This Malware Looks Familiar: Laymen Identify Malware Run-time Similarity with Chernoff faces and Stick Figures

Nathan VanHoudnos
Software Engineering Institute
Carnegie Mellon University
nmvanhoudnos@cert.org

William Casey
Software Engineering Institute
Carnegie Mellon University
wcasey@cert.org

David French
Software Engineering Institute
Carnegie Mellon University
french@cert.org

Brian Lindauer
Software Engineering Institute
Carnegie Mellon University
lindauer@cert.org

Eliezer Kanal
Software Engineering Institute
Carnegie Mellon University
ekanal@sei.cmu.edu

Evan Wright*
Anomali Inc.
evan@Anomali.com

Bronwyn Woods†
Turnitin
bwoods@turnitin.com

Seungwhan Moon
Language Technologies Institute
Carnegie Mellon University
seungwhm@cmu.edu

Peter Jansen
Language Technologies Institute
Carnegie Mellon University
pjj@cmu.edu

Jaime Carbonell
Language Technologies Institute
Carnegie Mellon University
jgc@cs.cmu.edu

ABSTRACT

Classifying unknown malicious binaries into malware families provides valuable information to security professionals. The task of recognizing a malicious binary (i.e., attributing it to a previously observed attack pattern) is widely considered a difficult task requiring extensive domain expertise. In this work, we offer a new approach which focuses on transforming the the recognition problem domain from system traces to Chernoff faces, thereby engaging the facial recognition aptitude of laymen.

To do so we (i) curated a expert tagged dataset of malware variants, (ii) instrumented behavior trace monitors for each variant, (iii) constructed a simple, graph based feature set from the run-time behavior, and (iv) visualized low-dimensional representations of these system call graphs with stick figures and Chernoff faces. We then selected the three families with the largest variation and asked non-experts on Amazon Mechanical Turk to classify binaries between these three families using the generated visual representations, a task that would otherwise be delegated to experts. We found that non-experts completed the task with between 63% and 86% accuracy, and when aggregated, these non-expert labels successfully trained a classifier to a similar level of performance as the ground truth labels. Although simple, the experiments conducted provide a novel evaluation of the inherent difficulty of malware recognition tasks. Additionally new operational possibilities for effective human in the loop malware recognition are indicated and discussed as future work within the research prospectus.

ACM Reference format:
Nathan VanHoudnos, William Casey, David French, Brian Lindauer, Eliezer Kanal, Evan Wright, Bronwyn Woods, Seungwhan Moon, Peter Jansen, and Jaime Carbonell. 2017. This Malware Looks Familiar: Laymen Identify Malware Run-time Similarity with Chernoff faces and Stick Figures. In *Proceedings of EAI International Conference on Bio-inspired Information and Communications Technologies, Hoboken, New Jersey, USA, March 2017 (BICT'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The risks and impacts of malware continue to increase, so also the amount and rate of its production [18]. New operating systems and software, such as web browsers, are targeted within hours and days of release with exploit frameworks that probe for vulnerabilities and generate attacks and malware. Still organizations must protect themselves and their digital assets, forming an emerging cyber security role where time critical operational decisions are needed. Often a best response will incorporate the evidence available when actions are needed, therefore knowledge and its timeliness are important.

In operational security settings, once a given software artifact (of unknown provenance) is determined malicious the next critical task is to determine how it relates to a reference group of previously

[*] Main contributions were while employed at CMU Software Engineering Institute.
[†] Main contributions were while employed at CMU Software Engineering Institute.

BICT'17, Hoboken, New Jersey, USA
© 2017 ACM. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

known malicious software artifacts (i.e., its family). By linking an artifact to prior experience the defender may respond strategically: to mitigate malware, employ firewall protections, utilize software-dissection tools, remove persistent attackers, delay the adversary, or adjust security policies to thwart the adversarial trends. However, the tasks associated with classifying a malicious artifact are widely considered technically difficult requiring extensive domain expertise and time intensive manual processing. As such, the problem is typically considered beyond the layperson's capacity and the primary approach relies on reverse engineering - a resource limited and time costly proposition. The current classification approach thus faces a problem of scale as the labor and financial cost of expert reverse-engineers is high while talent cannot meet current demand [26]. To address this problem within existing operational security infrastructures we consider leveraging the strengths of non-experts.

Our approach considers human in the loop decision systems for malware recognition and how the decision artistry of both experts and non-experts can be utilized within active learning systems. Because the work is tightly coupled to contemporary malware classification processes (affect the resulting recognition problem) we proportion a discussion to as much of the data preparation issues as possible.

We include a discussion of significant steps in the transformation, which we summarize as (i) creation of a dataset of malware variants labeled by experts, (ii) derived behavior sequences with a trace monitor, (iii) constructing feature vector from trace data, and (iv) reduction to low-dimensional representations of data features for visualization with stick figures and Chernoff faces, and finally (v) face and figures recognition experiments presented to non-experts on Amazon Mechanical Turk.

2 RELATED WORK

We overview relevant work in three main areas underlying the problem and experiment: malware classification, human in the loop machine learning, and visualization.

2.1 Malware Classification

Modern system architectures and applications are constructed as volumes of programs, data files, and resources collectively termed *artifacts.* Artifacts are frequently updated and patched making trusted verification and provenance management even more difficult. In this context attackers find ample 'attack surface' and means to go un-recognized. We consider the basic recognition problem: given a single artifact we wish to determine previously- or well-known malware samples or groupings that are related. Approaches to this problem are grouped into three stages of recognition: Static detection, Dynamic detection, and Behavioral detection.

Static detection scrutinizes the artifact as it appears in storage (on disk), as such it includes applying match heuristics such as filenames, hash function values, or membership for specific strings/patterns such as Antivirus or YARA signatures [33]. Malware authors, aware of these techniques and wanting not to be recognized, have coopted code re-writing and obfuscation techniques that make files appear vastly different but with otherwise identical logic. This adversarial tactic usually defeats match heuristics such as file hashing or most other static detection facilities. Among the plethora of malware obfuscation attacks[34], the tactic of *polymorphic malware* is used to slightly modify in each compiled payload for the purpose of obscuring hash function values.

To overcome this, defenders have considered a second type of recognition focused on what an artifact does rather than what it contains. *Dynamic detection* scrutinizes the outcome when the artifact is stimulated by loading or executing within the appropriate environment. A common example will scrutinize the dropped files or system properties altered consequently by the artifact stimulation. Often this is done by applying static detection to artifacts consequential to stimulating the instance under consideration. In [25], the value of dynamic analysis for recognition was demonstrated on polymorphic malware.

As an extension of dynamic detection, is *behavioral detection* which considers not only the discrete outcomes from stimulating artifacts, but also the fine grain sequence of system events. The idea is to capture not only the consequences of execution, but also the implementation pattern that achieves the design goal. We employ a *trace monitor* as a means to observe the sequence of Abstract Program Interface (API) events, thus logging interactions between the artifact and the system kernel. Trace monitors can be implemented within kernel mode or in user mode with *binary instrumentation*, a method that augments the artifact with additional profiling logic.

Several themes to mechanize malware classification are actively researched and pertinent. One theme is the advancements of automated systems for static and dynamic detection, including an emphasis on model-based classification [22, 21, 15]. Nonetheless a strong need for expert human evaluation remains [20], thereby indicating the importance of retaining the human in the loop. Supervised and unsupervised classification of malware features has been considered with considerable previous work in areas of both static and dynamic features[14], and further dimensionality reduction and supervised classification has been addressed [3]. Behavioral analysis also is found in several studies that indicate that API tracing allows the behavior of obfuscated malware to be clearly grouped together in a family[31, 6].

2.2 Human in the loop machine learning

Several papers have addressed leveraging crowdsourced labels to train a machine learning model. This combined crowdsourcing-machine learning approach has been shown successful across various domains, most notably in Computer Vision [29, 17], Machine Translation [5, 2], etc. A recent line of work [32, 23] has employed active learning strategies in crowdsourcing, where the objective is to minimize the annotation cost by querying only the most informative samples from the pool of unannotated data. For example, an uncertainty-based active learning strategy [27] iteratively queries an unlabeled instance with the highest entropy of the class-posterior probabilities, thereby avoiding redundant queries to a model. In this work, we also use an active learning strategy to choose an optimized order of stimuli to show to annotators.

2.3 Visualization

Visualizing high dimensional data is well studied in the human perception literature. Approaches include representing a high dimensional scatter plot in fewer dimensions by finding a lower dimensional embedding of the data [9, 19] and glyph based methods that encode several dimensions into the characteristics of an icon such as Chernoff faces [7] or star glyphs [16]. The innate ability of humans to recognize faces makes Chernoff faces useful for a variety of human computer interface tasks [4]. In this work, we use both Chernoff faces and our own glyph representation of stick figures to visualize malware behavior.

3 DATA

The recognition problem presented to non-experts depends on several stages: (i) the reverse engineering and analysis to provide expert labels, (ii) the trace monitor system that renders behavior data, and (iii) an iterative process of feature engineering, exploratory data-analysis, and machine classification used to determine the non-expert recognition task, and (iv) presentation of visualization and experiments with non-expert human subjects. Non-experts are tested against experts, that is the expert labels are withheld and non-experts are asked to recognize a matching group for each instance. Non-expert performance is evaluated as the difference between their assignments and the expert labels. The recognition problem domain is transformed from traces to faces in stages ii and iii. While currently the transformation requires expert guidance (as discussed below), our future work will test if this can also be accomplished with data-agnostic, automated or non-expert driven methods.

3.1 Ground truth labels

In order to provide a data set that could be used as ground truth for these experiments, we applied an iterative process to identify malware families [12], using a large corpus of candidate malware files obtained from Virus Total [28]. We limited our scope to Microsoft Portable Executable (PE) files, since they comprise a substantial percentage of files likely to be available from public sources [24]. For each family, we started with a single exemplar and reverse engineered it sufficiently to understand both the program structure and behavior. Based on analyst intuition, we selected code believed to be indicative of the exemplar, comprising bytes of the file representing assembled x86 instructions, applied a normalization algorithm (described as the "PIC algorithm" [8]) to these bytes, and encoded the normalized bytes as a YARA signature [33]. We then applied this signature to the corpus of candidate malware files, and selected those candidate files that matched the signature for inclusion within a particular family. These family-included files were compared with each other by an analyst, using techniques such as section hashing [8, 11] and function hashing [8] to attempt to identify outliers or false positives, and this process was iterated until the selected signature produced no false positives from the large candidate corpus. This final signature was then assigned a name corresponding with a family, and files that matched the signature were declared to be part of the family. This process was applied to all malware families selected for these experiments.

3.2 Behavior data

Formally, given a class of events E and a system program p, a program's *trace* $t(p)$ can be defined as a temporal sequence of observed events over E. When a program is run within an operating system, the kernel provides services to a user program, and we therefore focus on these events at the boundary between kernel layer and user layer as the focus for a monitoring tool for what may be considered a kernel Abstract Program Interface (API).

As an example when a program seeks to write a file, perhaps malware to self replicate, the program must request this service from the kernel layer as creating a new file requires a system service. The event is observable via a well designed trace monitor because the service is performed by the kernel via a system call.

In our work, we design *trace monitor tools* abstractly around a set of system calls that would be required to achieve a wide range of system services. Then binary rewriting techniques are implemented to detect and observe these events, using Intel PIN tool library. The system[6] implements a set of monitors on 527 Windows kernel level functions, of which 238 are `Rtl`, functions 224 are `NT` functions and 17 are `Ldr` functions. The trace monitor then becomes a program that takes as input a program p and returns as output a trace $t(p)$ as a sequence over the events of E. Next the trace monitor is deployed in a scalable and safe sandbox setting. Since we execute malware capable of exploiting systems, we use Virtual Machines (VM) to provide both containment[1] and resetting[2] capabilities. Additionally VM technology provide scale, as we can copy the VM and deploy them across physical machines using a variety of hypervisor and virtualization techniques. Having assigned one clean VM to each malware, the only other significant design parameter is the *observational time budget* which maybe defined as the amount of time we allow the malware to run within the trace monitor before halting the observation process as well as terminating the malware process. We usually set the observational runtime budget to 30 seconds. Control of the system is achieved with a variety of scripts to manage the virtual machines, marshalling data and resetting machine states, loading/running executable within the trace monitors, etc.

We maintain the identity of each binary executable program as b its unique identity (provided by a cryptographic hash function), the trace monitor system thus derives a trace for each binary $t(b)$ and we additionally use the family association tag provided by the cataloging effort (described above) as $f(b)$.

3.3 Sample selection

Here we describe the process of selecting a non-trivial and meaningful sample for testing non-expert assistance in operational malware recognition. One problem often requiring experts to go back and reevaluate data involves inconsistent variance measures for separate classifiers. For example, if static detection expresses low variance while behavioral detection expresses high variance, a new malware feature (needing to be learned) could be implicated and human

[1] Containment ensures that a malware will not infect additional machines nor communicate back to a command server information concerning its testing environment.

[2] Virtual Machines can be check-pointed, meaning that their state can be saved and restored, this is useful for malware studies as an infected machine maybe quickly restored to a safe state prior to having any malware contact.

judgment may be preferred or necessary. We therefore incorporate a round of data exploration with principal component analysis (PCA) applied to the expert labeled data to find families with high behavioral variance (relative to the comparatively stable static detection characteristics). Surprisingly, we discovered that variation in behavior features (within each expert tagged family) are infrequent rather than common. In our problem, such variance was noted in few families, resulting in a re-evaluation task with only three families.

We essentially use expert tags and multiple classifier outputs to identify a re-evaluation task needed as part of the overall recognition problem. Arguing that such a task would otherwise be delegated to experts, additional utility for the recognition problem is gained if non-experts can perform the re-evaluate task with the same precision as experts. As future work, we would like incorporate non-experts into the data exploration and reduction phase.

To reduce dimensionality we summarize a trace $t(b)$ as a low dimensional representation of its run-time graph on the events E. Conceptually, we could construct this graph by setting events E as nodes and adding a directed edge between two nodes when the corresponding API calls appeared successively in the trace $t(b)$. We limited the graph to the first 1,000 lines (999 transitions) to standardize the edge weights between graphs. We expressed the call graph as an edge list matrix, where each row of the matrix represented a trace, and each column represented a count of transitions. This resulted in a matrix of 8,277 rows and 652 columns. We then constructed a low-rank approximation to this edge list matrix with Principal Components Analysis[10]. Note that four of the columns had zero variance across traces; we discarded them before performing PCA.

Empirically, the principal components strategy both separated the malware families quite well and revealed two regimes malware heterogeneity. Figure 1 displays box plots of the first principal component for the 29 families that have at least 100 instances. Note that the left of the figure displays families where the first principal component has very little variation within family, but can have appreciable variation between families. The right of the figure, in contrast, displays families that exhibit substantial within variation.

In early work, we selected families from the left of figure, with negligible within family variation. This made the classification task remarkably easy – any machine learning method and any collection of non-experts was able to clearly distinguish between these families without difficulty.

For this work, we chose families from the right of the figure, specifically, the three with the highest within family variation. In the figure, the chosen families are labeled with the arbitrary names Amlity, Squent, and Desper. Note that the families also show considerable overlap with each other on the first principal component. Since it is not feasible to release the ground truth dataset, we provide MD5 hashes of typical examples of these classes:
Amlity bb8789de18346097680ec25c540ccaa8,
Squent 00008f397be9eb4de57165b6ac35b931, and
Desper 00097ac658ea91c6b4e4272b3e95d56d. Further information can be found at Virus Total [28].

A Support Vector Machine (SVM) classifier was used to check the feasibility of distinguishing between these three malware families. Classification using the PCA vectors typically had 80% accuracy due to the presence of outliers. We scaled the PCA vectors to the unit interval by replacing them with their empirical cumulative distribution function, i.e. an empirical histogram transformation. This transformation did improve performance; the SVM cross validation accuracy on the transformed data rose to approximately 90%.

4 VISUALIZATION

4.1 Stick Figures

We set out to choose a visualization that would not be confusing for a layperson, but that also did not leverage pre-existing knowledge. For example, we considered the possibility of using birds to represent different classes of malware, but did not want to introduce a confound where experienced birders performed better because of our chosen representation. We settled on the idea of drawing fictional alien species, and decided to start with a style well within our artistic reach – stick figures.

Figure 2 displays example stick figures for the three families where Desper's are the first two rows with red heads, Amlity's are the third and fourth rows with green heads, and Squent's are the fourth and fifth with blue head. The height and arm angle were generated from the first scaled PCA vector, the number of hairs from the second, the number of sides in the polygon that forms the head from the third, and the length of the legs from the fourth. Note that the coloring was not present for the experiments to avoid biasing the results.

4.2 Chernoff faces

We adopted Chernoff faces as an alternative to stick figues because they have a greater capacity to encode higher dimensional data. For example, the `faces` function from the aplpack R library [30] gives 15 possible dimensions with which to construct the Chernoff faces.

To keep the stick figures and Chernoff faces comparable, we recycled to first four scaled principal components so that each represented several features. We used the default scaling and assignment order of the `faces` function. Specifically, the first principal component adjusted the height of the face, width of the mouth, the height of the hair, and the width of the nose. The second scaled principal component adjusted the width of the face, the amount of smiling, the width of the hair, and the width of the ears. The third scaled principal component adjusted the structure of the face, the height of the eyes, the style of the hair, and the height of the ears. The fourth scaled principal component adjusted the height of the mouth, the width of the eyes, and the height of the nose.

Figure 3 shows example Chernoff faces for each family generated from the same examples as in Figure 2. Note that the Chernoff faces appear to highlight the heterogeneity within families as compared to the stick figures.

4.3 Sensitivity analysis

Although the subjects visualized are simple and commonly familiar, individuals are likely to express differential responses to various features (e.g., face height). While we have sought to avoid confounds arising from preexisting knowledge of the visualization subject (as could arise when visualizing birds) human non-experts may unavoidably present differential responses or other confounds (e.g.,

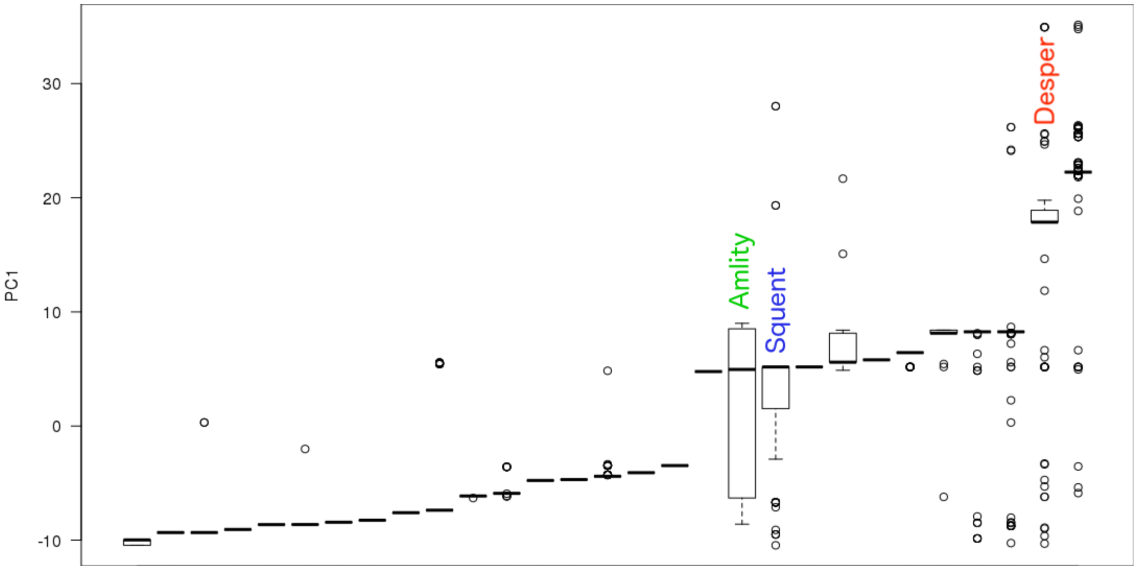

Figure 1: First principal component for 29 families, family names on the x-axis were redacted

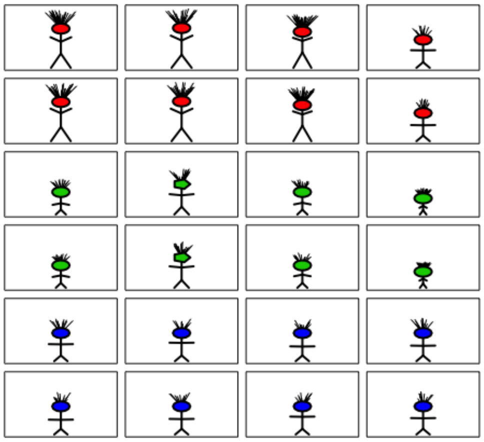

Figure 2: Stick figure examples.

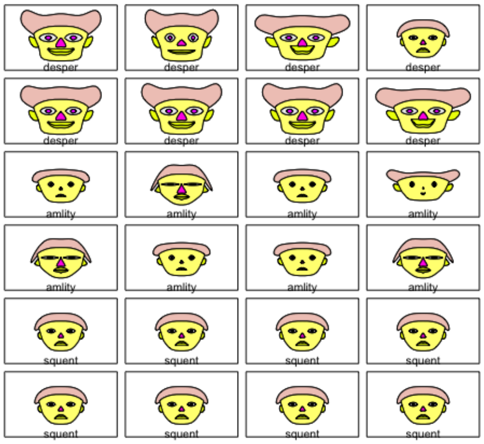

Figure 3: Chernoff face examples.

mood) difficult to control for. On the other hand, the sensitivity of the representation (faces and stick figures) on data can be explored with general numerical techniques (e.g., Monte-Carlo estimation) to evaluate how principal components are sensitive to data noise and model parameters. Additionally, techniques to make PCA more robust to outliers could be applied when sensitivities are determined. A more thorough approach involving model parameters and data sensitivities are considered as future work.

5 EXPERIMENTS

The experimental framework used PsiTurk [13] to present faces and stick figures to anonymous workers on Amazon Mechanical Turk [1]. Figure 4 illustrates the interface. Note that the users were provided with (i) ground truth labels for prior stimuli, (ii) immediate feedback on their accuracy, and (iii) an indication of their progress through the task.

For each of the experiments, the first 15 stimuli were practice stimuli, and the next 150 stimuli were graded for accuracy. We chose the order of stimuli to mimic that of an active learning strategy responding to oracle labeling for the task so that in future work we would be able to compare the difficulty that humans have with labeling a stimulus with the difficulty that a machine learner has. The pay scale for the experiments was based on accuracy, which gave the users an incentive to perform the work well. The schedule is illustrated in Table 1.

We report on four trials which compared the ability of users to distinguish between the Amlity, Squent, and Desper families of malware using the two types of visualization. They are detailed in Table 2.

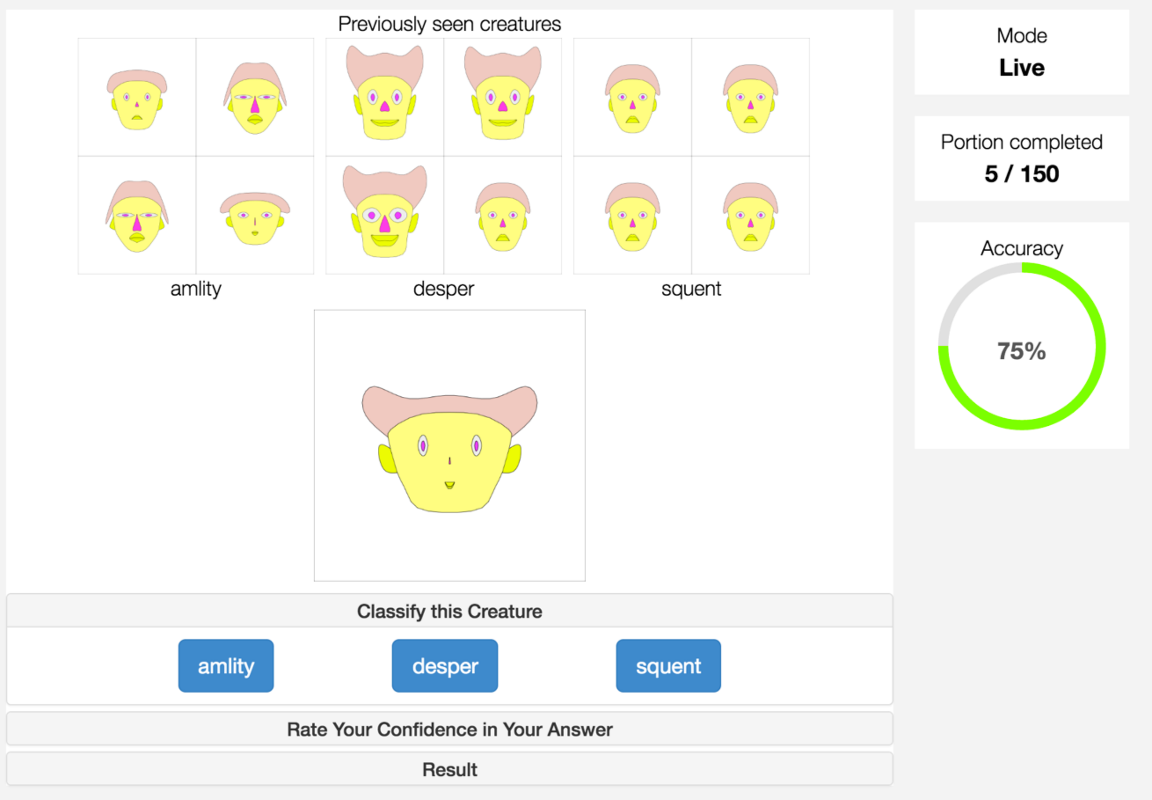

Figure 4: The User's interface to our experiment on Mechanical Turk

% Percent correct	Compensation
0%-44%	$0.50
45%-69%	$1.00
70%-89%	$2.00
90%-100%	$2.50

Table 1: Reward Structure for users

Trial	Users	Visuals
1	50	Chernoff faces
2	100	Chernoff faces
3	75	Stick Figures
4	75	Stick Figures

Table 2: Users and Visuals for each trial

6 RESULTS

We find that the non-expert users of Amazon Mechanical Turk are able to classify heterogeneous families of malware with limited training by using simple visualizations of malware runtime behavior. Figure 5 shows the error rates of the classifiers (SVM) trained with (i) the ground truth labels (blue line), and with (ii) the turkers' answers from Trial 1 and 2 (green line; averaged result), at varying number of stimuli labeled. It can be seen that non-expert answers, albeit with noisy labels, can still yield a classifier model that has a comparable accuracy to a model trained with ground truth labels.

We also find that the modality of visualization affects classification difficulty. Figure 6 displays a scatter plot of the proportion correct for each stimulus as averaged across all of the responses. The colors of the points correspond to the families: Desper's are red, Amlity's green, and Squent's blue. The left panel compares Trials 1 and 2 to examine if the proportion correct is similar between the two trials. Note that this appears to be the case, the points are tightly and evenly clustered around the 45 degree equality line. The right panel compares the proportion correct for the stick figures versus the Chernoff faces. Note that the stick figures are typically more difficult for the non-experts to classify, i.e. the fall below the 45 degree line.

7 DISCUSSION

One consequence of this result for the analyst is that, by using a visualization based on normalized features, the analyst can quickly and intuitively correlate findings in multiple feature spaces, especially when starting from ground-truth. For example, the relatively wide variance of the PC1 values for the family Amlity suggests that there are multiple sets of runtime instruction paths through exemplars of Amlity. Visual inspection of the Amlity Chernoff faces suggests that there may in fact be 2-3 ”primary” clusters of variants within this family. Applying function hashing to the exemplars of this family

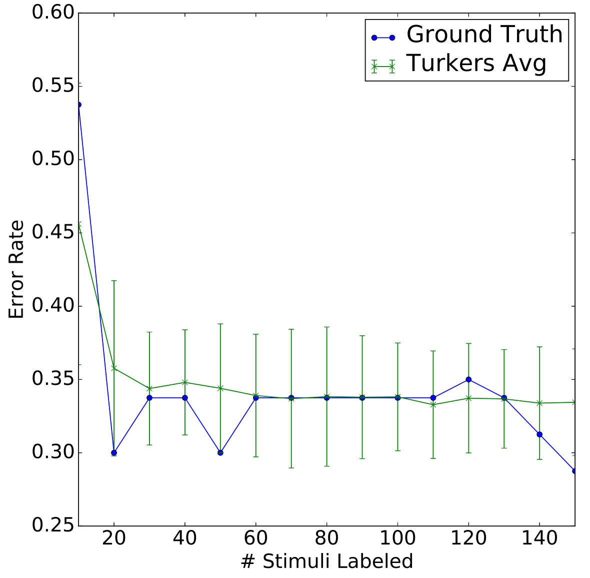

Figure 5: Comparison of error rates on a held-out test set of SVM classifiers trained with (i) ground truth labels and (2) with crowdsourced labels (Trial 1 and 2), at varying number of stimuli labeled. Standard deviation across classification models built from individual turkers' answers are shown in error bars.

reveals that there is significant functional overlap between exemplars, with at least two clusters of files possessing different counts of functions. Reverse engineering two exemplars, one from each candidate sub-cluster (e.g. file MD5 01a28bbf8cf0e6185dc7e97d9e7cc846 and file MD5 049652adb1580ff3c6fdfc6363f09768) reveals that, while both programs appear to be compiled with Visual Studio and to use MFC, the function that resolves the imports for each program (file MD5 01a28bbf8cf0e6185dc7e97d9e7cc846 address 0x402640, and file MD5 049652adb1580ff3c6fdfc6363f09768 address 0x401F80) differ in the stack variables used within the functions. In this way, the visualizations for both PCA and the Chernoff faces suggested variation within the family that was able to be confirmed by reverse engineering, even though the function used to identify the family (found at 0x402E70 in file MD5 01a28bbf8cf0e6185dc7e97d9e7cc846 and at 0x4024A0 in file 049652adb1580ff3c6fdfc6363f09768) was constant between exemplars.

Although this is but a single example, the underlying significance is that this technique may allow the analyst to better estimate the amount of reverse engineering that may be required to understand the various modes of a particular malware family. Combining this fact with an organizational or corporate structure in which many malware families must be understood in a relatively short amount of time, this gives an opportunity to prioritize and/or appropriately resource the efforts by first using automated and lower-skill (implying lower-cost) means to gauge the scope of analysis.

In addition, we find that non-expert labels obtained from the described process can build a classifier model that has a comparable error rate to that of ground truth samples. This suggests promise for our future work on a human in the loop malware analysis system.

8 ACKNOWLEDGEMENTS

Copyright 2017 ACM

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Carnegie Mellon® and CERT® are registered marks of Carnegie Mellon University.

DM-0004454

REFERENCES

[1] *Amazon Mechanical Turk: Artificial Artificial Intelligence.* URL: http://www.mturk.com.

[2] V. Ambati, S. Vogel, and J. G. Carbonell. "Active learning and crowdsourcing for machine translation". In: *LREC '10* (2010).

[3] Usukhbayar Baldangombo, Nyamjav Jambaljav, and Shi-Jinn Horng. "A Static Malware Detection System Using Data Mining Methods". In: *CoRR* abs/1308.2831 (2013). URL: http://arxiv.org/abs/1308.2831.

[4] Maya Cakmak and Andrea L Thomaz. "Eliciting good teaching from humans for machine learners". In: *Artificial intelligence* 217 (), pp. 198–215.

[5] Chris Callison-Burch. "Fast, cheap, and creative: evaluating translation quality using Amazon's Mechanical Turk". In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1.* Association for Computational Linguistics. 2009, pp. 286–295.

[6] William Casey et al. "Agent-based trace learning in a recommendation-verification system for cybersecurity". In: *Malicious and Unwanted Software: The Americas (MALWARE), 2014 9th International Conference on.* IEEE. 2014, pp. 135–143.

[7] Herman Chernoff. "The Use of Faces to Represent Points in K-Dimensional Space Graphically". In: *Journal of the American Statistical Association* 68.342 (1973), pp. 361–368.

[8] C Cohen and J Havrilla. "Function Hashing for Malicious Code". In: *CERT Research Annual Report.* Ed. by Software Engineering Institute. Carnegie Mellon University, 2009, pp. 27–29.

[9] Daniel Engel, Lars Hüttenberger, and Bernd Hamann. "A Survey of Dimension Reduction Methods for High-dimensional Data Analysis and Visualization". In: *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011.* Ed. by Christoph Garth, Ariane Middel, and Hans Hagen. Vol. 27. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 135–149. ISBN: 978-3-939897-46-0.

[10] Brian S Everitt and Graham Dunn. "Principal components analysis". In: *Applied Multivariate Data Analysis, Second Edition* (1993), pp. 48–73.

[11] D. French. "Beyond Section Hashing". In: *CERT Research Annual Report.* Ed. by Software Engineering Institute. Carnegie Mellon University, 2010, pp. 64–66.

[12] J Gennari and D French. "Defining malware families based on analyst insights". In: *Technologies for Homeland Security (HST), 2011 IEEE International Conference on.* Nov. 2011, pp. 396–401.

[13] Todd M Gureckis et al. "psiTurk: An open-source framework for conducting replicable behavioral experiments online". en. In: *Behavior research methods* (Jan. 2015).

[14] Qingshan Jiang, Xinxing Zhao, and Kai Huang. "A feature selection method for malware detection". In: *Information and Automation (ICIA), 2011 IEEE International Conference on.* 2011, pp. 890–895.

[15] Boojoong Kang et al. "Malware Classification Method via Binary Content Comparison". In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium.* RACS '12. San Antonio, Texas: ACM, 2012, pp. 316–321. ISBN: 978-1-4503-1492-3.

[16] Michael D Lee and Rachel E Reilly. "An Empirical Evaluationof Chernoff Faces, Star Glyphs, and Spatial Visualizationsfor Binary Data". In: *Australasian Symposium on Information Visualisation.* Ed. by Tim Pattison and Bruce Thomas. 2003.

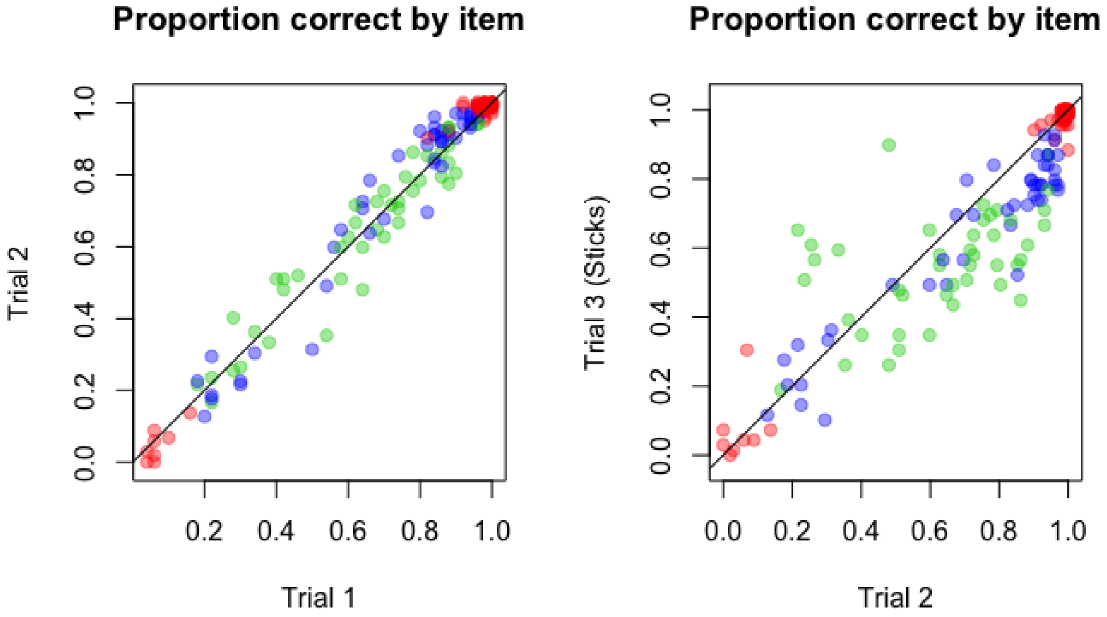

Figure 6: Comparison of the difficulty of labeling given stimuli across trials.

[17] Xin Li and Yuhong Guo. "Adaptive active learning for image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2013, pp. 859–866.

[18] Marta Lopez. "27% of all recorded malware appeared in 2015". In: (Jan. 2016). URL: http://www.pandasecurity.com/mediacenter/press-releases/all-recorded-malware-appeared-in-2015/.

[19] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[20] Dominik Maier, Tilo Müller, and Mykolai Protsenko. "Divide-and-Conquer: Why Android Malware cannot be stopped". In: *9th International Conference on Availability, Reliability and Security.* Ed. by SBA Research. Fribourg, Switzerland, 2014.

[21] Syed Bilal Mehdi, Ajay Kumar Tanwani, and Muddassar Farooq. "IMAD: In-execution Malware Analysis and Detection". In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation.* GECCO '09. Montreal, Qu&#233;bec, Canada: ACM, 2009, pp. 1553–1560. ISBN: 978-1-60558-325-9. DOI: 10.1145/1569901.1570109. URL: http://doi.acm.org/10.1145/1569901.1570109.

[22] Abedelaziz Mohaisen and Omar Alrawi. "Unveiling Zeus: Automated Classification of Malware Samples". In: *Proceedings of the 22Nd International Conference on World Wide Web.* WWW '13 Companion. Rio de Janeiro, Brazil: ACM, 2013, pp. 829–832. ISBN: 978-1-4503-2038-2. DOI: 10.1145/2487788.2488056. URL: http://doi.acm.org/10.1145/2487788.2488056.

[23] Seungwhan Moon and Jaime G Carbonell. "Proactive learning with multiple class-sensitive labelers". In: *Data Science and Advanced Analytics (DSAA), 2014 International Conference on.* IEEE. 2014, pp. 32–38.

[24] Dipl-Ing Maik Morgenstern and Hendrik Pilz. "Useful and useless statistics about viruses and anti-virus programs". In: *Proceedings of the CARO Workshop.* 2010.

[25] Chinmaya Kumar Patanaik, Ferdous A. Barbhuiya, and Sukumar Nandi. "Obfuscated Malware Detection Using API Call Dependency". In: *Proceedings of the First International Conference on Security of Internet of Things.* SecurIT '12. Kollam, India: ACM, 2012, pp. 185–193. ISBN: 978-1-4503-1822-8. DOI: 10.1145/2490428.2490454. URL: http://doi.acm.org/10.1145/2490428.2490454.

[26] Pamela Samuelson and Suzanne Scotchmer. *The Law and Economic of Reverse Engineering.* Levine's Working Paper Archive. David K. Levine, 2003. URL: http://EconPapers.repec.org/RePEc:cla:levarc:618897000000000538.

[27] B. Settles and M. Craven. "Training text classifiers by uncertainty sampling". In: *EMNLP* (2008), pp. 1069–1078.

[28] *Virus Total.* URL: https://www.virustotal.com/.

[29] C Wah. "Crowdsourcing and its applications in computer vision". In: *University of California, San Diego* (2006).

[30] Hans Peter Wolf and Uni Bielefeld. *aplpack: Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plotsummary, plothulls, and some slider functions.* R package version 1.3.0. 2014. URL: https://CRAN.R-project.org/package=aplpack.

[31] J-Y. Xu et al. "Polymorphic Malicious Executable Scanner by API Sequence Analysis". In: *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems.* HIS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 378–383. ISBN: 0-7695-2291-2. DOI: 10.1109/ICHIS.2004.75. URL: http://dx.doi.org/10.1109/ICHIS.2004.75.

[32] Yan Yan et al. "Active learning from crowds". In: *Proceedings of the 28th international conference on machine learning (ICML-11).* 2011, pp. 1161–1168.

[33] *YARA: The pattern matching swiss knife for malware researchers (and everyone else).* URL: http://virustotal.github.io/yara/.

[34] Ilsun You and Kangbin Yim. "Malware Obfuscation Techniques: A Brief Survey". In: *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications.* BWCCA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 297–300. ISBN: 978-0-7695-4236-2. DOI: 10.1109/BWCCA.2010.85. URL: http://dx.doi.org/10.1109/BWCCA.2010.85.