



ENDGAME.

**BinaryPig: Scalable Binary Data
Extraction in Hadoop**

Created By:
Jason Trost, Telvis Calhoun, Zach Hanif



ENDGAME.

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

IT'S ALL ABOUT THE BEN JAMINS

**MO MALWARE,
MO PROBLEMS**

E.

E.

05211972

BROOKLYN
MINT
FEDERAL RESERVE NOTE

03091997



Big Poppa

**NOTORIOUS
B. I. G.**

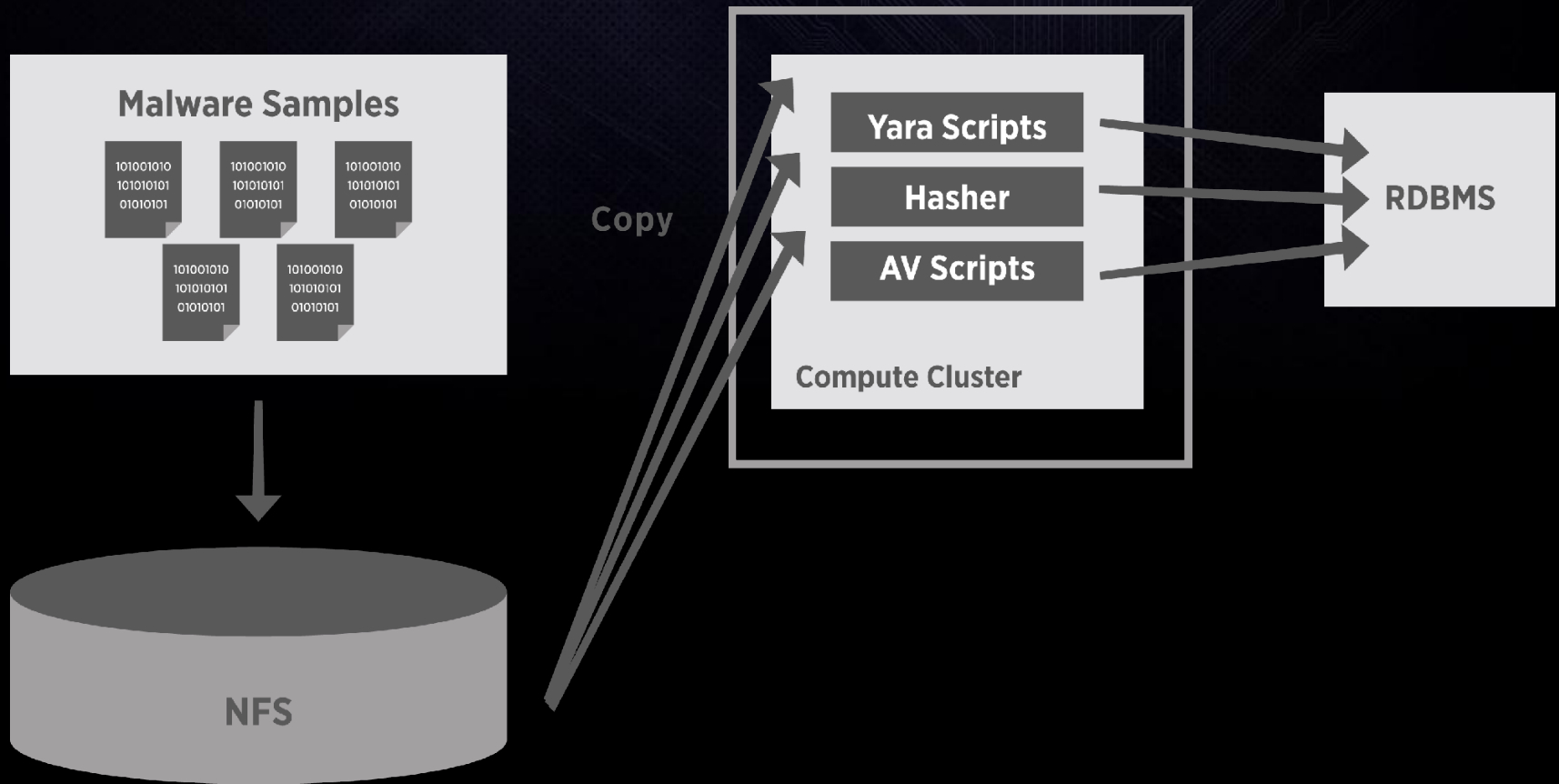
E.

E.

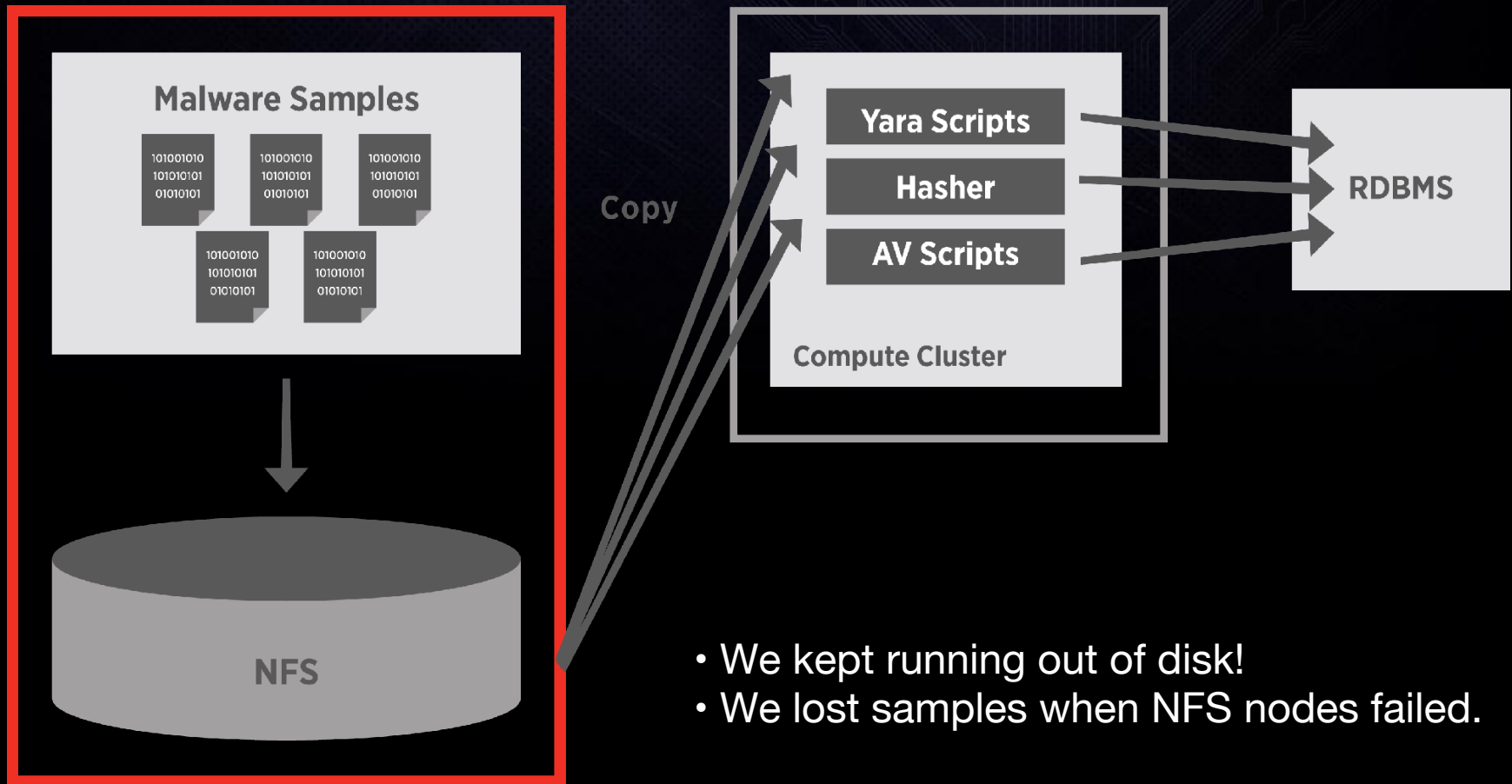
Malware data mining is useful

- **Threat intel feeds**
- **Contextual enrichment on events**
- **Machine learning models**

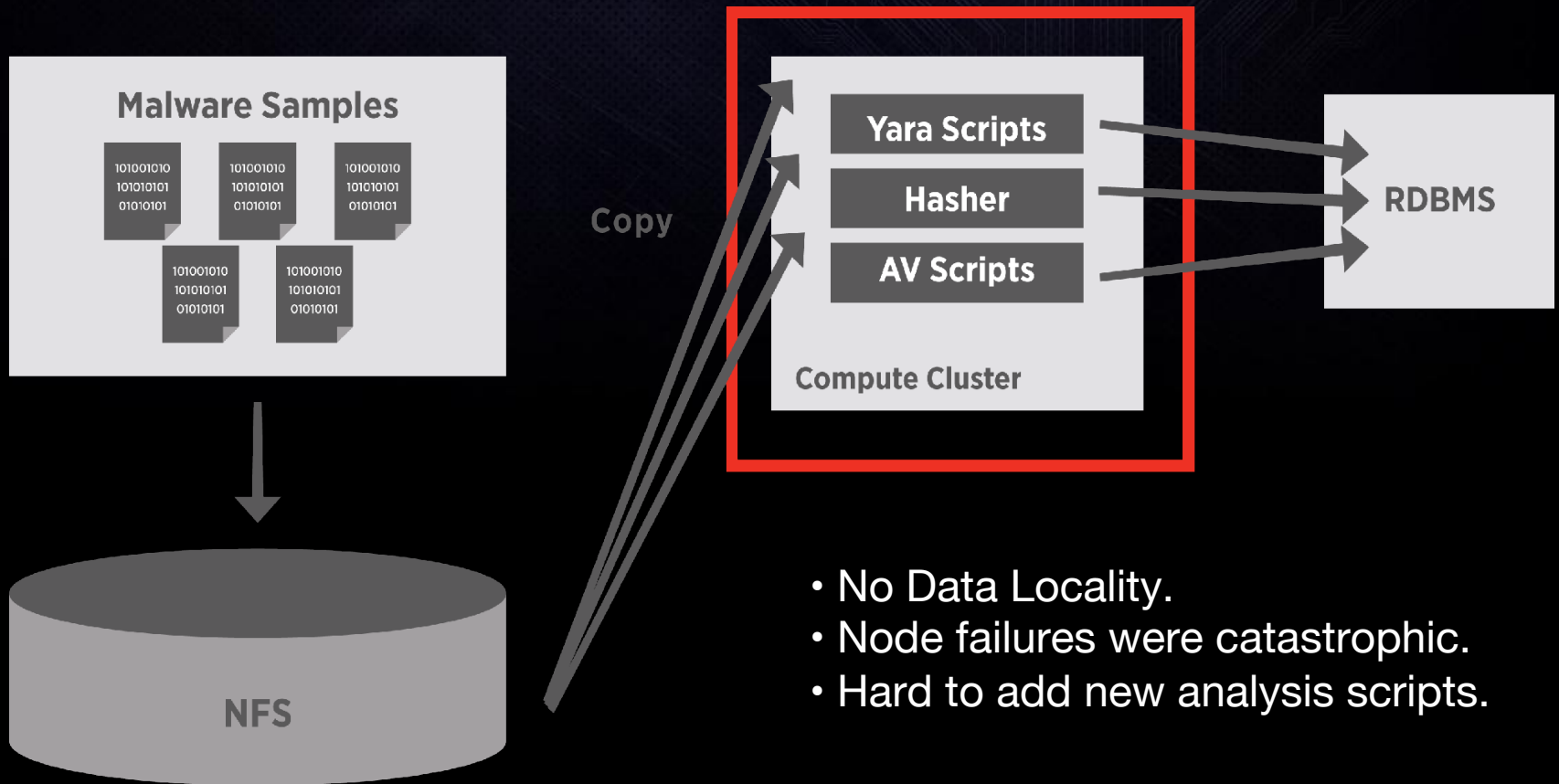
Pre-BinaryPig: Architecture



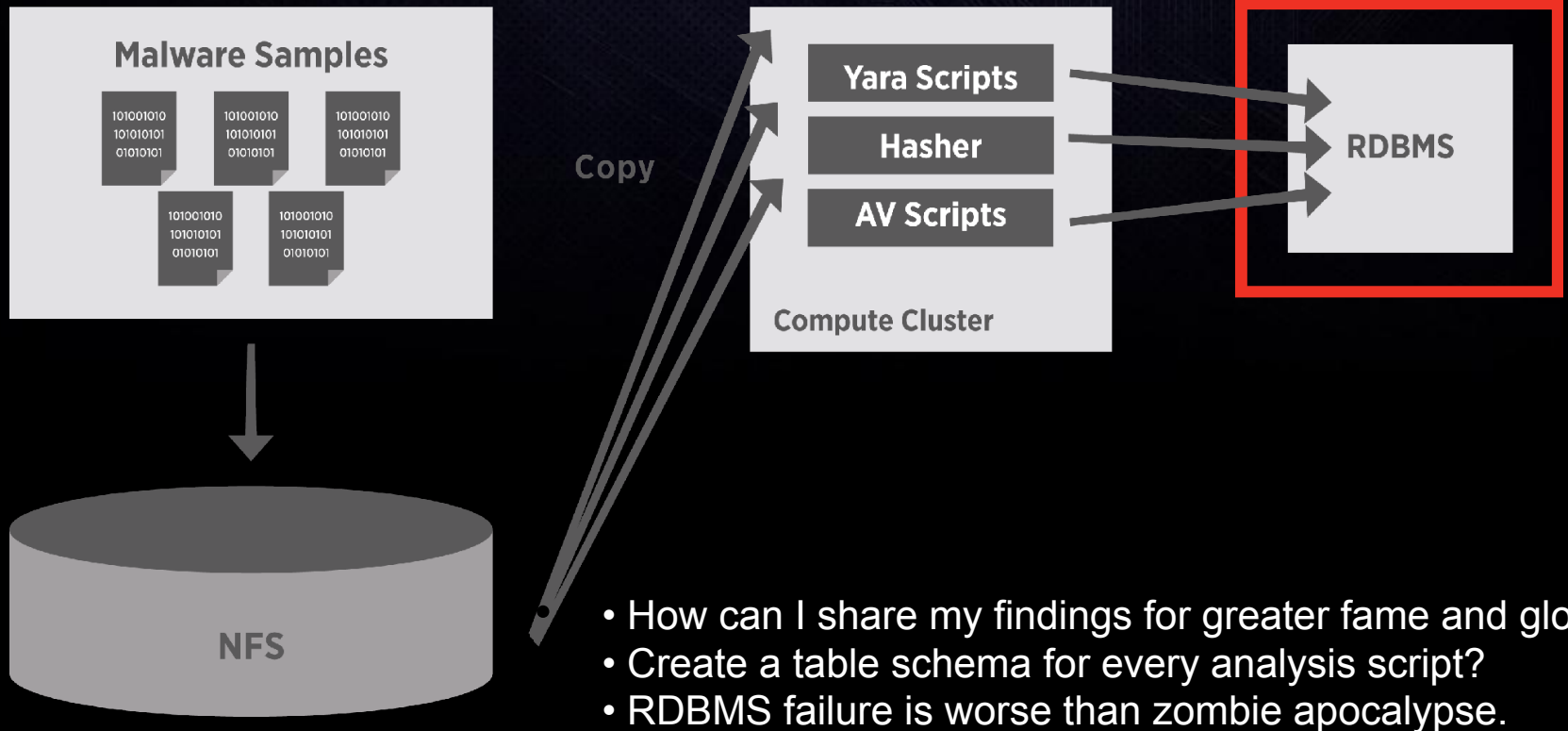
Pre-BinaryPig: Storage Issues



Pre-BinaryPig - Processing Issues



Pre-Binary Pig - Data Exploration Issues



We needed a system that...

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



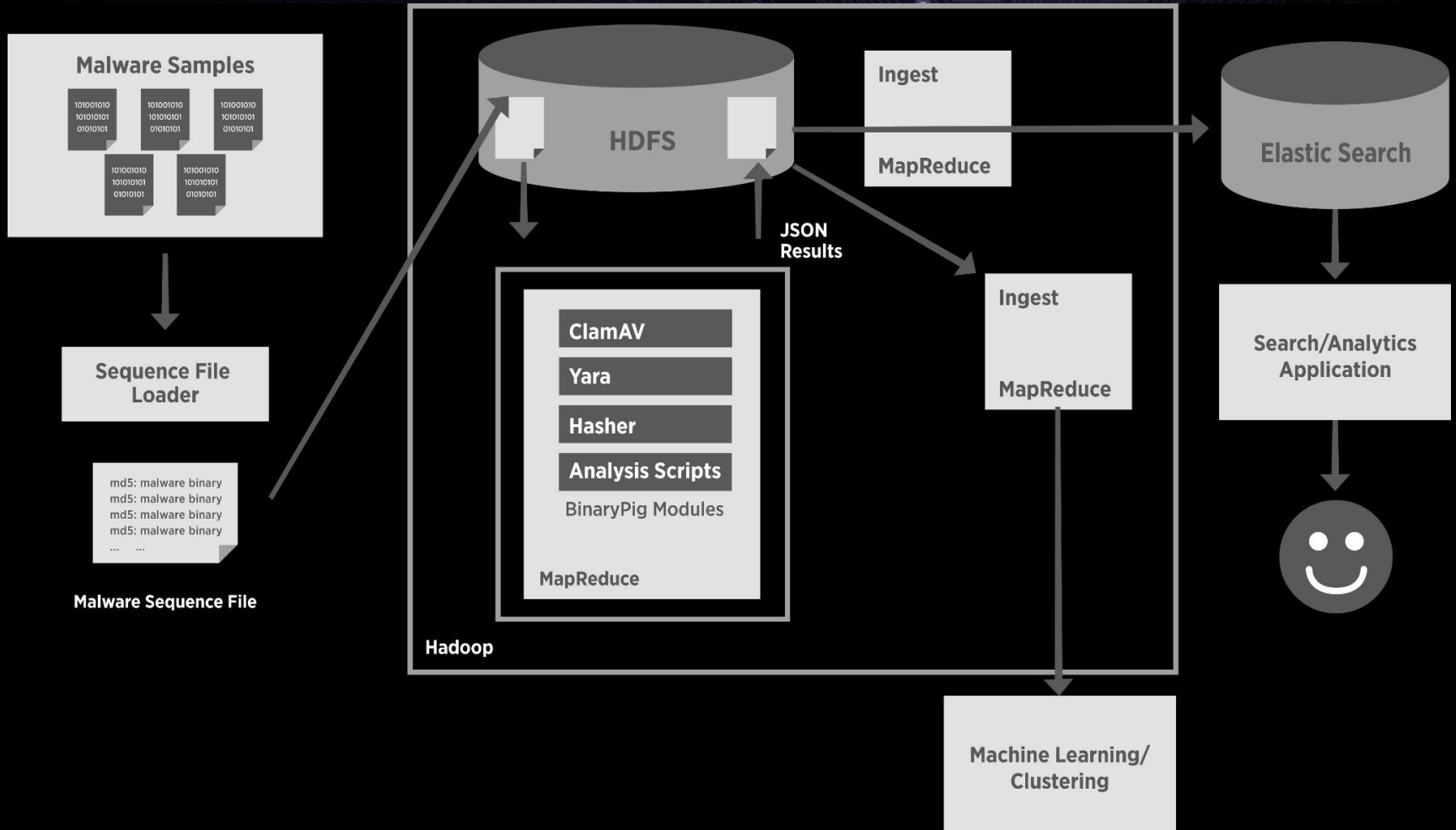
BinaryPig

FRAMEWORK FOR
PROCESSING SMALL
BINARY FILES USING
APACHE HADOOP AND
APACHE PIG

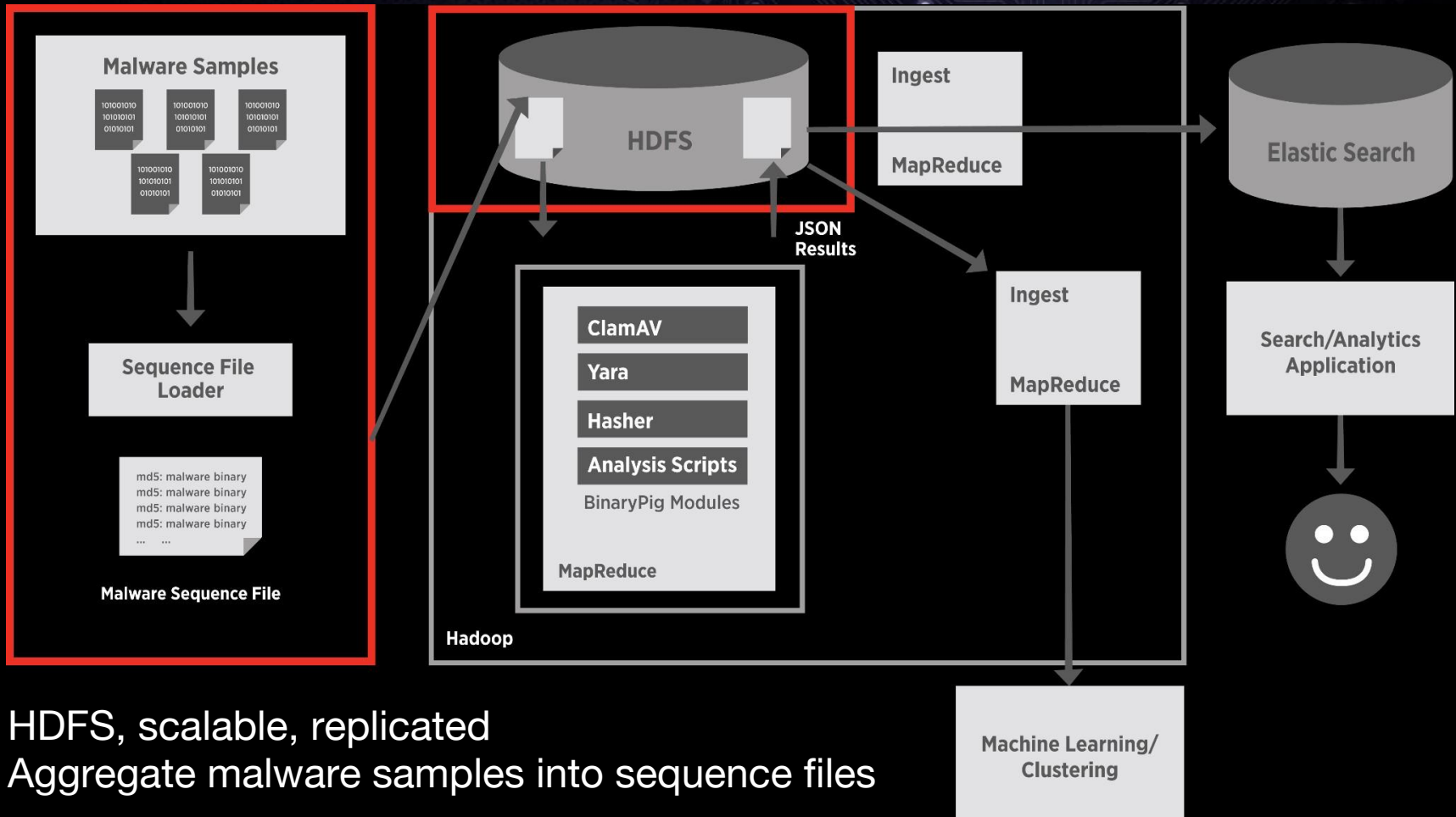
BinaryPig

- **Simple DSL**
- **Pluggable analytics**
- **Plays nice with existing tools**
- **Enables rapid iteration**

BinaryPig

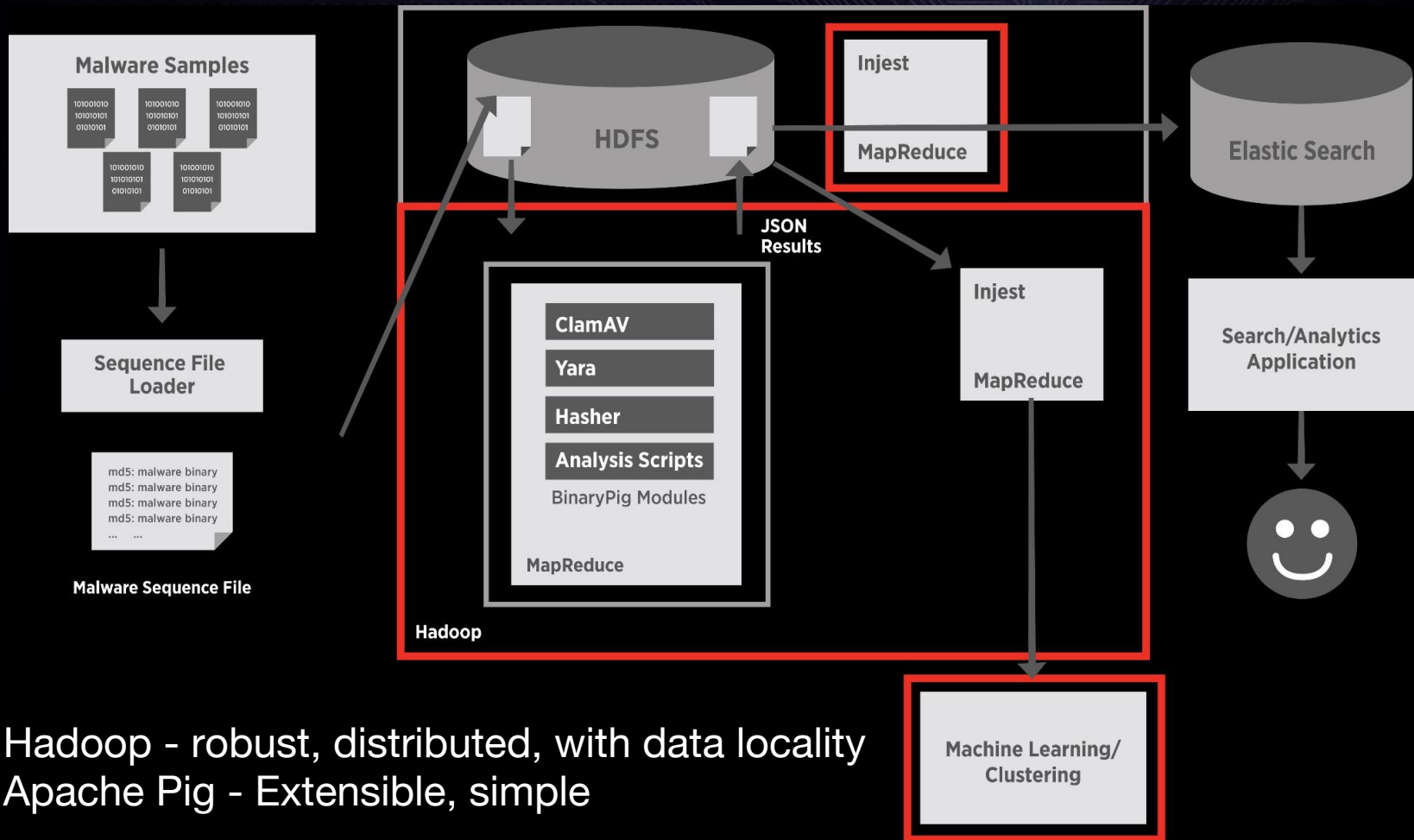


BinaryPig - Storage



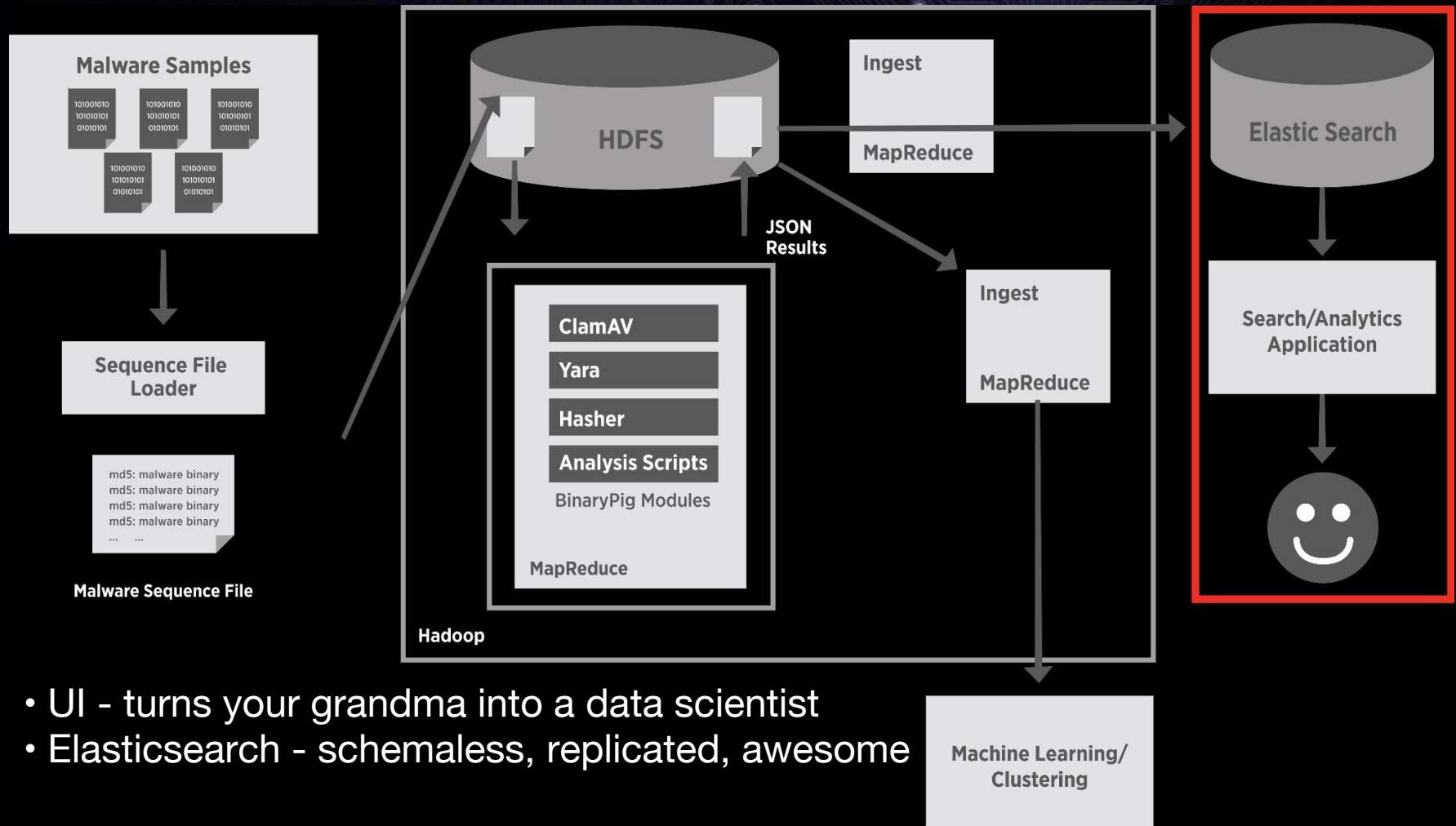
- HDFS, scalable, replicated
- Aggregate malware samples into sequence files

BinaryPig - Processing



- Hadoop - robust, distributed, with data locality
- Apache Pig - Extensible, simple

BinaryPig - Results Exploration



- 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
 - Note: local copy, not network copy
 - We optimized this to use /dev/shm/ instead
- Quick scripts are great for rapid iteration, but...
 - Interpreter startup time can dominate execution time
 - Creating small, long running, analytic daemons provides a huge speedup for frequently used tasks
 - i.e. the clamscand model of execution

BinaryPig: Loader Implementations

- Generic Script Loader
- Generic Daemon Loader
- ClamAV Loader
- Yara Loader
- Hashing Loader

BinaryPig: Scripting

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

BinaryPig

Search

Yara

ClamAV

PE Hash

PE Frame

Search

peid.packer:"Microsoft Visual C++ 8"

50 max results

Search

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

[61cbd81002f59ebc9e78ec8beeda7a25](#) [peframe]

filename: 61cbd81002f59ebc9e78ec8beeda7a25

md5: 61cbd81002f59ebc9e78ec8beeda7a25

sha1: a7dd3fd33b29235601993c842548727a6116

compile_time: 2002-02-10 12:21:49

dll: False

name: 61cbd81002f59ebc9e78ec8beeda7a25

sections: 1

size: 70656

packer: Microsoft Visual C++ 8

num_sections: 1

[62cc7394c5bc93c977718740a63f3327](#) [peframe]

name: IMAGE_DEBUG_DIRECTORY

Characteristics: 0x0

address: 0xDB90

offset: 0x0

TimeStamp: Thu Jan 1 00:00:00 1970 UTC

address: 0xDB94

offset: 0x4

MajorVersion: 0x0

address: 0xDB98

offset: 0x8

peframe.suspicious.api_antidebug	Count
UnhandledExceptionFilter	7674
TerminateProcess	6932
IsDebuggerPresent	3510
GetWindowThreadProcessId	2488
FindWindowExA	2108
FindWindowA	1827
OutputDebugStringA	1260

Showing 1 to 19 of 19 entries

Search:

peframe.meta	Count
FileVersion	25941
ProductVersion	25160
ProductName	23404
InternalName	22932
OriginalFilename	22878
CompanyName	21326
Translation	13676



Analysis and Results

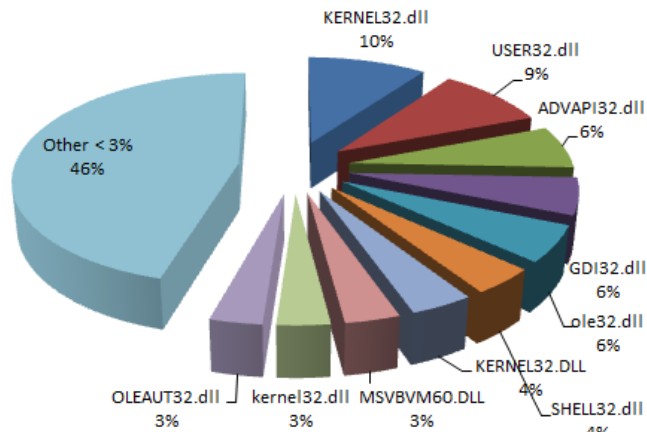


Malware Census

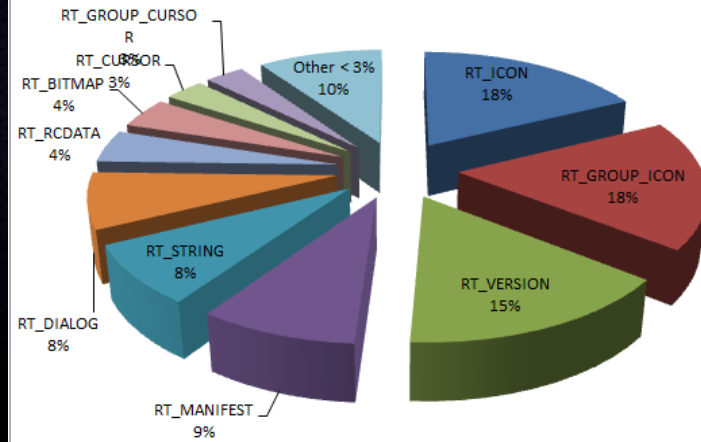
- 20 Million unique binaries
 - ~94% PE format
 - ~6% are mostly Android APK's
- 5 hours to run historical set

General Findings

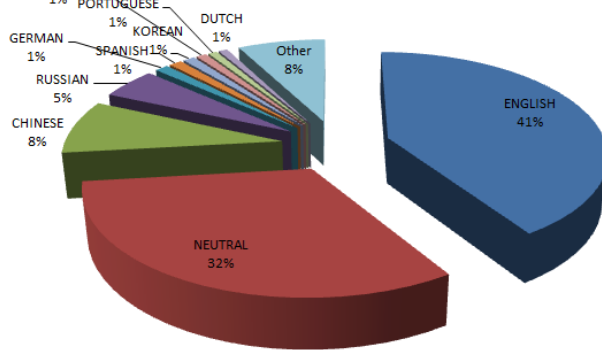
% Samples for Top 100 Imports



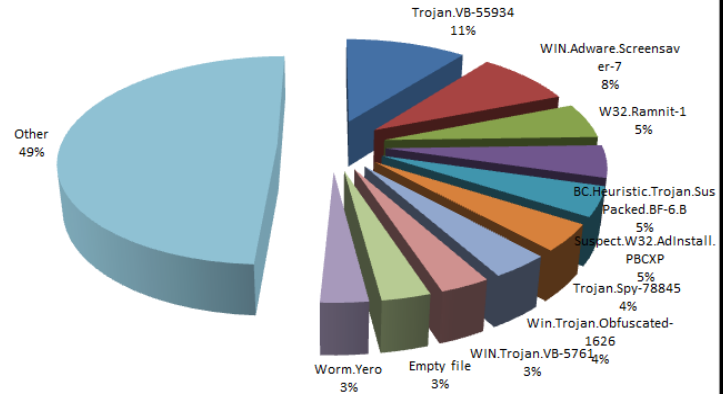
% Samples of Top 100 Resource Identifiers



% 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
 - Over 500 features in PEheader alone
 - Abundance of features requires pruning
- Interpretation and Validation of Results
 - Manual validation is an unfortunate reality
 - Care has to be taken to ensure that unsupervised learning provides meaningful results



DEMO





WRAP UP



So What?

- 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

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

