CS7038 - Malware Analysis - Wk04.1

Applying Static Analysis

Coleman Kane
kaneca@mail.uc.edu

February 2, 2017

Applying Your Static Analysis

There are numerous uses for your static analysis findings:

- Document unique contents that could be used to detect copies of this malware or malware family elsewhere
- Describe purpose, functions, or threat level of malware
- Build a utility that can extract structured metadata from future samples

This lecture will discuss some programmatic approaches that I use frequently to achieve these ends.

Additionally, there are a large number of tools out there to provide this service for us. We will focus a lot of our efforts on the engineering of components that would build these tools.

Data Management

It is common to build up a large amount of data from malware analysis. Managing this, in addition to formal malware analysis reports, becomes extremely important and challenging over time, as collected data grows.

I tend to look toward a database solution that has a programmatic API and is easy to integrate into my toolchain. In the class I'll be working primarily with two, but there are many options out there and you are all welcome to experiment with your favorites:

- SQLite - `https://sqlite.org`
- MongoDB - `https://mongodb.com`

There are numerous full platforms for managing this work. Below are a couple popular ones:

- CRITs: `https://crits.github.io/`
- MISP: `http://www.misp-project.org`

Sample Tracking

When dealing with numerous malware samples, we would like to distinguish the exact contents from the file names. For example, we could very easily call our malware `NOTEPAD.EXE` if we wanted to. Because of this, referring to samples by their filename is discouraged, and very quickly can present namespace collision issues.

The most common approach I've seen has been to use checksum (or digest) calculations as a de-facto unique id that is specifically derived from file contents. The common ones I've seen are listed below, and are traditionally dsitributed with most Linux systems:

- MD5 digest - `md5sum` - `https://tools.ietf.org/html/rfc1321`
- SHA-1 digest - `sha1sum` - `https://tools.ietf.org/html/rfc3174`
- SHA-256 digest - `sha256sum` - `https://tools.ietf.org/html/rfc6234`

MongoDB Simple Structured Data Ingestion

I like MongoDB for a number of reasons. It's easy to get started with, and doesn't necessarily enforce schema definition or planning up front, like most SQL databases tend to do.

Additionally, it comes with some nice programming interfaces:

- A JavaScript engine for internal querying and command-line interaction
- PyMongo - `https://api.mongodb.com/python/current/` - LightWeight Python interface
- MongoEngine - `http://mongoengine.org/` - A more structured Python interface with schema and similar features familiar to SQL or MVC developers
- package mgo - `https://godoc.org/labix.org/v2/mgo` - Interface for Go
- MongoDB API List - `https://api.mongodb.com/`

I will stick with using **PyMongo**. For most Python programs, you can install it and then import it into your code like so:

```
import pymongo
```

Easy MongoDB Setup

Generally speaking, MongoDB will automatically create databases and collections (which are kind of analogous to SQL Tables) where they don’t already exist.

If you’d like to do it manually:

```
bash$ mongo cs7038                 # Creates new "cs7038" database
MongoDB shell version: 2.4.9
connecting to: cs7038
> db.createCollection(’malware’)   // Creates new collection named "malware"
{ "ok" : 1 }
> show databases
cs7038  0.203125GB
local   0.078125GB
test    (empty)
> show collections
malware
system.indexes
```

The above steps will instantiate a new database and collection for you to begin storing your data.

Importing Data for a Sample

We will now discuss how to internalize the information gathered from some of the tools that we used on Tuesday. Recall that we were extracting structured data using `objdump` and `exiftool`.

Using our toolset, we have decided that we would like to import the following data into our database:

- MD5, SHA-1, SHA-256 digests as identifiers
- Filename(s) we have encountered the file as being named
- File type information
- List of section names (if an EXE)
- Company Name (if exists)
- Author (if exists)
- File Description (if exists)
- File size
- Compile, Creation, or Modification time from metadata (not filesystem)