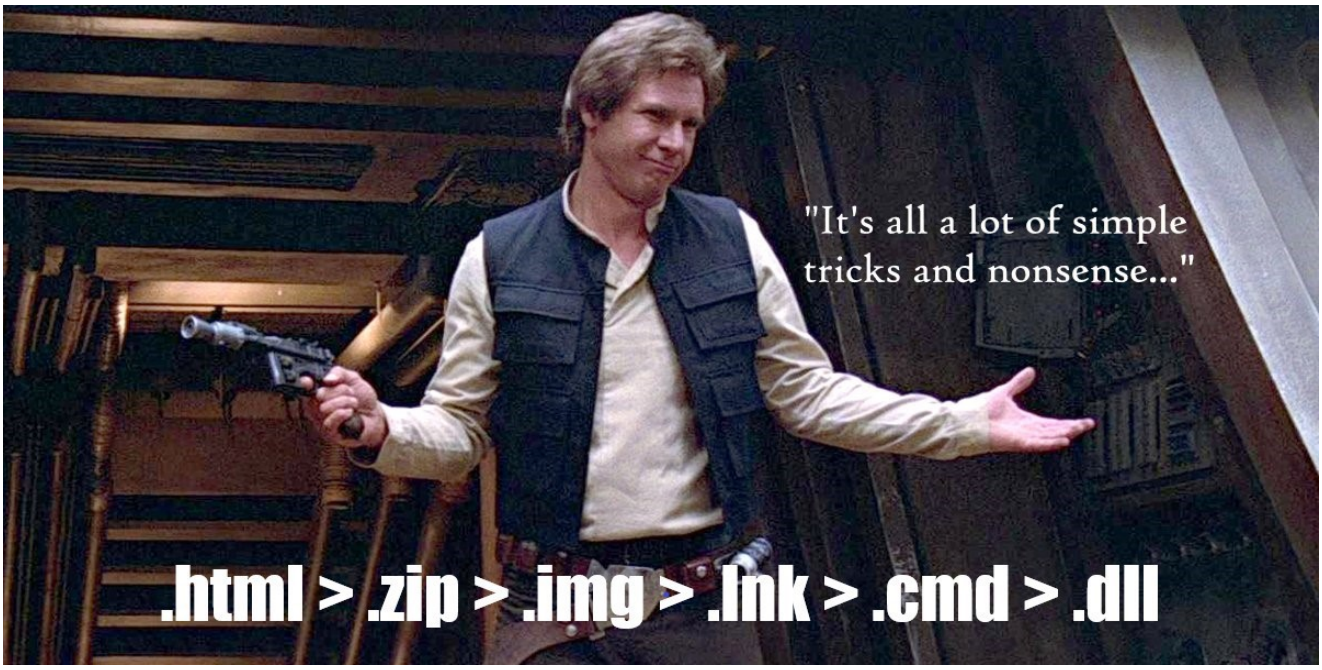
HTML Smuggling Detection

micahbabinski.medium.com/html-smuggling-detection-5adefebb6841

Micah Babinski December 28, 2022



Micah Babinski

Dec 28, 2022

.

15 min read

The most famous fictional smuggler that I could think of

Introduction

In this article I'll delve into HTML smuggling detection, following the detection engineering process I've described over my last two posts. This process includes research, testing, and development of new detection concepts. Unlike my previous posts however, this time we'll be observing, profiling, and detecting real QakBot malware in the lab. With this piece I hope to show how current and aspiring detection engineers can go beyond simulated, CTF-style challenges to study and detect real-world attacker techniques, flagging them for our SOC, and allowing our incident response colleagues to neutralize the threat.

Our Itinerary

After some background and defining the objective, I'll demonstrate how we can execute QakBot HTML Smuggling malware in the lab, tracking it's behavior in Splunk. Then, I'll sequence observable events of this attack, and identify useful detection points (which I think of as "building blocks," or links of the attack chain). I'll also introduce correlation (specifically chain rules), a crucial tool in your threat detection toolbox. To illustrate what this looks like, I'll discuss and demonstrate an as-yet-unreleased capability of Sigma, called Sigma Correlations. I'll wrap up by testing a new chain rule against 10 additional samples of QakBot HTML Smuggling malware to see how it performs.

Let's get started! 🏁🏁🏁

Background

Early on in my cybersecurity journey, John Strand of Black Hills Information Security said something that stuck with me: "There is no 'YOU HAVE BEEN HACKED' log." Event logs can be confusing and hard to read even on a good day. Deriving useful, actionable insights from logs is tricky! The challenge sometimes requires deep analysis and specialized techniques to be successful.

I've had that lesson in the back of my mind throughout my career in security, and it was with this in mind that a few months ago, I started seeing tons of posts like this one from @pr0xylife:

I like the way that pr0xylife and their peers summarize these attacks so succinctly, and I found myself repeating the sequence of file types out loud—almost like a weird form of poetry!

I wanted to learn more, but didn't know how to get started. I had reviewed some excellent research on QakBot by the team at Trellix, so I knew a bit about their techniques. (If you have never heard of QakBot, please read the Trellix report!) As I started following QakBot activity and digging deeper, I realized that pr0xylife and others were describing the subtle combinations and variations in the way QakBot could trick its victims and evade our defenses. Some examples:

- It uses , where a malicious HTML file containing encoded JavaScript is executed by the victim's browser, downloading the next stage of the payload.
- It uses to block sandboxing analysis.
- It uses a disk image format called an .iso file to evade the Mark-of-the-Web protection, as Red Canary .
- LNK files disguised to lure users into executing to hidden .CMD and .DLL files.
- And on and on with ever more deceptive tricks!

Viewed independently of one another, few of these behaviors will trigger an alert, much less get escalated, in many SOCs. So it seemed like a good use case for correlation, where multiple security event log entries are aggregated, sequenced, and timed in relation to each other to yield more sophisticated and high-fidelity alerts. If this sounds confusing, keep reading! By the end of this article, you should have a much better understanding of what correlation means and how you can put it to work for threat detection.

The Objective

There are some detections out there for HTML smuggling and other QakBot-connected techniques. These are mostly based on matching suspicious log events, such as this one from Elastic security:

Suspicious HTML File Creation | Elastic Security Solution [8.5] | Elastic

Identifies the execution of a browser process to open an HTML file with high entropy and size. Adversaries may smuggle…

www.elastic.co

I wanted to see if I could build my own detections, utilizing *correlation* (as opposed to simple matching) to make them resilient to subtle shifts in attack behavior.

> TBH, I was also sick of QakBot's dumb chicanery and wanted a reliable way to place it squarely within our sights.

Me, after seeing yet another QakBot variation

Initial Observation and Analysis of QakBot Malware

To accomplish my objective, I had to go beyond threat intelligence reports from security vendors and Atomic Red Team tests; I needed my own data created from real malware samples. I turned to an old favorite: malware-traffic-analysis.net by Brad Duncan (@malware_traffic). This site is among the most useful educational resources I have come across in security. In addition to Wireshark tutorials and hands-on network traffic exercises, the site offers quality analysis of real-world malware samples, including QakBot HTML smuggling files.

After making snapshots of my lab VMs, I fired up my virtual detection lab and visited a recent entry. Be careful, the files hosted on this site are unsafe!

Malware-Traffic-Analysis.net - 2022-12-09 - HTML smuggling leads to Qakbot, distribution/botnet…

NOTES: The HTML file used for smuggling was posted to VirusTotal today, and create/modify dates for the disk image &…

www.malware-traffic-analysis.net

I downloaded the artifacts zip file and extracted it to my downloads folder using the WinRAR utility built into my detection lab:

The IOC notes were extremely helpful in giving context to the included files. Among the notes were the following, which let me know that the infection chain started with the HTML file, called SCAN_DT6281.html.

```
2022-12-09 (FRIDAY) - HTML SMUGGLING FOR QAKBOT (QBOT) DISTRIBUTION TAG: AZD


DISTRIBUTION:

- Unknown source, possibly email --> HTML file --> password-protected zip archive --> extracted ISO image with .img file extension
```

I opened the HTML file in Chrome browser and noticed an immediate zip file download (bottom left below):

Seems legit
…sure, let's open up the zip file, why not?

The zip file was password protected but could be opened using the password displayed on the HTML page. The zip file contained a single .img file, which I know to be another mountable disk image file format. Double-clicking this mounted the file as the D drive on my system:

The shortcut LNK (SCAN_DT6281) and hidden folder (IncomingPay)
I also noticed a hidden directory called IncomingPay, which contained a .lnk file, two text files which contained (I kid you not) excerpts from the Wikipedia page on Psychology, a .cmd file 🤔, and a .lc file 🤷.

As a security professional and dutiful watcher of corporate security awareness training, all of my 🚩🚩🚩 were up and waving in the breeze. But this is for science, so I take the bait and double-click the LNK file, just as the attacker wants the victim to do. 🎣 A command line window appears for a few seconds, then disappears:

The LNK file target property is interesting:

```
C:\Windows\System32\cmd.exe /c IncomingPay\Issues.cmd A B C D E F G H I J K L M N O P
Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9
```

This tells me that I've run the Issues.cmd file located in the IncomingPay directory using CMD, which is important to know for later. Now back in Splunk, I start running queries to see what is happening on my victim VM:

Notice the commands logged immediately after I took the bait!
How do I know what searches to run? I don't really, but through my experience as an analyst and researcher I know that there are certain types of evidence that may appear in my logs — process executions, file creations, network connections, and the like. This is where on-the-job experience as a security analyst, or a good dose of CTF-style trainings (like those available at CyberDefenders or LetsDefend) come in handy.

After refreshing my searches a few times and wondering if I had done something wrong, I noticed a definite, incontrovertible sign of an intrusion: a burst of recon activity on my victim host:

I forgot to include in the screenshot, but the parent process for all these commands was wermgr.exe (what???)
As I was looking at these commands, I noted the time period in which they occurred — all within the span of a few seconds! Also, the processes were all spawned by an unusual parent — wermgr.exe (the Windows Error Reporting Manager). A good detection opportunity, perhaps? Event the most confused admin or overbuilt piece of [legit] software will not run all these suspicious recon commands within such a short timeframe, and *never* as children of a wermgr.exe. Swinging over to my network connections, I noticed that wermgr.exe had suddenly become extremely chatty with external IP addresses:

Uhhhhh……
Following this initial assessment of my data, I determined that:

- The malware sample contained a malicious initial access vector (duh, the rogues gallery of the .html, .zip, .img, .lnk, .cmd, and .lc files).
- The malware would download the zip file after opening the HTML page in a browser, and the zip file contained a mountable .img file.
- The mounted drive contained a malicious shortcut that would result in the execution of additional commands.
- An injected wermgr.exe process spawned an automated burst of recon activity and then connected to suspicious external IP addresses, presumably to send the attacker information about our system and network.

In Mitre ATT&CK terms, this all amounts to Initial Access, Defense Evasion, Discovery, and Command and Control. In other words: big oof.

Rogues Gallery

Breaking Down the Attack into Detection Building Blocks

Having performed an initial execution of the QakBot HTML smuggling technique in the lab and reverted to my VM snapshots, it was time to dig further into the logs to see what was happening.

As I reviewed the various types of events generated by the attack, I began to develop a plain-language narrative understanding of what was happening:

> "An attacker **sends** a victim an HTML file that purports to contain a report, invoice, or other document of interest to them. Basically, a phishing lure. When the victim **opens the file** in their browser, embedded JavaScript code executes, which **downloads or builds a password-protected zip file** on the victim's system. When the victim **unzips the archive**, the extraction **creates a mountable** disk image file. After **mounting the drive**, the user sees a shortcut that they believe will take them to the resource they are trying to find. But the shortcut actually calls a command or scripting interpreter that **executes other malicious files** that are hidden on the disk drive, leading to initial access compromise."

That's a wordy chunk of prose right there, but it makes sense in my head. If I am going to detect something, I need to understand it first! Note the bold text: I used my narrative to highlight events that I could use to detect the attack chain. Based on this review, I analyzed the logs from the attack, trying to isolate the following events:

1. Phishing email sent to victim containing an HTML attachment.
2. Creation of an HTML file in suspicious locations.
3. Opening of a stand-alone HTML file in a browser application.
4. Download/creation of a zip file by the browser application.
5. Opening/extraction of a password-protected zip file.
6. Creation of a mountable disk drive file format (.iso, .img, etc).
7. Mounting a drive.
8. Process execution on an external drive (either from an executable on that drive, or system executable touching files on that drive).

Whew! That's…a lot of events.

The Good News

The good news was, there are ways to detect nearly every event listed above! This means I had numerous ideas for how to query and filter available logs to extract the meaningful events (building blocks) for future detections.

The Bad News

The bad news was, none of these many events is, on its own, malicious. Again, there is no YOU HAVE BEEN HACKED log: each one of these events could occur in the course of normal business and be completely safe and benign. The solution would be to correlate these events, in an ordered or unordered sequence, grouping them by a common attribute like hostname, and triggering an alert when all of these occur within a time window, like an hour. The problem is, from my analysis and testing (more on that in a moment), chaining or *correlating* that many events would result in a brittle detection.

Brittle vs. Resilient

“Brittleness” describes the degree to which a detection idea falls apart in the face of subtle changes to attacker techniques, variations in actions performed by the victim, problems with logging, or other factors outside of our control. Brittle detections contrast with “resilient” detections, which are flexible and can withstand these subtle shifts.

It’s not as simple as saying “Brittle = bad and resilient = good.” A brittle detection could be highly-targeted, with a narrow “aperture.” The advantage could be that, if a brittle detection rule fires, there is a very high probability that it is a true positive. Resilient detections may have a wider aperture and may match more potential malicious activity. However, this could lead to false positives and a frustrated SOC if they are not developed with care.

With this in mind, I returned to my list of events from above.

Determining the Building Blocks to Test

In context, I realized that items one through three in the list above may not be good components of my HTML Smuggling detection. Lack of visibility and a high volume of innocent behavior would hinder item 1. I struck item two because I had limited ability to test this event, and item three proved unreliable depending on which browser I used. Plus, while not technically HTML Smuggling, a lot of QakBot activity uses URLs, rather than stand-alone HTML files, to deliver the initial payload.

Item five (extraction of a password-protected zip file) has lots of potential, and can be detected using this rule written by Florian Roth and inspired by the research of @SBousseaden. However, I left it out of my scope because 1) that event was not logging in my lab and 2) some documentation indicates that it is only applicable to certain operating systems.

After exploring my data, enumerating possible detection building blocks for my correlation, and winnowing that list down based on further analysis, I had the following building blocks:

1.

2.
3.
4.

With these building block concepts in place, I wrote or adapted a query and Sigma rule for each one, then correlated them into a chain rule.

Sigma Correlations

Correlation allows us to track log events through time. Rather than triggering an alert for each of the four events listed above, a chain rule correlation allows us to alert only when all four events occur in order on the same host by the same user, which is much more suspicious. Many SIEM products and their corresponding query languages support this type of chaining (although some do not).

To support this functionality, Sigma has a Correlations standard in progress that will allow us to write custom correlation rules in a common format then convert them to whatever SIEM product supports this logic. The draft standard can be reviewed here:

sigma-specification/Sigma_Correlations.md at main · SigmaHQ/sigma-specification

Sigma correlations is a new YAML-based extension that allows the expression of aggregations and relationships between…

github.com

What might this look like? **It's a draft standard, and subject to change**, but a simple example of a brute force chain rule might look like this:

```
action: correlationtype: temporalrule:    - many_failed_logins    -
successful_logingroup-by:    - Usertimespan: 1hordered: true
```

When the rule query "many_failed_logins" is matched followed by the "successful_login" rule within a one hour window, the correlation rule will fire. For a good example of a SIEM product that supports correlations, check out SumoLogic chain rules.

My draft correlations rule looks like this:

```
title: HTML Smuggling Activity - Chain Ruleid: 0952f2fa-e29b-4eb5-831c-
ce21520c56e3status: experimentaldescription: Detects HTML smuggling-style compromise
(such as HTML > ZIP > ISO/IMG/VHD > CMD/BAT/VBS > DLL). Includes rules to detect
zipfile dropped by browser, ISO/IMG/VHD/LNK file extraction, disk image mount,
followed by user-initiated process creation on an external drive.references:    -
https://blog.talosintelligence.com/html-smugglers-turn-to-svg-
images/#:~:text=HTML%20smuggling%20is%20a%20technique,directly%20on%20the%20victim's%2
    - https://www.malwarebytes.com/blog/news/2021/11/evasive-maneuvers-html-
smuggling-explained    - Original research and analysis performed off of QakBot
intelligence gathered at https://github.com/pr0xylife/Qakbot, https://www.malware-
traffic-analysis.net/, and https://github.com/executemalware/Malware-IOCsauthor:
Micah Babinskidate: 2022/12/27tags:    - attack.s0650    - attack.s0483    -
attack.initial_access    - attack.defense_evasion    - attack.execution    -
attack.t1564    - attack.t1566.001    - attack.t1566    - attack.t1027    -
attack.t1027.006    - attack.t1059    - attack.t1204    - attack.t1204.002action:
correlationtype: temporalrule:    - 1_win_zipfile_drop.yml    -
2_win_susp_file_extraction.yml    - 3_win_security_iso_mount.yml    -
4_win_process_creation_ext_drive.ymlgroup-by:    - ComputerName    - Usertimespan:
1hordered: truefalsepositives:    - Unknownlevel: high
```

This looks like many other Sigma rules you may have seen before, but has some unique elements. The action and type statements let you know this is a correlation rule of the temporal type. The four **rules** listed show the component rules in scope (these must be in the same directory as the correlation). The **timespan** specifies that the four participating rules must fire within one hour of each other, and the **group-by** statement defines fields in the rules which relate them to each other. Finally, the **ordered: true** statement lets Sigma know that these rules should occur in sequence.

> Again, the Sigma Correlations specification is in development, and is subject to change. Still, this will be a very useful expansion of Sigma's capabilities, so I wanted to preview it for you now!

Testing the Correlation

With this detection concept taking shape, and a hypothesis developed in the form of my correlation rule, it was time to test the detection with a larger sample size. I relied on the MalwareBazaar repository from Abuse.ch, which provides a helpful library of tagged malware samples. I downloaded 10 QakBot HTML samples, mostly reported by pr0xylife, ranging in date first seen from July 11 to December 22, 2022. I prepared the samples on my lab VM, and named each one according to its first seen date and its "humanhash" property (a random, unique sequence of human-readable words, such as ("dakota-earth-mockingbird-march"):

My 10 lovely HTML smuggling samples all ready to test
To track my results, I made a simple table in Google Sheets:

I made a clean, pre-test snapshot of my victim VM host to revert back to after each test, and started in on the test. After testing seven of the ten samples, my sequence of four building blocks had perfect coverage! When I hit the eighth sample, however, the fourth rule in the chain did not fire, because the .lnk file called RunDLL32.exe directly, instead of cmd.exe or wscript.exe. This was fine — I simply made an adjustment to the conditions of my Sigma rule, retested, and got perfect matches! I was delighted with the results and excited to share the detection use case with the community.

Bonus Round!

Many QakBot phishing attacks do not use .html attachments, but instead use malicious .pdf files with embedded links, or just good old-fashioned phishing links that point to zip files hosted on a compromised website. To test whether my detection was flexible enough to catch these instances as well, I tested them on a couple of recent examples from the IOC repository maintained by @ExecuteMalware, and found that these URL-based driveby attacks were also caught by the detection. The one documented here even included a .wsf (Windows Script File)—a file type with which I was unfamiliar but was nonetheless caught by my detection.

Windows Script File (.wsf) Delivered During a Recent QakBot Attack
Hooray for resilience!

Conclusion

Congratulations! You've reached the end of a lengthy post about some dense, complicated topics. You can find all the rules referenced here. Thank you for reading, and I hope you've enjoyed my ramblings on QakBot, HTML Smuggling, Sigma, and correlations. I am truly excited for Sigma Correlations to launch. It will be a big win for the detection engineering community and will allow us to share more sophisticated detection use cases.

I was really pleased with how my tests went, particularly when it showed that one of the building blocks in my chain rule had failed and needed adjustment! After all, why test if you think your work is already perfect? Lastly, this experience drove home for me what I had already started to believe — that we can't detect what we don't understand. There's no substitute for first-hand experience with real live malware if you are trying to see what it does.

Thanks to the Detection Lab project that I've enthused about in previous posts, this real-life experience is in reach for more people than ever. Lastly, thanks to pr0xylife, Brad (malware_traffic), and executemalware for providing pristine repositories of well-documented QakBot malware samples for us to access, analyze, and understand. These are truly an educational treasure trove!

Please feel free to send me ideas, comments, or suggestions on how I can improve. I am still new to this, and I welcome respectful feedback and critique in any form.

As always, happy analyzing! 🧐