An Analysis of the Practicality of Assessing Cryptographic Hashes for Malware Detection with VirusTotal and Wazuh File Integrity Monitoring

Zachary P. Pax
Riverland Energy Cooperative
N28988 WI-93
Arcadia, WI 54612, USA
February 2024
E-mail: zpax@riverlandenergy.com

ABSTRACT

This paper explores the feasibility of integrating automated cryptographic hash verification utilizing the VirusTotal API with Wazuh File Integrity Monitoring for the purpose of detecting malware on end-user production Windows endpoints. Additionally, it examines the approach’s efficacy in promptly identifying and optionally removing the identified malware from the endpoints, preferably before execution.

INTRODUCTION

Wazuh is a free and open-source security information and event management (SIEM) platform offering extended detection and response (XDR) capabilities. Initially being forked from the Open Source HIDS SECurity (OSSEC) project, Wazuh inherits a comprehensive array of functionalities, including system event monitoring and file integrity monitoring. This tool forms an integral component within the unique network security monitoring framework, referred to as the "Watchtower Stack," employed at Riverland Energy Cooperative [1].

During my research into endpoint security measures, I discovered that Wazuh has a guide on implementing cryptographic hash verification with VirusTotal from Wazuh File Integrity Monitoring events [2]. This procedure leverages the Wazuh Server to upload the cryptographic hashes of files, as monitored by Wazuh File Integrity Monitoring on each endpoint, to the VirusTotal API to solicit a result. This result is then made into a custom rule and offers alerts in a similar fashion to other rules created at Riverland Energy Cooperative.

After considerable planning**,** I implemented Wazuh File Integrity Monitoring to alert us on the contents of each user’s downloads folder when a file is downloaded if its extension is not on a centrally managed whitelist. In addition to this, various users were given explicit permission in their OSSEC configuration files to download other file types without creating an alert. This system proved incredibly effective at immediately alerting us about suspicious file downloads. Having already had this implementation of Wazuh File Integrity Monitoring setup and running, integrating it with the VirusTotal API was relatively simple, and the documentation proved sufficient for an initial proof-of-concept.

CONCEPT

The basic concept of monitoring downloaded files with VirusTotal and Wazuh FIM is straightforward:

1. A user downloads/adds a file to a monitored directory.
2. The Wazuh Agent on the endpoint, using regular expression formatting, determines if the file has an extension on your custom-made whitelist.
3. If the file's extension is whitelisted, no further action is taken.
4. If the file's extension is not whitelisted, Wazuh FIM will immediately calculate the cryptographic hash of the downloaded file on the endpoint and send it to the Wazuh Server.
5. Upon receiving notification that a file has been downloaded, the Wazuh Server will automatically connect to the locally installed version of Postfix to send email and SMS alerts [3].
6. After sending this email to the Postfix queue, the Wazuh Server will send the cryptographic hash(es) to the VirusTotal API and await a result.
7. After receiving the result from the VirusTotal API, the Wazuh Server will send a summary of the result from the API as well as a link to the VirusTotal result to the Postfix queue.
8. Optionally, when any result (malicious, not malicious, or unknown) is returned, a script or other program can be executed on the endpoint. This can be used to remove known-malicious files from the endpoint.

   Note: Because steps five, six, and seven happen incredibly quickly in succession, Wazuh will typically merge the alerts together to create one unified alert containing information on the file as well as the VirusTotal API result.

RESPONSE TIME

When a file is downloaded, the response time from the file being transmitted and receiving confirmation if it is malicious is an important consideration when determining whether this solution is practical in your environment. Moreover, should one be using a script to remove downloaded files identified as malicious, it is important that this is done promptly to prevent further infection.

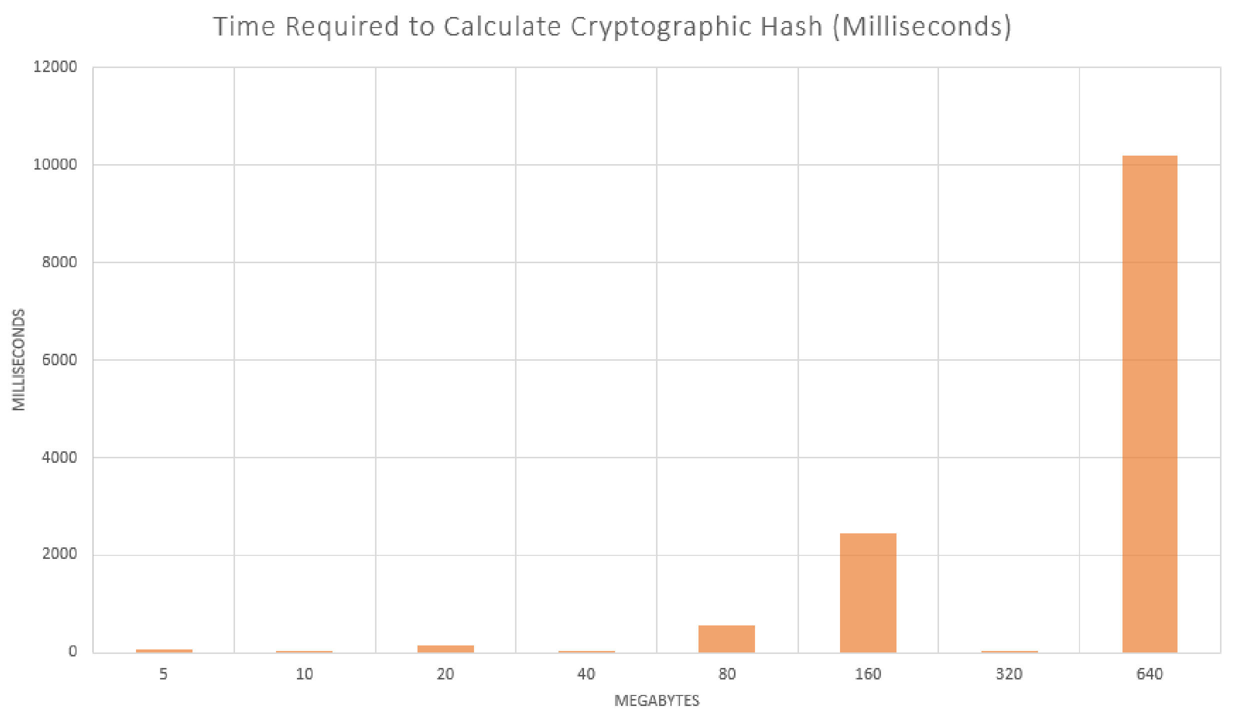
A real-world example of the server's and API's response time to a 205-megabyte .kmz Google Earth file being downloaded is below. Note that the stack proactively rescans the hash against the database after a file attribute changes, despite the contents of the file resulting in the same cryptographic hash.

Time (UTC)	**Action**	**Response Time**
13:22:23.717	Endpoint sends the cryptographic hashes of the file to the Wazuh Server, and Wazuh alerts that a file was added.	N/A
13:22:25.443	VirusTotal API responds that this file is not in the database.	≈1726 milliseconds
13:22:25.727	File modification time changes from 1709040141 to 1709040143 – this is likely Chrome "finalizing" the download or similar. This causes the server to reevaluate the cryptographic hash.	N/A
13:22:27.315	VirusTotal API responds that this file is not in the database.	≈1588 milliseconds

In this real-world example, if the 205-megabyte file being downloaded was a known-malicious file, from the time of hash-calculation it would be able to be actively removed in 1.59 to 1.73 seconds on the endpoint, should you choose to implement an active response.

This evaluation, however, leaves out an important aspect of determining the amount of time between a file being downloaded and having actionable data about whether it is malicious – the amount of time it takes to calculate the cryptographic hash of the file. To evaluate this, I used wmic [4] to expose the exact file creation time, and then calculated the time between the computer seeing the file being created and the Wazuh Server being sent the cryptographic hash of the file. I performed this for various file sizes to determine the strength of the relationship between file size and the amount of time required to calculate the cryptographic hash of the file.

File Size	**Created Time (Central Time)**	**Hash Received on Server (Central Time)**	**Response Time**
5 MB	8:06:53.626	8:06:53.683	≈57 milliseconds
10 MB	8:13:28.342	8:13:28.358	≈16 milliseconds
20 MB	8:13:35.899	8:13:36.064	≈165 milliseconds
40 MB	8:13:45.784	8:13:45.804	≈20 milliseconds
80 MB	8:13:56.892	8:13:57.443	≈551 milliseconds
160 MB	8:38:06.795	8:38:09.247	≈2452 milliseconds
320 MB	8:38:36.161	8:38:36.176	≈15 milliseconds
640 MB	8:38:59.824	8:39:10.012	≈10188 milliseconds

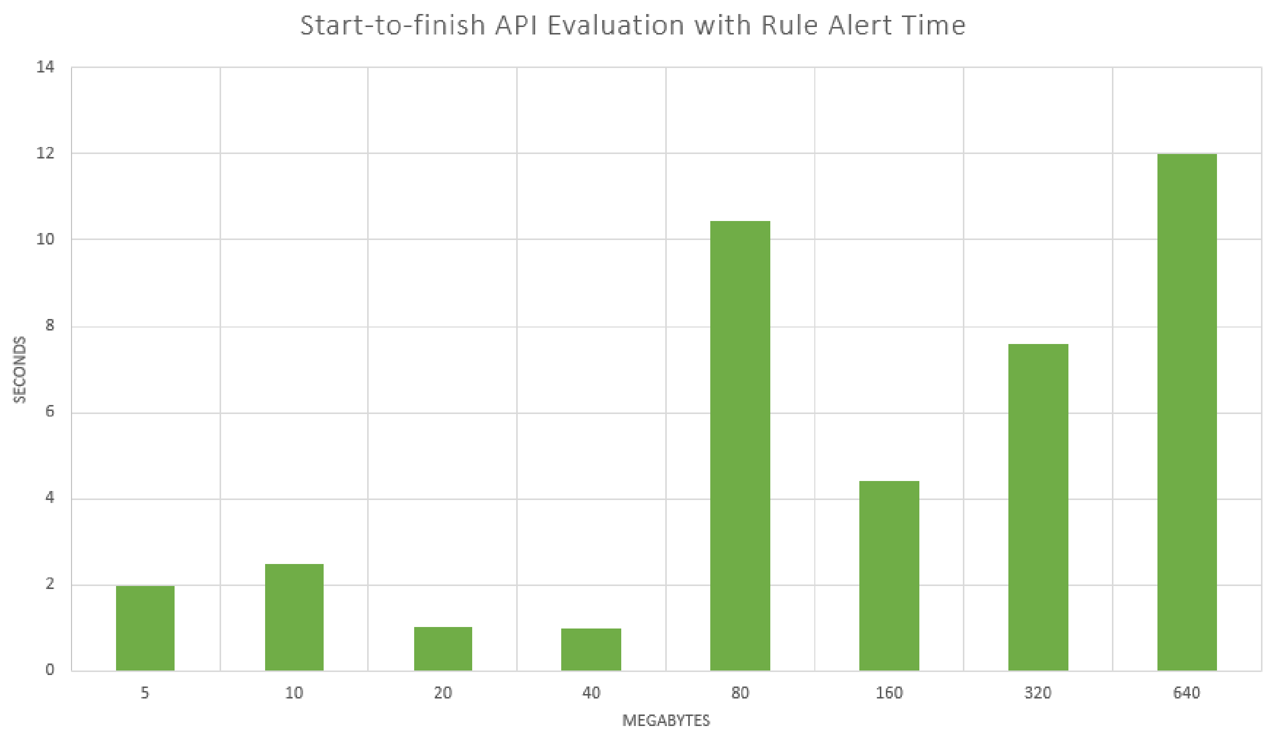


As can be seen in the data, there does appear to be a correlation between the size of the downloaded file and how long it takes to calculate the cryptographic hash of the file. This makes logical sense, considering that the entire file needs to be parsed to calculate the hash. That said, we can see high inconsistency despite this trend, with the hash of the 320-megabyte file being calculated faster than any other file's hash. This demonstrates that there must also be a high level of dependence on environmental factors, including network congestion and other disk or CPU intensive activities on the endpoint.

With this additional data, we can calculate the rough start-to-finish response times for various file sizes. This start-to-finish response time includes the file being created/downloaded, the file's cryptographic hash being calculated on the endpoint, the file's hash being sent from the endpoint and received by the server, the server sending the cryptographic hash to the VirusTotal API, the server receiving the response from the VirusTotal API, and the server alerting or activating an active response based on the result. This represents the response time on an endpoint running Windows 10 with an Intel Core i7-7700 CPU, 8 gigabytes of DDR4 RAM, and a SATA SSD.

The start-to-finish response times on the endpoint were as follows:

5MB file – 1963 milliseconds (1.963 seconds)
10MB file – 2468 milliseconds (2.468 seconds)
20MB file – 1026 milliseconds (1.026 seconds)
40MB file – ≈881 milliseconds (0.98 seconds) [exact number not preserved, assumes same API speed as 20MB file]
80MB file – 10425 milliseconds (10.425 seconds)
160MB file – 4402 milliseconds (4.402 seconds)
320MB file – 7578 milliseconds (7.578 seconds)
640MB file – 11992 milliseconds (11.992 seconds)



The vast differences in start-to-finish time outline the fact that the API itself has varying response times to the hashes it is sent. Should one invest in the private API instead of using the public API, I suspect that these response times will be much lower.

PRACTICALITY OF DEPLOYMENT

This solution, however convoluted it may seem, is simple to deploy. First, you need a working Wazuh Stack including the core components of Wazuh Server (Wazuh Manager), Wazuh Indexer, and Wazuh Dashboard. Whether you wish to deploy these as a distributed solution or as a unified appliance is at your discretion. In addition to this, you need to have the Wazuh Agent functioning on your endpoints. From here, you need to follow the instructions in Wazuh's documentation [2]. The amount of time to deploy this solution is trivial once the endpoints and the stack are prepared. That being said, you will almost certainly immediately notice that Wazuh File Integrity Monitoring is sending the cryptographic hashes of registry keys to the VirusTotal API. This causes both unwanted API calls that count towards your 500/day or 4/minute public API limit, and a plethora of unwanted emails depending on how you choose to configure the stack. Resolving this issue was seemingly impossible without manually changing all of the local OSSEC configuration files on our endpoints. However, over the course of six days, I troubleshooted with Wazuh developers and support staff [5][6] and we eventually developed this solution together:

```
<agent_config>

  <!-- Shared agent configuration here -->

<wodle name="syscollector">
   <disabled>no</disabled>
   <interval>1h</interval>
   <os>yes</os>
   <packages>yes</packages>
   <hotfixes>yes</hotfixes>
</wodle>

<syscheck>
  <registry_ignore arch="both" type="sregex">^HKEY_LOCAL_MACHINE\Software</registry_ignore>
  <registry_ignore arch="both" type="sregex">^HKEY_LOCAL_MACHINE\System</registry_ignore>
  <registry_ignore arch="both" type="sregex">^HKEY_LOCAL_MACHINE</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\batfile</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\cmdfile</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\comfile</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\exefile</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\piffile</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\AllFilesystemObjects</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\Directory</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Software\Classes\Folder</registry_ignore>
  <registry_ignore arch="both">HKEY_LOCAL_MACHINE\Software\Classes\Protocols</registry_ignore>
  <registry_ignore arch="both">HKEY_LOCAL_MACHINE\Software\Policies</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\Security</registry_ignore>
  <registry_ignore arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Internet
Explorer</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services</registry_ignore>
  <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\KnownDLLs</registry_ignore>
<registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurePipeServers\winreg</regi
stry_ignore>
  <registry_ignore
arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run</registry_ignore>
  <registry_ignore
arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce</registry_ignor
e>
<registry_ignore>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx</registry
_ignore>
  <registry_ignore
arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\URL</registry_ignore>
  <registry_ignore
arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies</registry_ignore
>
  <registry_ignore arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Windows</registry_ignore>
  <registry_ignore arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Winlogon</registry_ignore>
  <registry_ignore arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Active Setup\Installed
Components</registry_ignore>
</syscheck>
</agent_config>
```

By placing this block of XML in your agent.conf central configuration, you can effectively overwrite the default Wazuh FIM registry monitoring that is included in each endpoint’s OSSEC configuration file. This works by merging each endpoint’s OSSEC configuration file with the shared configuration.

By creating this shared configuration after remote command execution is enabled [6] on your Wazuh stack, you can harness the custom VirusTotal rules and/or overwrite them to an appropriate level in your environment depending on how you wish to develop the rest of the stack.

CONSIDERATIONS

When choosing to deploy this strategy, there are many considerations to contemplate. First, consider what types of files you want evaluated against the VirusTotal database. It is impractical to evaluate everything all the time. In addition, if you want to be alerted to new files downloaded of a certain type, due to the nature of how this solution works, they will need to be the same files you wish to have evaluated. It is also worth giving special permissions to people who regularly download or create files, unlike most files in your environment. This prevents unnecessary file evaluation and reduces alert fatigue. Another consideration is what you choose to do with the information the API returns. There is a publicly available Python script licensed under the GPL that will forcibly delete any file that VirusTotal believes is malicious. You may want to use this to remove malware that goes undetected by your anti-virus provider, or to be alerted about the presence of potentially malicious files to evaluate them yourself. Another possibility is creating a script that isolates the computer from the network if a malicious file is detected. These decisions must be made based on your risk tolerance, trust in your current anti-virus provider, and any established malware preservation protocols.

CONCLUSION

Although not a replacement for traditional anti-virus solutions, automatically sending the cryptographic hashes of certain downloaded files to the VirusTotal API for evaluation can dramatically improve your security posture, and can aide in detecting, mitigating, and/or removing known-malicious files that would otherwise go undetected in your environment.

References

1. Pax, Z. (n.d.). GitHub - ZacharyPax/WatchtowerStack: Repository for configuration files, scripts, code, and other information on the Watchtower Stack. GitHub. Retrieved February 26, 2024, from https://github.com/ZacharyPax/WatchtowerStack
2. Wazuh. (n.d.). VirusTotal integration - Malware detection · Wazuh documentation. https://documentation.wazuh.com/current/user-manual/capabilities/malware-detection/virus-total-integration.html
3. *Postfix feature overview*. (n.d.). https://www.postfix.org/features.html
4. Useful WMIC queries. (n.d.). TECHCOMMUNITY.MICROSOFT.COM. https://techcommunity.microsoft.com/t5/ask-the-performance-team/useful-wmic-queries/ba-p/375023
5. Stuti Gupta – Wazuh Team – (personal communication, February 2024)
6. Md. Nazmur Sakib – Security Engineer – B.S. Computer Science – Wazuh Incorporated – (2023, October 30) [Comment on the online forum post *Remove registry FIM alerts.*] Reddit. https://www.reddit.com/r/Wazuh/comments/17jnxqa/remove_registry_fim_alerts/
7. Wazuh. (n.d.). Centralized configuration (agent.conf) - Reference. https://documentation.wazuh.com/current/user-manual/reference/centralized-configuration.html