files automatically removed/quarantined. In such
a scenario, we may want to hand pick only the atomics we are going to run and copy only those
over to the system.

The second installation option will install both the Execution Framework and the "atomics" folder full of the atomic test definitions and supporting files. This is the installation option we
want to use for the labs in this class.

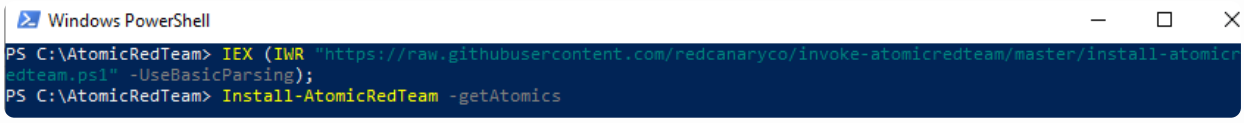
The third installation option simply downloads the library of scripted attacks without the execution framework.

Follow the instructions in the "Install Execution Framework and Atomics Folder" section of the Wiki. The installation commands should be run from a PowerShell prompt. You can start the PowerShell prompt by double clicking on the shortcut on the desktop.

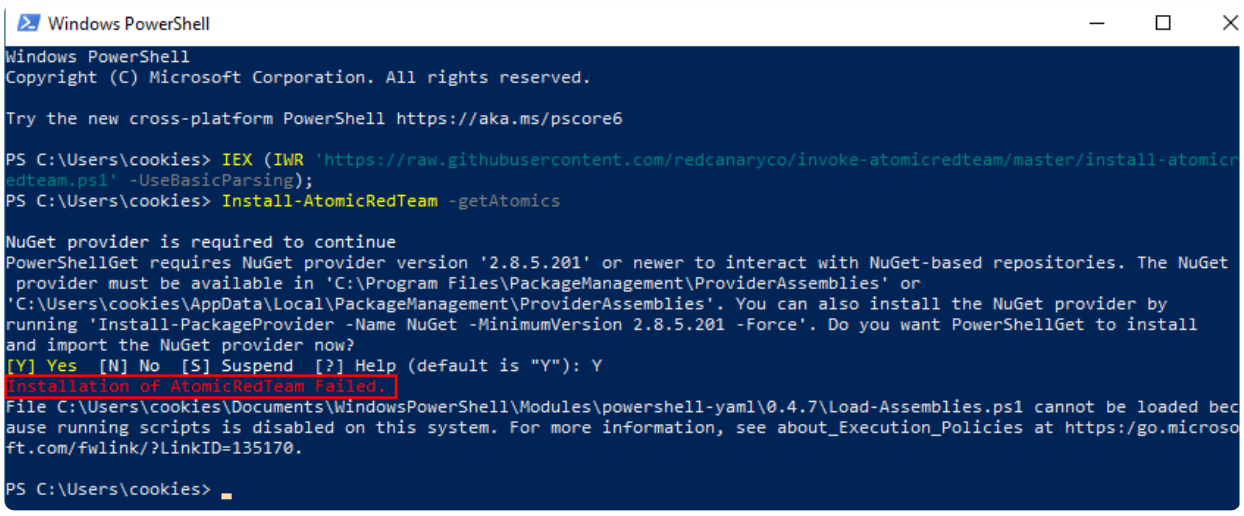


Paste the commands from the Wiki (also shown below) into the PowerShell prompt. You might
need to hit “Enter” after pasting the commands.

```
IEX (IWR "https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/maste

Install-AtomicRedTeam -getAtomics
```



You will be prompted to import the NuGet provider to which you should answer “Y”. If you have run this command before, you will need to add the “-Force” parameter to force the overwriting of the previous installation



You may receive an error during the installation as shown above. Notice that the last red line is telling us that the installation failed? The reason it failed is because there was a PowerShell
Execution policy in place preventing the running of scripts. If this occurs during your install, you will need to bypass this “safety feature” in order to use the execution framework. For simplicity in these labs, we will just bypass the execution policy completely for the current user
with the following command.

```
Set-ExecutionPolicy Bypass -Scope CurrentUser
```

There are options for just temporarily bypassing the execution policy, but we don’t cover those
here.

```
PS C:\Users\cookies> Set-ExecutionPolicy Bypass -Scope CurrentUser

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): Y
PS C:\Users\cookies>
```

Bypassing the execution policy outside of the lab environment may not be as straightforward
as in your own environment. If you run into problems, check out one of these methods is likely
to work. Method 12 is especially promising.

Now that we have installed the execution framework and the “atomics folder” containing the test definitions and simulated malware we may start seeing Windows Defender showing disapproval.

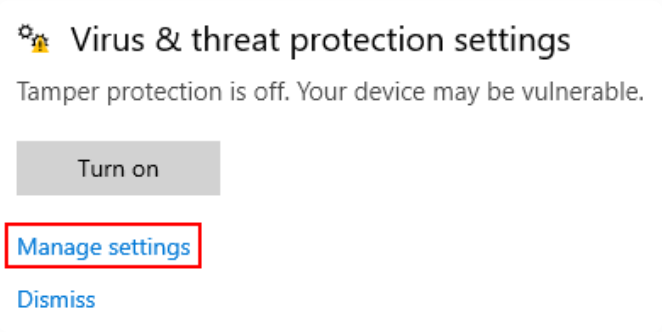
If we review the Virus and Threat Protection settings for Windows Defender, we can see some
needed files are being blocked, quarantined and/or deleted. In our lab environment, we are

going to exclude the atomics folder from being scanned by Windows Defender so that we can
run all of the atomic tests without being blocked. This allows us to confirm that we can detect
the attack even if preventative controls are bypassed. Remember, “Prevention is ideal, but detection is a must” an allowing emulation of each attack will help us prove out both.

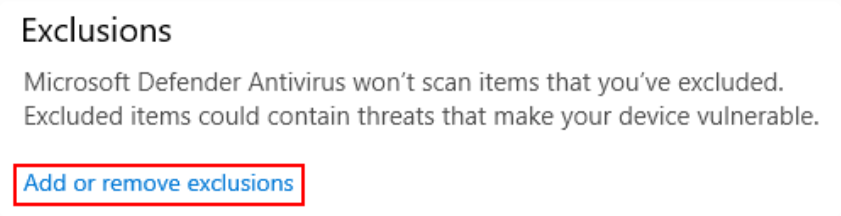
On the start menu search for “Virus and threat Protection” and launch.

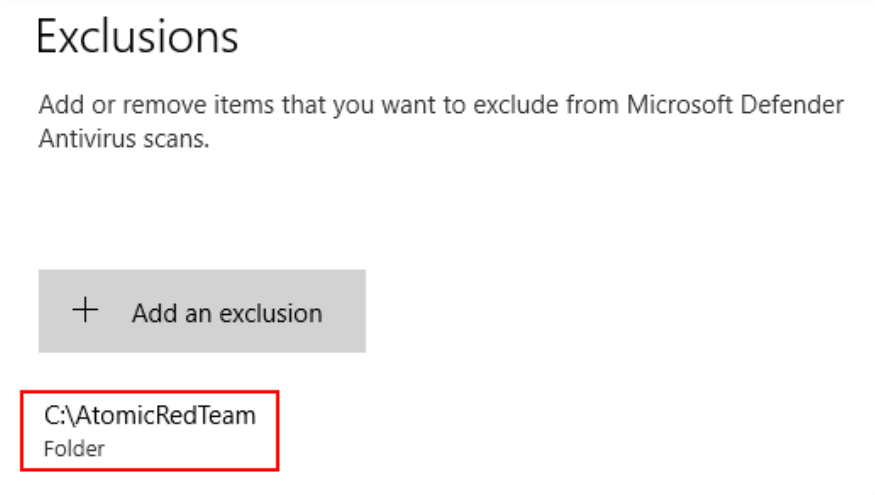
If prompted about “Sample submission” click “Don’t send”.

Under “Virus & threat protection settings” click “Manage settings”



Add an exclusion for the C:\AtomicRedTeam folder under the “Exclusions” section.



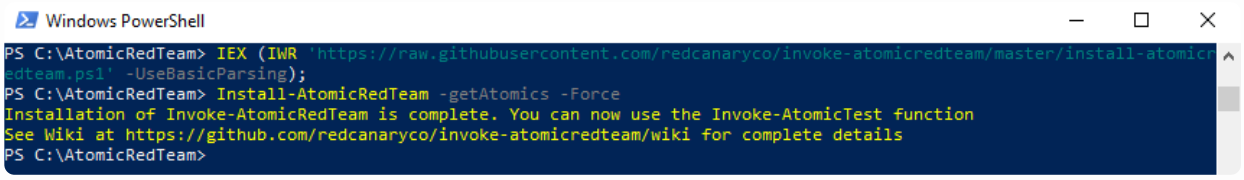


An alternative way to quickly add this exclusion from the PowerShell command line is given below for convenience but must be run from an administrative PowerShell prompt.

```
Add-MpPreference -ExclusionPath C:\AtomicRedTeam\
```

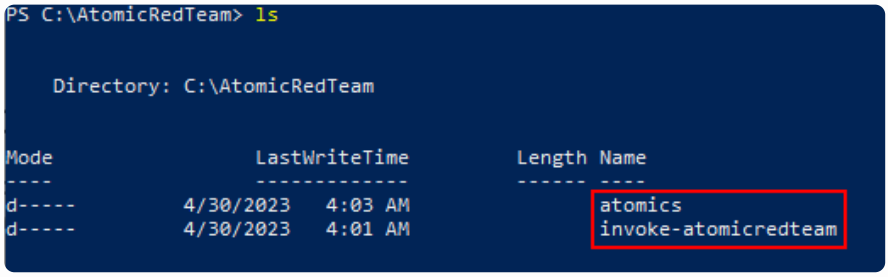
Now, run the installation command one more time (from the non-administrative PowerShell prompt) to re-download all the atomic files that may have been blocked or removed by Windows Defender during the initial install.

```
IEX (IWR "https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/maste

Install-AtomicRedTeam -getAtomics -Force
```

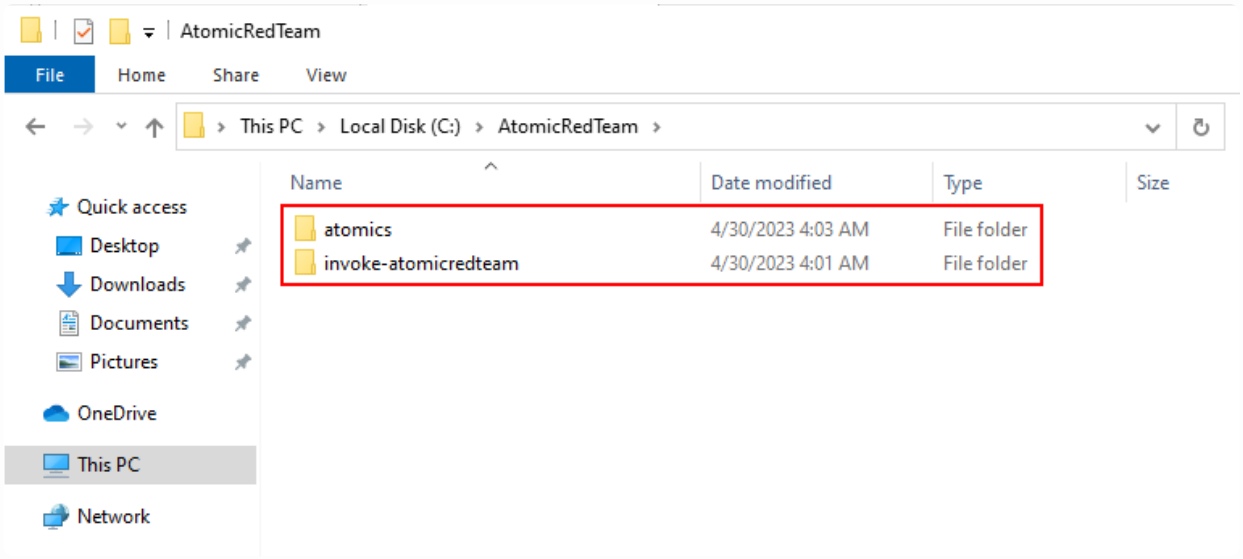


Occasionally, Windows Defender decides to block the install at this point. If this happens, you
need to completely disable Windows Defender for the installation.

To make sure everything was installed correctly, let's run the "Get-Module" PowerShell command. You should see the two items highlighted in red. They may not be positioned together.



You can also validate that you have the "atomics" and "invoke-atomicredteam" folders in the default installation folder of C:\AtomicRedTeam.



We purposefully gave instructions to let you run into the common hurdles encountered during
the installation. We could expedite the installation in the future by ensuring we bypass the execution policy and by adding the C:\AtomicRedTeam folder to the Windows Defender exceptions list before the installation attempt.

Atomic Red Team and the Execution Framework are now installed and ready for use. To make
sure the Execution Framework is always loaded you should configure your profile to automatically import this module as described in the next lab “Import the Atomic Red Team Module”.

- [x] End of Part 1

Part 2: Import Atomic Red Team Module

Objective: Ensure that the PowerShell Execution Framework modules are loaded and available
for use.

Lab VMs Needed: Windows PowerShell.

Instructions:

In the previous lab, we installed the Execution Framework. One step taken by the installer was
to import the Atomic Red Team module so that its functions are available for use. If you are still using the same PowerShell window where you installed Atomic Red Team, the needed modules will already be imported. For example, you could make a call to the Invoke-AtomicTest function without receiving an error message as shown below

```
PS C:\AtomicRedTeam> Invoke-AtomicTest

cmdlet Invoke-AtomicTest at command pipeline position 1
Supply values for the following parameters:
AtomicTechnique[0]:
PS C:\AtomicRedTeam>
```

We don't want to run a test right now, so press Ctrl+C to cancel the command. Now close the
PowerShell window and open a new PowerShell window. From this new window, type in "Invoke-AtomicTest" again. Now PowerShell is complaining that it doesn't recognize the command

```
PS C:\Users\cookies> Invoke-AtomicTest
Invoke-AtomicTest : The term 'Invoke-AtomicTest' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try
again.
At line:1 char:1
+ Invoke-AtomicTest
+ ~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Invoke-AtomicTest:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\cookies>
```

The Invoke-AtomicTest function is not recognized because the new PowerShell session/window does not have the Atomic Red Team modules loaded. If you want to run the modules you must import them into PowerShell again.

Run the following command to import the module again

```
Import-Module "C:\AtomicRedTeam\invoke-atomicredteam\Invoke-AtomicRedTeam.psd1" -F
```

Now the Invoke-AtomicTest command is available for use again.

```
PS C:\Users\cookies> Invoke-AtomicTest
Invoke-AtomicTest : The term 'Invoke-AtomicTest' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try
again.
At line:1 char:1
+ Invoke-AtomicTest
+ ~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Invoke-AtomicTest:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\cookies> Import-Module "C:\AtomicRedTeam\invoke-atomicredteam\Invoke-AtomicRedTeam.psd1" -Force
PS C:\Users\cookies> Invoke-AtomicTest

cmdlet Invoke-AtomicTest at command pipeline position 1
Supply values for the following parameters:
AtomicTechnique[0]:
PS C:\Users\cookies>
```

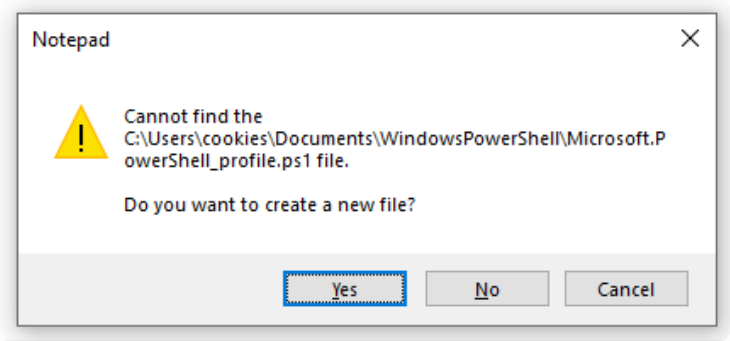
Press Ctrl+C to return to the PowerShell prompt.

If you are going to be using the execution framework across multiple PowerShell sessions it could get annoying to have to do the import each time. To make sure that the execution framework is always available for use, even after starting a new PowerShell session, we can add the import command to our PowerShell profile. Type the following in the PowerShell command prompt to edit your profile.

```
notepad $profile
```

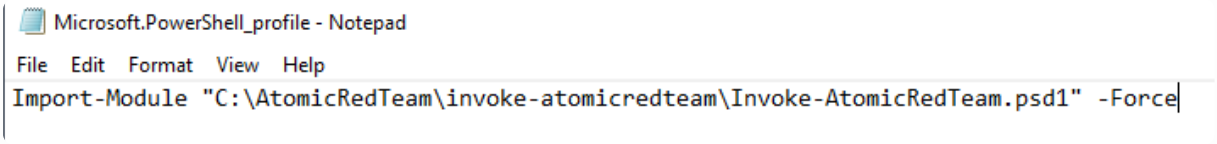
```
PS C:\Users\cookies> notepad $profile
```

Notepad should open and ask if you want to create a new file. Click "Yes"



In the Notepad file, add the Import-Module statement and save the file.

```
Import-Module "C:\AtomicRedTeam\invoke-atomicredteam\Invoke-AtomicRedTeam.psd1" -F
```



With the import statement in place in our profile, we can close and restart our PowerShell window and the execution framework modules will already be imported and available for use.

Now we are ready to use the framework to simplify the execution of the atomic tests from the
Atomic Red Team library of scripted attacks.

- [x] End of Part 2

Part 3: List Atomic Tests

Objective: Use the execution framework to list the atomic tests available for execution, along
with details of the commands, prerequisites and clean up.

Lab VMs Needed: Windows PowerShell.

Instructions:

Before we run any Atomic tests, let's use the execution framework to find out what tests are available for execution. Type the following into the PowerShell prompt.

```
Invoke-AtomicTest T1003 -ShowDetailsBrief
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1003 -ShowDetailsBrief
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

T1003-1 Gsecdump
T1003-2 Credential Dumping with NPPSpy
T1003-3 Dump svchost.exe to gather RDP credentials
T1003-4 Retrieve Microsoft IIS Service Account Credentials Using AppCmd (using list)
T1003-5 Retrieve Microsoft IIS Service Account Credentials Using AppCmd (using config)
T1003-6 Dump Credential Manager using keymgr.dll and rundll32.exe
PS C:\Users\cookies>
```

With the “ShowDetailsBrief” flag, you can see the MITRE Technique number (aka T#) and the atomic test name.

Now let’s take a look at the specific details of each test. We can do this by changing the “ShowDetailsBrief” flag to simply “ShowDetails”.

```
Invoke-AtomicTest T1003 -ShowDetails
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1003 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[********BEGIN TEST*******]
Technique: OS Credential Dumping T1003
Atomic Test Name: Gsecdump
Atomic Test Number: 1
Atomic Test GUID: 96345bfc-8ae7-4b6a-80b7-223200f24ef9
Description: Dump credentials from memory using Gsecdump.
Upon successful execution, you should see domain\username's followed by two 32 character hashes.
If you see output that says "compat: error: failed to create child process", execution was likely blocked by Anti-Virus.
  You will receive only error output if you do not run this test from an elevated context (run as administrator)
If you see a message saying "The system cannot find the path specified", try using the get-prereq_commands to download a
nd install Gsecdump first.

Attack Commands:
Executor: command_prompt
ElevationRequired: True
Command:
#{gsecdump_exe} -a
Command (with inputs):
C:\AtomicRedTeam\atomics\T1003\bin\gsecdump.exe -a

Dependencies:
Description: Gsecdump must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003\bin\gsecdump.exe)
Check Prereq Command:
if (Test-Path #{gsecdump_exe}) {exit 0} else {exit 1}
Check Prereq Command (with inputs):
if (Test-Path C:\AtomicRedTeam\atomics\T1003\bin\gsecdump.exe) {exit 0} else {exit 1}
Get Prereq Command:
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
$parentpath = Split-Path "#{gsecdump_exe}"; $binpath = "$parentpath\gsecdump-v2b5.exe"
IEX(IWR "https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/master/Public/Invoke-WebRequestVerifyHash.ps
1" -UseBasicParsing)
if(Invoke-WebRequestVerifyHash "#{gsecdump_url}" "$binpath" #{gsecdump_bin_hash}){
```

A bunch of information just scrolled across the screen, probably more than you could take in.

Let’s get just the details for one of the tests. Look at the previous output (ShowDetailsBrief) and choose one of the tests to view. We will choose to look at test number 1 (Gsecdump). We

can use the “TestNumbers” flag to specify test number 1.

```
Invoke-AtomicTest T1003 -TestNumbers 1 -ShowDetails
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1003 -TestNumbers 1 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[********BEGIN TEST*******]
Technique: OS Credential Dumping T1003
Atomic Test Name: Gsecdump
Atomic Test Number: 1
Atomic Test GUID: 96345bfc-8ae7-4b6a-80b7-223200f24ef9
Description: Dump credentials from memory using Gsecdump.
Upon successful execution, you should see domain\username's followed by two 32 character hashes.
If you see output that says "compat: error: failed to create child process", execution was likely blocked by Anti-Virus.
  You will receive only error output if you do not run this test from an elevated context (run as administrator)
If you see a message saying "The system cannot find the path specified", try using the get-prereq_commands to download a
nd install Gsecdump first.

Attack Commands:
Executor: command_prompt
ElevationRequired: True
Command:
#{gsecdump_exe} -a
Command (with inputs):
C:\AtomicRedTeam\atomics\T1003\bin\gsecdump.exe -a
```

These details contain the information about the test. In addition to describing the test, you can
see that the Executor for this test is “command_prompt” meaning it should be run from the command prompt (cmd.exe) and that it must be run with elevated privileges (ElevationRequired: True).

Notice that there is a “Command:” section and a “Command (with inputs):” section. The “Command (with inputs)” shows the command with all of the input arguments filled in. For example, the red #{gsecdump_exe} argument has been replaced with the path to the gsecdump.exe file.

```
Command:
#{gsecdump_exe} -a
Command (with inputs):
C:\AtomicRedTeam\atomics\T1003\bin\gsecdump.exe -a
```

We can also specify tests by name instead of number with the “TestNames” flag.

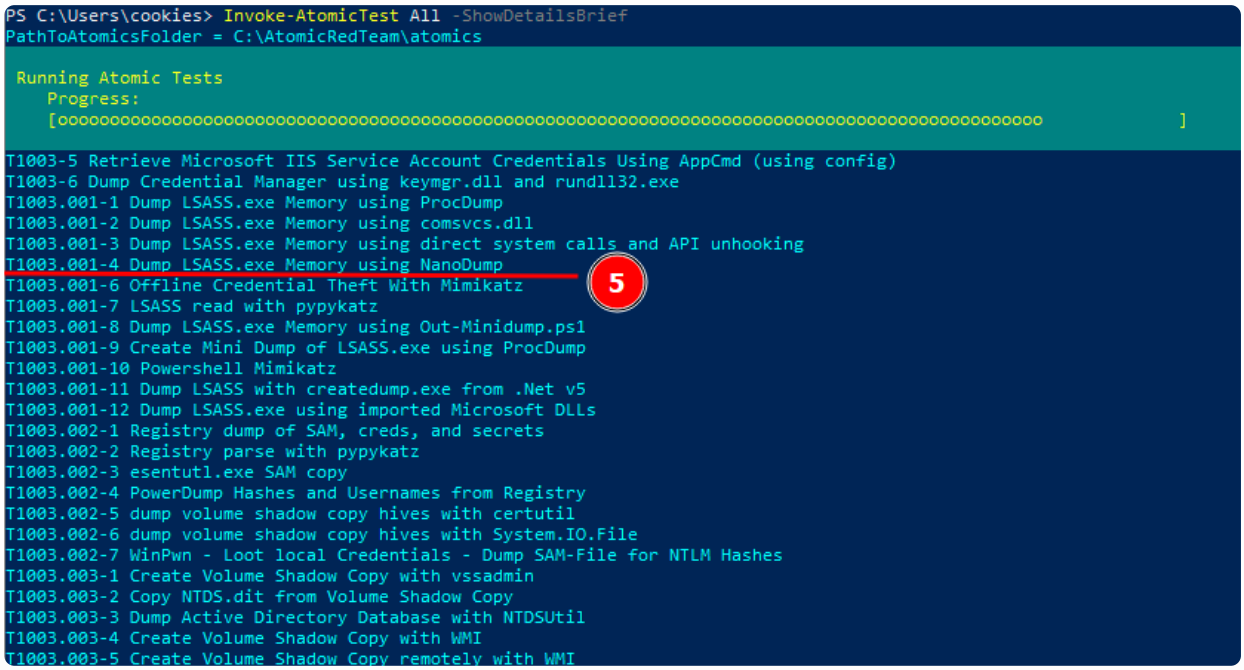
```
Invoke-AtomicTest T1003 -TestNames "Gsecdump" -ShowDetails
```

You can also provide multiple test numbers or names as a comma separated list as shown in the example below.

```
Invoke-AtomicTest T1003 -TestNumbers 1,2 -ShowDetails
```

Curious what other tests are available for execution on the current OS? Try listing them all by
using “All” in the place of the technique number.

```
Invoke-AtomicTest All -ShowDetailsBrief
```



Wow, there are a lot of tests in there. Note that only tests that apply to the current operating
system (Windows in this case) will be listed. It also does not list tests that include manual steps.
This is why you might see some missing test numbers, like T1003.001-5 in the image above.

- [x] End of Part 3

Part 4: Check or Get Prerequisites

Objective: Use the execution framework to check if your system meets the prerequisites for executing an atomic test. If dependencies aren't met, use the framework to download, install,
or otherwise meet the prerequisites.

Lab VMs Needed: Windows PowerShell.

Instructions:

Some Atomic Tests have dependencies on certain applications being installed or files being present. The dependencies or "prerequisites" must be satisfied in order to successfully run the
atomic test. Let's look at a test under Technique T1485 - Data Destruction that has a prerequisite.

```
Invoke-AtomicTest T1485 -ShowDetailsBrief
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1485 -ShowDetailsBrief
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

T1485-1 Windows - Overwrite file with Sysinternals SDelete
```

We will run the first test which securely deletes a file using the Sysinternals SDelete tool. If we try to run this test right off, we will get an error that “sdelete.exe” is not recognized as an
operable program. This is because SDelete does not come installed on a default Windows OS.

```
PS C:\Users\cookies> Invoke-AtomicTest T1485 -TestNumbers 1
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1485-1 Windows - Overwrite file with Sysinternals SDelete
C:\Users\cookies\AppData\Local\Temp\Sdelete\sdelete.exe : The term
file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct
'C:\Users\cookies\AppData\Local\Temp\Sdelete\sdelete.exe' is not recognized as the name of a cmdlet, function, script
and try again.
At line:1 char:1
+ C:\Users\cookies\AppData\Local\Temp\Sdelete\sdelete.exe -accepteula C ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (C:\Users\cookie...ete\sdelete.exe:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
Done executing test: T1485-1 Windows - Overwrite file with Sysinternals SDelete
PS C:\Users\cookies>
```

If we take a look at the markdown file that describes this test, we see that there is a prerequisite
that the “Secure delete tool from Sysinternals must exist on disk at specified location”.

Dependencies: Run with `powershell`!

Description Secure delete tool from Sysinternals must exist on disk at specified location (#{sdelete_exe})

Check Prereq Commands:

```
if (Test-Path #{sdelete_exe}) {exit 0} else {exit 1}
```

Get Prereq Commands:

```
Invoke-WebRequest "https://download.sysinternals.com/files/SDelete.zip" -OutFile "$env:TEMP\SDelete.zip"
Expand-Archive $env:TEMP\SDelete.zip $env:TEMP\Sdelete -Force
Remove-Item $env:TEMP\SDelete.zip -Force
```

We can use the “CheckPrereqs” flag to check if we meet the prerequisites before running the
test.

```
Invoke-AtomicTest T1485 -TestNumbers 1 -CheckPrereqs
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1485 -TestNumbers 1 -CheckPrereqs
>>
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

CheckPrereq's for: T1485-1 Windows - Overwrite file with Sysinternals SDelete
Prerequisites not met: T1485-1 Windows - Overwrite file with Sysinternals SDelete
        [*] Secure delete tool from Sysinternals must exist on disk at specified location ($env:TEMP\Sdelete\sdelete.exe
)

Try installing prereq's with the -GetPrereqs switch
PS C:\Users\cookies>
```

We see that we failed to meet the prerequisite. We can then use the “GetPrereqs” flag to satisfy
this dependency.

```
Invoke-AtomicTest T1485 -TestNumbers 1 -GetPrereqs
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1485 -TestNumbers 1 -GetPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

GetPrereq's for: T1485-1 Windows - Overwrite file with Sysinternals SDelete
Attempting to satisfy prereq: Secure delete tool from Sysinternals must exist on disk at specified location ($env:TEMP\S
delete\sdelete.exe)
Prereq successfully met: Secure delete tool from Sysinternals must exist on disk at specified location ($env:TEMP\Sdelet
e\sdelete.exe)
PS C:\Users\cookies>
```

The output indicates that the prerequisites have been successfully met. We can now successfully execute this test.

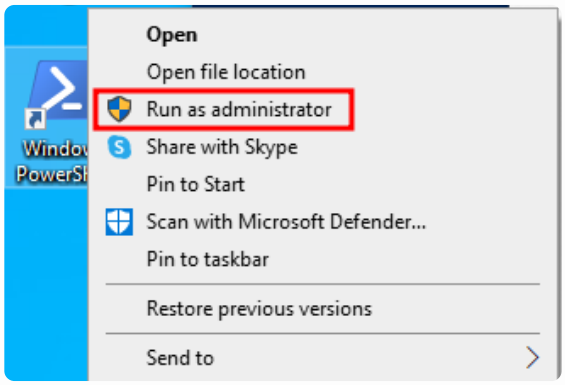
```
Invoke-AtomicTest T1485 -TestNumbers 1
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1485 -TestNumbers 1
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1485-1 Windows - Overwrite file with Sysinternals SDelete
SDelete v2.04 - Secure file delete
Copyright (C) 1999-2019 Mark Russinovich
Sysinternals - www.sysinternals.com
SDelete is set for 1 pass.
C:\Users\cookies\AppData\Local\Temp\T1485.txt...deleted.
Files deleted: 1
Done executing test: T1485-1 Windows - Overwrite file with Sysinternals SDelete
PS C:\Users\cookies>
```

Each atomic test specifies whether elevation is required in order to run the atomic test successfully. If a test requires admin privileges and you run the “CheckPrereq” from an unelevated context, the execution framework will report that you failed to meet the prerequisites
because “Elevation was required but not provided”.

```
PS C:\Users\cookies> Invoke-AtomicTest T1003 -TestNumbers 2 -CheckPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

CheckPrereq's for: T1003-2 Credential Dumping with NPPSpy
Prerequisites not met: T1003-2 Credential Dumping with NPPSpy
        [*] Elevation required but not provided
        [*] NPPSpy.dll must be available in local temp directory
Try installing prereq's with the -GetPrereqs switch
PS C:\Users\cookies>
```

To run an atomic test with elevated privileges, you will need to start PowerShell with the "Run
as administrator" option. To do this, right click on the PowerShell icon and then click on Run as administrator.



You will need to click "Yes" at the prompt to run as administrator.

Now that we know how to check and satisfy any dependencies for the atomic tests we want to
execute, we are ready to execute tests!

- [x] End of Part 4

Part 5: Execute Atomic Tests

Objective: Use the execution framework to execute atomic tests.

Lab VMs Needed: Windows PowerShell.

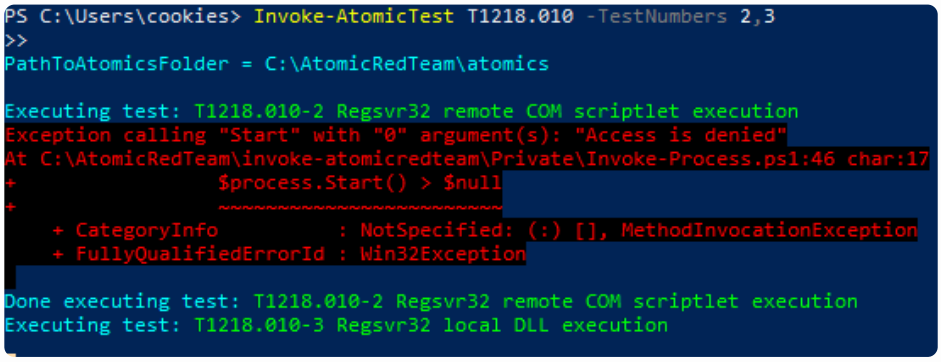
Instructions:

There are two methods that can be used to execute atomic tests using the PowerShell execution

framework: local and remote. In the “local” method, the framework executes the atomic tests
on the same machine that the framework is installed on. Whereas with the “remote” method, the execution framework is installed on one machine, and it is used to execute atomic tests on
a remote machine. You can read more about executing atomic tests remotely on the Wiki here.
In this lab, we will be using local execution to execute atomic tests on the same machine that
the execution framework is installed on.

We actually gave you a sneak peek at executing a test in the previous lab, but we will show some additional examples here. We can execute multiple tests by name or by number using a
comma separated list.

```
Invoke-AtomicTest T1218.010 -TestNumbers 2,3
```



Here we ran tests 2 and 3 of T1218.010. We could have specified these two tests by name instead of number as shown below.

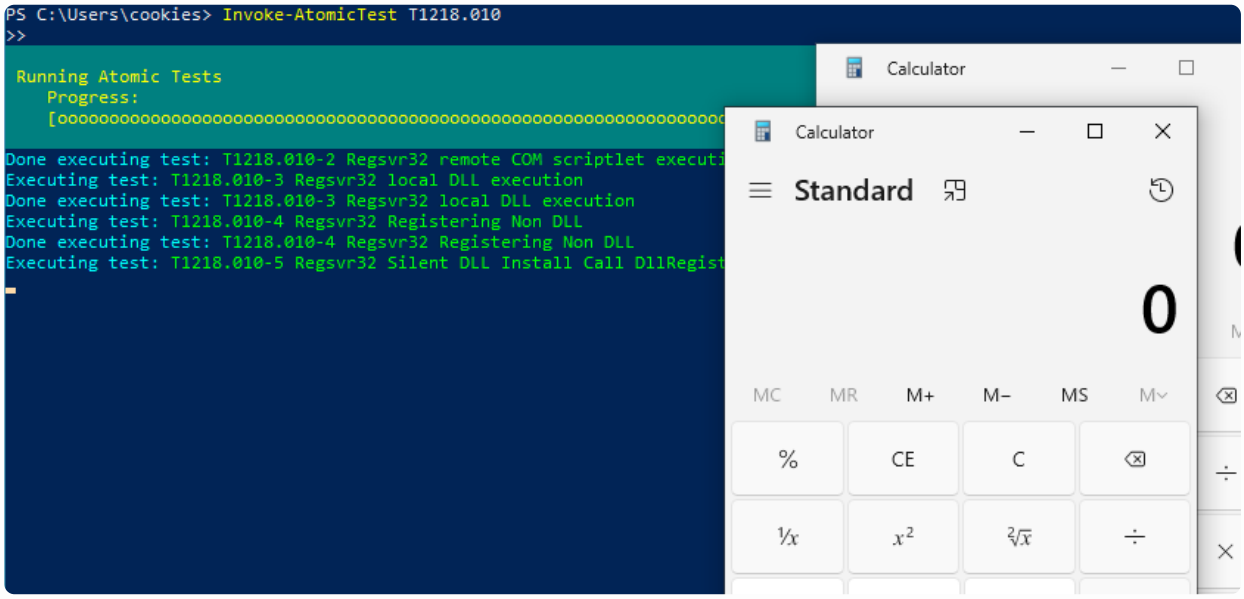
```
Invoke-AtomicTest T1218.010 -TestNames "Regsvr32 remote COM scriptlet
execution", "Regsvr32 local DLL execution"
```

The output shows an error while running test #2 with an “Access is denied” message. This could happen if the test requires execution from an elevated prompt and we didn’t provide it.
However, in this case, this error is a result of Windows Defender blocking this attack technique.
You might notice the popup message from Windows Defender in the corner stating that it blocked something malicious.

Test #3, Regsvr32 local DLL execution, ran successfully because we see the calculator application popup as noted in the description for this test.

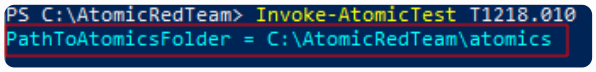
Executing each test individually might be time consuming. If you want to run all tests within a
specific T#, you can just specify the T#.

```
Invoke-AtomicTest T1218.010
```



Above, all of the tests in the T1218.010 technique ran with a single command.

Notice that when we run any test, the first line output on the screen is the "PathToAtomicsFolder".



The default installation location for both the execution framework and the atomics folder is "C:\AtomicRedTeam" on Windows or "~/AtomicRedTeam" on Linux\macOS. When the execution framework runs, it assumes that the atomic test definition YAML files are inside T#
folders at the default location of "C:\AtomicRedTeam\atomics".

If you want to run atomics from another location, you need to tell the execution framework where to find the atomic test definition YAML files by using the "PathToAtomicsFolder" flag as shown below.

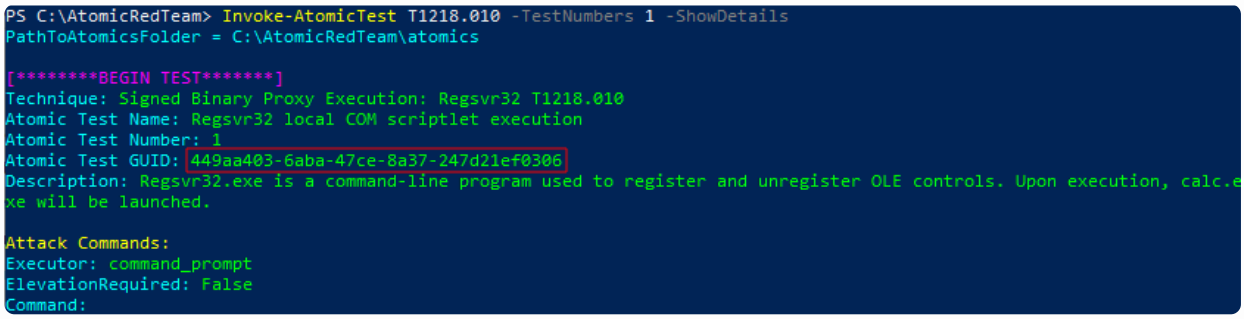
```
Invoke-AtomicTest T1218.010 -PathToAtomicsFolder C:\my-private-atomics
```

This would allow you to run your own custom/private atomics from a different folder

If you don't want to specify your custom path to the atomics folder for every test execution, you can add the following to your PowerShell profile.

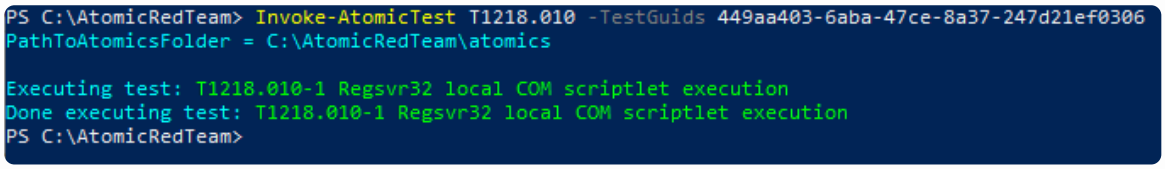
```
$PSDefaultParameterValues = @{"Invoke-AtomicTest:PathToAtomicsFolder"="C:\my-priva
```

Each atomic test has a unique identifier called the GUID. You won't see the GUID in the "ShowDetailsBrief" output, but you will find it in the "ShowDetails" output.



You can execute tests by specifying a GUID instead of a test name/number as follows.

```
Invoke-AtomicTest T1218.010 -TestGuids 449aa403-6aba-47ce-8a37-
247d21ef0306
```



You might be asking yourself "Why in the world would I do that?". While the GUID is not an easy thing to type on the fly, it comes in handy when writing scripts to automate the execution
of multiple tests to ensure that the script always runs the same test. The test numbers and names
change sometimes. The test number is simply an indication of the order the atomic test definition shows up in the YAML file and this can change as tests are added and removed. In addition, the test name can change as developers decide to make the name more descriptive or
fix typos.

Now a word on test execution timeout. The execution framework starts a hidden command window to execute each test but redirects its output to the window you ran the "Invoke-AtomicTest" command from. If the execution takes longer than two minutes by default, the process and its children will be terminated. You can specify an alternate timeout period with the "TimeoutSeconds" flag.

```
Invoke-AtomicTest T1218.010 -TestNumbers 1 -TimeoutSeconds 15
```

Note: Not all processes that are spawned during a test will be terminated after the timeout depending on how they are launched.

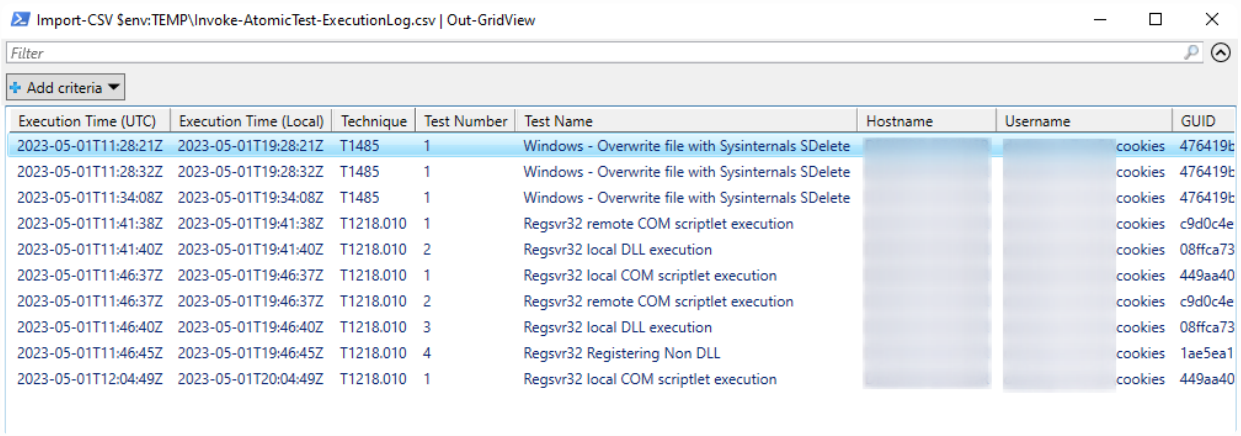
There is another flag called “-Interactive” that comes in handy when the commands being executed result in a prompt for the user give input. For example, some commands may prompt
the user to determine if they want to overwrite an existing file. If you don’t use the Interactive
flag, you won’t see the prompt and the command will timeout. The reason why “-Interactive” is not the default is because you can’t redirect command output to a file when executing with
this flag. Details on redirecting everything you see on the screen after running a test to a file
for logging purposes can be found here.

Another option you can use to execute all atomic tests is the “Invoke-AtomicTest All” command, but this is not recommended. The interaction of all the tests running at once can leave your system in an undesirable state.

By default, the details of which atomics were run are logged to the temp directory in a file called “Invoke-AtomicTest-ExecutionLog.csv”. You can view this file with the following command.

```
Import-CSV $env:TEMP\Invoke-AtomicTest-ExecutionLog.csv | Out-GridView
```

However, you may prefer to view this csv log file in Excel. The Excel
application is not included on the lab machine, so you won’t be able to open this view from there.



If you would like to specify a different path for your log file, you can do that with the “ExecutionLogPath” flag.

```
Invoke-AtomicTest T1218.010 -ExecutionLogPath 'C:\Users\cookies\log.csv'
```

Or, to set the execution log path permanently for all executions, you could add the following line to your PowerShell profile.

```
$PSDefaultParameterValues = @{"Invoke-AtomicTest:ExecutionLogPath"="C:\Users\cooki
```

This covers all the basic options for executing an atomic test. In the following labs, we will learn how to specify custom input arguments and clean up after test execution.

- [x] End of Part 5

Part 6: Specify Custom Input Arguments

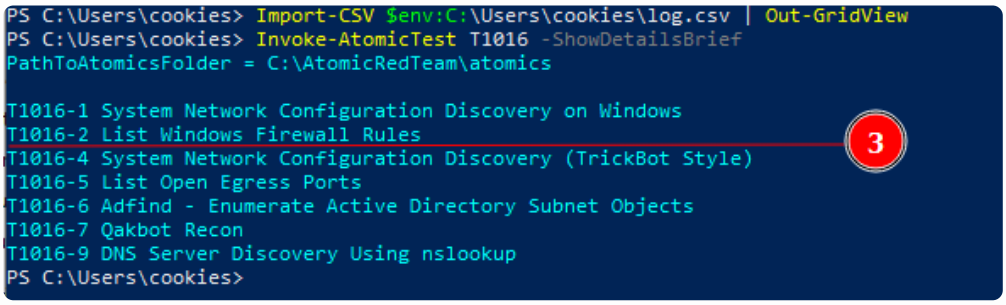
Objective: Specify custom input arguments during atomic test execution.

Lab VMs Needed: Windows PowerShell

Instructions:

To learn about input arguments, we will take a closer look at one of the tests under technique
number T1016.

```
Invoke-AtomicTest T1016 -ShowDetailsBrief
```



Notice that there is no test number 3. If you look at the details of test number 3 using the following link, you will see that it does not include Windows as one of the supported platforms.

- https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1016/T1016.md#atomic-tests

Atomic Test #3 - System Network Configuration Discovery

Identify network configuration information.

Upon successful execution, sh will spawn multiple commands and output will be via stdout.

Supported Platforms: macOS, Linux

auto_generated_guid: c141bbdb-7fca-4254-9fd6-f47e79447e17

Attack Commands: Run with `sh`!

```
if [ -x "$(command -v arp)" ]; then arp -a; else echo "arp is missing from the machine. skipping..."; fi;
if [ -x "$(command -v ifconfig)" ]; then ifconfig; else echo "ifconfig is missing from the machine. skipping..."; fi;
if [ -x "$(command -v ip)" ]; then ip addr; else echo "ip is missing from the machine. skipping..."; fi;
if [ -x "$(command -v netstat)" ]; then netstat -ant | awk '{print $NF}' | grep -v '[a-z]' | sort | uniq -c; else echo "netstat is mi
```

The supported platforms for test #3 are Linux and macOS. Since we are using the execution framework on a Windows machine, these tests don't apply and are not listed.

We continue on by looking at the information for test number 5.

Atomic Test #5 - List Open Egress Ports

This is to test for what ports are open outbound. The technique used was taken from the following blog: https://www.blackhillsinfosec.com/poking-holes-in-the-firewall-egress-testing-with-allports-exposed/

Upon successful execution, powershell will read top-128.txt (ports) and contact each port to confirm if open or not. Output will be to Desktop\open-ports.txt.

Supported Platforms: Windows

auto_generated_guid: 4b467538-f102-491d-ace7-ed487b853bf5

Inputs:

Name	Description	Type	Default Value
output_file	Path of file to write port scan results	path	$env:USERPROFILE\Desktop\open-ports.txt
portfile_url	URL to top-128.txt	url	https://github.com/redcanaryco/atomic-red-team/raw/master/atomics/T1016/src/top-128.txt
port_file	The path to a text file containing ports to be scanned, one port per line. The default list uses the top 128 ports as defined by Nmap.	path	PathToAtomicsFolder\T1016\src\top-128.txt

Notice that this test has 3 input arguments (aka inputs). The names of the input arguments are
"output_file", "portfile_url" and "port_file". Look at the table and you can see the default value for each of these arguments.

Let's use the PowerShell Execution Framework to look at the details for test #5.

```
Invoke-AtomicTest T1016 -TestNumbers 5 -ShowDetails
```

Scroll to the "Command" section.

```
Command:
$ports = Get-content #{port_file}
$file = "#{output_file}"
$totalopen = 0
$totalports = 0
New-Item $file -Force
foreach ($port in $ports) {
    $test = new-object system.Net.Sockets.TcpClient
    $wait = $test.beginConnect("allports.exposed", $port, $null, $null)
    $wait.asyncwaithandle.waitone(250, $false) | Out-Null
    $totalports++ | Out-Null
    if ($test.Connected) {
        $result = "$port open"
        Write-Host -ForegroundColor Green $result
        $result | Out-File -Encoding ASCII -append $file
        $totalopen++ | Out-Null
    }
    else {
        $result = "$port closed"
        Write-Host -ForegroundColor Red $result
        $totalclosed++ | Out-Null
        $result | Out-File -Encoding ASCII -append $file
    }
}
$results = "There were a total of $totalopen open ports out of $totalports ports tested."
$results | Out-File -Encoding ASCII -append $file
Write-Host $results
Command (with inputs):
$ports = Get-content C:\AtomicRedTeam\atomics\T1016\src\top-128.txt
$file = "$env:USERPROFILE\Desktop\open-ports.txt"
$totalopen = 0
$totalports = 0
```

There are two versions of the commands listed, one with the input arguments substituted in and
one without. See the “Command” vs “Command (with inputs)” sections. The area outlined in red in the “Command” section is the input argument, the default value for this input argument
is also outlined in red in “Command (with inputs)” section. The same is true for the input argument outlined in blue.

Let’s run the test and view the output. Because we aren’t specifying any input arguments, the
default values will be used.

```
Invoke-AtomicTest T1016 -TestNumbers 5
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1016 -TestNumbers 5
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1016-5 List Open Egress Ports
Directory: C:\Users\cookies\Desktop
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/1/2023   9:27 PM              0 open-ports.txt
7 closed
9 closed
13 closed
17 closed
19 open
```

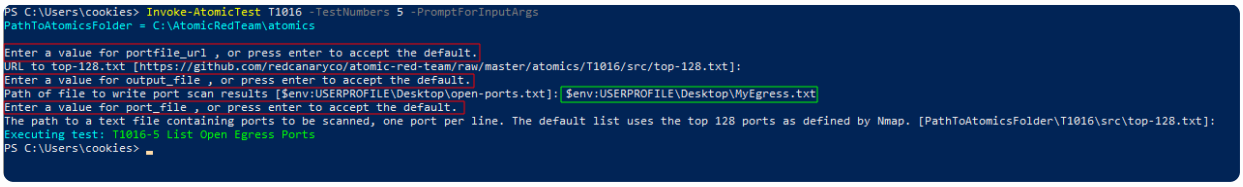
The output_file (c:\Users\cookies\Desktop\open-ports.txt) should have shown up on your desktop.

If we want to be prompted to enter our own values for the input arguments, use the "PromptForInputArgs" flag.

```
Invoke-AtomicTest T1016 -TestNumbers 5 -PromptForInputArgs
```

Here we are prompted to enter a value for each of the three input arguments as shown in red.
The default value for the input argument is shown inside the square brackets and can be accepted by just pressing Enter or Return. In this example we set one custom value for the output_file argument as shown in green.

```
$env:USERPROFILE\Desktop\MyEgress.txt
```

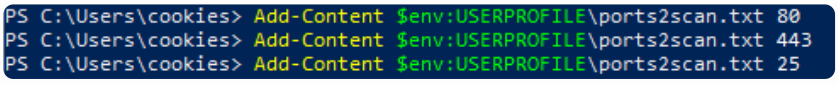


Now when we execute the test it will write the open ports to a file on the Desktop called "MyEgress.txt"

Being prompted interactively to enter values for input arguments is handy for interactive execution but there is also another way to specify input arguments which is especially useful
when using a non-interactive script to execute the test.

For this example, we will create a new ports.txt file with only a few ports in the list. The three
commands below will add 80, 443 and 25 to a file called ports2scan.txt

```
Add-Content $env:USERPROFILE\ports2scan.txt 80
Add-Content $env:USERPROFILE\ports2scan.txt 443
Add-Content $env:USERPROFILE\ports2scan.txt 25
```



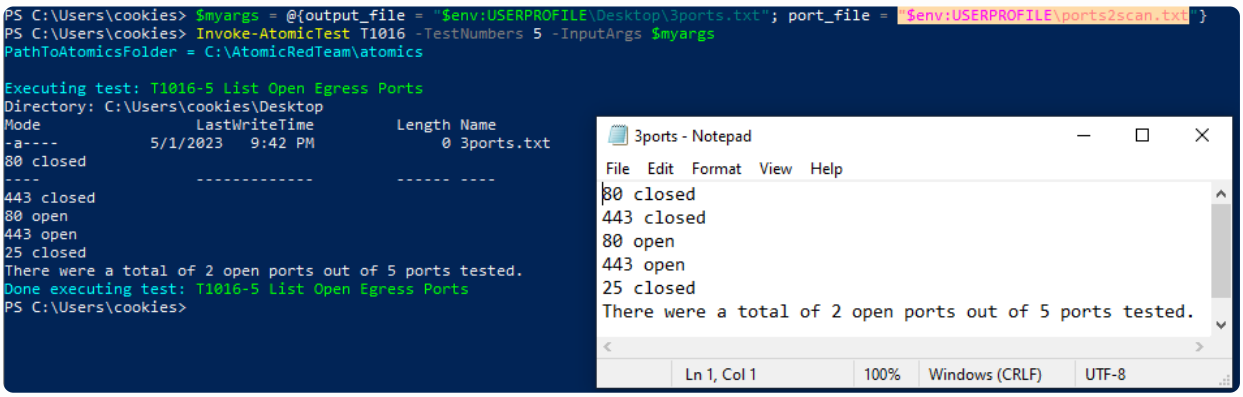
You can use the "cat" command to print the contents of your new ports2scan.txt file as shown
below.

```
cat $env:USERPROFILE\ports2scan.txt
```

```
PS C:\Users\cookies> cat $env:USERPROFILE\ports2scan.txt
80
443
80
443
25
```

We will use a variable called “myargs” to store our custom arguments which we will then pass
to the Invoke-AtomicTest call.

```
$myargs = @{output_file = "$env:USERPROFILE\Desktop\3ports.txt";
port_file = "$env:USERPROFILE\ports2scan.txt"}

Invoke-AtomicTest T1016 -TestNumbers 5 -InputArgs $myargs
```



We specified two of the three input arguments to use our own custom values instead of their
default value.

```
PS C:\Users\cookies> $myargs

Name                           Value
----                           -----
port_file                      C:\Users\cookies\ports2scan.txt
output_file                    C:\Users\cookies\Desktop\3ports.txt
```

With the $myArgs variable, we specified a port_file of:

C:\Users\cookies\port2scan.txt

And an output_file of:

C:\Users\cookies\Desktop\3ports.txt

This caused the test to check egress on 3 ports (80, 443, and 25) and to write its output to “3ports.txt” on the Desktop. This was all done without being interactively prompted for the input argument values.

This completes the custom input arguments lab. In the next lab we will be learning how to run

the cleanup commands in order to do some post-execution clean up.

- [x] End of Part 6

Part 7: Cleanup

Objective: Run the cleanup commands after executing an atomic test to reset the system and
prepare it for executing the test again.

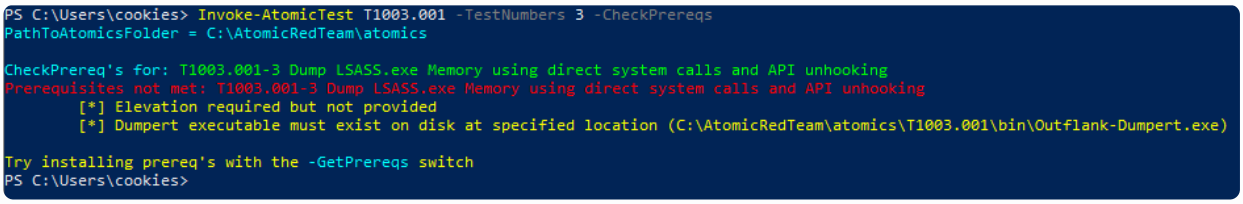
Lab VMs Needed: Windows PowerShell.

Instructions:

Some atomic tests create files that may contain sensitive information or otherwise clutter up the file system. Other tests may change settings to insecure values or stop services. It is often
desirable to delete the files created during atomic test execution or otherwise reset the system
to normal operating parameters. Many of the atomic tests include “cleanup_commands” which
do exactly that and we can use the execution framework to run these commands.

Let’s run through a full example of checking and satisfying prereqs, executing a test, and finally
cleaning up. For this we will use the “Dump LSASS.exe Memory using direct system calls and
API unhooking” test from T1003.001. First, check the prerequisites.

```
Invoke-AtomicTest T1003.001 -TestNumbers 3 -CheckPrereqs
```



As can be seen, in order to run the test to dump the LSASS.exe memory, we need to satisfy two prerequisites. The first can be satisfied by running PowerShell as an administrator. The second prerequisite specifies that the “Dumpert” executable must be found at the specified location. In order to obtain the Dumpert executable, we can run the test with the “GetPrereqs”
flag, as we did in previous labs.

```
Invoke-AtomicTest T1003.001 -TestNumbers 3 -GetPrereqs
```

```
PS C:\Users\cookies> Invoke-AtomicTest T1003.001 -TestNumbers 3 -GetPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

GetPrereq's for: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
Elevation required but not provided
Attempting to satisfy prereq: Dumpert executable must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe)
Prereq successfully met: Dumpert executable must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe)
PS C:\Users\cookies>
```

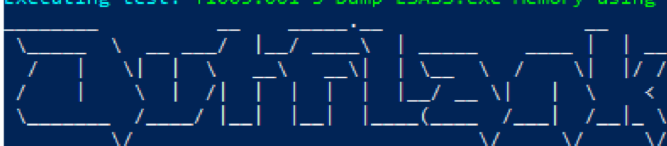
We have now satisfied the second prerequisite, but it still complains that elevation is required.
To solve this, we just need to start PowerShell as an administrator. Right click on the "powershell" shortcut on the desktop and click "Run as administrator".

Let's check prerequisites again.

```
PS C:\Windows\system32> Invoke-AtomicTest T1003.001 -TestNumbers 3 -CheckPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

CheckPrereq's for: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
Prerequisites met: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
PS C:\Windows\system32>
```

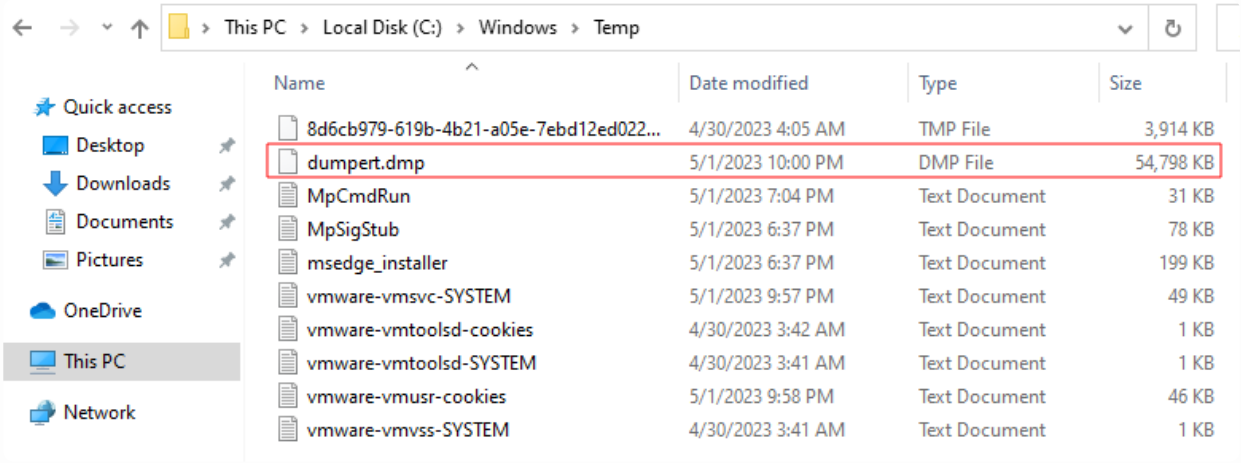
Now that the prerequisites have been met, we can run the test.

```
PS C:\Windows\system32> Invoke-AtomicTest T1003.001 -TestNumbers 3
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
```



```
                                  Dumpert
                               By Cneeliz @Outflank 2019
[1] Checking OS version details:
        [+] Operating System is Windows 10 or Server 2016, build number 19043
        [+] Mapping version specific System calls.
[2] Checking Process details:
        [+] Process ID of lsass.exe is: 668
        [+] NtReadVirtualMemory function pointer at: 0x00007FF82B74D4F0
        [+] NtReadVirtualMemory System call nr is: 0x3f
        [+] Unhooking NtReadVirtualMemory.
[3] Create memorydump file:
        [+] Open a process handle.
        [+] Dump lsass.exe memory to: \??\C:\Windows\Temp\dumpert.dmp
        [+] Dump succesful.
Done executing test: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
PS C:\Windows\system32>
```

You see that the lsass dump file was created in the location indicated -
`C:\Windows\Temp\dumpert.dmp`



Sometimes after running Atomic tests there are artifacts that are left over that are sensitive in
nature. This is one of those examples. We don't want to leave the dump file laying around. After you have finished the test, you can run the cleanup command to make sure the lsass dump
is not left on disk. We can check the details for the test to see what the cleanup command will
do.

```
Cleanup Commands:
Command:
del C:\windows\temp\dumpert.dmp >nul 2> nul
```

The cleanup command simply deletes the dumpert.dmp file, without out printing out any errors
if it doesn't exist.

In order to clean up after the test, we invoke the test with the "Cleanup" flag.

```
Invoke-AtomicTest T1003.001 -TestNumbers 3 -Cleanup
```

```
PS C:\Windows\system32> Invoke-AtomicTest T1003.001 -TestNumbers 3 -Cleanup
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing cleanup for test: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
Done executing cleanup for test: T1003.001-3 Dump LSASS.exe Memory using direct system calls and API unhooking
PS C:\Windows\system32>
```

Look at the location where the file was saved to make sure it has been removed.

```
ls C:\Windows\Temp
```

This completes the lab on executing cleanup commands. You have now learned how to use the

execution framework to execute atomic tests, including setup, cleanup, and custom inputs.

- [x] End of Part 7

Part 8: Bluespawn EDR

Bluespawn as a stand-in for an EDR system. Normally full EDRs like Cylance and Crowdstrike are very expensive and tend not to show up in classes like this. However, the folks at University of Virginia have done an outstanding job with BlueSpawn.

BlueSpawn will monitor the system for "weird" behavior and note it when it occurs. We will be starting BlueSpawn and then running Atomic Red Team to trigger a lot of alerts.

Before we begin, you can download BLUESPAWN from the link provided below. Please keep in mind the architecture of your operating system before downloading it.

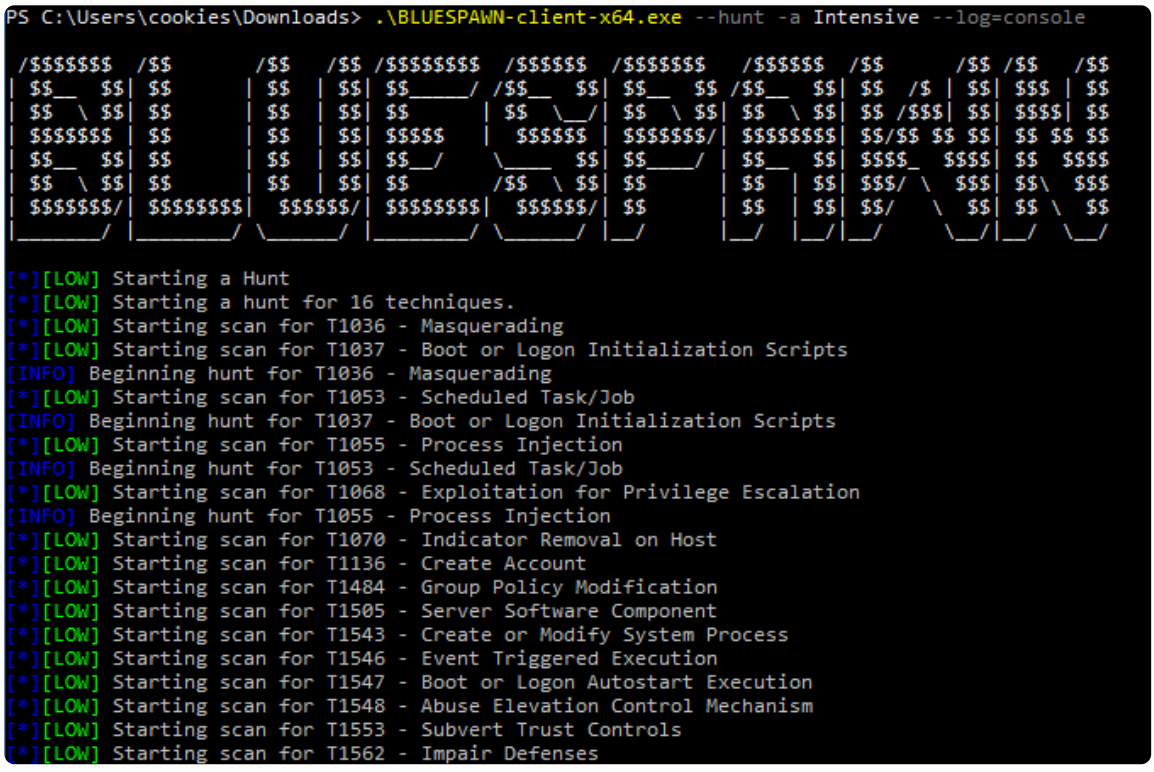
- https://github.com/ION28/BLUESPAWN/releases

First, let's disable Defender. Simply run the following from an Administrator PowerShell prompt:

```
Set-MpPreference -DisableRealtimeMonitoring $true
```

Let's get started by opening a Terminal as Administrator:

```
.\BLUESPAWN-client-x64.exe --hunt -a Intensive --log=console
```

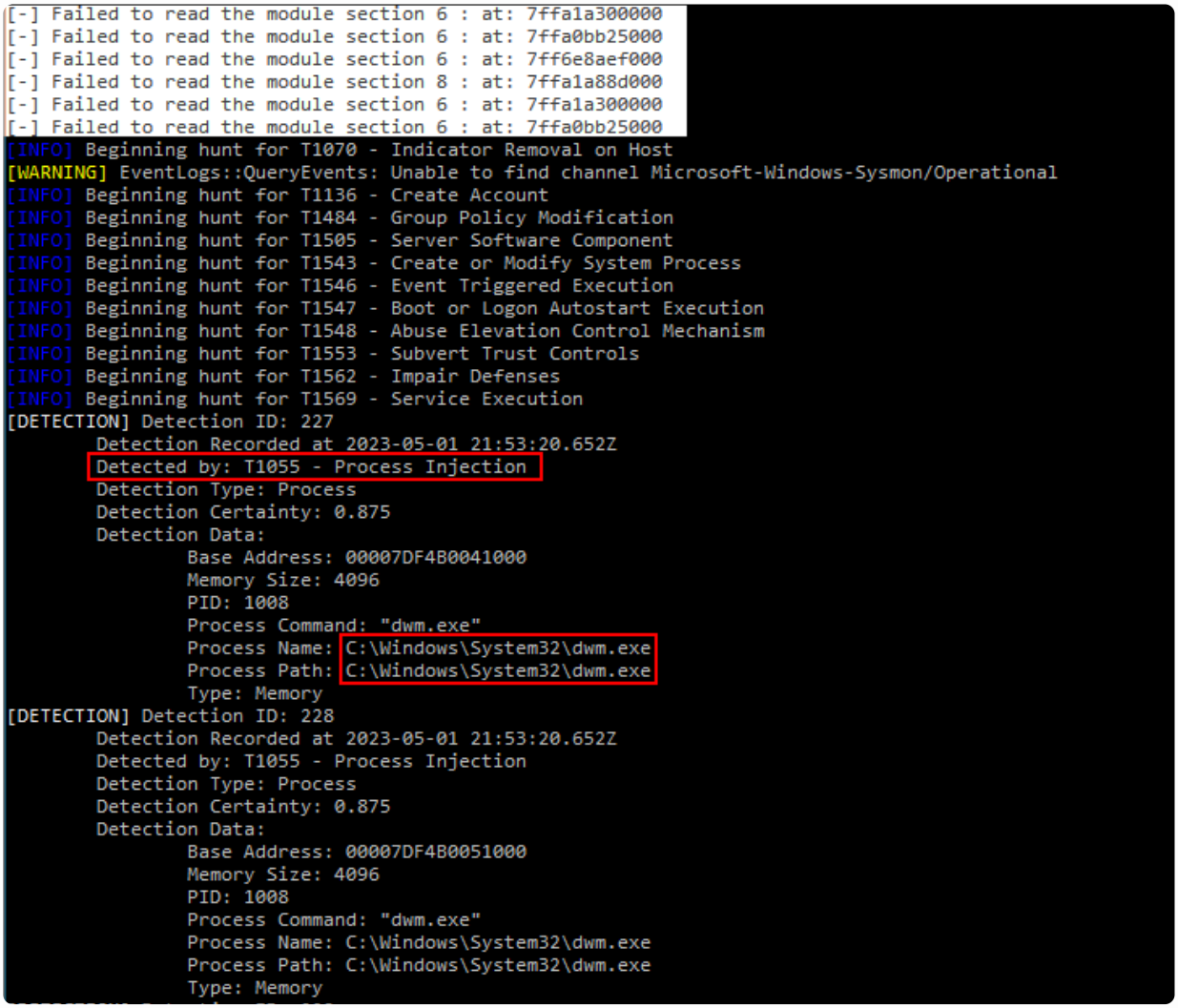


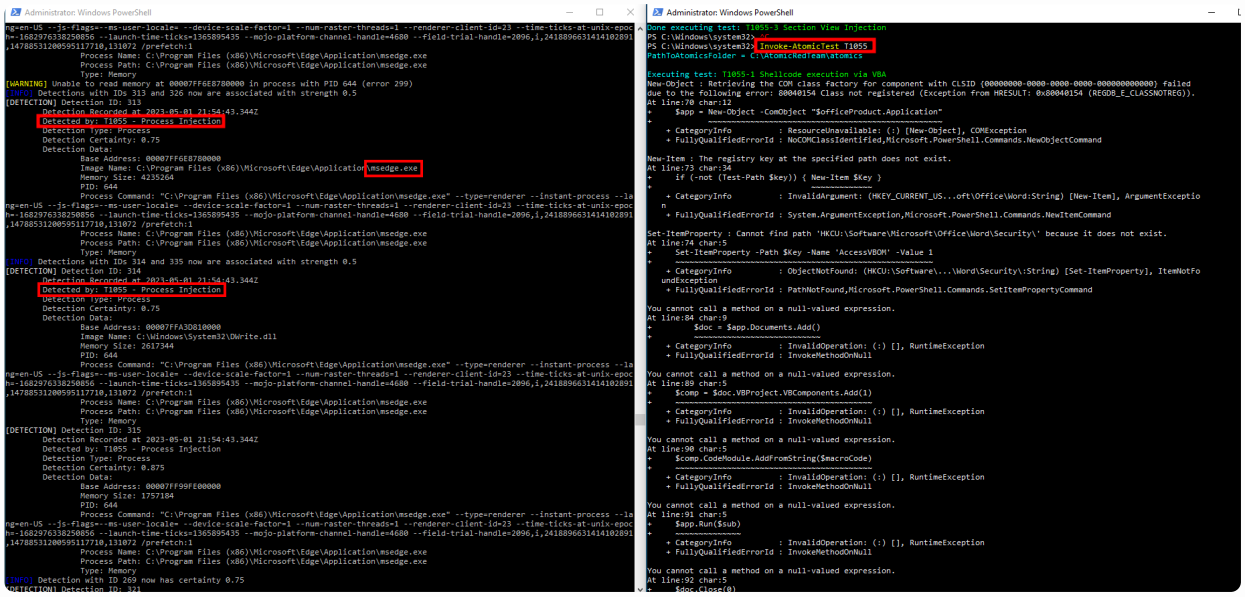
Now, let's use Atomic Red Team to test the monitoring with BlueSpawn, we need to invoke all the Atomic Tests.

Special note... Don't do this in production... Ever. Always run tools like Atomic Red Team on test systems. We recommend that you run in on a system with your EDR/Endpoint protection in non-blocking/alerting mode. This is so you can see what the protection would have done, but it will allow the tests to finish.

```
Invoke-AtomicTest T1055

OR

Invoke-AtomicTest All
```

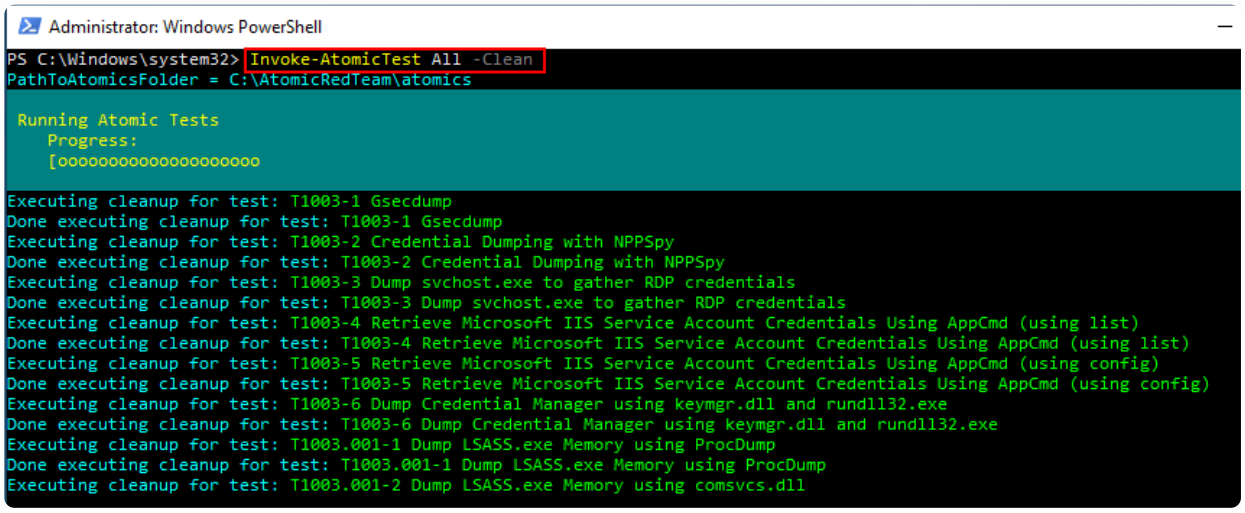
You should be getting a lot of alerts with Bluespawn :





Now, let's go back to the PowerShell prompt and clean up:

```
Invoke-AtomicTest All -Cleanup
```



☑ End of Part 8