Chapter 1: Basic Static Malware Analysis

DATA SCIENCE IN SECURITY

What is Data Science?

- Data science is a growing set of algorithmic tools that allow us to understand and make predictions about data
- generally, has three subcomponents:
  - machine learning
  - data mining
  - data visualization

What is Data Science?

- In the security context
  - Machine learning algorithms
    - learn from training data to detect new threats
    - have been proven to detect malware that flies under the radar of traditional detection techniques (like signatures)
  - Data mining algorithms search security data for interesting patterns
    - such as relationships between threat actors
    - might help us discern attack campaigns targeting our organizations.
  - Data visualization renders sterile, tabular data into graphical format
    - make it easier for people to spot interesting and suspicious trends.

we cover all three areas in depth in this course and show you how to apply them

Why data science matters for security?

- It is critically important for three reasons
  1. Security is all about analyzing data
     - files, logs, network packets, and other artifacts
     - too much manual in Traditional techniques
       - Need handcrafted techniques for each type of attack
  2. Number of cyberattacks has grown dramatically
     - 2008: about 1 million unique malware executables
     - 2012: there were 100 million
     - 2018: more than 700 million
     - manual detection techniques are no longer reasonable

Why data science matters for security?

- It is critically important for three reasons
  3. data science is the technical trend of the decade
     - both inside and outside of the security industry
     - it will likely remain so through the next decade

Applying data science to malware

- malwares are the primary means of threat actors
  - gain a foothold on networks
  - subsequently achieve their goals
  - E.g. ransomwares
  - Some skilled government funded attackers avoid using malware altogether to fly under the radar

Applying data science to malware

- Using a specific application of security data science
  - we aims to show more thoroughly how data science techniques can be applied to a major security problem
  - You can apply it to other areas of security
    - detecting network attacks
    - phishing emails
    - or suspicious user behavior

almost all the techniques you'll learn, can be applied to building data science detection and intelligence systems in general

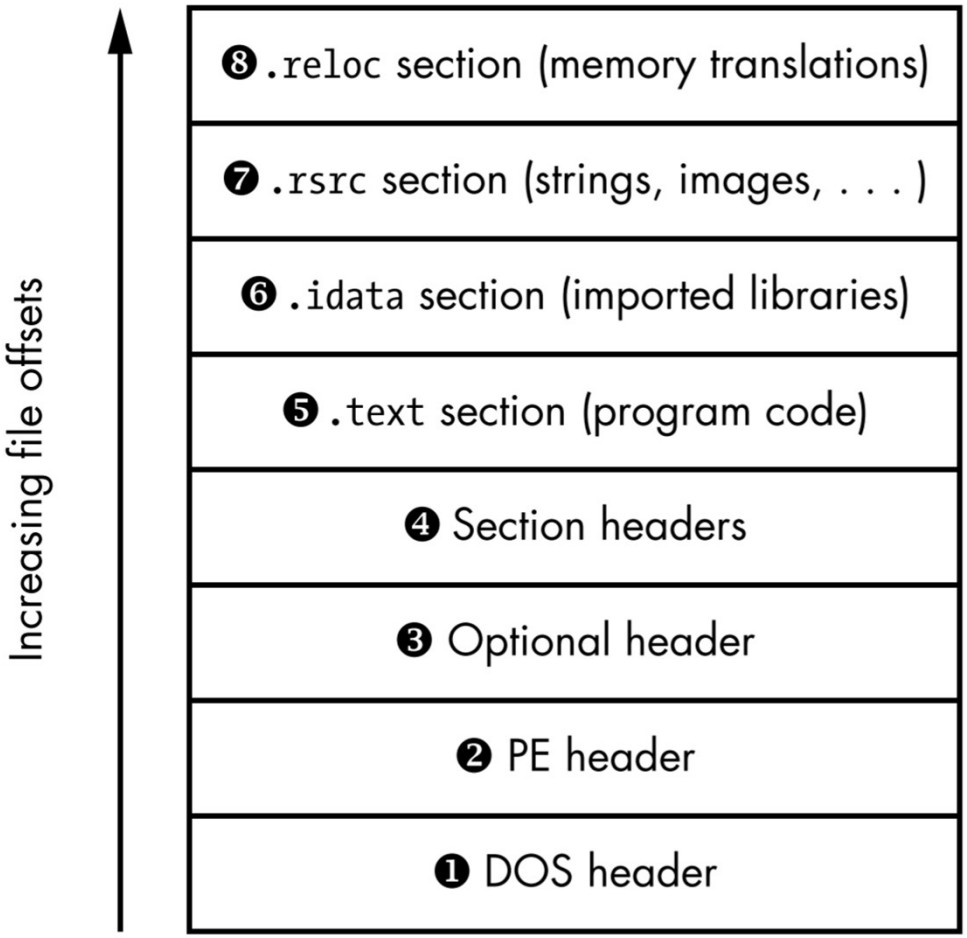
Basic Static Malware Analysis

- Performed by analyzing a program file's
  - disassembled code
  - graphical images
  - print-able strings
  - and other on-disk resources
- It refers to
  - reverse engineering without actually running the program.
- It has some shortcomings
- It can help us understand a wide variety of malware

the Microsoft windows portable executable Format

- describes the structure of Windows program files
  - .exe, .dll, and .sys files
- It is needed in static malware analysis
- was originally designed to do the following
  - Tell Windows how to load a program into memory
  - Supply resources a running program may use in its execution
  - Supply security data such as digital code signatures

the Microsoft windows portable executable Format



the Microsoft windows portable executable Format

- The PE Header
  - defines a program's general attributes such as
    - binary code, images, compressed data, and other program attributes.
  - also tells us whether a program is 32- or 64-bit
  - provides basic but useful information to the malware analyst.
    - includes a timestamp field that can give away the time at which the malware author compiled the file.

the Microsoft windows portable executable Format

- The Optional Header
  - is actually ubiquitous in today's PE executables
    - contrary to what its name suggests.
  - It defines the location of the program's entry point
  - It also defines
    - the size of the data that Windows loads into memory
    - the Windows sub-system
    - the program targets
    - and other high-level details about the program
  - can prove invaluable information to reverse engineer
    - E.g. program's entry point tells where to begin reverse engineering

the Microsoft windows portable executable Format

- Section Headers
  - describe the data sections contained within a PE file.
  - A section is a chunk of data that either
    - will be mapped into memory
    - Or inform the operating system about some aspect of the loading process
  - also tell Windows what permissions it should grant to sections
    - whether they should be
      - Readable
      - Writable
      - or executable

the Microsoft windows portable executable Format

- The .text Section
  - Each PE program contains at least one section of code
    - marked executable in its section header
    - almost always named .text
- The .idata Section
  - also called imports
  - contains the Import Address Table (IAT)
    - lists dynamically linked libraries and their functions
    - is among the most important PE structures
    - it reveals the library calls a program makes
      - can betray the malware's high-level functionality

the Microsoft windows portable executable Format

- The Data Sections
  - can include sections like .rsrc, .data, and .rdata
  - store items such as
    - mouse cursor images
    - button skins
    - Audio
    - other media used by a program.
  - E.g. .rsrc section contains printable character strings

the Microsoft windows portable executable Format

- The .reloc Section
  - A PE binary's code is not position independent
    - it will not execute correctly if moved from its intended memory location
  - .reloc section allows code to be moved without breaking.
    - It tells the Windows to translate memory addresses if the code has been moved
    - translations usually involve adding or subtracting an offset from a memory address

Dissecting the pe Format using pefile

```
$ pip install pefile
```

```
$ python
>>> import pefile
>>> pe = pefile.PE("ircbot.exe")
```

```
for section in pe.sections:
  print (section.Name, hex(section.VirtualAddress),
    hex(section.Misc_VirtualSize), section.SizeOfRawData )
```

Examining Malware images

- PE files may contain Interesting images

```
$ mkdir images
$ wrestool -x fakepdfmalware.exe -output=images
$ icotool -x -o images images/*.ico
```

Examining Malware Strings

- Strings can provide a quick sense of what may be going on inside
  - often contain things like
    - HTTP and FTP commands that download web pages and files
    - IP addresses and hostnames that tell you what addresses the malware connects to
    - and the like

Examining Malware Strings

- Strings can provide a quick sense of what may be going on inside
  - even the language can hint at a malware binary's country of origin
    - though this can be faked
  - a string may be find that explains in leetspeak the purpose of a malicious binary

Examining Malware Strings

- Strings can provide a quick sense of what may be going on inside
  - can also reveal more technical information about a binary
    - you may find information about
      - the compiler used to create it
      - the programming language the binary was written in
      - embedded scripts or HTML
      - and so on
- Malware authors can obfuscate, encrypt, and compress all of these traces
  - even advanced malware authors often leave at least some traces exposed

Examining Malware Strings

- Using the strings Program

```
$ strings filepath | less
```

- finds all printable strings with a minimum length of 4 bytes
- you can change the minimum string length using the –n option

Analyzing strings Dump

- the challenge is to understand what the strings mean

```
$ strings ircbot.exe > ircbotstring.txt
```

```
[DOWNLOAD]: Bad URL, or DNS Error: %s.
[DOWNLOAD]: Update failed: Error executing file: %s.
[DOWNLOAD]: Downloaded %.1fKB to %s @ %.1fKB/sec. Updating.
[DOWNLOAD]: Opened: %s.
--snip--
[DOWNLOAD]: Downloaded %.1f KB to %s @ %.1f KB/sec.
[DOWNLOAD]: CRC Failed (%d != %d).
[DOWNLOAD]: Filesize is incorrect: (%d != %d).
[DOWNLOAD]: Update: %s (%dKB transferred).
[DOWNLOAD]: File download: %s (%dKB transferred).
[DOWNLOAD]: Couldn't open file: %s.
```