Improved Malware Classification Through Sensor Fusion Using Disjoint Union *

Charles LeDoux[1], Andrew Walenstein[2], and Arun Lakhotia[1]

[1] Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA, U.S.A.
cal@louisiana.edu arun@louisiana.edu

[2] School of Computing and Informatics, Computer Science Program, University of Louisiana at Lafayette, Lafayette, LA, U.S.A.
walenste@ieee.org

Abstract. In classifying malware, an open research question is how to combine similar extracted data from program analyzers in such a way that the advantages of the analyzers accrue and the errors are minimized. We propose an approach to fusing multiple program analysis outputs by abstracting the features to a common form and utilizing a *disjoint union* fusion function. The approach is evaluated in an experiment measuring classification accuracy on fused dynamic trace data on over 18,000 malware files. The results indicate that a naïve fusion approach can yield improvements over non-fused results, but the disjoint union fusion function outperforms naïve union by a statistically significant amount in three of four classification methods applied.

1 Introduction

A fundamental problem encountered in the automatic classification of malware is constructing features extracted from the malicious programs such that accurate classification is achieved. A significant, underlying cause of this difficulty is the fact that program analysis has general limits to accuracy and that weaknesses are often exploited by malware through *program obfuscation* [8]. Thus, all malware classification work that utilizes features generated from program analysis face the threat of inaccurate or missing features that can lead to inaccurate generalization and misclassifications. Effective methods are therefore needed to address this omnipresent problem for malware classification. One such promising method is using *data fusion* [10]. Data fusion involves combining data from disparate sources such that the information is in some sense "better" (more complete or

* ©Springer. 2012. This is the author's version of the work. It is posted here with permission of Springer for your personal use. Not for redistribution. The definitive version was published in Information Systems, Technology and Management Communications in Computer and Information Science, 2012, Volume 285, Part 7, 360-371, DOI: 10.1007/978-3-642-29166-1_32 The final publication is available at www.springerlink.com URL: http://www.springerlink.com/content/x02884k103700654/

accurate, say) than would be possible if the sources are considered independently. *Sensor fusion* is a type of data fusion that combines data outputs from multiple *sensors* which, in the case of malware, are often program analyzers such as disassemblers or tracers.

In order to perform any data fusion, a *fusion function* must be defined. This task is relatively simple when the types of information extracted by the different program analyzers are fundamentally different, since the information can be combined independently. The issue, however, is not so simple when the sensors measure similar but not identical types of information, with differing levels of accuracy. Consider the problem of fusing program behavior data extracted by two different program tracers: one generating system call traces by intercepting system calls and another generating instruction traces by running the program in a specially-instrumented emulator. Both tracers extract information about behavior, but with many potential differences; they are not perfectly substitutable. Additionally, the tracers are susceptible to different obfuscations, so that an obfuscated program might successfully hide salient behaviors from one tracer but not the other. Moreover, while fusing the results might yield a more complete picture, how can we know that the fusion does not serve to multiply the errors of both and lead to decreased accuracy? Thus, an important open problem in malware classification is knowing how to fuse outputs from program analyzers generating related information.

In this paper, we propose a two step approach to solving the problem of fusing similar data in malware classification. We first find a common abstraction that permits combination, and then use a *disjoint union* fusion function. The key insight is the realization that the commonalities and differences in the outputs of the program analyzers may provide useful information for malware classification. That is, apart from the direct benefit of combining two sources of data for a more complete picture, we propose that there is *additional useful information* to be mined from how sensors agree and disagree. The proposed disjoint union approach is utilized to fuse results of the program tracers `CWSandbox` [9] and `Anubis` [2]. We report on an experimental evaluation of the proposed fusion approach using over 18,000 malicious files and four machine learning classifiers. The results show that accuracy improved by a statistically significant amount for three of four classifiers when using the disjoint union fusion as compared to either a naïve union of features or using any single tracer output.

2 Fusing Related, Imperfect Program Data

To achieve the promises of sensor fusion, important problems must be solved in terms of finding a suitable common basis for fusion and in defining a fusion function that yields maximal benefit and minimizes potential problems.

2.1 The Unreliable Sensor Problem

Malware classification faces a unique set of difficulties. First, extracting a complete and accurate set of relevant features about programs is impossible in the

general since many of the program attributes that may identify related malware are not generally computable [8]. Second, analyzers attempting to extract similar results can have a myriad of variances in models and assumptions resulting in outputs containing varying information with differing qualities. For example, program tracers running different versions of operating systems (patched versus unpatched, for example) can generate traces that are different even for identical programs.

The problem is further compounded by the existence of an adversarial context. Malware authors frequently utilize techniques know as *obfuscations* to prevent complete or accurate extraction of program properties. For example, Chen et al. [7] found that 40% of the 6,700 malicious files they examined utilize "anti-VM" or "anti-debugger" evasion methods. This adversarial context is distinct from the classic adversarial classification problem [14] in which adversaries attack the learning algorithms that utilize the features rather than feature extraction itself. Thus, taken in the context of both the limitations of feature extraction and the adversarial element, sensor fusion appears to be particularly advantageous.

2.2 Fusing Sensors for Robustness and Completeness

In this paper we are specifically concerned with what Boudjemaa et al. [6] classify as "fusion across sensors." Fusion across *sensors* combines information obtained from multiple sensors measuring the *same* attribute. In malware, a possible example is fusing results from two different disassemblers in order to combine the strengths of both. Given the limitations of program analysis reviewed above, one can expect several benefits due to fusion across sensors:

1. **Completeness of information.** Two different traces of a program collected under different execution conditions can yield a more complete profile of its possible behaviors. For example, a more complete set of system calls might be collected.
2. **Combination of strengths.** All obfuscations necessarily target some class of sensors [8] due to the impossibility of perfectly obfuscating all properties to all data collectors [4]. So, in a heterogeneous collection of sensors, a given set of obfuscations may negatively affect only a portion of the collection. For example, a junk byte insertion obfuscation can prevent correct disassembly by a linear sweep disassembler, but does not affect a recursive traversal disassembler [13].
3. **Decoupled integration.** While it is possible to create new tools which combine the strengths of existing tools (Kruegel et al. [13] combine aspects of recursive traversal and linear sweep disassembly, for example), it is generally difficult to create such new tools on a continual and ongoing basis. It would advantageous to instead be able to define a fusion function that combines the strengths of existing tools in a fully decoupled manner not requiring repeated algorithm revision and combination of current best-of-breed implementations.

2.3 Sensor Fusion Function Problem

In the domain of program analysis, combining multiple analyzers can be a difficult problem because the models of program information generated by the program analyzers often differ between analyzers, even analyzers measuring the same property. For example, consider the case of the tracers `CWSandbox` and `Anubis`. Both tracers generate reports of interactions with the system. `Anubis`, however, reports registry keys that are "created or opened", whereas `CWSandbox` reports registry key opens and creates separately. Some method for resolving the differences between models must be adopted.

A second important problem for defining a sensor fusion function is to define one that combines the strengths and not the weaknesses of individual sensors. If this problem is not handled correctly, it is possible that composing an accurate sensor with a less accurate sensor will result in a loss of improvement due to the inaccuracies introduced.

3 Sensor Fusion Using Disjoint Union

We describe a general method for fusing the outputs of program analyzers in cases where the analyzers extract the same class of attributes but have differences in their outputs. The heart of the approach is the construction of a feature set using a disjoint union. We illustrate the approach by deriving a fuser for two dynamic program tracers.

3.1 Approach Through Disjoint Union of Features

The proposed approach fuses *features* as illustrated in Figure 1. For this form of fusion, the main design questions are how to assure the features can be combined, and deciding on a fusion function. In our approach, we ensure the ability to combine features by defining a *feature abstraction* and use *disjoint union* as our fusion function.

Definition. Let I be an ordered set indexing a set of program analyzers run on some input program. Let $\Theta = \{\sigma_i \| \forall i \in I\}$ be the sets of outputs corresponding to the program analyzers in I. A *feature abstraction* for Θ is then defined as

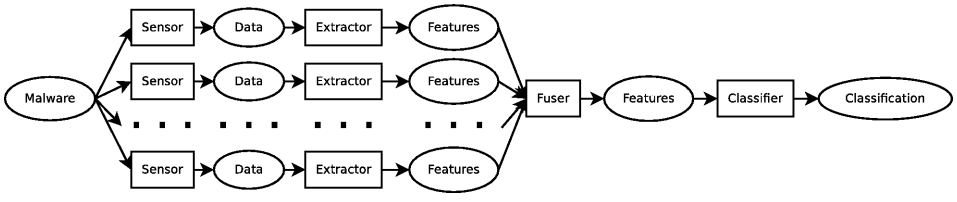

Fig. 1: Feature fusion in malware classification

the pair (F, A) where F is a set of features, and A is the family of functions $A = \{\alpha_i \| \forall i \in I, \alpha_i(\sigma_i) \subset F\}$ such that $\forall i, j \in I, s_i \in \alpha_i(\sigma_i), s_j \in \alpha_j(\sigma_j), [(s_i = s_j) \rightarrow (\sigma_i \approx_{s_i} \sigma_j)]$, where $x \approx_z y$ means analyzer output x is similar to analyzer output y according to some attribute z.

Example Given disassembler D and program tracer T which both provide information about the set of function calls the program may perform, let the output of the disassembler, σ_D, be a list of system calls with abstract parameter types, and the output of the tracer, σ_T, be a sequence of calls with their concrete parameters. Let us assume that the system calls in σ_D and σ_T are represented by their symbolic names. We define a feature abstraction (F, A) over $\Theta = \{\sigma_D, \sigma_T\}$ such that F is the set of all symbolic names contained in either σ_D or σ_T and both α_D and α_T are functions which remove the parameters of the system calls and leave only the symbolic names. We then consider that $\sigma_D \approx_f \sigma_T$ if a call to function f is present in both σ_D and σ_T. In Figure 1, each of the Extractor components implements one of the α_i abstraction functions.

Definition. Let $\{S_i \| i \in I\}$ be the sets of features extracted via the feature abstraction function from the program analyzers indexed by I. That is, $S_i = \alpha_i(\sigma_i)$. Then, *fusion by disjoint union* is defined as $\bigsqcup_{i \in I} = \bigcup_{i \in I} \{(f, i) \| \forall f \in S_i, i \in I\}$. This gives us a set containing the ordered pairs (f, i) where f is a feature extracted from the output of sensor i.

Example Let there be three sensor which generate outputs σ_1, σ_2, and σ_3. Feature abstraction functions α_i are applied to each σ_i to generate $S_1 = \{f_1, f_2\}$, $S_2 = \{f_2, f_3\}$, and $S_3 = \{f_1\}$. The fused feature set, $\bigsqcup_{i \in I} = \{(f_1, 1), (f_2, 1), (f_2, 2), (f_3, 2), (f_1, 3)\}$, indicates that feature f_1 was extracted from the outputs of sensors 1 and 3, feature f_2 was extracted from the output of sensors 1 and 2, and feature f_3 was extracted only from the output of sensor 2. In Figure 1, the Fuser component implements the fusion function.

The proposal to use disjoint union is based on the following hypothesis: the information about which sensors generate which output is additional information that will assist in correct classification, so preserving it in the fusion will aid classifiers. There are two intuitions underlying this hypothesis.

1. **Information Present in Differences.** Programs that are related are likely to generate similar sets of commonalities and differences in the outputs of sensors. Consider two programs x, y, both containing an obfuscation against sensor 1. If sensor 2 is not susceptible to this obfuscation, then S_2 will contain certain system calls that S_1 does not. Since the disjoint union records that the hidden system calls are found by sensor 2 and not by sensor 1, this information can be used to determine both x and y use the obfuscation.
2. **Reputation.** Not all program analyzers for a given attribute will generate equally reliable information. For example, classification accuracy might be significantly higher using one sensor rather than another simply because one sensor generates more consistent sets of features for related inputs than

the other does. By labeling which sensor generates which feature, this is information the classification algorithms can use to factor (in some way) the reliability of input sensors into account.

If the above hypothesis is true, then fusion using disjoint union should tend to result in better classification performance than either naïve union or classification with individual sensors.

3.2 Trace Fusion Example

Our example fusion is of two web-based behavioral analysis systems, `Anubis` (formerly TTAnalyze) [2] and `CWSandbox` [9]. Both of these sensors trace program execution and report on program behavior, particularly interactions with the operating system through system calls. `Anubis` and `CWSandbox` collect data at different system layers and run programs in differing runtime environments. `Anubis` runs in a modified version of the system emulator Qemu and works by inserting itself between the operating system and the hardware [19]. Instead of the operating system directly communicating with and controlling the hardware, it instead communicates with the virtualized hardware provided by `Anubis` through Qemu. `CWSandbox`, on the other hand, uses DLL hooking to insert itself between the Application Binary Interface (ABI) and the operating system [5]. In Windows, the ABI is contained in a collection of DLL files. Whenever one these DLLs are loaded into memory `CWSandbox` modifies the functions in the DLL so that system calls are first "hooked" and control flow redirected to `CWSandbox` instead of directly going to the operating system.

These two analyzers are appropriate choices to illustrate the fusion approach because there must be both expected overlap and differences between the sensor outputs. The hypothesis behind using disjoint union as the fusion function is that the differences between sensor outputs provide additional information. If there are either no differences or no commonalities, then this hypothesis will not hold true. Since `Anubis` and `CWSandbox` collect similar but not identical data using two different methods susceptible to different obfuscations, these requirements are held.

In order to fuse the outputs of `Anubis` and `CWSandbox`, we first need to define the abstract features and the abstraction function. The abstract features used are an adaptation of those presented in [5] and [17]. A feature is an action taken upon a resource, divided into three parts: category, action, and data. The category of a feature corresponds to the type of object that an action was taken upon. These are things such as DLLs, files, processes, etc. For example, the data stored for files is the path to the file and the data stored for registry actions is the registry key. The abstraction function removed all actions unique to a single sensor and converted a few actions to a common abstract feature. For example, in the case of `CWSandbox`, file open events are turned into file read events, and file copy events are turned into file create events. Actions unique to a single sensor were not included because our focus in the paper is on the in case where sensor information is nominally the same. Thus eliminating these actions

ensure measured improvements in classification accuracy are due to combining strengths of the tracers rather than provision of a more comprehensive collection of features. The list of possible features together with some example concrete data are given in Table 1.

Table 1: List of possible features with concrete examples

Possible feature types			**Concrete example**	
Category	**Actions**	**Data**	**Action**	**Concrete Data**
DLL	load	path	load	`c:\windows\system32\mlang.dll`
Mutex	create	name	create	`aashea`
Registry	create, modify, read	key	read	`hklm\software\classes\chkfile`
File	create/modify, read	path	read	`c:\windows\winvrn.exe`
Process	create	name	create	`c:\windows\charmapnt.exe`

4 Evaluation

An experiment was conducted to evaluate the hypothesis that fusion by disjoint union can succeed in improving classification, and that the additional information in the pattern of sensor responses helps combine strengths of sensors without combining weaknesses. The essential test is to compare trained classifier performance for statistically significant differences when using different feature sets on three treatments: single sensor, regular union fusion, and disjoint union fusion. Specifically, the experiment tests the following:

1. **Increased Performance.** The application of sensor fusion using disjoint union is expected to increase the accuracy of malware classification. This is tested by comparing the accuracies obtained when no feature fusion is used to the accuracies obtained when feature fusion is used. If the fusion function increases accuracy, we should be able to observe statistically significant differences in the dependent variable of classifier accuracy.
2. **Increased Performance Due to Patterns of Differences.** It is possible that any increase in accuracy observed in the fusion case is due solely to the composition of information from multiple sensors and no additional improvement is made through the use of patterns of sensor differences. We test against this by comparing accuracies obtained using disjoint union as the fusion function to accuracies obtained using regular set union as the fusion function. If the disjoint union provides additional useful information to the classifiers, we should be able to observe a statistically significant increase in the dependent variable of classifier accuracy.

4.1 Data Set

The sample of malware used contained 18,422 executables and was composed of six malware families plus a set of benign (non-malicious) executables gathered from three different sources. The files belonging to the Agent, Parite, PcClient, and the Benign families were obtained from a commercial anti-virus vendor. The Banker and SDBot families were provided by OffensiveComputing.net. The Storm class was self-collected from the Storm botnet. The benign files were collected from system and third party executables from clean Windows 2000 and Windows Vista systems. Table 2 gives the number of malware files in each family.

Table 2: Malware files in sample, organized by family

Family	**# Files**	**% of Collection**	**Source**
sdbot	5,956	32.33%	OffensiveComputing.net
agent	1,930	10.48%	Commercial AV
benign	5,460	29.64%	Clean Systems
banker	1,118	6.07%	OffensiveComputing.net
pcclient	1,562	8.49%	Commercial AV
parite	1,841	9.99%	Commercial AV
storm	575	3.12%	Self Collected

4.2 Procedure

The overall procedure followed was to submit the malware files to the sensors and retrieve the raw data, extract the features from the raw data, create the fused and non-fused feature sets, determine classification accuracies, and perform statistical analysis. The feature set reductions, cross validation, and the classifications for the paired t-tests were performed using the open source tool `RapidMiner` [16]. The dependent pairs t-tests were performed using the Data Analysis ToolPak in Microsoft Excel 2007.

Feature Set Generation. There were four types of feature sets extracted. The `Anubis` feature set contained only features that were detected by `Anubis`, the `CWSandbox` feature set contained only features detected by `CWSandbox`, the Union feature set was created using union as the fusion function, and the Disjoint feature set was created using disjoint union as the fusion function. The total number of features for each feature type are given in Table 3. Information gain was used to reduce the features used to a manageable subset. The number of features detected by each sensor are given in Table 4

Table 3: Number of features by feature set

Feature Type	Before Reduction	After Reduction
`Anubis`	24,767	661
`CWSandbox`	40,854	906
Union	58,924	1,363
Disjoint Union	58,924	1,265

Table 4: Number of features by sensor set

Detected By	Number of Features
Only `Anubis`	18,070
Only `CWSandbox`	34,157
`Anubis` and `CWSandbox`	6,697
`Anubis` or `CWSandbox`	58,924

Accuracy Measure Collection. The learning algorithms used for classification are: Naïve Bayes, Rule Induction, Decision Tree, and K-Nearest Neighbor (KNN) as implemented by the `RapidMiner` [16] application. 10-Fold Cross Validation was used to obtain accuracy estimations for all combinations of classifier and feature set types. Hold-Out classification was performed to obtain measurements for the statistical tests. This was due to limitations of `RapidMiner` as it did not provide mechanisms to collect the needed data using Cross Validation.

Statistical Tests. Paired t-tests were performed only against combinations of feature sets using the same classifier. The experiment was focused on how the feature sets rather the classifier affected classification accuracy. In the same way, statistical tests were not performed to compare classifications using the `CWSandbox` feature set against those using the `Anubis` feature set.

4.3 Results

The overall accuracies obtained from Cross Validation are given in Table 5. The bold accuracies indicate the feature type with the highest accuracy for each classifier. For the Naïve Bayes classifier, the `CWSandbox` feature set achieved the highest accuracy with 81.40%. For the Rule Induction, Decision Tree, and KNN classifiers, Disjoint Union had the highest accuracies with 91.30%, 92.83%, and 95.30% respectively.

Table 5: Classification accuracies

Feature Type	Naïve Bayes	Rule Induction	Decision Tree	KNN
`Anubis`	67.78%	75.32%	78.20%	86.29%
`CWSandbox`	**81.40%**	86.98%	88.91%	91.65%
Union	76.45%	84.11%	90.48%	92.85%
Disjoint	79.88%	**91.30%**	**92.83%**	**95.30%**

The p-values from the paired t-tests are given in Table 6. The difference in accuracies between the two feature sets is statistically significant if the p-value is less than 0.05. The places where statistical significance is *not* achieved are indicated by bold text. This is only between the Union and `CWSandbox` feature sets for the KNN classifier (6.11×10^{-2}), the Disjoint and `CWSandbox` feature sets using the Naïve Bayes classifier (2.02×10^{-1}), and the Disjoint and Union feature sets using the Decision Tree classifier (8.41×10^{-2}).

Table 6: P-values from the paired t-tests

Feature Types	**Naïve Bayes**	**Rule Induction**	**Decision Tree**	**KNN**
Union & Anubis	1.00×10^{-16}	1.00×10^{-16}	1.00×10^{-16}	1.00×10^{-16}
Disjoint & Anubis	1.00×10^{-16}	1.00×10^{-16}	1.00×10^{-16}	1.00×10^{-16}
Union & CWSandbox	2.29×10^{-6}	1.11×10^{-4}	1.33×10^{-4}	6.11×10^{-2}
Disjoint & CWSandbox	2.02×10^{-1}	9.42×10^{-7}	8.58×10^{-9}	1.63×10^{-9}
Disjoint & Union	1.00×10^{-15}	1.00×10^{-13}	8.41×10^{-2}	1.02×10^{-6}

4.4 Discussion

The results support the hypothesis that sensor fusion using disjoint union improves the accuracy of malware classification. As can be seen in Table 5, disjoint union obtained a statistically significant increase in accuracy over the classifications performed without feature fusion in all but one case. For the Naïve Bayes classifier, the `CWSandbox` feature set had an accuracy of 80.40%, while disjoint union had an accuracy of only 79.88%. However, according to Table 6, the p-value for these two classifications is 0.202 indicating this difference in not statistically significant. All increases in accuracy disjoint union obtained over one of the non-fused feature sets, however, were statistically significant. For the Rule Induction, Decision Tree, and KNN classifiers, the `CWSandbox` feature set obtained accuracies of 86.98%, 88.91%, and 91.65%, while disjoint union obtained higher accuracies of 91.30%, 92.83%, 95.30%. The p-values for these classifications were 9.42×10^{-7}, 8.58×10^{-9}, and 1.63×10^{-9}.

The results also lend evidence to support the hypothesis that the additional information captured by disjoint union contributes to the improvement in classification accuracy. This can be seen by comparing the accuracies of Disjoint features to Union features. Looking at Table 5, we find that Disjoint consistently achieves a higher accuracy than Union (79.88% vs. 76.45%, 91.30% vs. 84.11%, 92.83% vs. 90.48%, 95.30% vs. 92.85%) and these increases are statistically significant, except when using the Decision Tree classifier (p-values of 1×10^{-15}, 1×10^{-13}, 1.02×10^{-6}, and 8.41×10^{-2}). Lending further evidence is the fact that while Disjoint consistently achieved higher accuracies than no fusion, Union

only performed better than the `CWSandbox` feature set when using the Decision Tree and KNN classifiers (90.48% vs. 88.91% and 92.85% vs. 91.65%), and only the increase using the Decision Tree classifier was statistically significant with a p-value of 1.33×10^{-4} (KNN had p-value of 6.11×10^{-2}).

5 Relations to Other Work

The concept of utilizing patterns of differences in sensor outputs to understand malware was previously explored in several prior works. Kang et al. [12] and Balzarotti et al. [3] use divergences in the execution behavior of the same malware running on both reference hardware and an emulation environment to automatically detect circumstances where the malware is detecting the emulated environment and modifying its behavior as a result. Allen et al. [1] describe a method called cross-view diff which uses discrepancies between different views of the same data structure to detect malware attempting to hide by modifying the data structure. While these three works cleanly illustrate the promise of exploiting the patterns of differences between sensor outputs, they only used these differences to identify malware which contains a specific type of obfuscations. The present work instead utilizes the insight to improve machine learning classifiers.

The idea of using data fusion in malware classification has also been explored by previous works. Islam et al. [11] combine function length frequency and printable string features. While Islam et al. did not have a specified motivation for the features chosen to combine, Lu et al. [15] combine features which are meant to be complementary, specifically static and dynamic features. Walenstein et al. [18] provide a review of work combining program metadata with various other data. None of these works address the problems associated with fusion across sensors.

6 Conclusion

In this paper, we have introduced a method of performing sensor fusion using disjoint union to combine the strengths of program analyzers ("sensors") while minimizing their weaknesses in the context of malware classification. We present an implementation of our approach and evaluate it through an experimental case study. We found that the application of sensor fusion using disjoint union typically increased the accuracy of classification by a statistically significant amount. Additionally, the results from the case study lend evidence to the additional information provided through the use of disjoint union rather than a naïve union contributing to this increase in accuracy.

One question left by this paper to be addressed in later work is the nature of the additional information provided through the use of disjoint union. It is evident that it is useful to provide the classifier with provenance information on the features, but the question still remains: Why? There are several possible answers to this question. It could be that obfuscations are affecting the sensors differently, or perhaps the provenance information helps the classifier decide

which features are trustworthy from which sensor. It is worthwhile to explore this question further.

7 Acknowledgments

This research work was sponsored in part by funds from Air Force Research Lab and DARPA (FA8750-10-C-0171) and from Air Force Office of Scientific Research (FA9550-09-1-0715). We would like to thank Craig Miles, Anshuman Singh, and Daniel Hefner for help with test design and implementation.

References

1. Allen, W.H., Ford, R.: How not to be seen II: The defenders fight back. IEEE Security & Privacy 5(6), 65–68 (2007)
2. Anubis: Analyzing unknown binaries (June 2011), `http://anubis.iseclab.org`
3. Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., Vigna, G.: Efficient detection of split personalities in malware. In: Network and Distributed System Security (NDSS) (2010)
4. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Killian, J. (ed.) Advances in Cryptology – CRYPTO 2001: Proceedings of the 21st Annual International Cryptology Conference (2001)
5. Bayer, U., Kruegel, C.: TTAnalyze: A tool for analyzing malware. Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference (2006)
6. Boudjemaa, R., Forbes, A.: Parameter estimation methods for data fusion. NPL Report CMSC 38(04) (2004)
7. Chen, X., Andersen, J., Mao, Z., Bailey, M., Nazario, J.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: Proceedings of the IEEE International Conference on Dependable Systems and Networks. pp. 177–186. Anchorage, AK, U.S.A. (2008)
8. Collberg, C., Nagra, J.: Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Addison-Wesley Professional (2009)
9. CWSandbox: behavior-based malware analysis (June 2011), `http://mwanalysis.org`
10. Hall, D., Llinas, J.: An introduction to multisensor data fusion. Proceedings of the IEEE 85(1), 6–23 (1997)
11. Islam, R., Tian, R., Batten, L., Versteeg, S.: Classification of malware based on string and function feature selection. Cybercrime and Trustworthy Computing, Workshop 0, 9–17 (2010)
12. Kang, M.G., Yin, H., Hanna, S., McCamant, S., Song, D.: Emulating emulation-resistant malware. In: Proceedings of the 1st ACM Workshop on Virtual Machine Security. pp. 11–22. ACM, Chicago, Illinois, USA (2009)
13. Kruegel, C., Robertson, W., Valeur, F., Vigna, G.: Static disassembly of obfuscated binaries. In: Proceedings of the 13th USENIX Security Symposium. pp. 255–270. Usenix (2004)
14. Laskov, P., Lippman, R.: Machine learning in adversarial environments. Machine Learning 81, 115–119 (2010)

15. Lu, Y., Din, S., Zheng, C., Gao, B.: Using multi-feature and classifier ensembles to improve malware detection. Journal of C.C.I.T. 39(2) (November 2010)
16. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid prototyping for complex data mining tasks. In: Ungar, L., Craven, M., Gunopulos, D., Eliassi-Rad, T. (eds.) Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 935–940. ACM (2006)
17. Trinius, P., Willems, C., Holz, T., Rieck, K.: A malware instruction set for behavior-based analysis. Tech. Rep. TR-2009-07, University of Mannheim (2009)
18. Walenstein, A., Hefner, D., Wichers, J.: Header information in malware families and impact on automated classifiers. In: Proceedings of the 5th International Conference on Malicious and Unwanted Software. pp. 15–22. IEEE CSP (2010)
19. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using CWSandbox. IEEE Security & Privacy 5(2), 32–39 (2007)