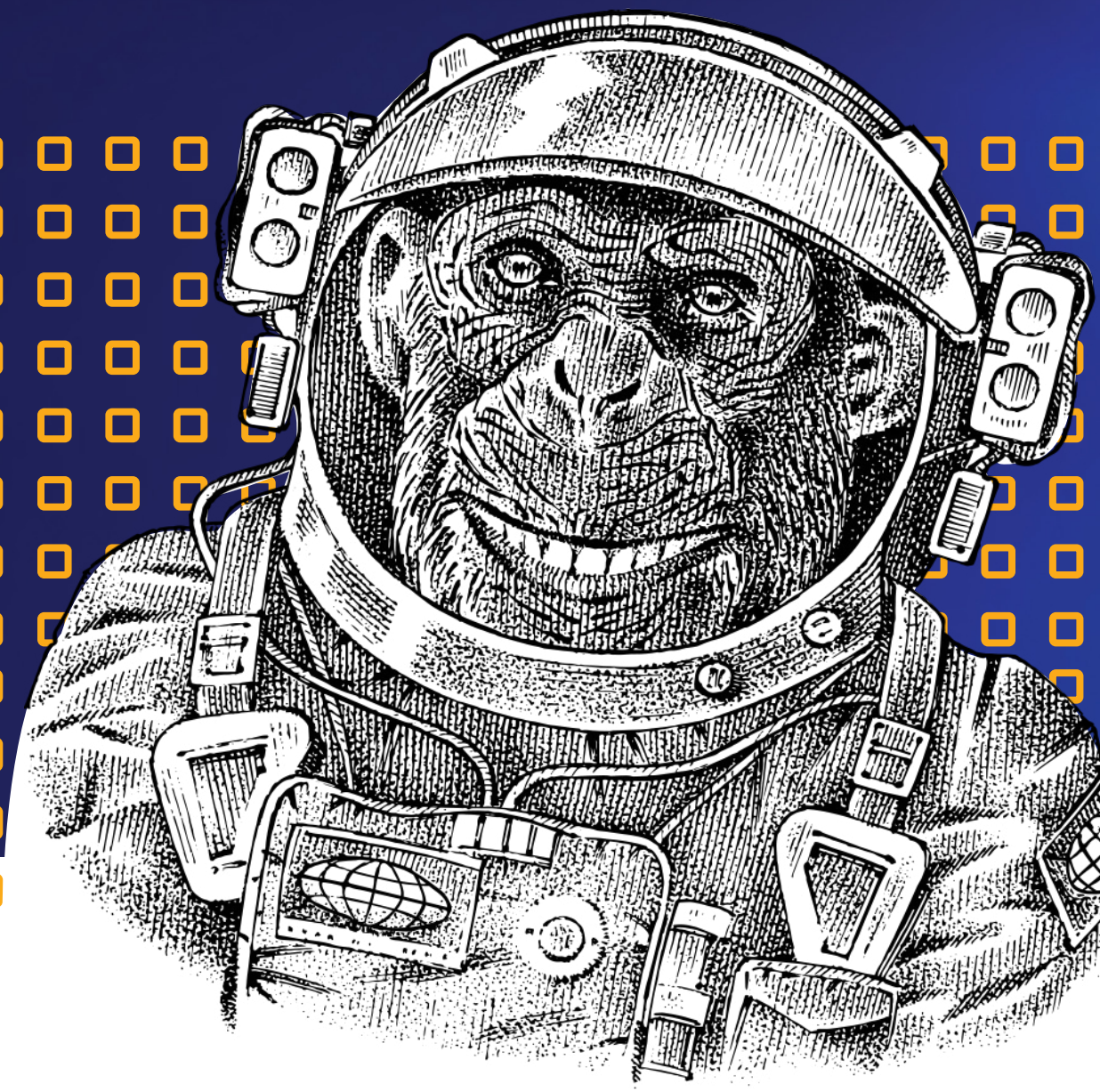
PENTERA | Pentera Labs™ Research Series

# The LOLBAS Odyssey: Finding New LOLBAS, and How You Can, Too

**Empower your security teams to proactively defend against the use of LOLBAS in an attack vector.**

**Nir Chako**



## Table of contents

## Purpose

By reading this article you will learn how you can find unknown LOLBAS, and empower your security teams to proactively defend against the use of LOLBAS in an attack vector.

## Executive summary

LOLBAS (Living-Off-the-Land Binaries-And-Scripts) is a known technique hackers can use to stay under the radar by utilizing legitimate tools for malicious activities. As the use of LOLBAS is a growing trend in cybersecurity attacks, we wanted to look into exploring innovative ways of finding unfamiliar binaries that malicious hackers could use to exploit organizations.

With more than 3000 binary files on Windows, discovering new LOLBAS can be challenging. This led us to develop an automation-driven approach to our research, which resulted in the discovery of 12 new LOLBAS files in just four weeks. That's an increase of 30% in known LOLBAS downloaders - plus a few executors!

Because LOLBAS can do just as much damage in a full attack scenario as a new vulnerability, the goal of our research was to shine light on this often overlooked threat, as well as to highlight how automation can be used in cybersecurity research.

We hope you learn something new, and try out our research for yourselves. Good luck!

## Does it apply to my organization?

Yes. Assuming your organization uses computers.

## Who should read this?

Security researchers and red teamers engage in the exploration of novel attack and research techniques.

Defense teams and blue teamers who are responsible for the configuration of the organization's security systems and conducting investigations of security breaches.

CISOs responsible for defining the organization's defense methodologies.

## Intro

In a recent research we conducted at Pentera Labs, we found a treasure of a dozen new types of LOLBAS files attackers can use, with many more waiting to be uncovered. If you're curious to learn how we did it, which new LOLBAS files we found, and how you can find new ones too, then join me on the LOLBAS Odyssey.

In this research article, I'll take you on an adventure from start to finish. We'll start by understanding what LOLBAS is, show how to run a script to find the next unknown LOLBAS and propose a framework for future LOLBAS identification. Along the way, I'll share knowledge, tools, and techniques that can help stay ahead of LOLBAS-based attacks. In the end, I'll leave you with something to think about.

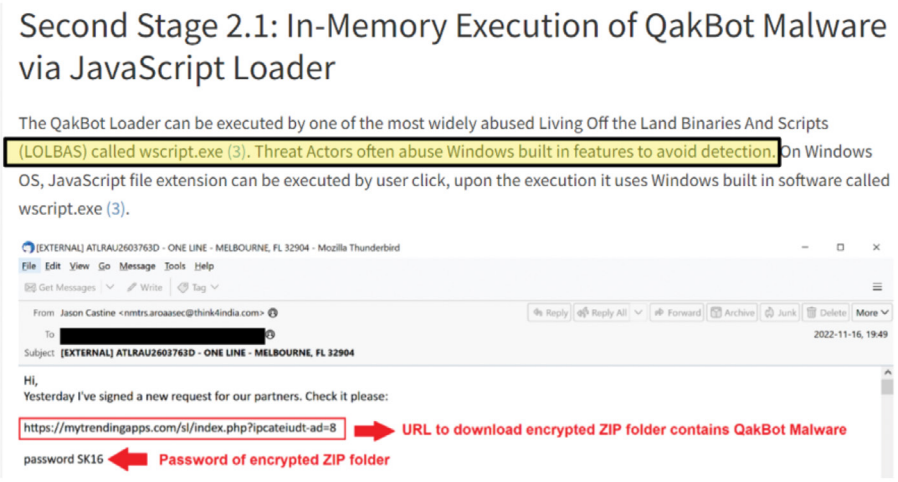
## What is LOLBAS?

LOLBAS, Living-Off-the-Land-Binaries-and-Scripts, is an attack method in which hackers use legitimate binaries and scripts that are already part of the system. Just like a paratrooper in a hostile environment grabs a stick from the ground and covers themselves with leaves, i.e utilizing what's around them, an attacker in an unknown cyber field will use its existing programs for malicious functions.

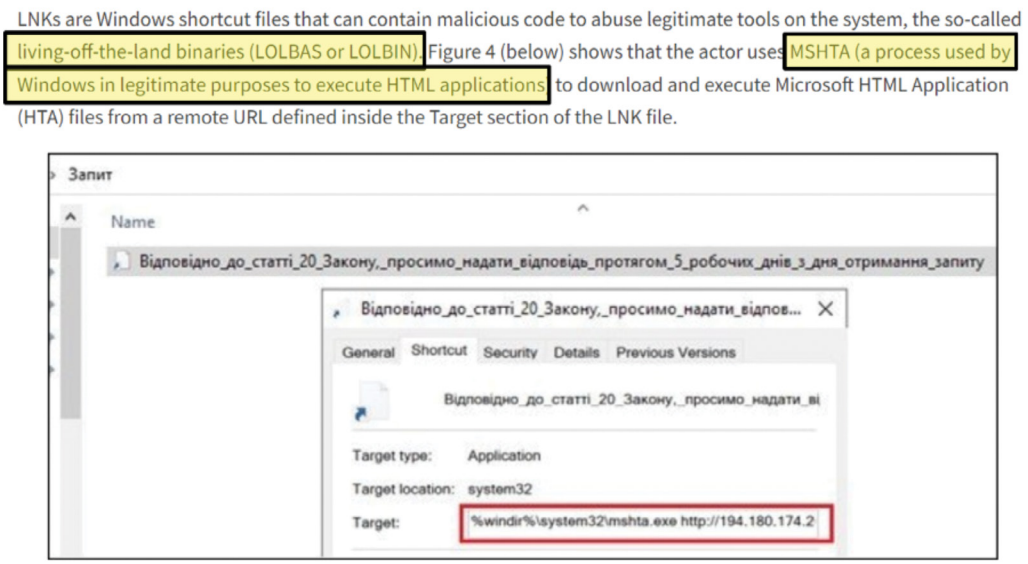
And exactly as the color and the type of the local leaves will be best to conceal the paratrooper, the local legitimate tools will be best to conceal the illegitimate use of them. When a hacker uses Excel.exe to download malware, it looks like Excel tried to get a picture online. That's why it's hard for security teams to distinguish between the legitimate and malicious activities of trusted system utilities.

## An Evergreen Type of Cyber Attack

LOLBAS isn't a new concept. Yet, it is still one of the most growing trends in cyber-security attacks! For example, just a few months ago, QakBot malware used WScript.exe to execute malware written in JavaScript to avoid detection.



In another cyber attack against the Security Service of Ukraine, MSHTA.exe was used to legitimately execute a malicious HTML application.




Hackers will always try to find a way to abuse legitimate tools on your systems and use them against you. LOLBAS continues to be relevant in cyber attacks due to its ability to remain undetected. Its effectiveness lies in leveraging legitimate and built-in system tools to carry out malicious activities, making it challenging for security solutions to detect it.

This ongoing relevance underscores the importance of continuously adapting defensive strategies to detect and mitigate the usage of LOLBAS in cyber-attacks, which is where our journey starts.

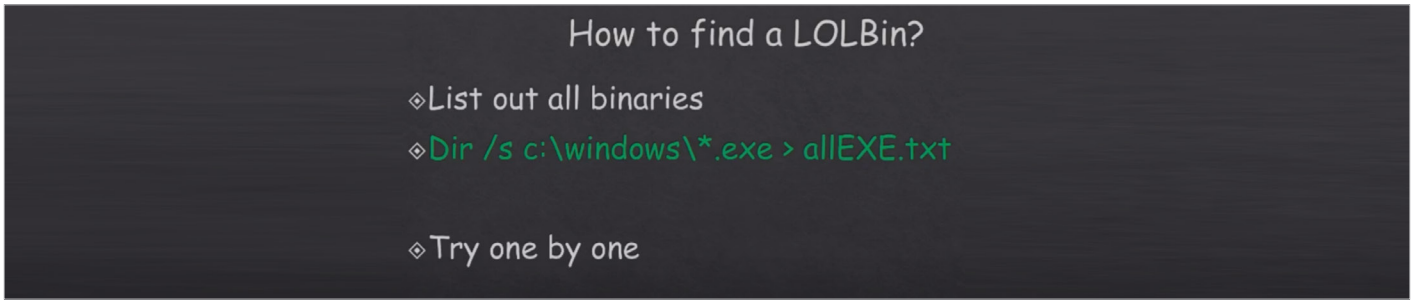
## The Manual "One by One" Approach

The story of our LOLBAS Odyssey starts with research I conducted, in an attempt to enlarge the arsenal of LOLBAS downloaders we execute as part Pentera's verification. I reviewed the official LOLBAS Project, a comprehensive list of all binaries, scripts and libraries that can be used for Living Off the Land techniques, and looked for a Microsoft Office LOLBAS. I decided to try Excel as the new downloader.

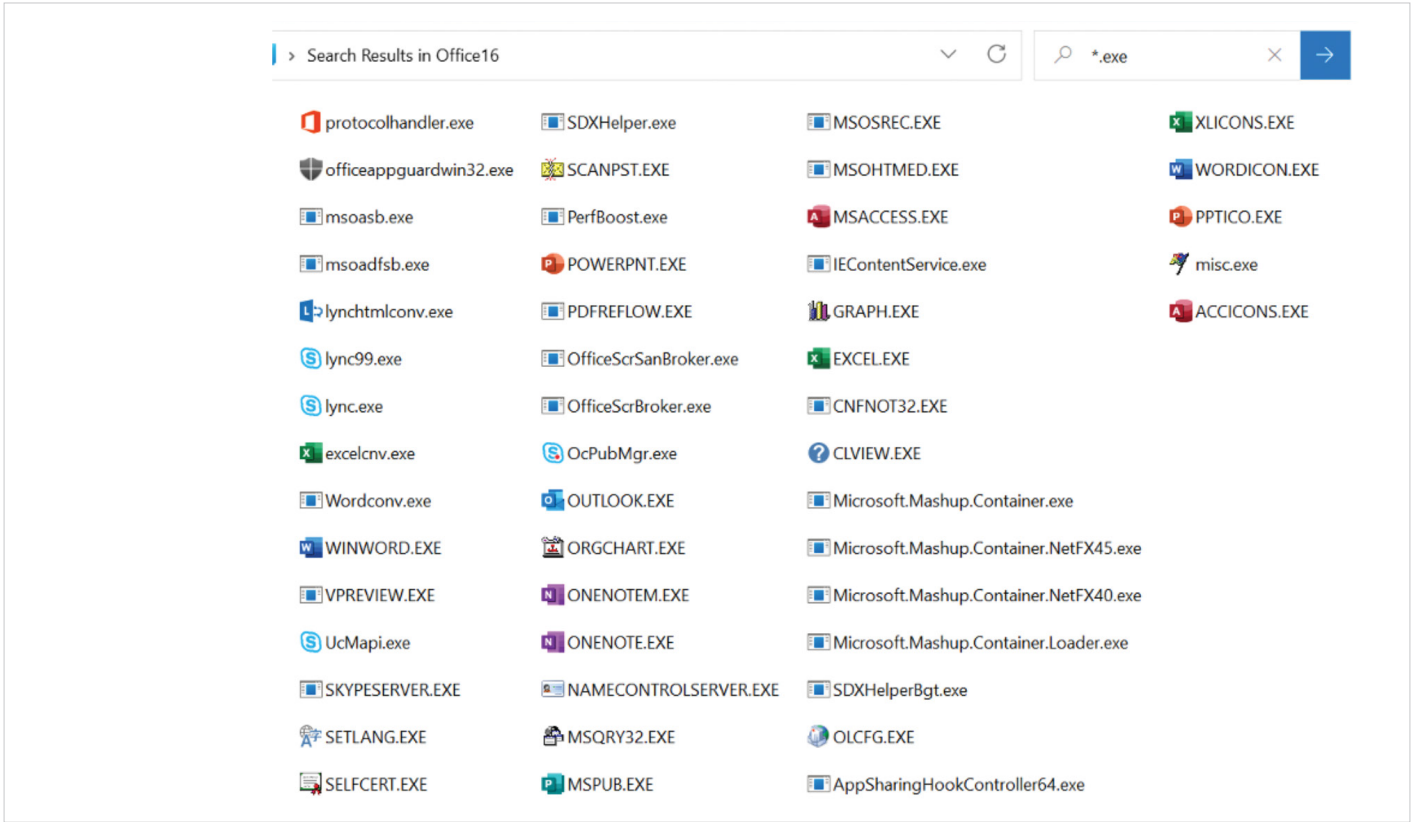
Yet, I quickly discovered that Excel couldn't be used as a downloader in cases when the attacker gains system privileges, since its system settings aren't compatible. It wasn't what we expected, but we're not the type of people who give up easily.

We shifted gears and turned to the master – Oddvar Moe. Oddvar is the founder of the official open-source LOLBAS project. Back in 2018, Oddvar gave a great talk about LOLBAS. In the talk, he recommended a method for finding new LOLBins (LOLBAS before 'scripts' were part of the initials):

- First – List all the binaries.
- Second – Try them one by one.




So that's exactly what we did. We listed all the binaries in the Office suite installation folder –

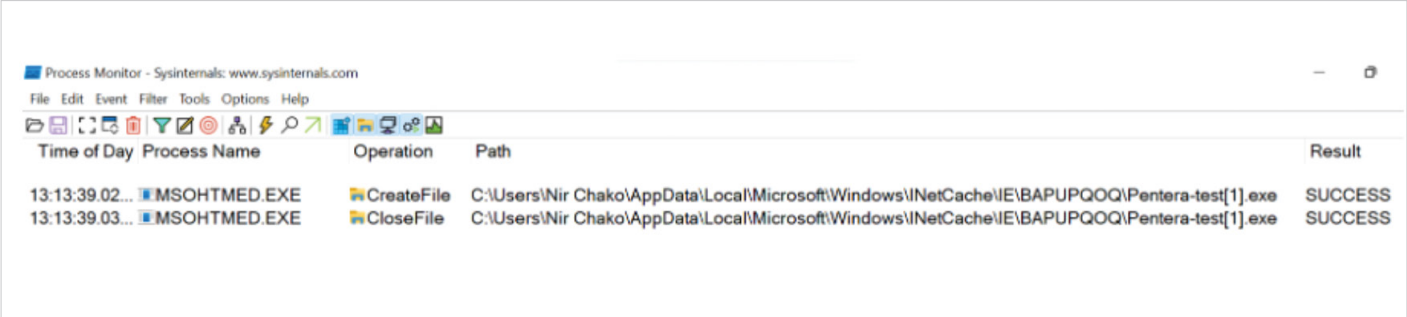



And tried them – one by one. [Watch how in this video]

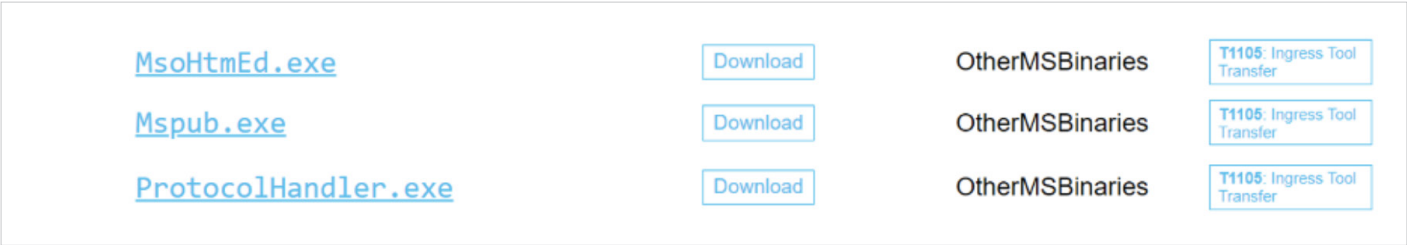
We kept it simple and tried to run the executables with a URL to download a file as the only argument. We initiated an HTTP server and tried to execute various Office suite binaries. And as you can see in the demo video, we tried to trigger MSOHTMED.exe as a downloader, to download the file Pentera-test.exe.

As you can also see, we received a GET request in the HTTP server, which means that the MSOHTMED binary tries to GET something from the HTTP server, in other words – it is trying to download the file.

After finding the LOLBAS download trigger, it's easy to find the location of the downloaded files by tracking the downloader with ProcMon.




Within two hours of the manual "one by one" approach we found three(!) new LOLBAS downloaders from the Microsoft Office suite and contributed them to the community. Not too shabby...




I was quite pleased with the results and thought about looking for some more new LOLBAS files on the Windows OS itself. But there was a problem, a big problem.

There are more than 3000 executable files as part of the Windows operating system.
Going over all of them one by one will be an extremely tedious task!

But since we used a specific manual algorithm for running the executables, the solution became crystal clear –

AUTOMATION. Research automation.

## If You Want Something, Go and GET It

The first action the automated solution needs to take is generating a download attempt. The tool needs to list all the binaries, go over them one-by-one and try to trigger a potential downloader.

To do so, we ran the simplest command structure that could initiate a download from an HTTP server. Its structure includes only two parts –

- The path of the potential downloader
- A URL to download the file from

```
Downloader.exe    http://localhost:8000/Downloader.exe
```

The code itself looks something like this:

TriggerPotentialDownloaders.py

```
source = r'C:'
source = Path(source).expanduser().glob('**/*.exe')

for source_file in source:
    params = [f'{source_file}', f'http://localhost:8000/{source_file}']
    proc = subprocess.Popen(params)
```

Go over all the exe files

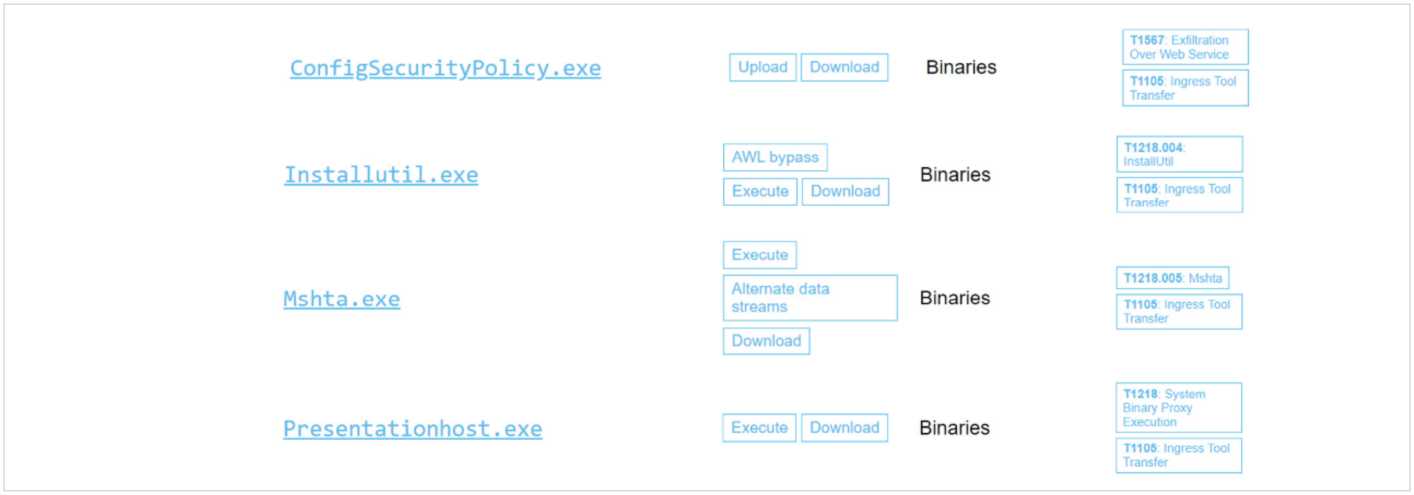
The second part is receiving feedback on the download attempt. This part includes an HTTP server, like the one we used in the manual approach. The HTTP server log records gives us the indication about the attempt to download a file.

Did you notice that we put the name of the potential downloader as the path to the file we want to download? Now, with the HTTP server running, if we receive an HTTP GET request, we can just check the path of the URL, and within it we'll find the name of the downloader that made the request.

DownloadHttpServer.py

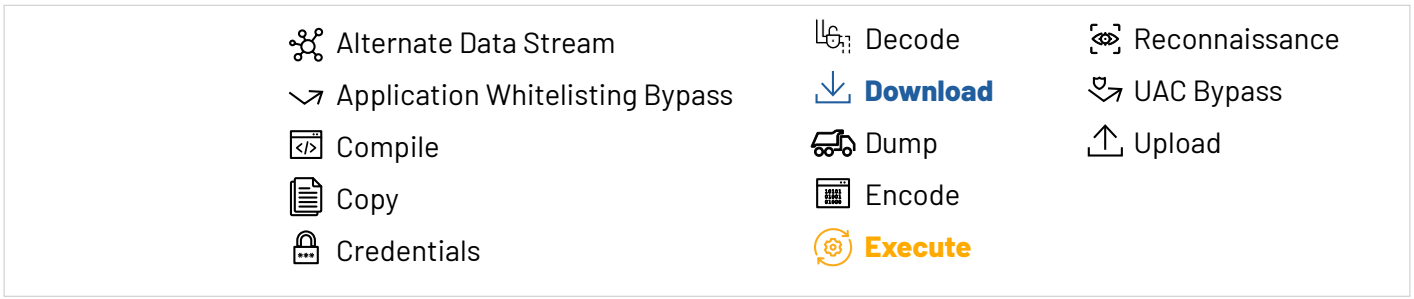
```
    httpd = SocketServer.TCPServer(("", PORT), Handler)
    httpd.serve_forever()

class GetHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_GET(self):
        if self.path.endswith('.exe'):
            exe_file = self.path
            print(exe_file)
```

Using this automated method, we managed to find six more downloaders! All in all, we discovered nine new downloaders! That's almost a 30% increase in the official LOLBAS downloaders list.




- A few more are still in review with the LOLBAS project (Outlook.exe and MSAccess.exe)

At this point, we were so excited about our findings that we decided not to stop after finding LOLBAS downloaders. Instead, we decided to continue to find new LOLBAS with other functionalities.




## .exe ^_^ cute

In a complete attack chain, a hacker will use a LOLBAS downloader to download more robust malware. Then, they will try to execute it in a stealthy way. LOLBAS executors allow attackers to execute their malicious tools as part of a legitimate looking process tree on the system.

Before we continue with the automated algorithm of finding new executors, I would like to point out that there's another small addition to the general logic of the script . This is an addition of the hyphen (-), dash (—) and slash (/). A lot of executable files allow using different flags that affect the execution flow of the program. For example, using ping with '- a' won't just ping the IP address but will also try to resolve the address hostname. Now, instead of testing just one possibility of executing a file, the script covers more options. 3(,-,/) times 26 times 2 (the ABC and abc) = 144.

```
Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
            [-r count] [-s count] [[-j host-list] | [-k host-list]]
            [-w timeout] [-R] [-S srcaddr] [-c compartment] [-p]
            [-4] [-6] target_name

Options:
    -t             Ping the specified host until stopped.
                   To see statistics and continue - type Control-Break;
                   To stop - type Control-C.
    -a             Resolve addresses to hostnames.
```

Just like before, we'll start by iterating over all the binary files on the system, trying to execute a "FILE_TO_EXECUTE" by passing it as an argument of the executor.

```
Executor.exe    File_To_Execute.exe
```

Then we do the same with hyphen, dash and slash while iterating over all the ABC letters (lowercase and uppercase).

See examples below:

TriggerPotentialExecutors.py

```
for source_file in source:
    params = [f'{source_file}', f'{PATH_TO_EXE_HELPER}']
    proc = subprocess.Popen(params)

    for letter in string.ascii_lowercase:
        params = [f'{source_file}', f'-{letter}', f'{PATH_TO_EXE_HELPER}']
        proc = subprocess.Popen(params)

        params = [f'{source_file}', f'--{letter}', f'{PATH_TO_EXE_HELPER}']
        proc = subprocess.Popen(params)

        params = [f'{source_file}', f'/{letter}', f'{PATH_TO_EXE_HELPER}']
        proc = subprocess.Popen(params)

    for letter in string.ascii_uppercase:
```

In this example you might notice there is no HTTP server as an indicator of a trigger. This begs the question – how can we receive feedback about the malware execution by the potential executor?

In this scenario, our feedback is based on the way that the operating system manages the process tree. As you can see, if we run the notepad from cmd, we can look up the process parent and get a clear indication of the executor by the name of the process parent. [Watch how in this video]

To implement that logic in our research, we developed a helper with the task of finding the name of its parent process and writing it to a log file, if it was executed. This enabled us to get our much-needed indication of the test file execution by the executor.

ExeHelper.exe

```
pid = GetCurrentProcessId();
hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
Process32First(hSnapshot, &pe32);

do {
    if (pe32.th32ProcessID == pid) {
        parentPid = pe32.th32ParentProcessID;
        break;
    }
} while (Process32Next(hSnapshot, &pe32));

GetProcessImageFileName(hProcess, nameProc, sizeof(nameProc) / sizeof(*nameProc));
std::wcout << "Process Name -  " << nameProc;
```

We managed to find **three new executors** by using this approach! SCP, sftp and our beloved MsoHtmEd.

Sftp was found as an executor with the '- D' flag.

```
C:\Users\Nir Chako>sftp
usage: sftp [-46aCfpqrv] [-B buffer_size] [-b batchfile] [-c cipher]
            [-D sftp_server_path] [-F ssh_config] [-i identity_file]
            [-J destination] [-l limit] [-o ssh_option] [-P port]
            [-R num_requests] [-S program] [-s subsystem | sftp_server]
            destination
```

[Watch how we found the executor]

Afterwards, we tried reversing the process and indeed we found out that the usage of –D really leads us to the use of the CreateProcess API, which is the windows API call for running a new process. In our case – it ran the Exe helper. In potential future cases, it might execute malware as part of a cyber attack campaign.

```
mov     [rsp+108h+lpCurrentDirectory], r13 ; lpCurrentDirectory
mov     [rsp+108h+lpEnvironment], r13 ; lpEnvironment
mov     [rsp+108h+dwCreationFlags], r14d ; dwCreationFlags
mov     [rsp+108h+bInheritHandles], 1 ; bInheritHandles
call    cs:__imp_CreateProcessW
test    eax, eax
jnz     loc_7FF76DFA0773
```

## A Future-Ready Framework: Automated Static Analysis

Don't stop me now...

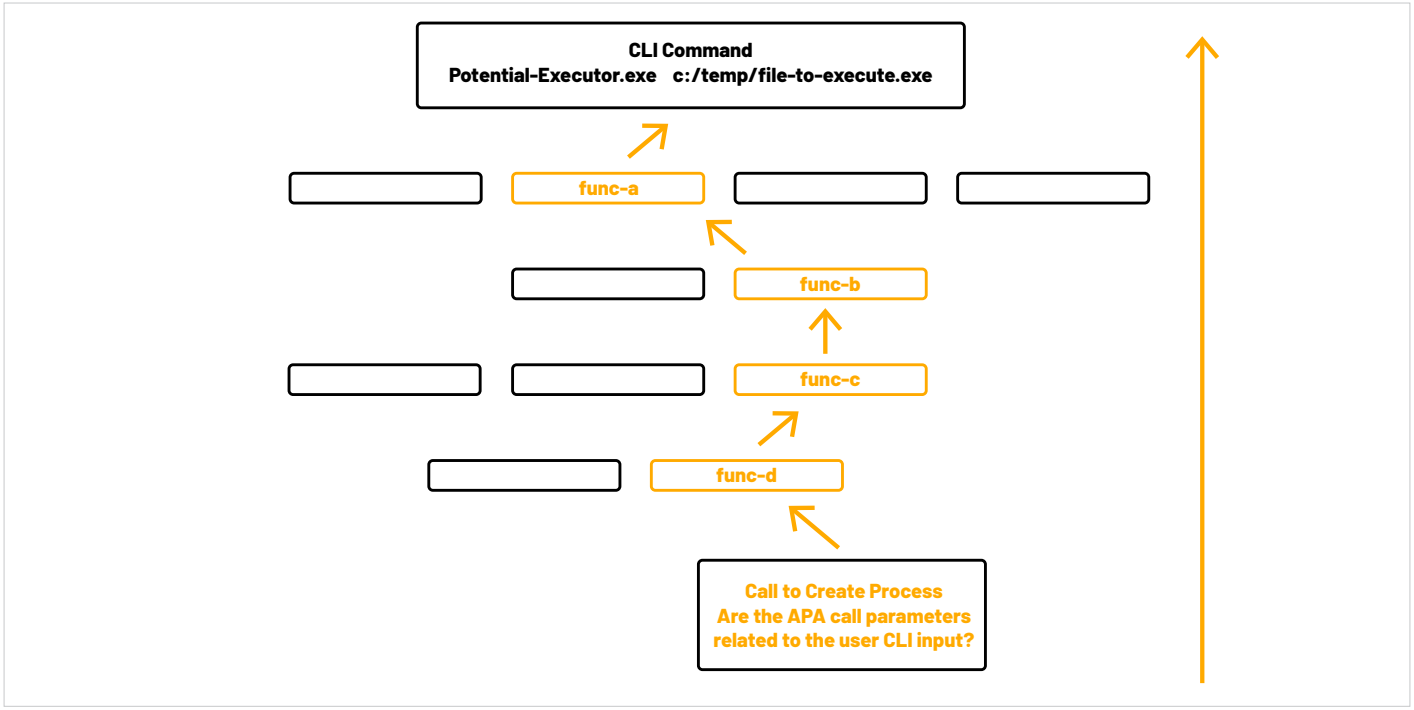
So far, we used  a dynamic analysis method to find new LOLBAS. We attempted to figure out what a manual process of "poking" binaries would entail and then we automated those actions. But this is not the end of our journey.

We have a vision for an automated static analysis approach.

In a similar manner to the dynamic analysis approach, the first thing we'll do is to model our manual approach. Then, we'll automate it. The manual approach might look something like this:

1. Look for API calls relevant for execution, just like CreateProcess. Other API calls will be relevant for other LOLBAS functionalities, such as URLDownloadToFile for downloaders.
2. Reverse our way up to the execution point with the command line and arguments.
3. Understand if there's any user input that might influence the arguments passed to this API call and, consequently, the outcome.

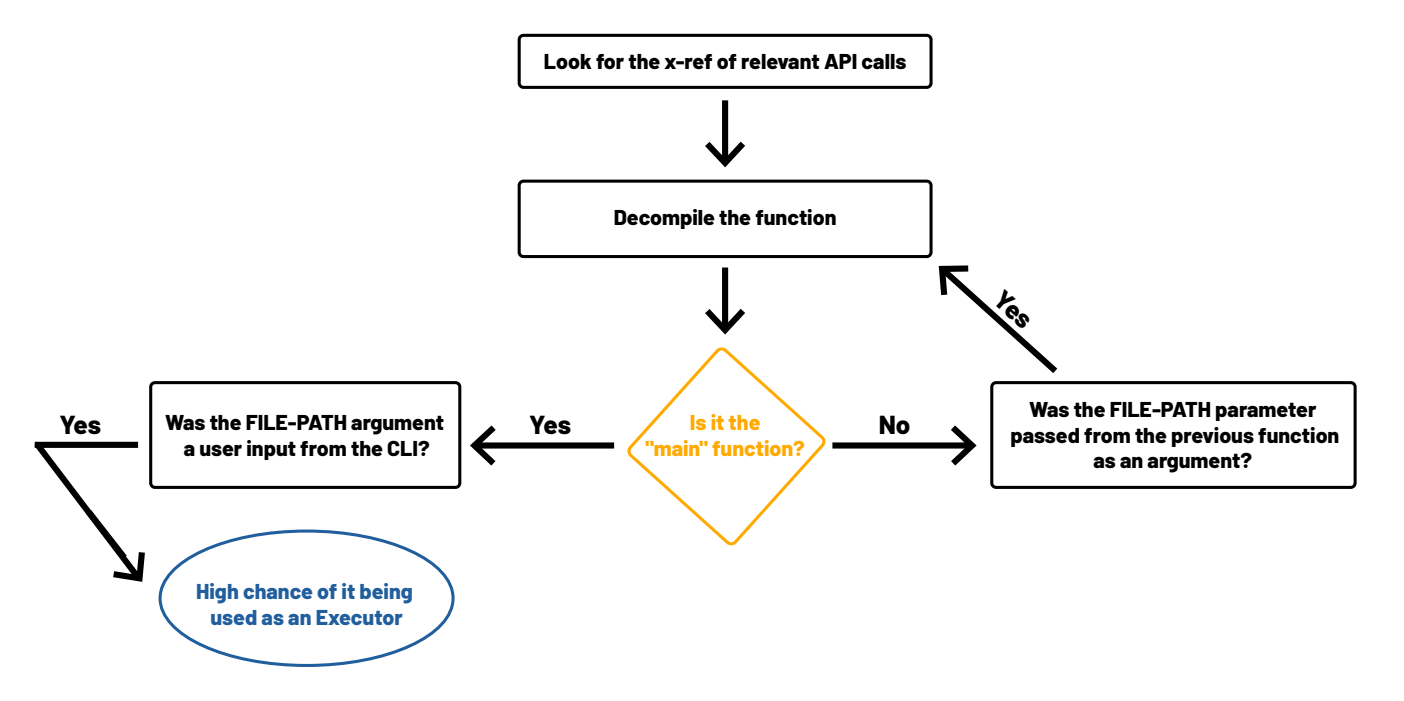
Here's a graphic representation of what it would look like:

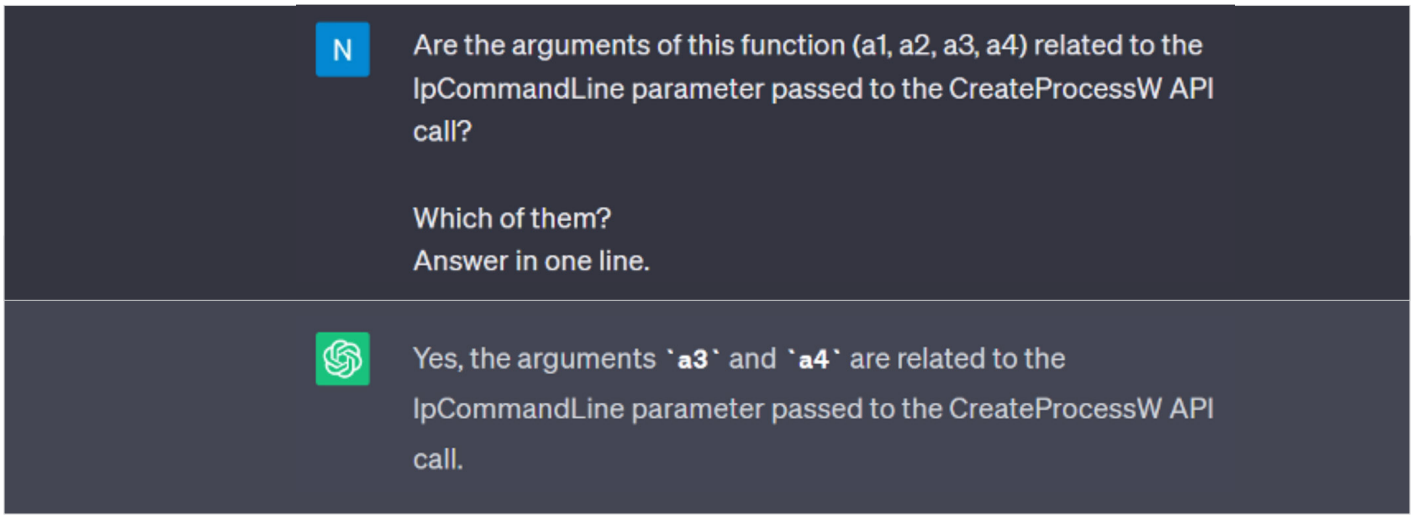



In this case, we can trace our way up from the call to the API function through the code branches until we get to the main function and the CLI arguments.

This process can be automated by orchestrating two different tools. IDApython does the trick for the first part of the algorithm – finding cross-references of interesting API calls and decompiling the functions. The second part of the algorithm, trying to understand if there's a connection between the arguments passed from one function to another, isn't so trivial to develop. In the meantime, instead of breaking our heads, we ran a pretty good POC using ChatGPT to answer this question.

Here's what it looks like:




The proposed static approach pays more attention to the low-level details when compared to our previous dynamic analysis. This includes how the program is built by automating the reverse engineering process to get an understanding of the code itself. As a result, this approach provides better insights into the structure, behavior, and potential issues of the code.

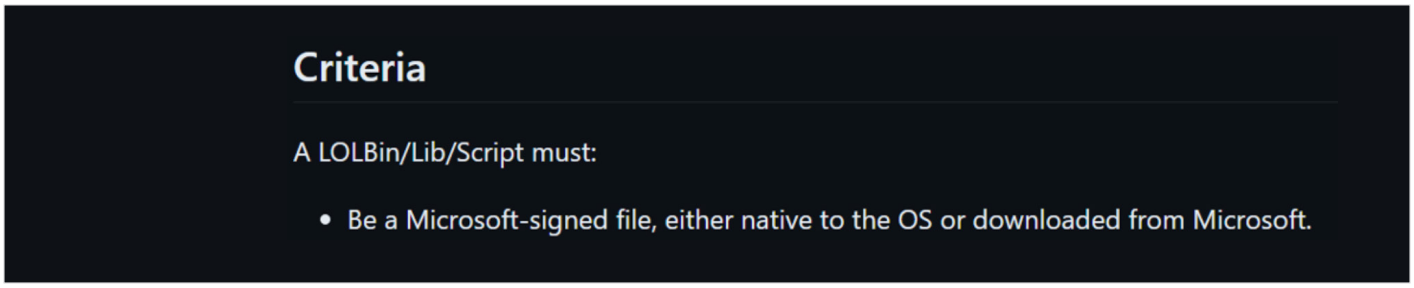
## Takeaways for Red /Blue Teams and Researchers!

We hope our research presents valuable insights on how to discover new LOLBAS for Red/ Blue Teamers and Security Researchers. We also propose an approach that enables leveraging automation for research purposes when we're dealing with a wide range of potential targets.

Let's not wait to hear about an unknown LOLBAS taking part in the next cyber attack campaign. Prior knowledge about these threats can help organizations effectively boost their security measures and mitigate potential risks. Our research provides a roadmap for proactive defense strategies, enabling security professionals to anticipate and counteract evolving threats in the ever-changing landscape of cybersecurity.

## What if I told you...

One last thing before we go. You should be aware that the official LOLBAS project has criteria. These criteria dictate that a LOLBAS must be either – a Microsoft signed file that is native to the OS or downloaded directly from Microsoft.




But...
**But what if I told you that hackers don't really care about official lists?**

What if they have the basic intelligence of you using Zoom, Slack or PyCharm? Can't these tools be used as Downloaders and Executors? Can't hackers use them, the same way they use the official LOLBAS, against you?

Just something for you to think about...

## About the author

**Nir Chako** is a Senior Security Researcher at Pentera Labs. His primary research areas are Network Defense, Linux OS and DevOps Security. Prior to Pentera, Nir spent two and a half years at CyberArk Labs as a Researcher and Research Team Leader and was also the Team Leader of an Israel Defense Force (IDF) Red Team.

For any questions, feel free to contact Nir at labs@pentera.io.

## About Pentera

PENTERA

Pentera is the category leader for Automated Security Validation, allowing every organization to test with ease the integrity of all cybersecurity layers, unfolding true, current security exposures at any moment, at any scale. Thousands of security professionals and service providers around the world use Pentera to guide remediation and close security gaps before they are exploited.

For more info, visit: pentera.io

PENTERA | Pentera Labs™ Research Series

# The LOLBAS Odyssey: Finding New LOLBAS, and How You Can, Too