Tuning: Endpoint Standard Policy Good, Better, Best

Table of contents

Tuning: Endpoint Standard Policy Good, Better, Best

Introduction

Overview

Good, better, best is a stair-step approach to achieving a restrictive policy that fits your organizational needs. This resource built based on the knowledge our Professional Service Consultant team gains with each customer implementation. The intention of this guide is to help customers start thinking about how they can utilize their increased visibility to build advanced policies.

Good, better, best is just one method for getting your devices into more restrictive policies. There are other methods, and as we design new TTP's and other indicators even this approach may change. We will update this post as we learn and adapt.

This tutorial takes you through considerations and best practices when tuning your security policies within VMware Carbon Black Endpoint Standard.

Audience

This tutorial is intended for Carbon Black Cloud Administrators who are familiar with VMware Carbon Black Cloud Standard. Understanding your Corporate security goals is also helpful.

Pre-Requisites

Ensure that you have a subset of your environment currently deployed.

It is recommended to review Configuration vs. Interop guidance to understand order of operations related to reputation approvals.

Policy Overview

Predefined Policies

Predefined policies are devised as templates for common use cases. You can assign sensors to these policies, change the policy settings, or duplicate the settings to create a new policy. You cannot delete predefined policies.

Policy	Description	Note
Monitored	Monitors endpoint application activity and logs events to the Dashboard. This policy has no preventive capabilities.	Use the data that this policy provides to evaluate policy rule implementation needs.
Standard	Blocks known and suspected malware, and prevents risky operations like memory scraping and code injections. Newly deployed sensors are assigned this policy by default. *It is the recommended starting point for new deployments*.	Review and refine the Standard policy rules to avoid unnecessary blocks or false positives that are triggered by in-house or custom software applications, which may have reputations that the Carbon Black Cloud does not recognize.
Advanced	Extends the capabilities of the Standard policy. It blocks operations from system utilizing, and prevents from riskier behaviors that are more likely to be false positives.	Use a phased roll-out approach to implement any new or Advanced policy rules. We recommend assigning Advanced policies to a group of pilot endpoints, and watching for false positives or blocks on legitimate software before rolling them out to more endpoints.

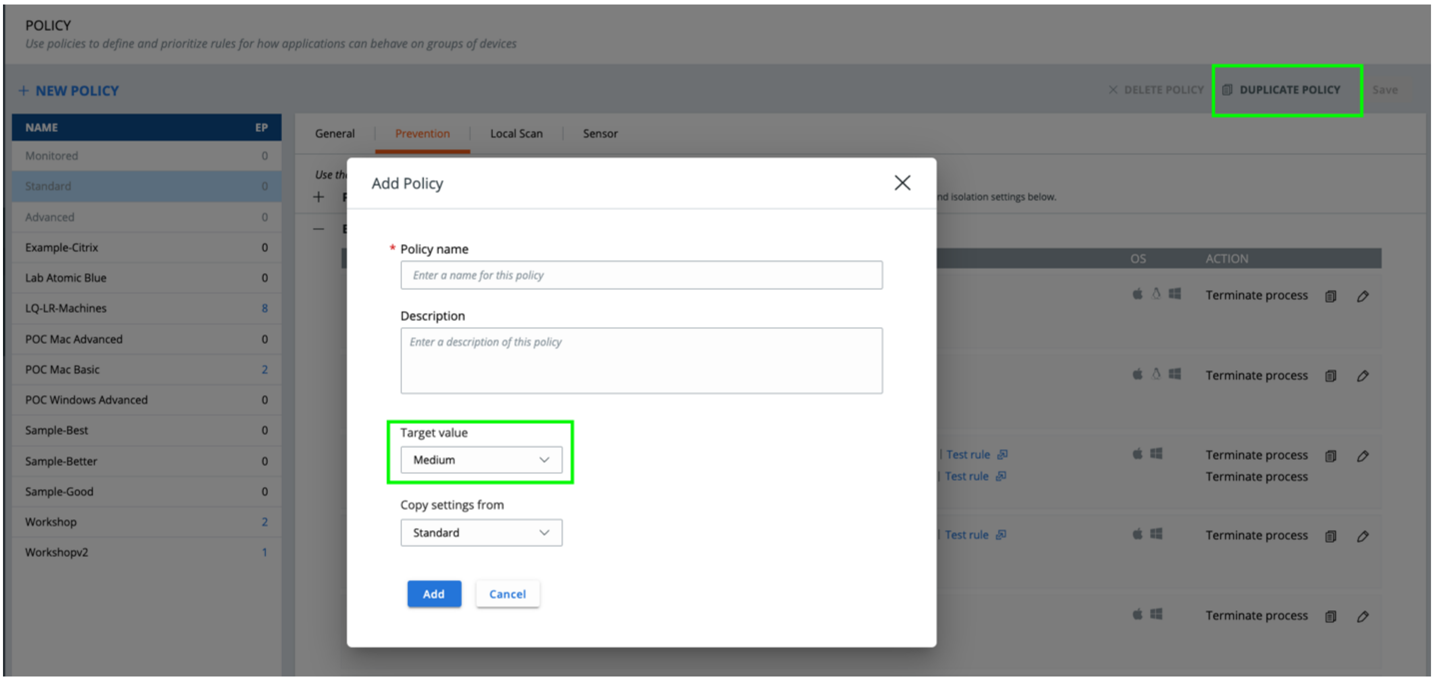
Based on your organizations risk threshold start in the predefined policy that best suits those needs.

I.E. If your organization currently has an AV deployed and you working towards a migration plan starting in Monitored with no prevention would work as you have some signature-based protection with the AV solution and visibility through VMware Carbon Black Cloud. If you don’t have any protection on your assets it would be advisable to begin in Standard and tune with the below guidance.

Duplicating a Predefined Policy

Duplicate your desired policy.

Name the copy something meaningful (e.g. “MyCorp-Standard”) assign a target value of Medium to match the other predefined policies. Make all edits and modifications in the duplicated policy so it is clear as to what changes have been made.



POLICY

Use policies to define and prioritize rules for how applications can behave on groups of devices

NAME	EP
Monitored	0
Standard	0
Advanced	0
Example-Citrix	0
Lab Atomic Blue	0
LQ-LR-Machines	8
POC Mac Advanced	0
POC Mac Basic	2
POC Windows Advanced	0
Sample-Best	0
Sample-Better	0
Sample-Good	0
Workshop	2
Workshopv2	1

Add Policy

* Policy name: *Enter a name for this policy*

Description: *Enter a description of this policy*

Target value: Medium

Copy settings from: Standard

Achieving “Good”

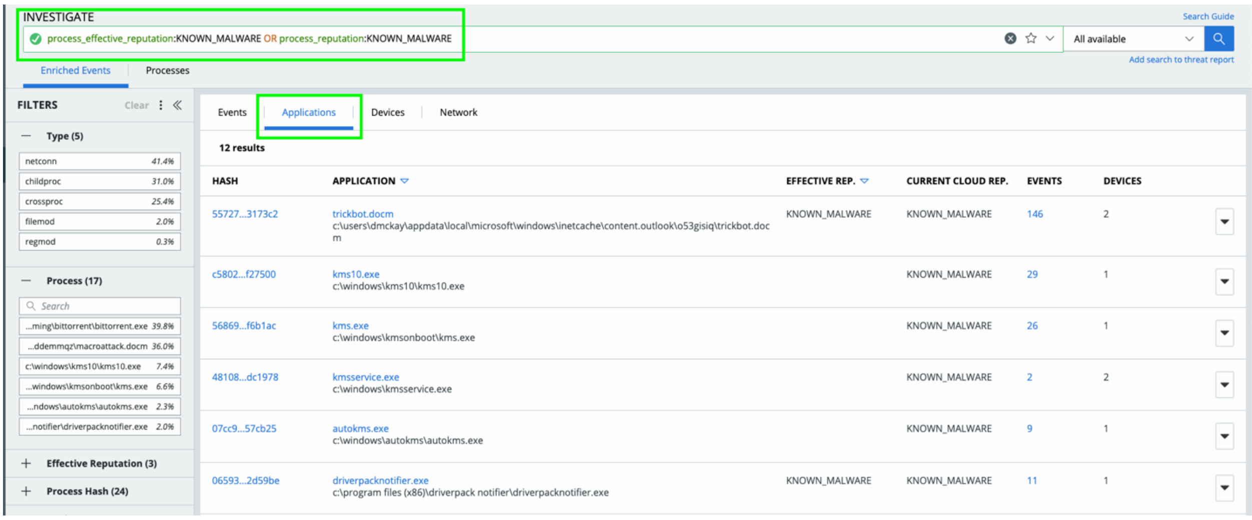
Blocking Known and Suspect Bad Files.

Start by identifying all files classified as **Known Malware** in your organization.

On the Investigate Tab, run the following query:

```
process_effective_reputation:KNOWN_MALWARE OR process_reputation:KNOWN_MALWARE
```

Sort to the Applications Tab.



INVESTIGATE

process_effective_reputation:KNOWN_MALWARE OR process_reputation:KNOWN_MALWARE

Type (5)

netconn	41.4%
childproc	31.0%
crossproc	25.4%
filemod	2.0%
regmod	0.3%

Process (17)

...ming\bittorrent\bittorrent.exe	39.8%
...ddemmqz\macroattack.docm	36.0%
c:\windows\kms10\kms10.exe	7.4%
...windows\kmsonboot\kms.exe	6.6%
...ndows\autokms\autokms.exe	2.3%
...notifier\driverpacknotifier.exe	2.0%

Effective Reputation (3)

Process Hash (24)

12 results

HASH	APPLICATION	EFFECTIVE REP.	CURRENT CLOUD REP.	EVENTS	DEVICES
55727...3173c2	trickbot.docm c:\users\dmckay\appdata\local\microsoft\windows\inetcache\content.outlook\o53gisiq\trickbot.docm	KNOWN_MALWARE	KNOWN_MALWARE	146	2
c5802...f27500	kms10.exe c:\windows\kms10\kms10.exe		KNOWN_MALWARE	29	1
56869...f6b1ac	kms.exe c:\windows\kmsonboot\kms.exe		KNOWN_MALWARE	26	1
48108...dc1978	kmsservice.exe c:\windows\kmsservice.exe		KNOWN_MALWARE	2	2
07cc9...57cb25	autokms.exe c:\windows\autokms\autokms.exe		KNOWN_MALWARE	9	1
06593...2d59be	driverpacknotifier.exe c:\program files (x86)\driverpack notifier\driverpacknotifier.exe	KNOWN_MALWARE	KNOWN_MALWARE	11	1

Here you are able to review any application identified as KNOWN_MALWARE, review for any concerns/ to implement approvals only if needed.

Next you will want to identify any **PUPs** and **Suspect Malware** within your environment.

On Investigate Tab, run the following queries:

```
process_effective_reputation:PUP OR process_reputation:PUP
```

PUP’s or Potentially Unwanted Programs are just that, an application that can cause machines to run slowly, display unexpected ads, or can be used to install other software which may be harmful to your system. Because PUP’s introduce poor hygiene to your environment it is important to review the applications for validity and block when possible. If you decide that an app (with the reputation of a PUP) is something we want to let run we can approve the hash or allow the file. Or, if you are unsure, create an "Allow and Log" and allow this to run but keep an eye on it.

Allow and Log Tips:

*We need to get the path in which it is running. We can be very specific if we only want this file to run on a certain person’s machine, or we can make it so it could run out of multiple people’s user download file. We will assume in the below example we want this file to run and the username is “Joe,” or that we can have multiple users run this file by using a wildcard * in place of a username.*

For example…

**\Users\Joe\Downloads\Setup_DriverDoc_2015.exe

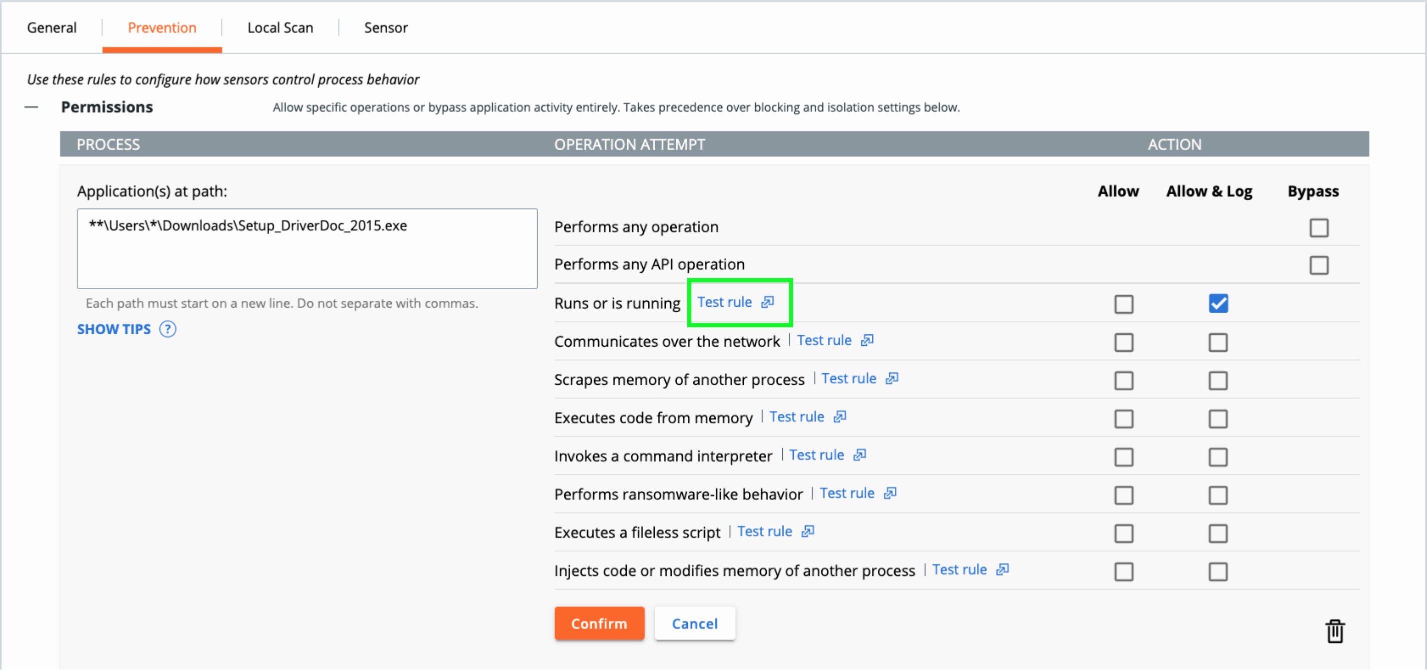
**\Users*\Downloads\Setup_DriverDoc_2015.exe

Another few examples could be like the below:

**\Program Files*\MediaMall*.exe

**\Program Files (x86)\MediaMall\**

*NOTE: Use of wildcard characters * matches one or multiple characters within the same directory level. Use of double asterisk (**) matches one or multiple directory levels.*



General | Prevention | Local Scan | Sensor

Use these rules to configure how sensors control process behavior

— **Permissions** Allow specific operations or bypass application activity entirely. Takes precedence over blocking and isolation settings below.

PROCESS	OPERATION ATTEMPT	Allow	Allow & Log	Bypass
Application(s) at path: **\Users*\Downloads\Setup_DriverDoc_2015.exe Each path must start on a new line. Do not separate with commas. SHOW TIPS ?	Performs any operation			☐
	Performs any API operation			☐
	Runs or is running | Test rule	☐	☑	
	Communicates over the network | Test rule	☐	☐	
	Scrapes memory of another process | Test rule	☐	☐	
	Executes code from memory | Test rule	☐	☐	
	Invokes a command interpreter | Test rule	☐	☐	
	Performs ransomware-like behavior | Test rule	☐	☐	
	Executes a fileless script | Test rule	☐	☐	
	Injects code or modifies memory of another process | Test rule	☐	☐	

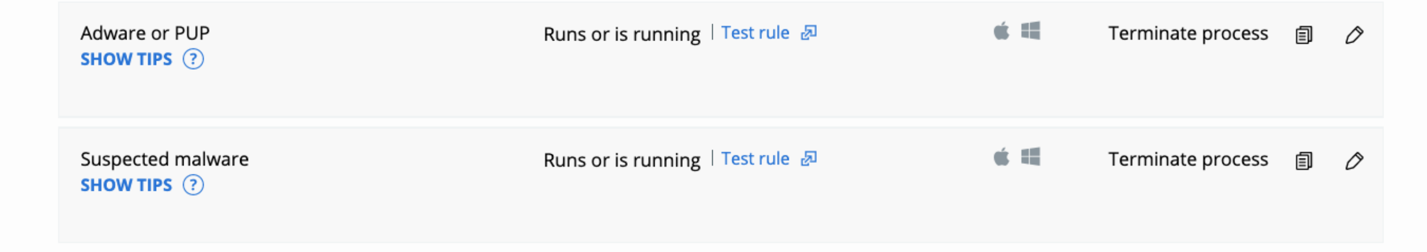
Be sure to run "Test rule" to verify it is working as expected. Review Configuration vs. Interoperability to understand how to handle permissions vs. allow listing and the order of operations the Carbon Black Cloud sensor follows.

Repeat for **Suspect Malware**.

```
process_effective_reputation:SUSPECT_MALWARE OR process_reputation:SUSPECT_MALWARE
```

PRO TIP: *any of these queries can be saved by selecting the star in the query bar. Option to save for your enterprise or individually.*

Once review is complete add the following rule to your policy (note this currently exists in both the Standard and Advanced predefined policies):



Adware or PUP SHOW TIPS ?	Runs or is running | Test rule	Terminate process
Suspected malware SHOW TIPS ?	Runs or is running | Test rule	Terminate process

Now we are making some progress! We've gone from "good" to "better," and now let's start working from "better" to the "best" type of policies.

Achieving “Better”

Blocking Unknown and Not Listed Applications.

Process	Definition
Unknown Application	Application reputation set to Unknown, for example, a new application added when the sensor was offline or unable to connect
Not Listed Application	No reputation information to supply to the sensor; typically means the hash is new. Helps protect against zero-day malware and is frequently assigned to new hashes/updated applications.

In the Standard and Advanced policies we restrict Unknown and Not Listed applications from scraping memory of another process, and performing ransomware-like behavior because adversaries will often deploy homegrown tools to steal credentials for lateral movement and/or attempt to encrypt and destroy local data. To verify the applications that fall into this category leverage the ‘Test rule’ functionality to identify potential blocks and mitigate accordingly.

In order to achieve a “Better” policy we will want to go beyond the defaults. Injection of code is something adversaries can use to masquerade i.e. notepad (trusted application) injecting code into calc is a common demonstration but adversaries can do this with homegrown software. BUT this is something legitimate applications can use as well. Lets investigate to see if there are ways for us to get more restrictive on Unknown/Not Listed applications.

On the Investigate Tab run the following:

Not Listed applications exhibiting code injection;

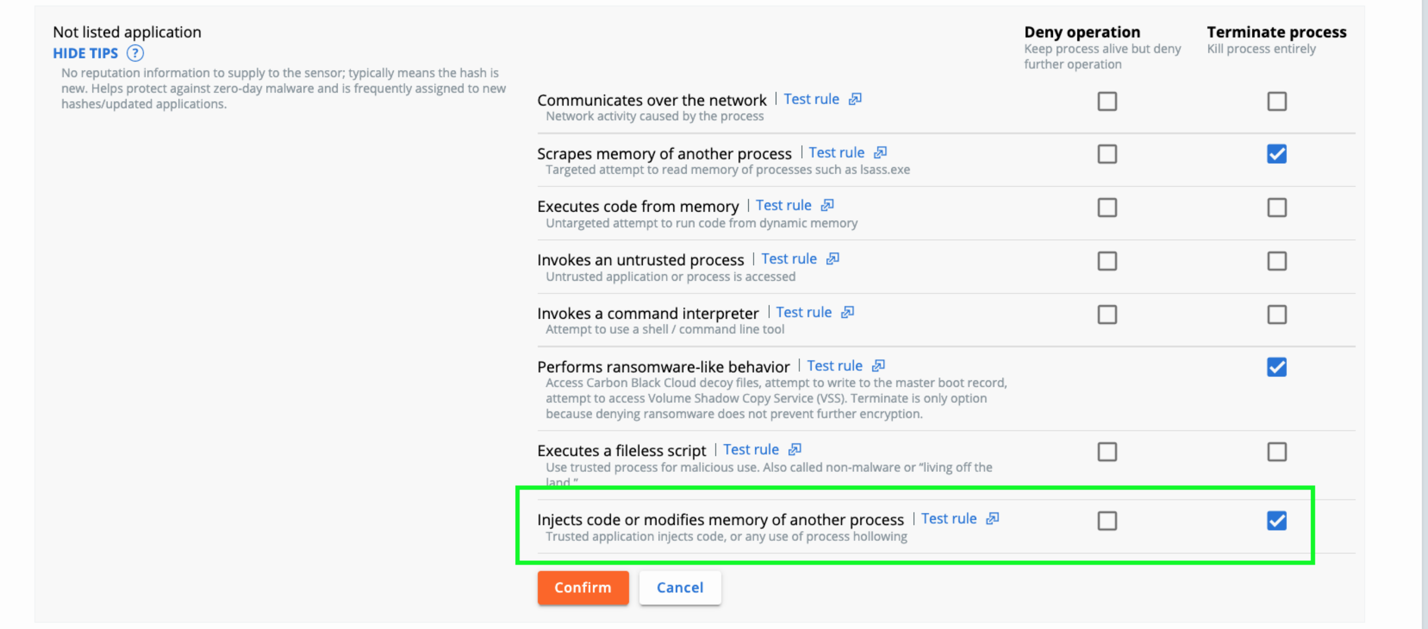
```
(process_effective_reputation:ADAPTIVE_WHITE_LIST OR process_effective_reputation:NOT_LISTED)AND(ttp:INJECT_CODE OR
ttp:HAS_INJECTED_CODE OR ttp:COMPROMISED_PROCESS OR ttp:PROCESS_IMAGE_REPLACED OR ttp:MODIFY_PROCESS OR
ttp:MODIFY_PROCESS_EXECUTION OR ttp:HOLLOW_PROCESS)
```

Unknown Applications exhibiting code injection;

```
process_effective_reputation:RESOLVING AND (ttp:INJECT_CODE OR ttp:HAS_INJECTED_CODE OR ttp:COMPROMISED_PROCESS OR
ttp:PROCESS_IMAGE_REPLACED OR ttp:MODIFY_PROCESS OR ttp:MODIFY_PROCESS_EXECUTION OR ttp:HOLLOW_PROCESS)
```

Then click on “Applications” for the time frame you have selected and review the applications list. If there are applications you want to allow to inject code or that you need to run for business purposes, you can allow these by creating a reputation approval or an allow rule. Tips types of approvals are defined in the Configuration vs Interoperability resource.

Work through the list and again, as always, test on a small subset of machines displaying this kind of activity.



Not listed application

No reputation information to supply to the sensor; typically means the hash is new. Helps protect against zero-day malware and is frequently assigned to new hashes/updated applications.

	Deny operation Keep process alive but deny further operation	**Terminate process** Kill process entirely
Communicates over the network | Test rule Network activity caused by the process	□	□
Scrapes memory of another process | Test rule Targeted attempt to read memory of processes such as lsass.exe	□	☑
Executes code from memory | Test rule Untargeted attempt to run code from dynamic memory	□	□
Invokes an untrusted process | Test rule Untrusted application or process is accessed	□	□
Invokes a command interpreter | Test rule Attempt to use a shell / command line tool	□	□
Performs ransomware-like behavior | Test rule Access Carbon Black Cloud decoy files, attempt to write to the master boot record, attempt to access Volume Shadow Copy Service (VSS). Terminate is only option because denying ransomware does not prevent further encryption.		☑
Executes a fileless script | Test rule Use trusted process for malicious use. Also called non-malware or "living off the land."	□	□
Injects code or modifies memory of another process | Test rule Trusted application injects code, or any use of process hollowing	□	☑

Other Not Listed or Unknown rules to consider? Use the 'Test Rule' capability to investigate:

- *NOT_LISTED "TRIES TO EXECUTE CODE FROM MEMORY"*
- *UNKNOWN "TRIES TO EXECUTE CODE FROM MEMORY"*
- *UNKNOWN "TRIES TO COMMUNICATE OVER THE NETWORK"*
- *NOT_LISTED "TRIES TO RUN AN UNTRUSTED PROCESS"*

Continue to explore the different behavioral prevention options to see what works best for your organization. You want to follow the concept of least privilege or zero trust anywhere possible so that you are not chasing adversarial behaviors; rather you are assessing how your organization needs to behave and restricting behaviors outside of the norm to ensure you are controlling deviations.

Lets explore this concept a bit more in 'Achieving Best' policy section.

Achieving “Best”

Restricting trusted applications to harden your security posture.

Going beyond reputation, we know that adversaries LOVE using trusted and built-in tools for malintent. Why, because if the application is trusted it will bypass many security and EDR solutions.

BUT where do you start?

A good place to start is looking at application that are commonly exploited.

PRO TIP: *I like to start with MITRE ATT&CKs Top Ten TIDs, Red Canary has a great resource for understanding the top techniques* here*.*

Command interpreters, specifically PowerShell, is a perfect target. How can we get more granular with PowerShell control? Beyond known bad behaviors with PowerShell how can we adopt the concept of least privileged?

Let’s consider PowerShell rules that can stop injection of code.

```
(process_name:pwsh.exe OR process_name:*.ps1 OR process_name:powershell*.exe OR process_name:*.psm1) AND
(ttp:INJECT_CODE OR ttp:HAS_INJECTED_CODE OR ttp:COMPROMISED_PROCESS OR ttp:PROCESS_IMAGE_REPLACED OR
ttp:MODIFY_PROCESS OR ttp:MODIFY_PROCESS_EXECUTION OR ttp:HOLLOW_PROCESS OR ttp:RAM_SCRAPING OR
ttp:READ_SECURITY_DATA)
```

PRO TIP: *You can approve by hash or allow rules on your PowerShell scripts so that they will never have a reputation of NOT_LISTED or UNKNOWN.*

How about PowerShell communicating over the network? Developers will often need this to pull something from GitHub or make additional calls with their scripts, this may not be as common with non-developer roles. As we know adversaries will use PowerShell to call to their C2 servers, so considering least privilege can we restrict this function on all devices or on a subset of devices?

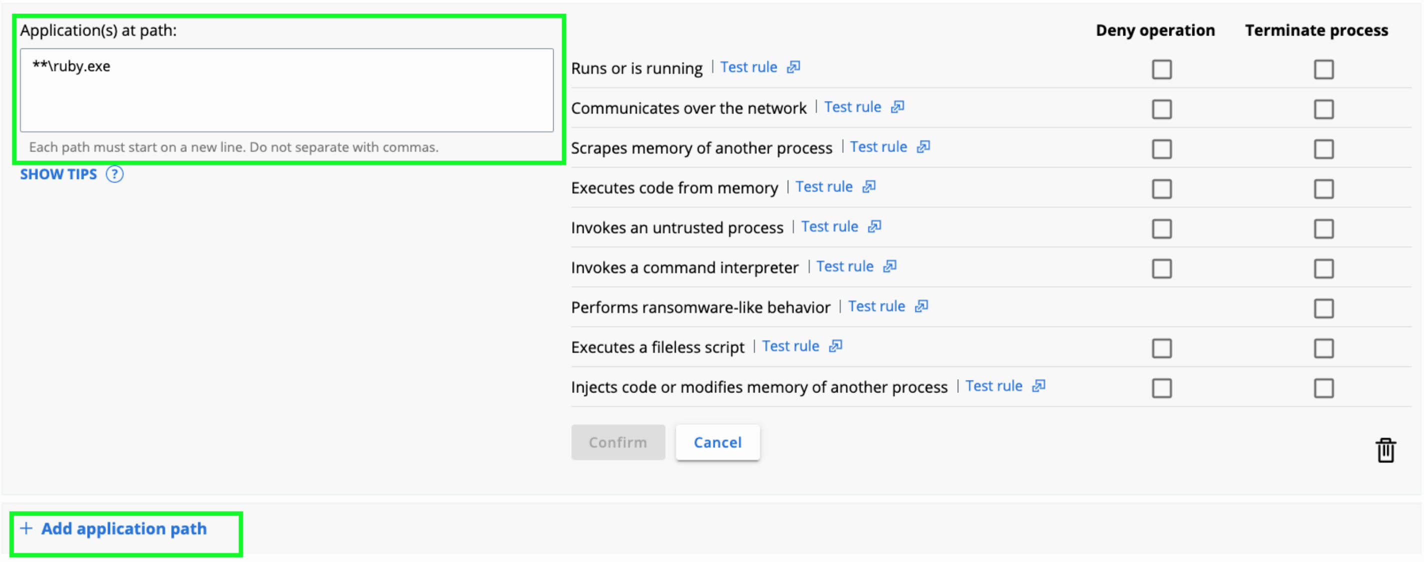
```
(process_name:pwsh.exe OR process_name:*.ps1 OR process_name:powershell*.exe OR process_name:*.psm1)AND
(childproc_name:cmd.exe OR childproc_name:powershell.exe OR childproc_name:cscript.exe OR childproc_name:wscript.exe
OR childproc_name:wmic.exe OR childproc_name:mshta.exe OR childproc_name:sh OR childproc_name:zsh OR
childproc_name:csh OR childproc_name:bash OR childproc_name:tcsh OR childproc_name:python)
```

Other applications to consider?

- Ruby
- Office Applications
- Java
- Web Browsers
- Your Homegrown Applications

Repeat the investigate process with your desired applications.

Leveraging the ‘test rule’ capability you can continue forming these queries across your enterprise without needing to format the query syntax.



Application(s) at path:

```
**\ruby.exe
```

Each path must start on a new line. Do not separate with commas.

SHOW TIPS ?

	Deny operation	Terminate process
Runs or is running | Test rule	□	□
Communicates over the network | Test rule	□	□
Scrapes memory of another process | Test rule	□	□
Executes code from memory | Test rule	□	□
Invokes an untrusted process | Test rule	□	□
Invokes a command interpreter | Test rule	□	□
Performs ransomware-like behavior | Test rule		□
Executes a fileless script | Test rule	□	□
Injects code or modifies memory of another process | Test rule	□	□

Confirm Cancel

+ Add application path

Conclusion

Summary

Good, better, best is a stair-step approach to achieving a restrictive policy that fits your organizational needs. This resource is built based on the knowledge our Professional Service Consultant team gains with each customer implementation. The intention of this guide is to help customers start thinking about how they can utilize their increased visibility to build advanced policies.

Good: *Blocking Known and Suspect Bad Files.*

Better: *Blocking Unknown and Not Listed Applications.*

Best: *Restricting trusted applications to harden your security posture.*

About the Author

Kirk Hasty; My name is Kirk Hasty, and I am a Sr. Technical Product Manager for VMware Carbon Black. My passion is helping our customers continually improve their security posture. I have helped our customers in the field as a consultant and a solution architect before moving over to the product side. I have worked in multiple companies as an IT administrator and as a cyber security SOC person. I enjoy teaching others how to review their network and focusing on how to help them protect their corporate environment. We are stronger as a community than individually, so let's help each other get better and learn together.