Malware Analysis of Executable Files and Source Code

Group: Provaxin

Priydarshi Singh
180561

Devanshu Singla
190274

Shubhankar Gambhir
190835

Aman Agrawal
190102

Niket Jain
190547

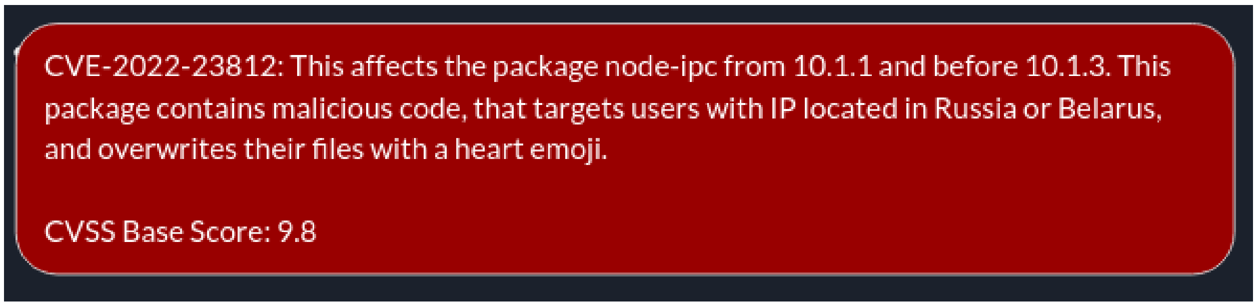
Varenya Srivastava
190943

Motivation

We know that malware attacks are becoming increasingly common in today's world. So there is an ever-increasing need to build tools to identify and restrict malwares.

Usually, malwares run as executable files. So we decided to build a classifier that can detect whether a file is malware or not. We chose to do this using machine learning models, instead of using static rules, because static rules can only detect malware that follows a standard behavior, whereas machine learning models can be used to detect unseen types of malware too.

Further, usually antivirus tools are capable of detecting malwares in the form of executable files. However, there was a recent incident wherein the maintainer of a popular open-source library (node-ipc) added malware in the code of the library to protest against the war in Ukraine. This was the cause of CVE-2022-23812.


CVE-2022-23812: This affects the package node-ipc from 10.1.1 and before 10.1.3. This package contains malicious code, that targets users with IP located in Russia or Belarus, and overwrites their files with a heart emoji.

CVSS Base Score: 9.8

With this incident, there was a lot of uproar in the open-source community to develop ways to mitigate such attacks in the source code supply chain. For this, we have developed two products: pronpm and propip. These products are basically wrappers for the package managers of nodejs (npm) and python (pip). These wrappers just install the libraries like npm or pip would do, and then run malware analysis on the installed code.

Malware Analysis of PE files - Model 1

We used the EMBER dataset (https://github.com/elastic/ember) for this model. It consists of 1.1 million PE files, which has 400K malicious, 400K benign and 300K unlabeled files. There are about 2300 features like entropy of binary, headers related data, etc.

Since we trained and tested the models on our personal computers, we had to limit the total size of the training set to 200K samples (including both benign and malicious files) and then tested the model on a set of 200 K samples. We tried many different models, and then calculated the score for them. The various accuracies that we obtained were as follows:

Model	**Accuracy**
Random Forest Classifier	89.3%
Decision Tree Classifier	84.2%
ADA Boost Classifier	83.1%

The accuracies are good but not as good as we would expect from a critical tool like the one that detects whether or not a file is a malware. So we improved this in the second model.

Malware Analysis of PE files - Model 2

Model 1 was a generic model to detect all classes of malware, it was not as accurate as we would want a useful malware detector model to be. So we decided to build a new model for a specific type of malware, with a reduced number of features. From the previous set of ~2300 features, we selected 54 features based on the approach by Viorel et al.[1] With this reduced feature space and reduced malware domain (restricted to a specific class of malware), we were able to obtain an accuracy of >98% using the K nearest neighbors model. The code and the dataset are provided.

[1] Alexandru-Sebastian, Tufiş-Schwartz. “Detection of Malicious Applications using Machine Learning Methods”

One limitation of this model is that it doesn't work properly for a malware belonging to a different class of malwares, as was demonstrated in the demo.

Malware Analysis of Source Code - pronpm

This was the major product of our project. Our tool pronpm first installs the mentioned package and then performs a dynamic analysis of the javascript code which is present in the node_modules directory.

The javascript code runs in a sandboxed environment. We use an open-source tool (https://github.com/HynekPetrak/malware-jail) to create the sandbox. This sandbox works by redefining the major javascript functions (like eval, fetch, etc.) to just log the arguments instead of executing the original functions. This way, the malware is unable to do any damage, while we are able to understand how it works.

Currently, the tools flags a package as a malware if it downloads a malicious executable, or if it makes a request to a malicious URL, which is identified by using the IP Quality Score API. We also tried using the Google SafeBrowsing API but it did not work for URLs that we know to be malicious, based on real malware samples, so we chose to use the IPQS API.

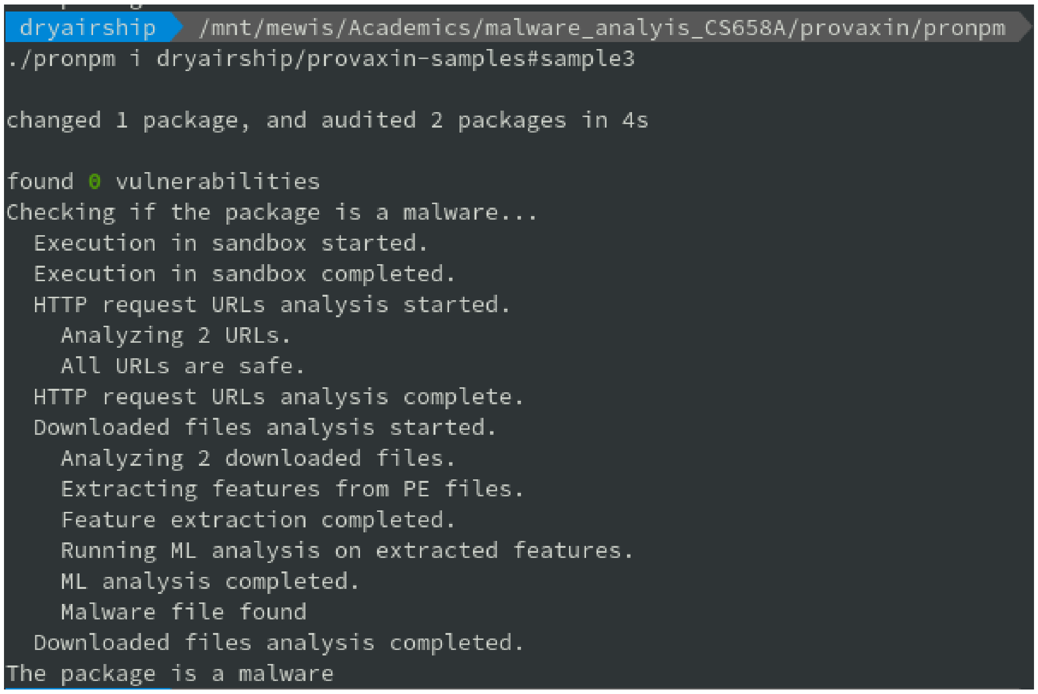
Note that currently, the pronpm tool only supports the `install` (or `i`) command, and can only be used to install one package at a time, like this:

`./pronpm install package-name`

A limitation of this tool is that it currently does not support checking those packages for malware that don't have an index.js file, or those that don't do anything when they are imported (ie. they just expose functions).

For demonstration, we also hosted 4 different samples of npm packages on github, which can be installed using regular npm as well. To install the package (eg, for sample 3) use the name `dryairship/provaxin-samples#sample3`. There are four tags (sample1, sample2, sample3 and sample4) on the same package, which can be used to test the tool.

A sample execution looks like:



```
 dryairship  /mnt/mewis/Academics/malware_analyis_CS658A/provaxin/pronpm
./pronpm i dryairship/provaxin-samples#sample3

changed 1 package, and audited 2 packages in 4s

found 0 vulnerabilities
Checking if the package is a malware...
  Execution in sandbox started.
  Execution in sandbox completed.
  HTTP request URLs analysis started.
    Analyzing 2 URLs.
    All URLs are safe.
  HTTP request URLs analysis complete.
  Downloaded files analysis started.
    Analyzing 2 downloaded files.
    Extracting features from PE files.
    Feature extraction completed.
    Running ML analysis on extracted features.
    ML analysis completed.
    Malware file found
  Downloaded files analysis completed.
The package is a malware
```

(Note that the line "found 0 vulnerabilities" is generated by npm itself, and not our tool).

Please refer to the README file for more examples.

Malware Analysis of Source Code - propip

We couldn't build any sandbox for python files, so we decided to perform a static analysis of the python code. The code is checked for potentially vulnerable calls and imports. It then calculates a score for the python file. The imports are also recursively checked for dangerous code.

We have built the core of propip which performs the actual analysis. The generation of a complete wrapper would require knowledge of the location where the python code is installed (which is not fixed, unlike node_modules in the case of npm).

Conclusion

We built two novel tools - propip and pronpm that have potential for actual use in real world systems. As developers continue to use open-source tools for development purposes, and with increasing incidents of cyberwarfare, the need for our tools will become all the more relevant.