The process from start to finish is relatively easy. Included will be the necessary configs and files for analysis, some of which are not available in the github. At this time, everything can be self-contained in one VM or machine so long as that system supports nested virtualization. On Intel chips, VT-x and/or VT-d should be enabled. The system should also be using a debian based linux distro for deployment.

First, follow these steps to configure and install cuckoo on your host. This part will take some effort as proper host/guest configuration must be followed to prevent the malware from leaking beyond the confines of the analysis engine.

To begin acquiring samples, create a directory on your storage platform entitled the name of your sample. How you create your organizational structure is up to you. The first script you're going to want to use is the dateSearch.py script. Before executing this script, the date range must first be edited. Open up the script in a text editor and enter your given start and end dates for the search. While you're editing the file, any other signifiers from the VT api can be added. When the script is run, it will prompt your for additional signifiers as well as engine signatures. These will look something like TSPY_ZBOT.BWP. More info can be found here.

Now you should have a number of json files, one per month, stored in ./samples. These files contain a list SHA256 hashes that identify the given samples. Each json file will contain a maximum of 300 hashes distributed over the month. More can be acquired by using an additional API call to gather the next 300 hashes by issuing another search call on the given offset hash, but we opted against this as we needed to limit our API usage.

Next, the downloader.py script should be used. This script will pull both the sample binary and report from VT. Do note that this will *not* pull down behavioral data as this is a waste of an API call. This script will prompt for a ceiling of samples per month to download. This serves as a hard cap to prevent more API calls then needed being used. The max number of samples is then pulled from VT, evenly distributed per month to keep sane data over time. This prevents us from seeing skewed evolution data generated by a higher concentration of samples in a particular short time period.

This script puts the sample binary in the ./bins folder and the sample reports in the ./reports folder. The binaries are named with their appropriate hashes for easy lookup. The reports are stored as json files named by their hashes. As they are in json format, they can be kept in their respective folder for easier lookup. A full spec of the report contents can bee seen here.

Next, the binaries can be ingested into Cuckoo for intense behavioral analysis. Cuckoo ships with a submit.py script that allows you to submit multiple binaries at once by specifying directory paths containing all the samples. Other cuckoo specifiers can be added at this stage, documented here. After the binaries have been submitted, it may take minutes to weeks for full processing to be completed.

Once completed, all the uncovered behavioral analysis will be available for viewing. If the optional Cuckoo web interface was started, they can be easily viewed through that portal. All of the extracted data will be stored in ./cuckoo/storage/analyses/<analysis ID>/. As we were only interested in the system call data, extractor.py was created to perform the required marshaling and extraction of data.

Extractor.py requires some configuration for use. First, the username, password, database name and host must be entered on lines 31-34 to allow proper access to the sql database. Second, the path to the analyses folder must be entered on line 9. Lastly, lines 56-71 contain the code to properly tag the malware with the sample identification. Since cuckoo treats all samples ingested the same, the analyst must use signifiers extracted from the analyses and VT data to properly identify the sample in question. In some cases, Cuckoo was unable to extract the proper date from the sample due to mixed formats. In this case, data from VT scans was cross-referenced to properly tag each sample with the required type.