CMSC 449
Malware Analysis

Lecture 3

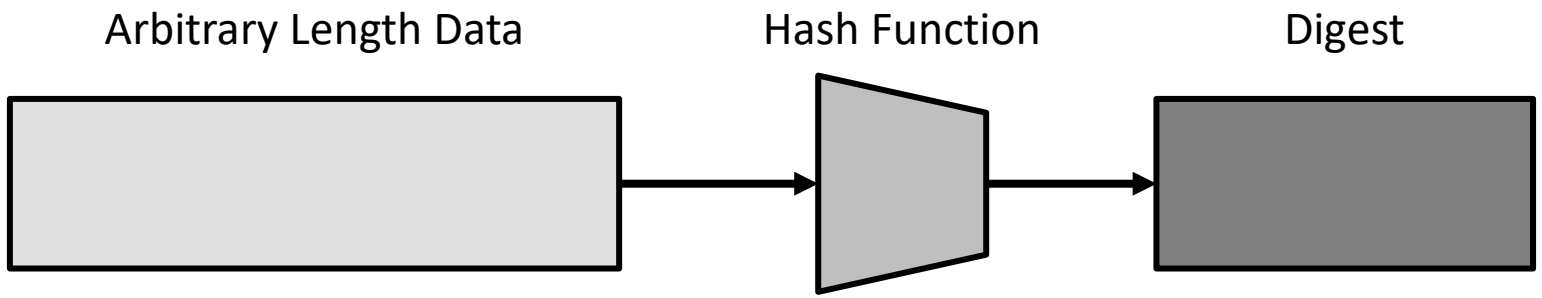
Hashing and Packing

File Hashing

Hash Functions

- Maps an arbitrary-length input to a fixed-length output

- Properties of hash functions:
  - Same input always produces same digest
  - Cannot “reverse” hash digest to find input
  - Very unlikely that two different inputs share a digest



Malware File Hashes

- Files are just a sequence of bytes
  - Use that as input to a hash function

- Malware analysts use file hashing to keep track of individual malware samples
  - Functions as a “unique ID” for the file
  - Shared in open-source threat reports

Common File Hashing Algorithms

- MD5 033f9150e241e7accecb60d849481871
- SHA-1 09067fd23539df1ece704a92b2dca8e32f20f7c8
- SHA-256 5013a9fc3766f0c065d44c9f6a6a8c0101811d7df4860dd50cf627a0d28ed007

- Cryptographically secure – extremely unlikely to have collisions

Downsides of File Hashing

- If a single byte of the file changes, the hash will completely change too

- Malware authors use lots of simple tricks for changing the hash of their malware without changing functionality
  - Like just appending random bytes at the end

Similarity Hashing

File Similarity Problems

- **Clustering:** Grouping similar files together
- **Nearest-Neighbor Lookup:** Given a specific file, search for the files that are most similar to it
- In naïve cases:
  - Clustering is $O(N^2)$ - compare every file to every other file
  - NN Lookup is $O(N)$ - compare given file to every other file
- (Actual clustering / NN algorithms are usually faster)

Similarity Hashing

- Comparing the contents of two files is slow!
  - Especially if they are large

- Instead, use a **similarity hashing function** to compute a similarity digest for each file
  - Same input always produces same similarity digest
  - **Similar inputs produce close similarity digests!**
  - Digests may be fixed or variable length depending upon algorithm

Similarity Hashing

- Can approximate how alike two files are by comparing their similarity digests

- Digests are short, so significantly shorter comparison time

Notable Similarity Hashes

- **SSDEEP:** General-use similarity digest, originally for spam email detection
- **TLSH:** Similarity hash developed specifically for file similarity
- **LZJD and BWMD:** Larger digest size tradeoff for other benefits. BWMD maps file into Euclidean space.
  - Developed by Dr. Raff – Check out the DREAM lab!
- **VHash:** VirusTotal's proprietary hash. No public information.

Metadata Hashing

When File Contents Aren't Similar

- Malware may have very different contents but similar behavior
  - Packing, obfuscation, polymorphism, etc.
- These techniques can drastically change file contents
  - Especially executable code
  - File metadata is often least impacted
- These techniques can defeat similarity hashing and many other kinds of static analysis

Metadata Hashing

- Provide select types of file metadata as input to a hashing function
  - Files with same digest share all of these metadata values
- Can index a database based on metadata hash digest
  - Allows fast querying over extremely large malware collection
- Trick is figuring out which metadata fields to hash!
  - Algorithms based on many different kinds of metadata

Imphash

- Use the imported functions in the order they are listed in the Import Address Table (IAT) as input to a hash function
- The linker builds the IAT based on the order imported functions are called in source code
- So files with the same Imphash probably have very similar source code

Weaknesses of Imphash

- High false positive rate when a file contains few imports

- A technique called runtime linking hides imports from the IAT
  - We'll talk about this later!

pehash

- Based on specific fields from a PE file's:
  - IMAGE_FILE_HEADER
  - IMAGE_OPTIONAL_HEADER
  - IMAGE_SECTION_HEADERs

- Very low false positive rate due to inclusion of many fields

Weaknesses of pehash

- Very strict hash – even small changes in metadata can cause related files to not be identified

- Usually defeated by packers, since they often add/change PE sections in the file

Other Metadata Hashes

- **Rich header hash:** Hash the contents of the Rich header, an undocumented header that appears in all files linked using the Microsoft linker

- **RichPE hash:** My own metadata hash! Based on fields in both the Rich header and PE headers.

- Have also seen hashes of resources, file signatures

VirusTotal Demo

Lab03-03.exe

Cluster and NN Lookup Demo

MOTIF Dataset

Packers

Packers

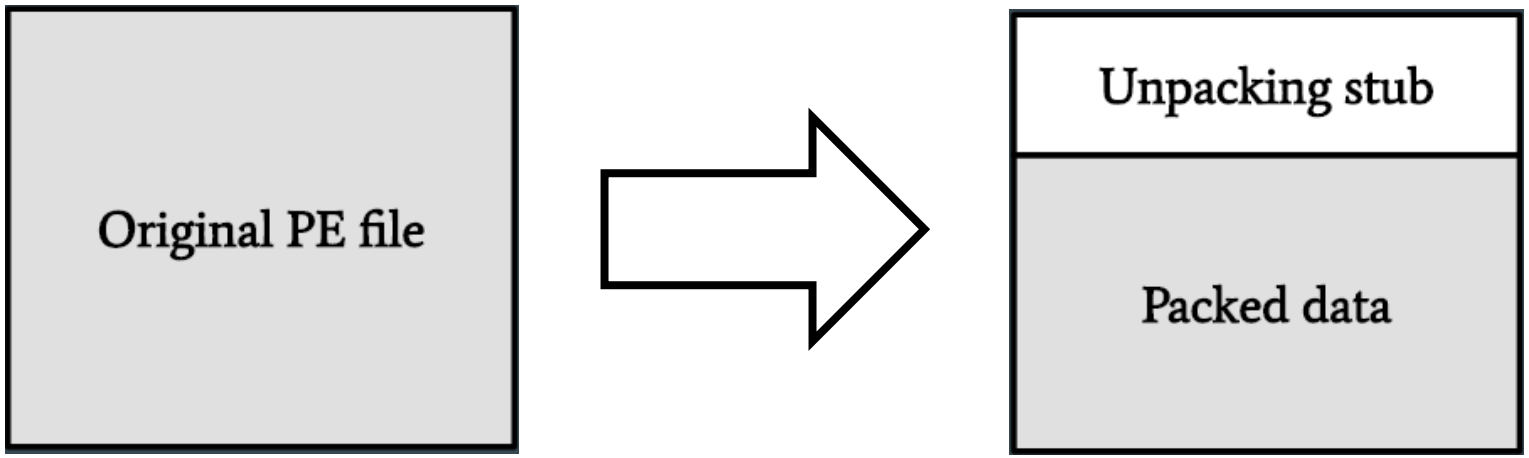
- Malware authors want to make it difficult for you to perform static analysis on their malware

- Use packers to hide:
  - Executable code
  - Strings
  - Imports

How Packers Work

- Compress original program and add an unpacker stub

- When the packed executable is run, the stub unpacks the compressed program into memory and runs it



Indicators that a File is Packed

- File / Section entropy > 7
- Few readable strings
- Unusual section names
- Imports resolved using runtime linking
- Sections with unusual raw / virtual sizes

- PEiD, DIE, VirusTotal are decent at detecting packers
  - Notice lots of some false positives for some packers though

Entropy

- A byte has 2^8 possible values, so a truly random sequence of bytes has an entropy of 8
- Executable code usually has an entropy around 4-6
- Obfuscated / encrypted data usually has an entropy over 7, often near 8

Runtime Linking

- Malware authors don't want you to be able to easily analyze a program's imports

- Can hide a file's imports until it is run by using runtime linking
  - Resolves imports as the file runs
  - Can import functions that are not listed in the IAT

How Runtime Linking Works

- LoadLibrary – Gets a handle to any DLL file on a system
- GetProcAddress – Gets address of any function in a DLL
- Together, allows a program to import a function from any DLL
- There are other ways to do runtime linking, but this is by far the most common technique

Packing Indicators Demo

Lab01-02.exe

Lab01-03.exe