A Scalable, Ensemble Approach for Building and Visualizing Code-Sharing Networks over Millions of Malicious Binaries

Joshua Saxe, Invincea Labs
Thanks to: Giacomo Bergamo, Robert Gove, Alex Long, Sigfried Gold

Invincea Labs, 2014
Identifying code-sharing networks: a key to advancing our ability to understand and detect malware

- Malware authors reuse work: graphical images, code, infection and persistence strategies

- Identifying reused work allows for analysts to reuse analysis

- Shared code networks are important pieces in the attribution puzzle

- Accurate shared code detection over huge volumes of malware is one piece of the next generation of malware detection approaches (we believe)

**BUT:** hard technical problems have stood in the way of accurately recovering malware code-sharing networks

Invincea Labs, 2014
Problem 1, *scale*: huge volumes of malware to evaluate for similarity

A similarity method is most useful when it can scale to real-world volumes of unique malware samples.
Problem 2: Accuracy, defeating multiple malware obfuscation methods

<table>
<thead>
<tr>
<th>Blackhat malware author obfuscation strategy</th>
<th>Whitehat analyst feature recovery response</th>
<th>Limitations of whitehat approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packing</td>
<td>Dynamic analysis + memory dump</td>
<td>Unclear when we should dump memory</td>
</tr>
<tr>
<td>Code randomization</td>
<td>Abstract from code to control flow / data flow graph structures</td>
<td>No abstraction will defeat all code randomization techniques</td>
</tr>
<tr>
<td>Anti-debugging / anti-virtual machine</td>
<td>Static analysis</td>
<td>What if the sample is also very well packed?</td>
</tr>
<tr>
<td>Thin virtual machines and randomized bytecodes</td>
<td>Dynamic analysis + triggering mechanism</td>
<td>Triggering / code coverage is non-trivial and manual</td>
</tr>
</tbody>
</table>

Malware obfuscation makes it hard to always be right when we say whether or not two malware samples belong in the same lineage. Each individual similarity analysis technique can be defeated by a smart adversary.

Invincea Labs, 2014
More detailed summary of state of the practice: surface feature similarity

- **Ssdeep**: file format independent byte-level homology detection
- **Pehash**: portable executable similarity with a Boolean range
- **Imphash**: portable executable similarity with a Boolean range
- **These are the best approaches because they scale, but they are brittle and low-cost for adversaries to defeat**

---

**peHash: A Novel Approach to Fast Malware Clustering**

*Georg Wicherski*

RWTH Aachen University

*gw@mwcollct.org*

**Abstract**

Data collection is not a big issue anymore with available honeypot software and setups. However, malware collections gathered from these honeypot systems suffer from generating multiple binaries with distinct message digest sums that belong to the exact same specimen and therefore pollute malware databases as well as automated analysis.

**SimHash: Hash-based Similarity Detection**

*Caitlin Sadowski*

University of California, Santa Cruz

supertiles@ucsc.edu

*Greg Levin*

University of California, Santa Cruz
glevin@cs.ucsc.edu

December 13, 2007

**Abstract**

Most hash functions are used to separate and obscure data, so that similar data hashes to very different keys. We propose to use hash functions for the opposite task an aid to search. A standard technique in similarity detection is to map features of a file into some high-dimensional space, and then use distance within that space as a measure of similarity. Unfortunately, this typically involves computing the distance between all...
Current research: usually deeper but often non-scalable feature similarity

- Dynamic execution approaches have the “coverage problem”, and take a long time to execute each sample
- Deep static approaches are time consuming and can also have a coverage problem
- The call graph is not directly observable in the program text and can only be fully recovered using often intractable symbolic execution techniques
- Ultimately, deep methods are necessary but are very hard to scale and have inherent limitations

Invincea Labs, 2014
Our idea:

Scalable *and* deep shared-code network identification using ensemble code-sharing, locality sensitive hashing, and approximate nearest neighbor search

• While a malware author may succeed at obfuscating some subset of the malware feature domains, it’s difficult to succeed in obfuscating across all software feature domains

• By integrating similarity sensors over multiple domains and combining sensor readings on a *pairwise* basis we can more accurately detect sample similarity

• *Probabilistic data-structures (feature hash indexing and stochastic feature counting)* allow us to scale to millions of samples

Invincea Labs, 2014
Evidence for the merits of the ensemble approach: four similarity matrices on the same ~2K samples (white boxes show families we have detected)

Combining matrices can produce better results

Looking at individual matrices can reveal what obfuscation techniques individual families use

Invincea Labs, 2014
Evidence for the merits of the ensemble approach: four similarity matrices on the same ~2K samples

(white boxes show families we have detected)
Evidence for the merits of the ensemble approach: four similarity matrices on the same ~2K samples.

(white boxes show families we have detected)
Evidence for the merits of the ensemble approach: four similarity matrices on the same ~2K samples

*white boxes show families we have detected*
Evidence for the merits of the ensemble approach: four similarity matrices on the same ~2K samples (white boxes show families we have detected)
Evidence for the merits of the ensemble approach: four similarity matrices on the same ~2K samples (white boxes show families we have detected)
Scalable similarity network detection prototype design

Static and dynamic feature extraction

Semantic strings data  Graphical asset / image data

PE metadata  Markov partition method for dynamic API call traces

Ensemble Matrix

Feature “sampling” and indexing approach, “OR” technique over matrices to get final similarity matrix

Scalable locality sensitive hashing based similarity index

Visualization

Invincea Labs, 2014
Deep dive into how we calculate sample similarity:
Step 1, assigning malware features appropriate weights

**Goal:** Not all features extracted from malware should be considered equally when looking for shared code / shared provenance relationships between samples, the features that are more *rare* should be considered to be more important when they match.

**Problem:** Counting occurrences of features at scale is hard; unbounded number of features, unbounded number of observations, bounded memory and CPU resources (*example:* we estimate there would be 5.1 billion unique printable string features for a 30 million sample dataset)
**Solution:** Morris counting for compact approximation of item counts

- Store only exponent values in counter
- Randomized counter increment value approximates true magnitudes

and ... adapted bloom filter for indexing of Morris counters

- Shared Bloom filter’s properties; no false negatives, prone to false positives
- 8-bits per register, one hash function, 32MB data structure
Step 2, comparing malware samples for similarity: weighted Jaccard calculation

- Weighted Jaccard measures the size of the intersection of A and B, weighted by the importance of the features in the intersection (we use rarity, with some caveats, as a proxy for importance)

- Good, intuitive measure of similarity between two sets of features extracted from malware samples

- Thus would be appropriate to compute ‘weighted Jaccard’ similarity matrix for each of our ensemble components

- ... but it’s slow to compute for every pair of malware samples

Universal problem in similarity network detection: computing pairwise similarities is **quadratic**
Step 3, scaling pairwise similarity calculations:
Binary version of weighted ‘Minhash’ algorithm

- Minhash is a dimensionality reduction technique
- Allows for fast *approximate* computation of pairwise set similarities
- We’ve modified the minhash algorithm to further reduce feature vectors to binary vectors
- We can then use fast XOR based Hamming-distance calculation to approximate minhash, which in turn approximates weighted Jaccard

Hash malware features using $k$ 32-bit hash functions
Pull out minimum value under each of $k$ hash functions
Modulo each value by 2 to produce 1-bit value
Hamming distance over bit vectors, adjusted by null hypothesis (no set overlap) to predict Jaccard distance

Invincea Labs, 2014
Overall Similarity-Network based Clustering Performance

- 10k extremely well labeled samples used in test
- Consistent results in tests where the evaluated samples the same, regardless of sample size
- Perfect performance on 10 out of 21 clustering tests
- Precision of 1.0 on all tests
- Dramatically outperformed ‘ssdeep’ baseline (about double average adjusted rand index)
- Highly scalable; we’ve clustered 2M samples and could cluster more with appropriate hardware

Dynamic & static obfuscation

Invincea Labs, 2014
Demonstration of malware similarity network / clustering tool

Invincea Labs, 2014
Questions and Discussion