Variant: A Malware Similarity Testing Framework

Jason Upchurch[1,2,3] and Xiaobo Zhou[3]
[1] Center of Innovation, United States Air Force Academy, CO, USA
jason.upchurch.ctr@usafa.edu
[2] Intel Security Group, Intel Corporation, Hillsboro, OR, USA
jason.r.upchurch@intel.com
[3] Department of Computer Science, University of Colorado, Colorado Springs, USA
{jupchurc,xzhou}@uccs.edu

***Abstract*—This paper describes Variant, a testing framework for projects attempting to locate variants of malware families through similarity testing. The framework is a series of tests and data standards to evaluate recall and precision in tools that attempt to statically measure similarity in implementation of compiled software, specifically in determining code reuse in compiled software to identify malware variants. The paper offers a malware test dataset that has been manually analyzed to provide a gold standard dataset to be used in current and future malware variant detection works. This set provides a much needed resource in standardizing results across numerous works that have, so far, been tested against datasets that are either not reproducible, algorithmically derived, or both. The framework and dataset provided in this paper are used to test several malware detection approaches published in academic works or used in industry. Finally, the paper brings alignment of testing and reporting methods used in malware variant detection to those used in other statical testing methods used in industry and academia.**

I. Introduction

Malware identification is challenging. Traditional signature analysis used in the antivirus industry requires that malware be first identified from known and unknown software. The volume of files that must be analyzed is increasing at a staggering pace, consisting of more than 400K files per day as of the writing of this paper. With the large volume of files queued to be analyzed and triaged, the demand for new and robust solutions is ever present. As such, malware analysis, similarity comparisons of malware samples, and clustering of malware samples is an active area of research in industry and academia.

Papers have been published on finding malware similarity through byte code analysis [1][2][3][4], call graph comparison [5], file similarity [6][7][8][9], and behavioral similarity [1][10]. While these works analyze their respective solutions for performance and accuracy, reproducing these tests for peer review and comparison is problematic. As of this writing, no standard testing set nor procedure has been proposed to evaluate current malware analysis variant detection that measures accuracy against a human analyst and is reproducible for peer review and future works.

Our contribution in this paper is to provide a dataset and testing scheme that is reproducible for direct comparison of results. The paper will comprise of a malware dataset that was manually analyzed by professional malware analyst, a detailed evaluation scheme, and results of the test for publicly available or reimplemented tools used for malware similarity and clustering. Further, the paper will discuss the performance measures used in testing so that future works and/or transitions will have a basis for comparison.

II. Related Research and Motivation

In statistical analysis, a $Gold\ Standard\ Test$ refers to a test or measurement that is the best available under reasonable conditions. Until now, these datasets have been derived through algorithm or sampled through malware repositories without manual analysis. For instance, in [1], samples where identified through algorithm with malware names reported in Virus Total [11] as input. The problem with this approach is that Virus Total's database is ever changing, making it impossible to reproduce the exact test for peer review and/or comparison. In addition, the nature of choosing a dataset by antivirus naming is that antivirus identification is well known to be inexact and flawed. Further, even if these problems where solved, the evaluation is such that it tests the ability of a newly proposed algorithm to match that of another older algorithm, namely the signature driven algorithm of common antivirus products.

The failure of the signature system of antivirus products is one of the driving forces in malware variant detection research. As such, a comparison of proposed solutions cannot be measured on the ability to find malware identified by these signatures. Other works such as [12] have sought to algorithmically change source code in a way to create variants of software packages. While this may be a superior approach to signature identification, the test was targeted toward general similarity comparisons rather than malware, specifically.

To further complicate the issue, in the wild, malware is produced by numerous attackers, malware writers, and packaging algorithms. This leads to a wide variety of techniques used to avoid signature detection and similarity analysis. This crowd sourced approach to malware generation produces a variety of malware variants that are very difficult to reproduce in the lab through algorithm. Thus, the only available solution is to use actual malware that has been sampled from this large crowd source as the basis for comparison. However, as mentioned above, the sampling should not be derived from signature detection algorithms as this would lead only to an evaluation of a new method to produce old results.

As the goal of the paper is to produce a $Gold\ Standard\ Test$, it bares discussion of what this test should comprise. First, is the dataset itself. The dataset should represent data that is found in the wild. Knowledge of

the dataset should be derived from the best available source under reasonable conditions. The tests against the dataset should enable comparison of performance in terms of both speed and accuracy on like systems. While similarity tests exists in works such as [12], we know of no tests with both a fixed dataset and standard, comparative results for future works that have been published specifically for malware variant detection.

Our contributions in this paper are:

1) A New Gold Standard Dataset for malware variant detection
2) A Standardized Nomenclature to bring malware variant detection inline with other statistical classification projects
3) An initial comparative analysis of projects within the field of malware variant detection based on static analysis

III. Method: Measurements and Dataset

A. Measurements

Variant is presented as a solution toward evaluating malware variant detection works. As a statistical analysis $Gold\ Standard\ Test$, Variant will report on accuracy in terms of $Precision$, $Recall$, and $F_1Measure$. Recall that $Precision$ is the positive predictive value, or the faction of retrieved samples that are identified correctly. In this type of binary classification, $Recall$ is the sensitivity of an algorithm to retrieve relevant instances. In simpler terms, $Precision$ is those samples picked from a field that are correct, divided by the total picked. $Recall$ is those samples picked from a field that are correct, divided by the total number of actual positive samples.

$Precision$ and $Recall$ are specific terms that describe general accuracy and, most often, as one increases, the other decreases. To find a balance between these values it is common to take the harmonic mean of $Precision$ and $Recall$ to generate $FMeasure$ or balanced F_1score. For the purposes of this paper we formally calculate the following values:

1) $Recall = \frac{RelevantSamples\ \cap\ RetrievedSamples}{RelevantSamples}$
2) $Precision = \frac{RelevantSamples\ \cap\ RetrievedSamples}{RetrievedSamples}$
3) $FMeasure = 2 \cdot \frac{Presicion \cdot Recall}{Presicion + Recall}$

Performance is also of concern with respect to variant detection with suspicious files being submitted at near 1/2 million per day to the antivirus providers. However, performance is also divided into two measurements that can impact end points. First is the intake processing speed. This performance measure is the time it takes for an intake system to process the malware or suspicious software sample so that it can be compared to other samples. For example, for traditional hashing, where the sample is compared to other samples by taking a cryptographic hash of the sample and comparing it to other hashes, this processing time is the time it takes to generate the hash itself (without comparison). The generation of these signatures is often not dependent on previous signatures and is, therefore, linear in nature. Thus scaling projections of signature generation is straight forward in these cases.

The next performance measure is the time it takes to conduct the comparison itself. While signature generation time is linear in the test cases provided in this paper, the comparison time in similarity analysis is not. Some of the methods included in this paper compare via sorted lists (LSH or other techniques) which is $O\ n\ log\ n$ while others conduct a pair-wise comparison which is On^2. Each of these present unique problems in transition. The On^2 pair-wise comparison is CPU bound, requiring ever increasing computations to meet demand. The $O\ n\ log\ n$ sorted indices comparison is memory bound, requiring ever increasing memory to account for the indexed tables used to calculate similarity. As we describe each of the tested projects for these papers, we will briefly discuss the implications of scalability for each of them.

B. Dataset

Dataset generation by algorithm is problematic as explained above. As such, we have obtained a dataset that was sampled at random via inputs in various incidents and analysis by professional malware analysts, whom grouped the dataset into eight distinct groups. Some of the malware samples within the dataset were originally packed. We include the actual MD5s of the samples within this paper as the dataset is relatively small and this ensures that the dataset remains static and available in the future for the comparative accuracy tests.

The Malware set used consists of 85 samples. Within the set the malware has been put into 8 separate groups by function similarity. Analysis information was derived manually by malware professionals. Additional meta data has been gathered for each malware sample from Virus Total, Mcafee and or Dell SecureWorks Counter Threat Unit.

Group 1: Consists of 12 malware samples. 10 of the 12 are identified as Ziyang RAT and were originally packed with Armadillo v1.71
25721aa47fb29fcba9de1f3406d9f8d6
31da84e9dd9b865a7d0e4c3baa7b05a2
35f65bd2c9ff5c46186f84f19a3a7d18
3ce19fc2a1a6a42b8450d477a9919de2
47cc260cf70fc81995f651dc1c5b172a
718c6e47512bec8c585320d087041ace
7b30b4d95ed988081ec9fe3908df409e
8d64f279400d8e1f8bf2170d148203a7
90a219684b3b815d6b6c1addd5e28c5b
b6e1a2048ea6bd6a941a72300b2d41ce
cbef7e4cd8b51b0b38fdbeab6486f89a
ea66e664bdf530124ff7993a4ad510d4

Group 2: Consists of 19 malware samples. The 19 samples are identified LinseningSvr and were originally packed with Armadillo or Aspack
0f171ff1a80822934439edaa7be1023b
10d7989355b5fc2915a18004df4f9074
156085a7cd31d272486193df10d7e26e
1a56c6eb1cd54ce642bdfd59168da127
35185b8c5e3cb928c97919aa5ad01315
356c9314ae95a18f3fef630e04f4d8b6
3f7601f0aeb5e391638a597c15f80c9f
4734d158048c398f2ae44c035487e249
47803deb563d9ff917369b8c97c22a7e
49361de55268ff2ee67add42d359248d

5a5d2c6fe70521efd875fecc961ff75a
5fa46b686c3a5e27fd4dfe0e1fbb1145
89e9bed692611692e244ed294c9904cc
9951f026f491ef90037a59f305269273
a90194c071aefeb21331385ad7115fbc
a9a53cd80a12519429a9a40f9d34e563
b14ad1298928bb33613eb8e549c93e9e
d414c721c60df0282481df77c0c1cdae
e4cdfa15a38034e6ae7f80334e7d6a14

Group 3: Consists of 20 malware samples. The 20 samples are identified as BeepService and were originally packed with Armadillo v1.71.
0625b5b010a1acb92f02338b8e61bb34
15cb44831bdd295bb3c0decf7cea0dc0
1f3c731aed7d8085eb2d15132819cb8b
2393b93a762d4990ec88d25c9e809510
3a282da31bf93cfaaa8b5a11d441483b
3aa3846284b6e7112da90e1d5e4e7711
3c6ff8b69513bf338a2d5b3440b9a8cd
463a12f92652fc82b3c6e53bb917ecf2
4e95cb057f351af0f7c972800a07f350
52b8063f663563d549ec414a7caf38f9
54dc517c9f62dc5d435fb8bac0fd59f9
59534c90c3234fbdc82492d1c1b38e59
660b856f485fb8fa0ecb3533d88d405e
6b8ea95a729551fde76a28244cb95ac1
726d77fe00b4c00df1bb2c5afd05ad21
73b8facac3e946354a89e58d308d8ebd
99f67381b3b389f0e6120603019e0ef9
a0f71497ca4c4c62c094c1843693381e
d5caf69c7a2ac416131133e0b1623066
e8ee22223b6475d7b3ef8f51383df1ef

Group 4: Consists of 13 malware samples. The 13 samples are identified as SimpleFileMover and were originally packed with Armadillo v1.71.
00d0382fe1b02b529701a48a1ee4a543
139ddf5aeff4602bf8168fe13a63da30
36093314059a9e7b95025437d523d259
59ee8762316018862d7405b595267d8d
5d7c34b6854d48d3da4f96b71550a221
5ff93637082c96de1650facdce95a970
721c56a617dfd2cecade790d9e9fa9ce
731089e10e20b13095df2624b6eb399f
8f73b7653ebf20f66a961cc39249b2e3
9cf67106cd1644125b773133f83b3d64
9f546188e0955737deffc5cec8696d9a
dc1a284e82f4f38a628b84b0e43e65d5
e9d0a062b9b72d4c46ad9a70c80ade13

Group5: Consists of 5 malware samples. The 5 samples are identified as DD Keylogger.
0e058126f26b54b3a4a950313ec5dbce
12b0e0525c4dc2510a26d4f1f2863c75
78f2acc3309e1e743f98109a16c2b481
96c28bddba400ddc9a4b12d6cc806aa3
b13ab523e89d9bb055aee4d4566ab34f

Group 6: Consists of 10 malware samples. The 10 samples are identified as a PUP by McAfee and were originally packed with Armadillo v1.71 or InstallShield 2000.
02106ca9b92b8c268951b08fe3f80341
3685dd53195dae257528731ddd21d5c6
649d5fd5cbe4f5e1e16be346f8aee937
70306b773629f7fa71b301b5ee5bf087
8cdd5484985c62e38703314ffd472642
99fded4bc652f2bd0e2085c2ea085aca
da33550dbcd821ead57f64349b04a7d4
dcc267ca10304fd38bc926dc778108ca
edb7d0e894e929a5892bfc0b52c4752d
f46b1a119a53c16736a6b593df5f7eaf

Group 7: Consists of 3 malware samples. The 3 samples are identified as backdoors by McAfee and were originally packed with Armadillo v1.xx - v2.xx.
27e4610a97265110c7b79b2ba93f77f8
88653dde22f723934ea9806e76a1f546
d08c54ed480c9cd8b35eab2f278e7a28

Group 8: Consists of 3 malware samples. The 3 Samples are identified as SvcInstaller and were originally packed with Armadillo v1.71.
4a12f4646fe052392641533944d240d1
bc55ba7467d5d62ac0b5c42a2c682fd6
f23ee51aa4a652266c2c1666bc15e15b

For performance measurements with regard to signature generation and comparison times, the 85 samples included above do not represent a large enough dataset to project real world results. As one might expect, including 10K or 1M MD5s in a paper would not be useful. However, access to a randomized dataset of malware in large numbers from large repositories is likely to produce similar results for speed tests. Thus we acknowledge this is not an ideal solution; however, the solution to a randomized selection of malware for speed tests is practical.

The tests are targeted toward understanding malware variant detection in terms of binary classification as well as performance characteristics in term of signature generation speed and comparison for all pairs clustering. To understand these measurements, we plot a ROC curve of precision and recall, varying the threshold for each comparative result from a similarity score of .01 to 1 using our manually analyzed dataset detailed above. For performance measurements, we plot the signature generation time of our 85 sample dataset, plus 10K randomly selected samples in 1K increments. We conduct and visualize the comparison times for each candidate solution with the same inputs.

C. Candidate Solutions

1) CTPH: Context Triggered Piecewise Hashing [9] was proposed in 2006 and implemented in the open source project ssdeep. The project is a general file similarity tool in that the entire content of the file is used to determine similarity. The tool is not completely input agnostic as hash windows are started and stopped on triggers based on hashes of file content. A fixed size window of 7 bytes slides through the content and the least significant 6 bits of a FNV hash are concatenated to the signature. Trigger points are dispersed across the file by modulating over a value that is determined by file size. Thus the signature is file size dependent and only files of similar size can be compared. A second signature with a modulating value of half the block size is also stored to compensate for this weakness.

As CTPH signatures are concatenated, the straight forward comparison of these signatures is On^2, though their very small size reduces this issue in small datasets (O is very small). However, in large datasets, this would become an issue in practice. Work has been conducted to address the performance issues of CTPH [13][14]; however, as ssdeep is widely used in industry, the ssdeep implementation of CTPH is used in this paper.

2) TLSH: The TLSH is a locality sensitive hashing scheme developed at Trend Micro[6] and, like CTPH and sdhash, it operates on the entire content of a file. The approach uses a sliding window hash that is temporarily stored into a byte array. Quartile points are then calculated from this array to allow for the selection of an abstracted view of the array, which is both easy to compare to other signatures and a small fixed length signature.

The comparison of TLSH signatures calculates a distance measure such that a measurement of 0 is exactly (or nearly exact) the same as the comparative file. The distance measure can exceed $1K$ with radically different files. This type of measurement was identified as superior in [6]; however, it presents an issue when comparing test results with other tests. In all the other tools tested for this paper, a weighted similarity measure from $0\ to\ 1$ is calculated to evaluate sample likeness. To deal with this disparity, we turn to the paper that describes TLSH. In the paper, the authors point out the value 300 as a point of diminishing returns, noting that the false positive rate (1-$Precision$) and Detect rate ($Recall$) flatten at this point. To compare this tool with others, the measurement of distance was converted to a value of $0\ to\ 1$ by subtracting the distance from 300 then dividing by 3. Any value below 0 was bounded to 0. For completeness, we plot the recall vs precision ROC curve of both the bounded TLSH and the natural TLSH output.

3) sdhash: The similarity digest sdhash was introduced by Roussev in 2010 in work [15]. The approach attempts to limit false positives by picking out features that are likely to be unique. The uniqueness of the features relies on entropy calculations and a large empirical study of file features. In addition to the feature selection, sdhash stores these features in Bloom filters, which address both comparison of arbitrary input sizes and efficient comparison. The sdhash implementation is a general file similarity tool that designed for forensic file analysis that takes a file as input and the signature output represents the file in its entirety. In works [7][8], the authors point out several implementation flaws and a design flaw that would allow some manipulation of the similarity score.

4) BitShred: BitShred was introduced in 2009 by Jang and Brumley at CMU-Cylab [16]. It was further refined in works [17][1] and is a malware/software specific variant detection scheme. The approach was further developed to incorporate behaviors into signatures for comparative analysis and variant detection [1]. While this approach is reported to be promising, only the static analysis features of BitShred have been reimplemented for the purposes of this test.

The static analysis portion of BitShred incorporates a sliding window hashing scheme across the executable section of the malware. In the latest publishing, window sizes of both 4 and 12 are used to generate the hashes, which are stored in fixed-sized, bit-indexed arrays. The arrays are similar to Bloom filters in that the hash is effectively truncated and stored as a bit indexed by the truncated hash. The two arrays can then be compared linearly through a sliding OR/AND operation to produce a Jiccard Similarity estimate such that $J(F_a, F_b) = \frac{(F_a \cap F_b)}{(F_a \cup F_b)}$ is approximated by $J(F_a, F_b) \approx \frac{S(B_a \wedge B_b)}{S(B_a \vee B_b)}$; however, while the comparison of two arrays is a linear calculation, comparison of multiple arrays is n^2 exponential.

We have reimplemented the static portion of the BitShred for comparative analysis for this paper. As our implementation was built to reproduce the similarity comparison functionality of the static analysis portion of BitShred only, we make no claims that our implementation produces similar speeds; therefore, results for performance with respect to execution times have been extrapolated from the latest work [1].

In the latest work, BitShred has been implemented on a map/reduce framework with multiple map nodes. They report that signature generation times are linear, as expected, and can be produced at a rate of 4m 40s for 655,360 samples and 2m 25s for 327,680 samples. The generation times are on a 320 map node distributed system, which is approximately 2300 samples/per second on the distributed system or 7.1 samples per second per node. While our in house implementation is slightly faster in signature generation times (7.7 samples/sec on a single node), our comparison times are significantly slower. The work reports a comparison time of approximately 21,850,000 comparisons/sec on a 320 node system. This works out to approximately 68.3K comparisons/sec per node; therefore our reporting will project this comparison time.

5) First Byte: The First Byte malware similarity method was introduced in 2013 by Upchurch and Zhou in work [2]. It was realized after researchers in works [18][3] proved that simple disassembly can dramatically improve location of variant segments of code. The First Byte method was based on a full disassembly of malware, linking similar basic blocks to other basic blocks to form a graph from which relationships could be visualized. The basic blocks where normalized by extracting only the first byte of each instruction and using those as inputs to the sliding hash window. The project used sliding window hashes of size 5 or 10 to create a bit indexed array such as used in BitShred

Comparisons are completed in the same manner as in BitShred, that it a n^2 based many to many comparison. This limitation lead to the development of a Locality Sensitive Hashing (LSH) scheme to represent the feature set that was fixed length and ordered. Thus one to one comparisons remain ordered as with the original bit array, but look ups in a many to many comparison are $n\ log\ n$ rather than n^2. To conduct the variant test for First Byte, which was developed to detect code blocks rather than entire variants, the blocks signatures for the malware samples where combined into one signature to represent an entire malware sample. (A later paper will describe the approach in detail)

Additionally, First Byte originally used IDA Pro [19] to extract basic blocks from the malware samples. While IDA Pro is widely used in industry, the extraction of basic blocks in this manner proved too slow for en masse comparisons. While IDA Pro's recursive disassembly allows for more accuracy in identifying variants, a linear sweep disassembly proved more than an order of magnitude faster. A brief comparison of a

recursive vs linear sweep disassembler results within First Byte are included in the results; however, linear sweep is used throughout the rest of the paper (abbreviated FB-L in graphs and tables).

IV. Experimental Results

All experiments were performed on an Linux64 machine (Intel 3.5 GHz 4770K / 32GB memory) unless otherwise noted.

A. Choosing Solution Options

1) BitShred Window Size: The work [1] describes BitShred with window sizes of 4 and 12, which represent a hash for every 4 and 12 consecutive bytes, respectively, within the executable section of a software package. To simplify comparative results with other tools, we first choose a single window size within BitShred in which to conduct the rest of the comparative tests.

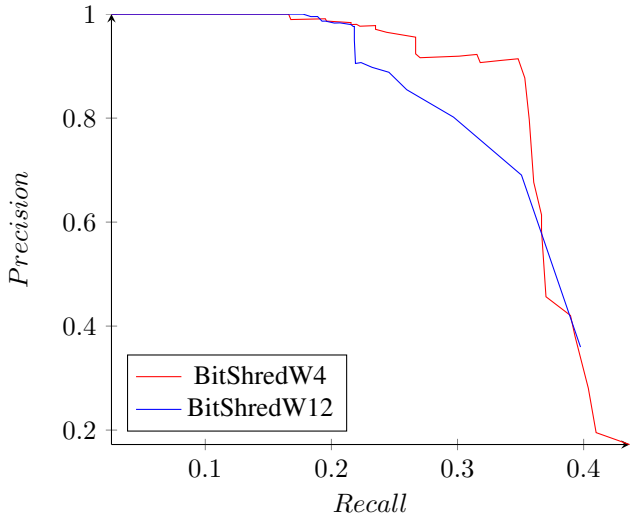

Fig. 1. Precision/Recall Curve for BitShred Windows

In figure 1, the two window sizes for BitShred are compared to each other. We do not address the signature size to avoid collisions, we instead use the maximum signature size, 32K, as reported in the paper to eliminate any noise in comparison. Signature size is an important component of speed of comparison; however, our performance comparisons are extracted from the paper and are, therefore, not affected by this choice.

As the graph in figure 1 reveals that a window size of 4 generates a superior ROC curve (best is to the upper right), this window size was chosen for subsequent tests.

2) TLSH Bounding: The Fig 2 shows the effect of bounding TLSH to a value between 0 and 1. As expected, the ROC curves are unaffected by the bounding except that recall increases to near $recall = 0.5$ at $Precision \approx 0.23$ where the bounding removed those results. While this concerns only those where $Precision$ requirements are very low, it should be noted for completeness.

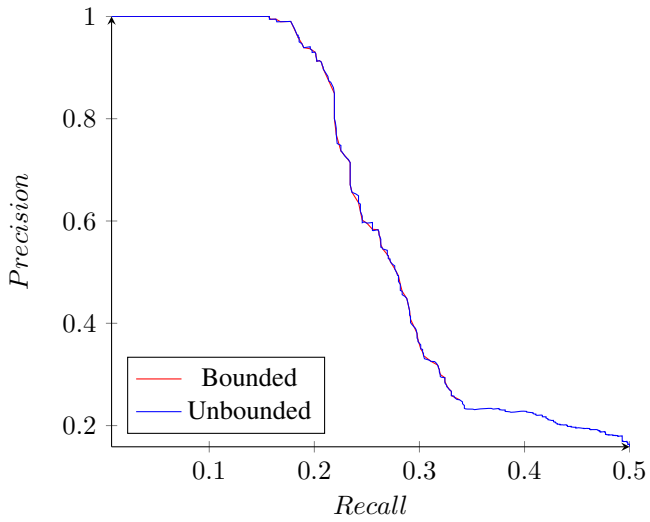

Fig. 2. Precision/Recall Curve for TLSH

3) First Byte Implementations: Two variations are available for comparisons using the First Byte technique. Recall in the original work [2], the extraction of basic blocks allows for the identification of libraries through a bit-array based filter. As this work uses the same process, it is possible to remove library features from signatures of malware samples. Therefore, we will examine the effect of this option with respect to $FMeasure$ and a $Recall$ - $Precision$ ROC curve. In addition, as the linear sweep disassembler is much faster at generating signatures (4.1sigs/sec vs 0.13sigs/sec), it is worth examining the cost in accuracy.

As can be seen in Fig 3, the $FMeasure$ peaks steeply for linear with libraries, linear without libraries, and recursive with libraries. The removal of libraries from the recursive is peaked at $threshold = 0.01$. The ROC curve for the four techniques in Fig 4 shows that the precision for recursive never falls below $Precision = 0.925$. The peak $FMeasure = 0.755$ for all of the techniques is "recursive with libraries", though both linear sweep methods peek at $FMeasure \approx 0.71$. The $FMeasure$ performance with both linear sweep options are nearly identical throughout the curve of both graphs. While the FMeasure in the linear sweep disassembly is much more sensitive to changes in threshold, we will use the First Byte implementation with linear sweep disassembly and exclude the libraries in further tests.

B. Binary Classification Comparison

The goal of a variant detection scheme for malware, as with any binary classification system, is to correctly identify malware variants in a field of unknowns. To predict the field performance of these solutions, we test and measure if an implementation can correctly identify malware variants, how many instances of the identification are incorrect, and how many in the field have been missed. As explained above, we will measure these attributes with a manually sorted grouping of malware variants that was not algorithmically generated. This will measure the performance of a potential solution to that of a human analyst rather than another algorithm.

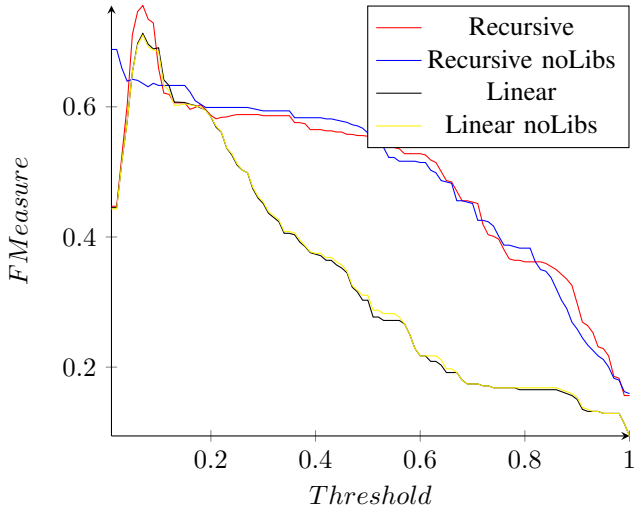

Fig. 3. FMeasure vs Threshold for First Byte Types

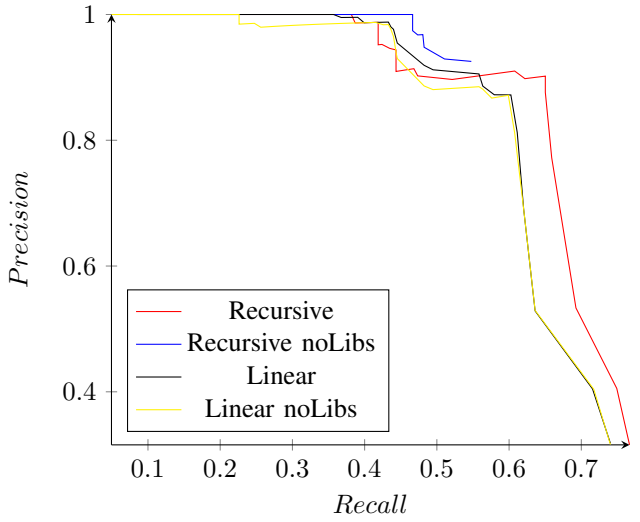

Fig. 4. Precision/Recall Curve for First Byte Types

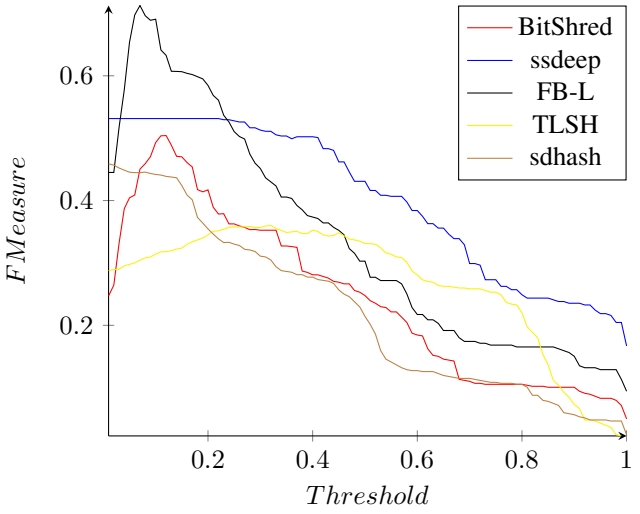

Fig. 5. FMeasure Comparison

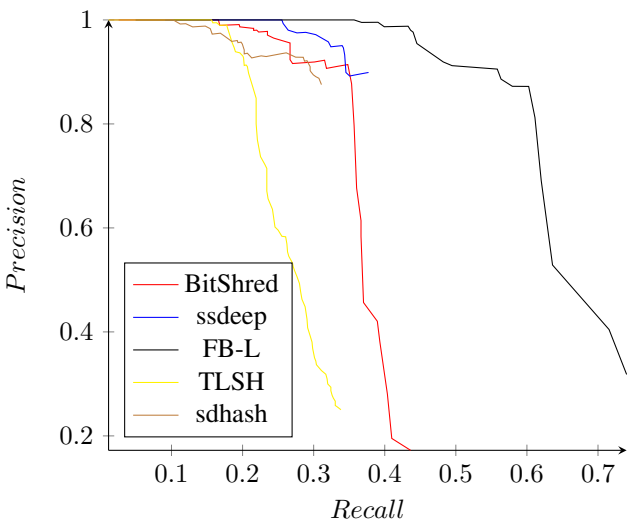

Fig. 6. Precision/Recall Curve Comparison

For generalized binary classification performance, the peak $FMeasure$ of each of the techniques is most useful. The graph in Fig 5 displays these values for the tested solutions. BitShred maximizes $FMeasure$ at $FMeasure = 0.504$; ssdeep at $FMeasure = 0.531$; First Byte at $FMeasure = 0.710$; TLSH at $FMeasure = 0.360$; and sdhash at $FMeasure = 0.446$. While BitShred, First Byte, and TLSH all peak within their respective $FMeasure$ plot, both ssdeep and sdhash are maximized at the minimum possible threshold.

For a more tuned classification performance, we have plotted the ROC curve to contrast the tradeoffs in precision and recall. The graph in Fig 6, plots the points of recall and precision while varying threshold levels from 0.01 to 1 (the trivial case of $threshold = 0$ was omitted for graph focus). For cases where $Precision = 1$, the tools display a large range of recall values. Table I shows the recall levels for high degrees of precision. First Byte has the highest recall measurement in the test with $Precision = 1$ where $Recall = 0.357$. The only other tool to push out of the teens at that precision level is ssdeep. At $Precision = 90$, First Byte again has the best performance with BitShred performing second best. In table II, $Precision$ is measured at a required $Recall$ level. The only project to reach a recall level beyond $Recall = 0.50$ in the test was First Byte with a maximum $Recall = 0.74$ at

TABLE I. RECALL AT PRECISION

$Precision$	1	0.9
BitShred	0.166	0.348
ssdeep	0.255	0.343
First Byte	0.357	0.558
TLSH	0.156	0.207
sdhash	0.102	0.296

TABLE II. PRECISION AT RECALL

$Recall$	0.5	0.3
BitShred	na	0.919
ssdeep	na	0.971
First Byte	0.74	1
TLSH	na	0.35
sdhash	na	0.89

TABLE III. SIGNATURE GENERATION TIMES

	Per Sec	Gen 10K Signatures
BitShred	7.1	1408
ssdeep	117	85.1
First Byte	4.1	2384
TLSH	128.7	77.7
sdhash	348.4	27.8

TABLE IV. ALL-PAIRS COMPARISON PROJECTIONS

	Type	Max All-Pairs/Node/Day	apparent 0
BitShred	n^2	108K	$1.53 * 10^{-5}$
ssdeep	n^2	339K	$1.5 * 10^{-6}$
First Byte	$nlogn$	345M	$2.5 * 10^{-4}$
TLSH	$nlogn$	147M	$5.84 * 10^{-4}$
sdhash	n^2	175K	$5.6 * 10^{-6}$

$Precision = 0.318$. All tools were able to retrieve at the 30% level, with First Byte having no false positives at this level.

C. Performance Comparisons

1) Signature Generation: Performance of the test tools varies widely. The signature generation times of First Byte, having to disassemble each malware sample, are the slowest of the group, with an average of 4.1 signatures/sec. The fastest signature generation was sdhash, which measured 348.4 signatures/sec. For in-processing malware, First Byte is capable of generating $\approx 350 K signatures/day/node$ in its current research state. The fastest, sdhash, is capable of more than $30 M signatures/day/node$. In Fig 7 , the plot shows the dramatic difference in signature generation times over our Gold Standard dataset and the 10K additional samples.

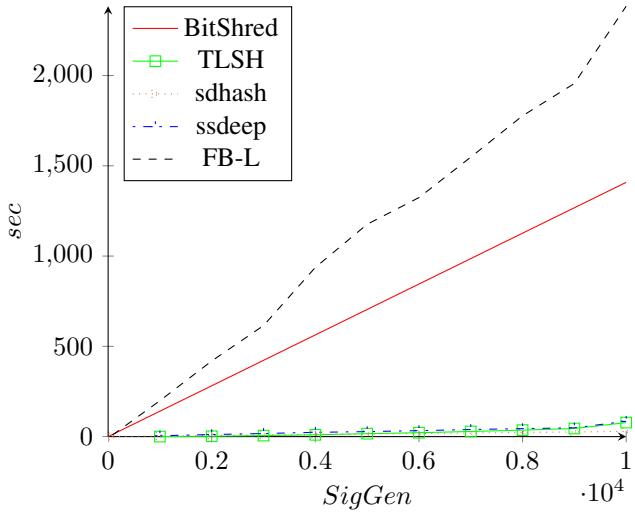

Fig. 7. Signature generation performance per node

2) All-Pairs Comparison: Signature generation times are linear in nature for all of the tools tests. In an all-pairs similarity clustering, each submission must be evaluated against all others, which leads to exponential comparison times as sample sets grow. A one to many comparison is linear with respect to field size with this method. There are known methods to reduce this comparison time, such as LSH, which is known to be $nlogn$ and is used in two tools in this test, TLSH and First Byte.

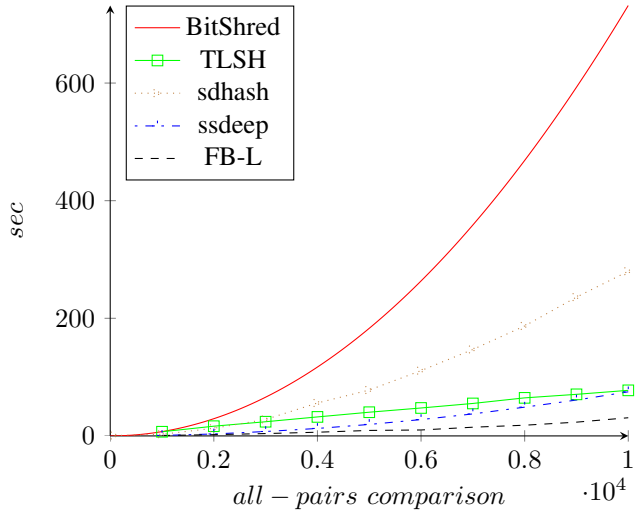

Fig. 8. Time to conduct all-pairs comparison

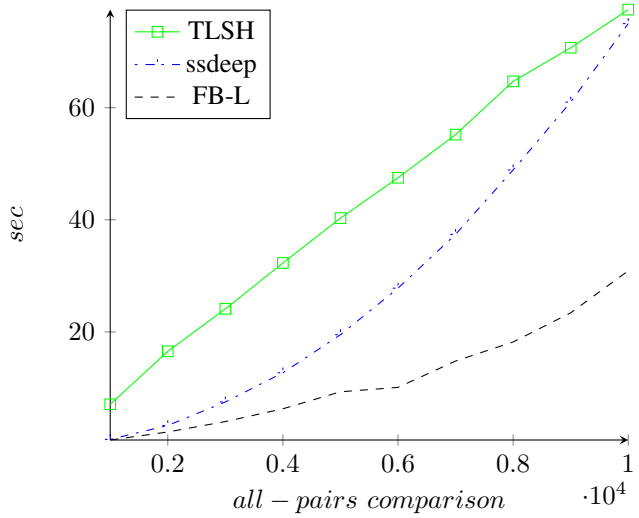

Fig. 9. Comparison Preformance

In all-pairs comparison, the effects of algorithm choice become apparent. As seen in Fig 8, two of the three n^2 comparisons, BitShred and sdhash, consume hundreds of more node seconds than the rest. The CTPH tool ssdeep, though n^2 can perform inline with other tools with datasets in the 10K range. However, to project computation times of much larger datasets, the operation time was computed at 10K samples for each of the algorithms, n^2 for ssdeep, BitShred, and sdhash; and $nlogn$ for TLSH and First Byte. As seen in Table IV, ssdeep performs in league with TLSH and First Byte due to its very small operation time. However, as seen in Fig 9, the ssdeep algorithm is clearly exponential and thus its maximum samples it can process in a day is 339K, where the $nlogn$ algorithms can process 345M and 147M for First Byte and TLSH respectively. As LSH is memory bound for lookup tables, these numbers may not be achievable. For instance, First Byte is limited to $\approx 2 M signatures/GB$ of available RAM.

TABLE V. PUBLISHED MEASUREMENTS VS TESTED MEASUREMENTS

	Published		Tested (Peak $FMeasure$)	
	Recall	Precision	Recall	Precision
BitShred[1]	.928	.932	.348	.914
TLSH[6]	.945	.935	.260	.583
sdhash[6]	.371	.995	.295	.910
ssdeep[6]	.312	.999	.337	.898
First Byte	na	na	.602	.872

V. CONCLUSION

This paper has presented Variant: a malware similarity testing framework. The framework is a series of tests on a known, static dataset, which is the best possible under reasonable conditions to date. The paper outlines classical, binary, classification nomenclature to standardize the results across the tested approaches to malware variant detection in this paper and in future works. We demonstrate the value of this test by testing 5 malware variant detection approaches used in research and industry.

A. Inconsistencies

Variant locates several inconsistencies in reporting of recall and precision of other projects. In the various published works on tools, datasets have been generated by algorithms geared toward selection of sets by signature detection, limited manipulation of source code, or limited manipulation through byte code changes. These derived datasets produced recall and precision measurements for the various tools in our tests that were much higher than our manually analyzed set that was selected from wild malware.

While the origin of these inconsistencies are unknown, we submit the first place to look is in the test sets used in other projects. As these sets have been derived by algorithm, the malware does not likely represent the breath of variation in wild malware samples. Variant allows for the direct comparison of works using static analysis techniques to detect malware variants.

In Table V, we show that our manual selection of wild malware produces results inconsistent with other selection processes used in previous works. As our selection process represents a small portion of wild malware and is not derived or identified by another algorithm, we submit these results are representative of real world tests and deployments.

B. Reproducibility

Our dataset is static. While we encourage others to expand upon the dataset by manually selecting malware that has been grouped by professionals, the original dataset can be reproduced simply by obtaining the malware identified by MD5 in this paper. Since our dataset can be reproduced exactly, future testing of malware comparative tools, with regards to binary classification measurements, can be compared directly to the included results outlined in this paper and with other tools that publish based on the described dataset. This is an essential component of peer review and validation that has been absent in this field.

C. Measurement Alignment

Malware similarity works are relatively new to the greater binary classification field. The development of malware similarity works and their subsequent efficacy measurements have been reported in terms of false positive rates, detection rates, or other accuracy measurements that are related to the standard binary classification terms (such as $FalsePositiveRate$ = $1-Precision$), but are not aligned with reporting standards in the greater binary classification works. We submit that reporting of measurements be aligned with other binary classification works as report as such in this paper.

D. Comparitive Results

As outlined in [12], performance metrics are divided into three categories for these types of binary classification works. First is the preprocessing stage, where signatures are generated and times are measured. Second is the comparative stage, where a linear or all-pairs comparison is conduced and time is measured. Lastly, is the measurement in terms of binary classification. We identify these areas in our testing approach and produce results for 5 implementations from previous malware classification works to demonstrate the need for such a test. In conducting our tests, we uncover some insights with regards to our test candidates.

In terms of speed, the approaches used in TLSH with respect to both signature generation and comparative speed are clearly preferred. While signature generation is 2.5x that of sdhash, this is more than compensated for through the use of an $nlogn$ all-pairs comparison through the use of an LSH scheme. However, in terms of classification of malware, TLSH could not produce an $FMeasure > 0.4$.

In terms of binary classification, First Byte is a superior approach with our test set. First Byte produces the highest $FMeasure = 0.71$, highest $Recall = 0.740$, and $Precision = 1$ at $Recall = 0.357$. Performance with respect to comparative times is also a sound approach for the project as the comparative measurements are the fastest within the 5 projects tested. However, classification performance comes at the cost of time needed to generate signatures. First Byte is 85x slower at generating signatures than the fastest time, which was produced by sdhash; however, comparison times will quickly overcome the signature generation times in large datasets when using sdhash. First Byte is 30x slower at generating signatures than TLSH, which also has a desirable $nlogn$ comparison algorithm.

Overall results are a matter of the specific needs of the transition project. If 365K inputs per node, per day are enough to satisfy project requirements, First Byte in its current state is the best solution tested. However, if memory bounding is an issue with expected dataset sizes, the LSH solutions offered in TLSH and First Byte may not meet requirements. Here, ssdeep and BitShred offer a CPU bounded solutions that could produce $n\ to\ m$ results in reasonable times on distributed systems so long as n was bound to a small fraction of m.

E. Future Work

Future work should seek to expand the manually analyzed and selected dataset to thousands of samples; however, the

man power and expertise intensive task of selecting malware through a full capabilities analysis makes this goal daunting. We would encourage others with solutions in development to use this work for comparative analysis in their work and those with improved versions of the included tools to test using our dataset or a larger published dataset derived under the conditions outlined in this paper.

REFERENCES

[1] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 309–320, ACM, 2011.

[2] J. Upchurch and X. Zhou, "First byte: Force-based clustering of filtered block n-grams to detect code reuse in malicious software," in *Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on*, pp. 68–76, IEEE, 2013.

[3] F. Adkins, L. Jones, M. Carlisle, and J. Upchurch, "Heuristic malware detection via basic block comparison," in *Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on*, pp. 11–18, IEEE, 2013.

[4] D. Raygoza, "Automated malware similarity analysis."

[5] X. Hu, T.-c. Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs," in *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, (New York, NY, USA), pp. 611–620, ACM, 2009.

[6] J. Oliver, C. Cheng, and Y. Chen, "Tlsh–a locality sensitive hash," in *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*, pp. 7–13, IEEE, 2013.

[7] F. Breitinger, H. Baier, and J. Beckingham, "Security and implementation analysis of the similarity digest sdhash," in *First International Baltic Conference on Network Security & Forensics (NeSeFo)*, 2012.

[8] F. Breitinger and H. Baier, "Properties of a similarity preserving hash function and their realization in sdhash.," in *ISSA*, pp. 1–8, 2012.

[9] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital Investigation*, vol. 3, Supplement, no. 0, pp. 91 – 97, 2006. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[10] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Network and Distributed System Security Symposium (NDSS)*, Citeseer, 2009.

[11] "Virus total," 2015.

[12] F. Breitinger, G. Stivaktakis, and H. Baier, "Frash: A framework to test algorithms of similarity hashing," *Digital Investigation*, vol. 10, pp. S50–S58, 2013.

[13] L. Chen and G. Wang, "An efficient piecewise hashing method for computer forensics," in *Knowledge Discovery and Data Mining, 2008. WKDD 2008. First International Workshop on*, pp. 635–638, Jan 2008.

[14] F. Breitinger and H. Baier, "Performance issues about context-triggered piecewise hashing," in *Digital forensics and cyber crime*, pp. 141–155, Springer, 2012.

[15] V. Roussev, "Data fingerprinting with similarity digests," in *Advances in digital forensics vi*, pp. 207–226, Springer, 2010.

[16] J. Jang and D. Brumley, "Bitshred: Fast, scalable code reuse detection in binary code (cmu-cylab-10-006)," *CyLab*, p. 28, 2009.

[17] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: Fast, scalable malware triage," *Cylab, Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-Cylab-10-022*, 2010.

[18] A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su, "Detecting code clones in binary executables," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, pp. 117–128, ACM, 2009.

[19] Hex-Rays, "Ida pro disassembler."