Matthew Green

I am a DFIR and detection guy from Sydney Australia.

(sitemap)~$ type to search

Navigation

» Home
» About Me
» Talks / Projects
» XML Feed

Binary Rename 2

29 May 2019 » posts

This is my second Binary Rename post, for the first and a detailed description of what Binary Rename is, please see: Blue Team Hacks - Binary Rename.

In my first post, I talked about telemetry and how we could leverage WMI Eventing for a niche detection usecase. In this post I am focusing on static detection, that is assessing files on disk. I am going to describe differences between both Yara and Powershell based detections, then share the code.

Yara Detection

Firstly Yara - Yara is a command line driven tool used mainly for pattern matching in malware or detection use cases. Rule based, though strings or binary patterns - matching can be leveraged with logic like boolean, counts or regular expressions. Although traditionally pattern based, Yara is modular and expandable such that a “PE” module is available focusing on querying common binary attributes. The PE module allows you to create rules targeted specifically to the PE file format and file headers, providing functions which can be used to write more effective rules for PE file use cases.

The example I am using is leveraging pe.versioninfo InternalName attribute:

```
import "pe"

rule renamedCMD
{
    condition:
        pe.version_info["InternalName"] == "cmd" and
        not filename == "cmd.exe"
}
```

PE module import and InternalName rule for renamed cmd.exe

Our Yara use case is interesting as we require to compare an expected filename with the actual filename which is not typically a Yara capability. Florian Roth wrote about an “inverse” technique back in 2014 leveraging a Powershell script to obtain all files to be scanned and pass each filename into a yara scan as an external variable. The idea is a new yara instance is created for each file, passing in the relevant filename as the variable to allow comparison. In my code below I have expanded out the use case to cover x32 and x64 bit machines.

```
If ([System.Environment]::Is64BitOperatingSystem) {
    Get-ChildItem -Recurse -filter *.exe \ -ErrorAction SilentlyContinue |
        ForEach-Object {
            .\yara64.exe -d filename=$(($_.Name).ToLower()) rename.yar $_.FullName 2> $null
        }
}
Else {
    Get-ChildItem -Recurse -filter *.exe \ -ErrorAction SilentlyContinue |
        ForEach-Object {
            .\yara32.exe -d filename=$(($_.Name).ToLower()) rename.yar $_.FullName 2> $null
        }
}
```

Powershell: inverseYara.ps1

For execution we require the following files in our execution path:

- inverseYara.ps1
- yara binaries x86 or x64
- rename.yar

Then execution via a bat file or commandline as below:

```
c:\Users\REM\Desktop>powershell -ExecutionPolicy ByPass -f inverseYara.ps1
renamed7zip C:\staging\not7z32.exe
renamed7zip C:\staging\not7z64.exe
renamedCertutil C:\staging\notcertutil32.exe
renamedCertutil C:\staging\notcertutil64.exe
renamedCmstp C:\staging\notcmstp32.exe
renamedCmstp C:\staging\notcmstp64.exe
renamedCscript C:\staging\notcscript32.exe
renamedCscript C:\staging\notcscript64.exe
renamedMshta C:\staging\notmshta32.exe
renamedMshta C:\staging\notmshta64.exe
renamedMsiexec C:\staging\notmsiexec32.exe
renamedMsiexec C:\staging\notmsiexec64.exe
renamedPowershell C:\staging\notpowershell32.exe
renamedPowershell C:\staging\notpowershell64.exe
renamedPsExec C:\staging\notPsExec.exe
renamedPsExec C:\staging\notPsExec64.exe
renamedWinRar C:\staging\notrar32.exe
renamedWinRar C:\staging\notrar64.exe
renamedRegsvr32 C:\staging\notregsvr32.exe
renamedRegsvr32 C:\staging\notregsvr3264.exe
renamedRundll32 C:\staging\notrundll32.exe
renamedRundll32 C:\staging\notrundll3264.exe
renamedwmic C:\staging\notWMIC32.exe
renamedwmic C:\staging\notWMIC64.exe
renamedWscript C:\staging\notwscript32.exe
renamedWscript C:\staging\notwscript64.exe
renamedCMD C:\Windows\System32\cdm.exe
TotalSeconds = 75.0241635
```

Yara: Binary Rename detection results

This technique works very well from a detection standpoint, however in my testing performance does not appear to be optimal due to the overhead of generating a new yara process for each file scanned. It is worthy to note, the yara scan could be targeted without the filename match focusing on unexpected locations for the files in scope, but this doesn't match the binary rename usecase as required.

Powershell Detection

In this case, the preferred detection is moving to Powershell only. The Windows API provides access to PE attributes via the FileVersionInfo Class with support back to Powershell 2.0 /.NET 2. Speed is significantly improved and logic can be optimised adding additional items in the output that may aid analysis. In my script output below you can see I have added a sha1 hash to the output object.

```
FileVersionRaw     : 5.2.3790.3959
ProductVersionRaw  : 5.2.3790.3959
Comments           :
CompanyName        : Microsoft Corporation
FileBuildPart      : 3790
FileDescription    : Windows Command Processor
FileMajorPart      : 5
FileMinorPart      : 2
FileName           : C:\Windows\System32\cdm.exe
FilePrivatePart    : 3959
FileVersion        : 5.2.3790.3959 (srv03_sp2_rtm.070216-1710)
InternalName       : cmd
IsDebug            : False
IsPatched          : False
IsPrivateBuild     : False
IsPreRelease       : False
IsSpecialBuild     : False
Language           : English (United States)
LegalCopyright     : © Microsoft Corporation. All rights reserved.
LegalTrademarks    :
OriginalFilename   : Cmd.Exe
PrivateBuild       :
ProductBuildPart   : 3790
ProductMajorPart   : 5
ProductMinorPart   : 2
ProductName        : Microsoft® Windows® Operating System
ProductPrivatePart : 3959
ProductVersion     : 5.2.3790.3959
SpecialBuild       :
Sha1Hash           : 4B0B3A93A52DE953AAE6BAA6BFA54D31C1EB7155

TotalSeconds = 12.7369955
```

Powershell results: 6 times faster than yara!

Limitations

The biggest limitation with any static detection capability that queries the whole disk is performance. Leveraging Powershell and native Windows API seems to optimise performance significantly. Other optimisations added are setting CPU priority to Idle only and configuring logic to filter effectively to minimise processing footprint. Additional optimisations around

performance, could be targeted queries for specific staging locations of interest as part of a targeted detection.

One consideration to keep in mind is the Powershell method leverages the Windows API. Although not a huge concern for my usecase of renamed binaries in a living off the land scenario, if there was tampering with rootkit like functionality a raw collection would be preferred.

Final Thoughts

Hopefully you will find this summary useful, closing the loop on an open source detection capability for the Binary Rename use case. Feel free to reach out if you have any feedback, questions, or improvements.

Powershell and Yara detection code can be found here - Get-BinaryRename

Further reading

1) Green, Matthew. Blue Team Hacks - Binary Rename, 2019
2) The MITRE Corporation. Technique: Masquerading - MITRE ATT&CK™
3) MSDN. FileVersionInfo
4) Roth, Florian. Inverse Yara Signature Matching (Part 1/2), 2014
5) Roth, Florian. Inverse Yara Signature Matching (Part 2/2), 2014
6) YARA v3.10.0. PE Module

Share by: Post

Related Posts

- WMI Event Consumers: what are you missing? (Categories: posts)
- Cobalt Strike Payload Discovery And Data Manipulation In VQL (Categories: posts)
- Windows IPSEC for endpoint quarantine (Categories: posts)
- Local Live Response with Velociraptor ++ (Categories: posts)
- Live response automation with Velociraptor (Categories: posts)
- O365: Hidden InboxRules (Categories: posts)

« Blue Team Hacks - Binary Rename

O365: Hidden InboxRules »

© Matthew Green - https://github.com/mgreen27 - Powered by Jekyll.