ENDGAME.
BinaryPig: Scalable Binary Data Extraction in Hadoop

Created By:
Jason Trost, Telvis Calhoun, Zach Hanif
Bringing data science to cyber security, allowing you to sense, analyze and act in real time.
Agenda

• The Problem
• BinaryPig Architecture
• Code and Implementation Details
• Analysis and Results
• Demo
• Wrap-Up
Background

2.5 years
20M samples
9.5TB of malware
MO MALWARE, MO PROBLEMS
Malware data mining is useful

- Threat intel feeds
- Contextual enrichment on events
- Machine learning models
Pre-BinaryPig: Storage Issues

- We kept running out of disk!
- We lost samples when NFS nodes failed.
Pre-BinaryPig - Processing Issues

- No Data Locality.
- Node failures were catastrophic.
- Hard to add new analysis scripts.
Pre-Binary Pig - Data Exploration Issues

- How can I share my findings for greater fame and glory?
- Create a table schema for every analysis script?
- RDBMS failure is worse than zombie apocalypse.
We needed a system that...

- Scales to our historical data
- Recovers from failures
- Grows through scripting
- Supports dynamic schemas
- Searchable via the web
BinaryPig

- Simple DSL
- Pluggable analytics
- Plays nice with existing tools
- Enables rapid iteration
- HDFS, scalable, replicated
- Aggregate malware samples into sequence files
BinaryPig - Processing

- Hadoop - robust, distributed, with data locality
- Apache Pig - Extensible, simple
• UI - turns your grandma into a data scientist
• Elasticsearch - schemaless, replicated, awesome
Yet Another Framework?

- Malware tools didn't scale
- Hadoop does not play well with small binary files
- Hadoop did not integrate existing malware analysis tools
Code and Implementation Details
BinaryPig is easy to use!
BinaryPig Ingest Tools

- Generate sequence file from directory containing malware samples
  
  ./bin/dir_to_sequencefile <malwareDir> <hdfsOutputFile>

- Generate sequence file from archive
  
  ./bin/archive_to_sequencefile <archive> <hdfsOutputFile>
BinaryPig Loaders

- Converts raw data to a tuple
- Executing Loader
  - Executes a specific script/program on a file written to a logical path
  - Example: Hashing
- Daemon Loader
  - Writes binaries to a path, and provides those paths to an already running analysis process
  - Example: Clamd
Optimizations in BinaryPig

• To leverage pre-existing tools, we had to write malware binaries to the local filesystem on the worker nodes
  o Note: local copy, not network copy
  o We optimized this to use /dev/shm/ instead

• Quick scripts are great for rapid iteration, but...
  o Interpreter startup time can dominate execution time
  o Creating small, long running, analytic daemons provides a huge speedup for frequently used tasks
  o i.e. the clamscand model of execution
BinaryPig: Loader Implementations

- Generic Script Loader
- Generic Daemon Loader
- ClamAV Loader
- Yara Loader
- Hashing Loader
strings.sh:

#!/bin/bash
strings "$@

strings.pig:

define Loader com.endgame.binarypig.loaders.ExecutingTextLoader;
data = LOAD '$INPUT' USING Loader('strings.sh');
DUMP data;
BinaryPig supports non-PE32 files

- Handles more than just malware...
  - Image analysis
  - PDF data extraction
  - APK extraction
  - Any small binary files
## Web Interface

### Search

Search

```
peid.packer:"Microsoft Visual C++ 8"
50 max results
```

Showing 50 of 2,025 results for peid.packer:"Microsoft Visual C++ 8"

<table>
<thead>
<tr>
<th>peframe.auspicious.api_antidebug</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnhandledExceptionFilter</td>
<td>7674</td>
</tr>
<tr>
<td>TerminateProcess</td>
<td>6932</td>
</tr>
<tr>
<td>IsDebuggerPresent</td>
<td>3510</td>
</tr>
<tr>
<td>GetWindowThreadProcessId</td>
<td>2488</td>
</tr>
<tr>
<td>FindWindowExA</td>
<td>2108</td>
</tr>
<tr>
<td>FindWindowA</td>
<td>1827</td>
</tr>
<tr>
<td>OutputDebugStringA</td>
<td>1260</td>
</tr>
</tbody>
</table>

Showing 1 to 19 of 19 entries

Search:

<table>
<thead>
<tr>
<th>peframe.meta</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileVersion</td>
<td>25941</td>
</tr>
<tr>
<td>ProductVersion</td>
<td>25160</td>
</tr>
<tr>
<td>ProductName</td>
<td>23404</td>
</tr>
<tr>
<td>InternalName</td>
<td>22932</td>
</tr>
<tr>
<td>OriginalFilename</td>
<td>22878</td>
</tr>
<tr>
<td>CompanyName</td>
<td>21326</td>
</tr>
<tr>
<td>Translation</td>
<td>13670</td>
</tr>
</tbody>
</table>
Analysis and Results
Malware Census

• 20 Million unique binaries
  • ~94% PE format
  • ~6% are mostly Android APK's
• 5 hours to run historical set
General Findings

% Samples for Top 100 Imports

% Samples of Top 100 Resource Identifiers

% Samples for Top 100 Languages

% Samples for Top 100 ClamAV Classifications
Feature Extraction

• Our core motivation was to drastically improve the experience of validating research.
• Packer identification
  - Overall and Sectional Entropy
  - Kolmogorov Complexity
  - Section and resource names
  - Section flags
• Import tables
• Function Calls
• Resource hashes and subfeatures
Feature Depth

- PEHeaders are shallow
  - Easy to manipulate
  - Less resolution than reverse engineering features
  - File metadata is also low resolution
- Headers provide excellent fast features
- Headers are often ignored
- Work the analysis around the feature resolution
  - Ignore tight clusters, go for wide ones
  - Triage, not true classification
Clustering Results

- Triage for dynamic analysis winnowing
- Largest cluster: 377,882 samples
  - Three malware families contained within
  - Second largest: 124,894 samples
- Validation is tricky
  - Manual validation cannot be entirely avoided
  - Cluster meanings change with feature sets
  - Cannot just go off of AV results
Icon Features

- Pixel based features
  - Brightness
  - Color values
  - Pixel density
- Cryptographic and fuzzy hashing
  - Perceptual hashes
- Edge detection
Icon Results

- Icon clustering
  - Groups do not just include family lines
  - Copycat malware is shown as well
  - Clear indications of malicious intent
- Method of infection can be extrapolated
  - Phishing
  - Obfuscated executables
  - Adware (more than we expected)
  - False positives - popular software detection
Lessons Learned

• Feature Selection
  o Over 500 features in PEheader alone
  o Abundance of features requires pruning

• Interpretation and Validation of Results
  o Manual validation is an unfortunate reality
  o Care has to be taken to ensure that unsupervised learning provides meaningful results
DEMO
Wrap Up
• Rapid Iteration
• Feature extraction
• Clustering analysis for rapid malware triage
• Enables weekly AV scans with latest signatures over previous malware.
• Created binary classifier to improve sample collection and categorize new samples

So What?
Future work

• Compatibility with Pig 0.10.* and 0.11.*
• EC2 tutorial
• More examples/starter scripts
  • Inclusion of some of our Mahout tasks
  • Open source process for that is moving forward
• Better error logging and handling
  Messages should be stored in a separate DB
• Easier deployments
  • Analytic daemons
  • Dependency libs
  • Fabric/Salt/Puppet/Chef
BinaryPig is Open Source!

https://github.com/endgameinc/binarypig
Apache 2 License
We are hiring!

http://endgame.com/careers
QUESTIONS