Visualization of Shared System Call Sequence Relationships in Large Malware Corpora

Josh Saxe
Invincea Labs
josh.saxe@invincea.com

David Mentis
Invincea Labs
david.mentis@invincea.com

Chris Greamo
Invincea Labs
chris.greamo@invincea.com

ABSTRACT

We present a novel system for automatically discovering and interactively visualizing shared system call sequence relationships within large malware datasets. Our system's pipeline begins with the application of a novel heuristic algorithm for extracting variable length, semantically meaningful system call sequences from malware system call behavior logs. Then, based on the occurrence of these semantic sequences, we construct a Boolean vector representation of the malware sample corpus. Finally we compute Jaccard indices pairwise over sample vectors to obtain a sample similarity matrix.

Our graphical user interface links two visualizations within an interactive display. Our first view is a map-like visualization of similarity among the samples based on a reduced dimensional projection of our similarity matrix. Our second view provides insight into similarities and differences between selected malware samples in terms of the system call sequences they share. We also provide a set of interactive filters based on malicious behavioral traits. The integration of these views into an interactive, linked display allows users to comprehend the overall similarity structure of a malware corpus, inspect how behavioral traits distribute over the corpus, and to drill in to inspect local similarities and differences between samples.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive software (e.g., viruses, worms, Trojan horses)

General Terms

Security, Artificial Intelligence, Visualization

Keywords

Computer Security, Malware, Data Mining, Data Visualization

Work funded under the DARPA Cyber Genome program. Approved for public release, distribution unlimited. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

1. INTRODUCTION

In recent years researchers have applied a variety of automatic malware similarity analysis techniques to help address the deluge of new malware variants appearing on the Internet. While the research in this area has produced effective methods, there is a need to explore approaches that produce easily interpretable visualizations of malware corpora. Here we present one such approach, a system which visualizes shared system call sequence relationships within large malware corpora.

We base our work on an algorithm we term semantic sequence extraction. Semantic sequence extraction partitions a malware system call log so that the extracted system call sequences express meaningful functional blocks. By way of analogy, if system call logs are seen as long run-on sentences, our algorithm finds natural locations for punctuation marks. Once we have located these punctuations marks, we split the log into sequences by treating the marks as subsequence boundaries.

Our sequence partitioning algorithm is based on two intuitions: first, that if we represent a system call log as a Markov chain in which unique states are system calls plus the file paths, registry keys, or network tuples they operate upon, we can find meaningful partitions where improbable state transitions occur. This is because, we hypothesize, such transitions will tend to occur at the end of loops, functions and other oft-repeated control flow structures. Second, we intuit that logical breaks in these logs occur when a system call and its successor sharply diverge in terms of the similarity of their file paths, registry paths, or network tuples. We employ our partitioning approach to malware corpora to extract variable length subsequences that represent meaningful blocks of program behavior. Next we compare samples in terms of the sets of subsequences present in their behavior logs.

To visualize subsequence occurrence as well as sample similarities, we link together a number of displays within a prototype graphical user interface. The main panel of our interface displays the similarity structure of the entire malware corpus under analysis. Specifically, we employ a multi-dimensional scaling technique to project our sample similarity matrix onto a two-dimensional grid.

To support various malware analysis tasks, our interface supports a number of interactions. Users can highlight one or many samples on the grid to bring up an inspector panel on the top of the screen which renders a detailed view of which semantic sequences the selected malware executed. Panels on the left of the display provide the user with a set of filters, so that they can see how various behavioral traits distribute over the malware corpus.

This paper introduces our system and relates it to existing work on visualization of malware. The structure of the rest of this paper is as follows. Section II details the methods employed in each of our system components, describing our malware sandbox, semantic sequence extraction algorithm, sequence labeling algorithm and our similarity metric. Section III discusses the scalability of our system and the nature of the test data set we used in developing and testing our system. Section IV describes the methods by which we visualize our results, including a discussion of our sample subsequence visualization, our similarity map view, and our interactive filters. Section V describes a number of use cases for our system in the context of malware similarity exploration. For each use case, we give an example based on real-world malware. Section VI discusses related work. And Section VII discusses our plans for future work.

2. SYSTEM DESCRIPTION

Our system includes four components: a virtual machine sandbox for extracting malware behavior logs, a semantic sequence extraction and similarity calculation component, a labeling component which categorizes semantic sequences in terms of their system effect, and a visualization tool which presents analysis results to the user.

2.1 Instrumentation Environment

To extract a record of a malware sample's execution behavior we have leveraged two existing tools: QEMU, which is an x86 emulator, and Procmon, which is an instrumentation tool for the Windows operating system [2] [3]. Our procedure for extracting a behavior log for a given malware sample through these two tools is as follows.

A sample is loaded onto the QEMU-emulated sandbox's virtual hard drive. Procmon, responsible for collecting system call information for all processes, is enabled on the virtualized guest Windows XP operating system. The sample is then executed on the sandbox. The sample completes its run or times out after a global, predefined length of time, at which point the Procmon behavior log is extracted from the sandbox and the virtual operating system is powered down. For the tests discussed in this paper, we let samples run for a maximum of 10 minutes.

After running samples and collecting logs from Procmon, we prune the Procmon logs as follows. First, we identify the system process ID of the malware sample under analysis. Then we iterate over its system call log identifying places where the process or any of its descendants created new threads and child processes. Finally, we filter the behavior log so that only the malware process and its descendant threads and processes are represented in the log.

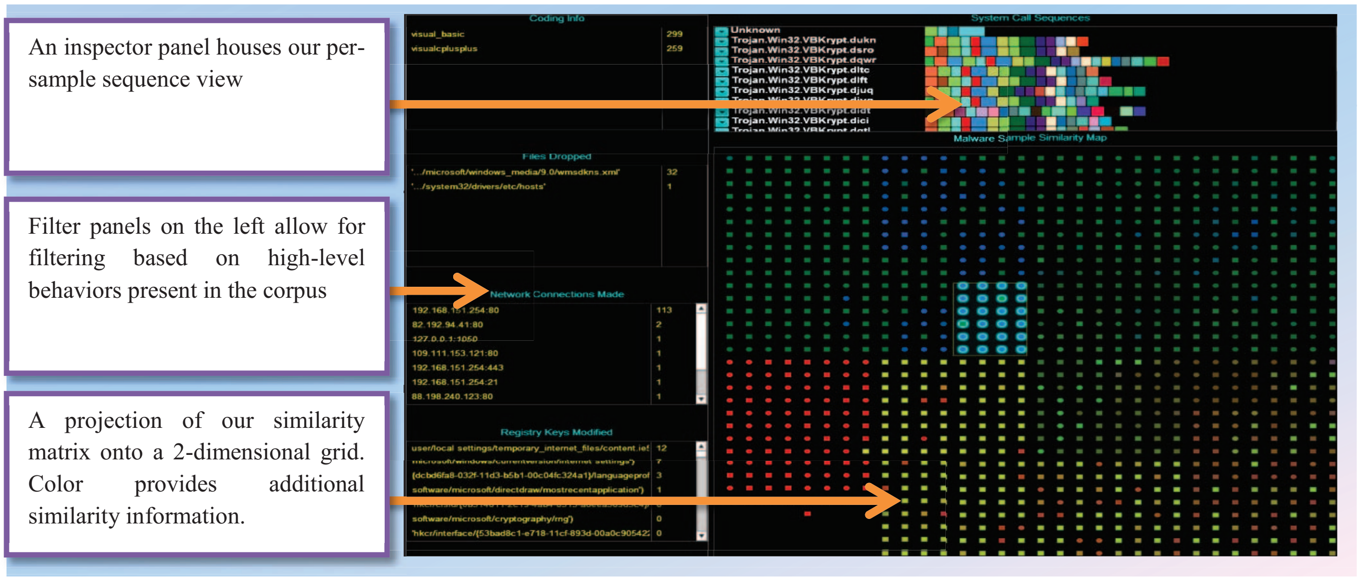
The system call data we inspect within the Procmon logs contains a number of fields, four of which we use in our analysis. The first is the thread ID, which we use to order the events into sequences based on thread. The second field is a timestamp, indicating when each system call was made. The third is an operation name, denoting the system call the subject process took in relation to the registry, the filesystem, or the network.

The fourth field contains information about the system or network object the operation acted upon. For registry operations, this field contains the fully qualified registry key path, for file operations, the fully qualified file path, and for network operations, the relevant source IP, source port, destination IP and destination port. Procmon records only TCP and UDP traffic, and not, for example, ICMP traffic. Thus our analysis currently only considers TCP and UDP network interaction. It should be noted here that for the purposes of this paper, by system call we mean the Procmon "operation" field; Procmon populates this field with a slightly abstracted version of the actual system calls provided by the Windows operating system.

2.2 Semantic Sequence Extraction

Once we have collected behavior logs from a corpus of malware samples, we employ our novel technique for partitioning each sample system call log into sequences of system calls that represent functional blocks (such as walking the registry key tree to find a key entry, or looping over ReadFile and WriteFile to copy a file from one location to another).

As mentioned in the introduction to this paper, our algorithm is based on the combination of two separate intuitions. First, we hypothesize that if we derive a Markov chain from the malware sample's system call log we can meaningfully partition the log at the points at which improbable state transitions between system calls occur. This, we conjecture, is because these improbable


An inspector panel houses our per-sample sequence view
Filter panels on the left allow for filtering based on high-level behaviors present in the corpus
A projection of our similarity matrix onto a 2-dimensional grid. Color provides additional similarity information.
Coding Info
Files Dropped
Network Connections Made
Registry Keys Modified
System Call Sequences
Malware Sample Similarity Map

visual_basic	299
visualcplusplus	259

'.../microsoft/windows_media/9.0/wmsdkns.xml'	32
'.../system32/drivers/etc/hosts'	1

192.168.151.254:80	113
82.192.94.41:80	2
127.0.0.1:1050	1
109.111.153.121:80	1
192.168.151.254:443	1
192.168.151.254:21	1
88.198.240.123:80	1

user/local settings/temporary_internet_files/content.ie!	12
microsoft/windows/currentversion/internet settings')	7
{dcbd6fa8-032f-11d3-b5b1-00c04fc324a1}/languageprof	3
software/microsoft/directdraw/mostrecentapplication')	1
[illegible]	0
software/microsoft/cryptography/rng')	0
'hkcr/interface/{53bad8c1-e718-11cf-893d-00a0c905422	0

Unknown
Trojan.Win32.VBKrypt.dukn
Trojan.Win32.VBKrypt.dsro
Trojan.Win32.VBKrypt.dqwr
Trojan.Win32.VBKrypt.dltc
Trojan.Win32.VBKrypt.dlft
Trojan.Win32.VBKrypt.djuq
Trojan.Win32.VBKrypt.diva
Trojan.Win32.VBKrypt.didt
Trojan.Win32.VBKrypt.dici
Trojan.Win32.VBKrypt.dqtl

Figure 1. An overview screenshot of our prototype Graphical User Interface.

Example Semantic Subsequence Extraction from Sample of Variant Type "menti-gtmr"

```
...
-----------------------------------------------------------------------------------------------
ReadFile      C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile      C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile      C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile      C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile      C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile      C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
-----------------------------------------------------------------------------------------------
RegQueryKey   HKLM/SOFTWARE/Microsoft/Windows-NT/CurrentVersion/Fonts
-----------------------------------------------------------------------------------------------
RegCreateKey  HKU/S-1-5-21-436374069-813497703-1177238915-1004/Software/Microsoft/GDIPlus
-----------------------------------------------------------------------------------------------
RegQueryValue HKU/S-1-5-21-436374069-813497703-1177238915-1004/Software/Microsoft/GDIPlus/FontCachePath
-----------------------------------------------------------------------------------------------
QueryOpen     C:/RUNME/ShFolder.DLL
QueryOpen     C:/WINDOWS/system32/shfolder.dll
CreateFile    C:/WINDOWS/system32/shfolder.dll
CreateFileMap C:/WINDOWS/system32/shfolder.dll
CreateFileMap C:/WINDOWS/system32/shfolder.dll
Load Image    C:/WINDOWS/system32/shfolder.dll
-----------------------------------------------------------------------------------------------
. . .
```

Figure 2. Example output from our sample sequence extraction algorithm, horizontal lines represent partitions.

state transitions represent the termination of program loops and the entry into and return out of functions which are called from multiple places within malicious source code. Our second intuition is that meaningful partitions may be located when a system call and its successor system call refer to highly divergent files, network objects, or registry keys.

Our sequence extraction algorithm involves the following steps. In an initial preprocessing step we sort the malware behavior log by thread ID, and then by timestamp, so as to "flatten" the thread structure of the log. Next we derive a Markov chain from the resulting set of behavioral sequences. The nodes in this chain are defined as unique system calls plus the system objects they act upon, so that, for example, a call to "WriteFile" which acts upon "C:\temp.txt" would comprise a node. Transition probabilities are calculated by adding up the number of times there we observe a transition between a given node and its successor node. To get the transition probability for an edge, we divide the transition count across that edge by the total transition count for that edge and all of its siblings.

Having constructed a Markov chain based on the malware's behaviors, we make another pass over the log, computing an additional parameter similarity score for each system call bigram. At a high level, this score is computed by comparing the parameter strings of each system call and the system call that immediately follows it; the more different they are, the lower this score. Specifically, the similarity between two sequential calls is computed as follows. If the two calls both operate on the registry, we tokenize their registry key parameters using the registry path separator "\". Then we lowercase normalize the resulting registry path tokens, and compute the Jaccard index similarity between the two sets of tokens; we regard this as the similarity between the parameter strings.

If the two calls operate on files, we perform the same tokenization operation, but for the file paths. If the calls invoke network operations, we treat them as having a similarity of "1" if they refer to the same foreign IP address and port, and as having a similarity of "0" if they refer to different foreign IP addresses and ports. Finally, if the two calls are of different classes (for example, a registry call and a file call), we set their similarities to zero. We do this because, at least for now, we do not have a more sophisticated way of comparing these two different parameter types.

Having computed both a transition probability and a parameter similarity for each system call transition in the sample system call log, we then iterate over the log one final time to insert partitions and extract semantic subsequences. To decide whether or not to

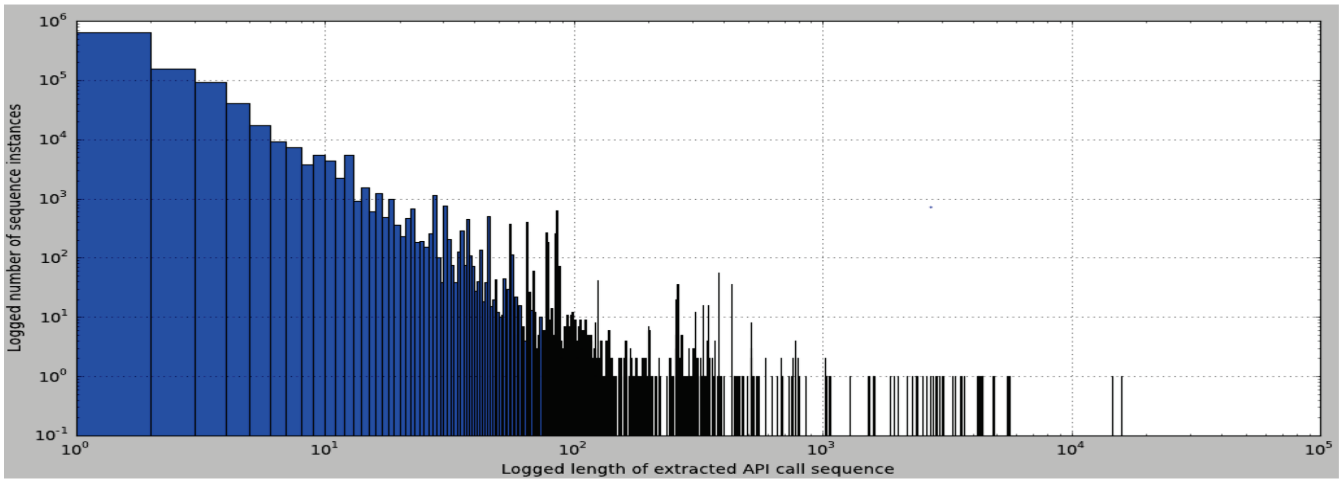


Figure 3. A histogram of semantic sequence lengths extracted from a sampling of malware executables.

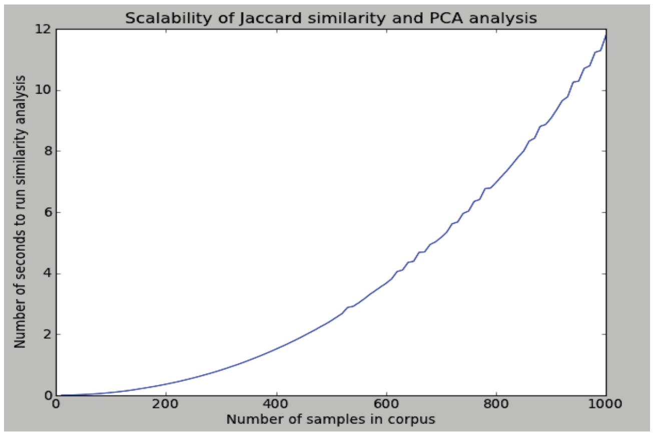


Figure 4. Effect of corpus size on run times on a single Intel Xeon e5645 core running at 2.4GHz with 32 GB of RAM.

insert a partition at any particular pair of system calls, we compute a weighted average between the system call transition's parameter similarity score and its transition probability, and then check to see if this score is below a threshold, which for the purpose of this paper we have set to 0.3.

If the score is below the threshold, we interpret this as meaning that there is a functional break in program behavior at this location in the system call log, and we insert a partition. When we have finished inserting partitions, we extract the system call sections between partitions as our semantic subsequences. For the experiments discussed in this paper, our parameter similarity score and transition probability make equal contributions to the weighted-average final score that we use to decide whether or not to partition. In the future, we plan to investigate optimal combinations of these weights as well as our threshold parameter.

Figure 3 displays a histogram of the lengths of the semantic sequences we extracted from a corpus of 1000 malware samples drawn a dataset we describe below in Section III. The charts shows that these sequences tend to be short, and follow an inverse power-law length distribution.

2.3 Sequence Labeling

To support visualization, we assign human readable, textual labels to the set of sequences we extract from the malware corpus under analysis. These labels are assigned based on simple regular expressions which we match against the system call arguments. For example, if a system call in a sequence modifies a registry key with the substrings "registry" and "Internet," we create a label for that sequence that describes it as modifying Internet settings. One sequence can have one or more labels, which we use to add descriptive information to our graphical user interface. In future work we plan to describe in depth our efforts to categorize semantic subsequences of system calls.

2.4 Similarity Matrix Computation

Having extracted variable length sequences from a malware corpus, we proceed to compute a similarity matrix. Specifically, we construct Boolean sample vectors based on the occurrence of these variable-length sequences in each sample. To eliminate noise, we throw out any sequences that do not occur in at least two samples. Finally, we compute a Jaccard index pairwise over the sample vectors. As we discuss below, our extracted subsequences and our sample similarities are subsequently used in our interactive visualizations.

3. DATASET AND PERFORMANCE

Throughout our development effort we have been testing our system against malware data. Specifically, we test our system on malware samples collected from the MD:Pro malware data feed, and validate our algorithms by checking whether they correspond to Kaspersky anti-virus engine classifications of the samples. Below we describe our dataset, ground truth, run times for our algorithms, and the accuracy of our system when it is measured against ground truth.

3.1 Data Collection

All of the malware samples we depict here were sourced from the MD:Pro malware feed. MD:Pro is a paid-subscriber malware feed service run by security consultancy Frame4. We sourced 100,415 unique malware samples from MD:Pro between February 2011 and June 2012, and, so far, have run 28,460 of these samples in our malware analysis sandbox. Additionally, we have input each sample into the Kaspersky anti-virus engine to generate ground-truth labels. Kaspersky was able to classify 30,104 of these samples with robust hierarchical classifications (such as "trojan.win32.jorik.skor.akr"), assigned a class of "unknown" to 41,830 samples and assigned low-fidelity labels (such as "generic win32 trojan") to the remaining 28,481 samples.

We hand-picked a dataset from our MD:Pro malware feed to generate the tests and examples in this paper. Because of the inconsistency of the Kaspersky labels (some Kaspersky label groupings, such as the "jorik" family, represent a very diffuse range of malware, whereas others, such as "webprefix" represent a very specific set of variants) we heuristically selected four Kaspersky defined malware classes that we believe share a similar range of behavioral traits. Specifically, we selected the "xtoober", "jorik" and "vbkrypt" families, alongside a random selection of malware samples Kaspersky labeled as "unknown" samples.

3.2 Validation

To verify that our sequence features and Jaccard similarity function are effective at capturing malware similarities, we have validated that they reflect malware-analyst defined ground truth. Namely, we filter our dataset down to include only samples that have complete Kaspersky classifications assigned to them by the Kaspersky anti-virus engine, compute a sample similarity matrix for these samples, cluster our sample similarity matrix, and then compute precision and recall statistics on our clusters based on how well they fit Kaspersky's classifications.

We compute precision and recall for a given clustering as the weighted average precision and recall for each cluster that we

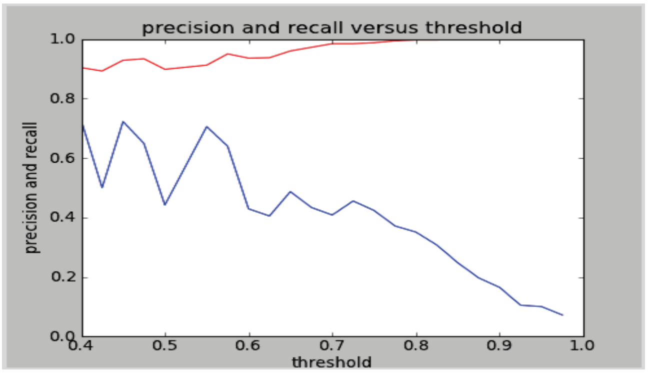
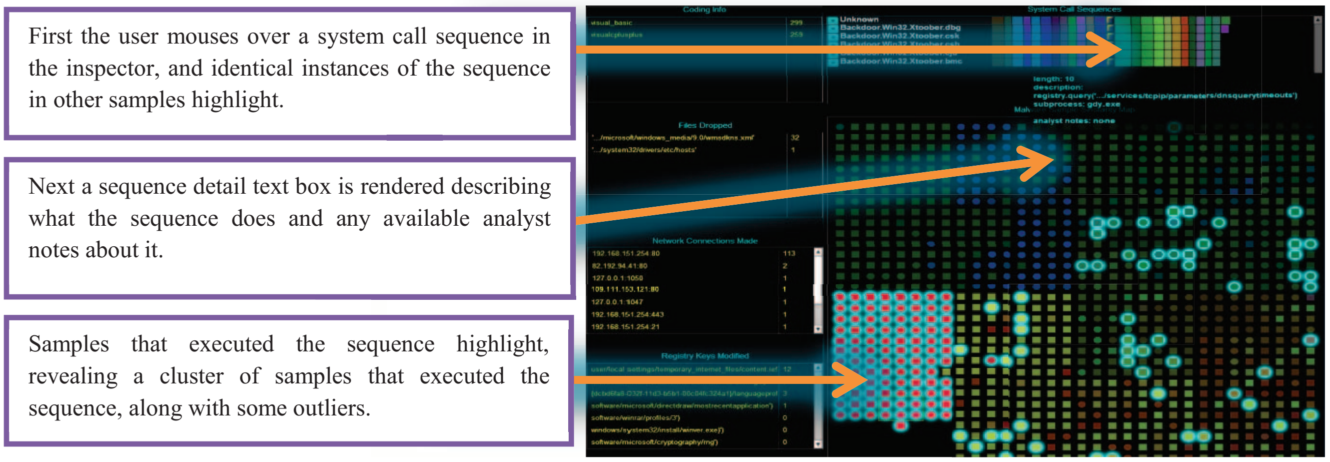


Figure 5. Change in clustering performance as similarity threshold grows. The red line represents precision and the blue line represents recall.


First the user mouses over a system call sequence in the inspector, and identical instances of the sequence in other samples highlight.

Next a sequence detail text box is rendered describing what the sequence does and any available analyst notes about it.

Samples that executed the sequence highlight, revealing a cluster of samples that executed the sequence, along with some outliers.

Figure 6. An example of semantic linking within and between our sequence and similarity map views.

generate, where the weight of each cluster is the number of samples it contains. For more information on precision and recall as measures of clustering accuracy see [4].

We use a graph-theoretic method to cluster the malware sample vectors. This method proceeds as follows. First we define some similarity threshold *t* and then construct a graph from our similarity matrix such that edges reflect similarities that pass the threshold. Then we identify, in greedy fashion, a maximal sub-clique in the graph, remove it from the graph, and repeat, until the graph is entirely decomposed into maximal sub-clique clusters. This algorithm is motivated by the fact that our maximal clique clusters all share edges representing similarities above our threshold *t*. Thus each node is guaranteed to share at least *t* percentage of its sequence set with the sequence set of every other node in its cluster.

Figure 5 shows how our precision and recall scores change as we increase the similarity threshold on our clustering algorithm. At a similarity threshold of 0.8 we can perfectly match the pairwise groupings of samples implied by Kaspersky's malware classifications, but for many clusters we fail to retrieve all of the relevant class. On the other hand, at a threshold of 0.55, we can retrieve most of the relevant samples in each cluster, and we can classify most samples correctly. These results demonstrate that our semantic subsequence features are effective at differentiating between malware of multiple classes with high accuracy.

3.3 Algorithm Run Times

Our sequence extraction algorithm scales linearly as malware corpus size grows, since it is a sample that runs on a per-sample basis. We found that our algorithm took an average of 0.34 seconds to extract sequences for each sample based in a test corpus of 917 malware samples system call traces. The samples in our test corpus had an average number of 311.67 system calls.

Figure 4 illustrates how our similarity matrix and Principal Components Analysis steps scale as corpus size grows. While our algorithms scale exponentially, we can accommodate malware corpora of thousands or tens of thousands of samples on off-the-shelf hardware within the span of a few hours. In the future, we plan to explore ways of scaling our system up to handle tens or hundreds of thousands of samples.

4. VISUALIZATION

Our interface contains three panels: a sequence inspector, which supports inspection of the actual subsequences that occurred during a malware sample's execution, a code sharing map, which provides a global view of behavioral similarity over a corpus of malware samples, and a set of filters, which allow a user to see how trait relationships distribute over the malware corpus. These three displays are linked, so that, for example, brushing over a sequence in the sequence inspector highlights all of the other occurrences of that sequence as well as all of the samples in the code sharing map that exhibit that sequence (see Figure 6).

4.1 Sample Similarity Map

Our similarity overview visualization is rendered as a two dimensional grid of samples. We compute the layout and coloring of our sample similarity grid by first performing Principal Components Analysis on our sample similarity matrix. We project the first three principal components onto a reduced

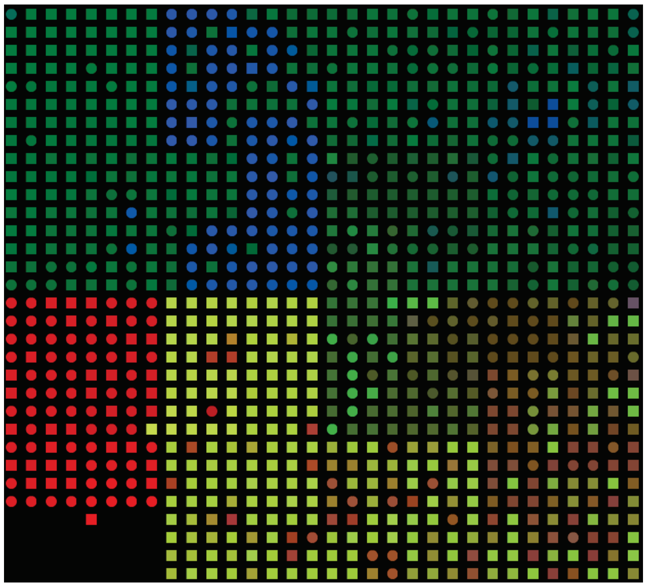


Figure 7. Malware samples arranged in a similarity grid layout with color representing additional spatial dimensions. A number of clusters can be discerned as colored shapes.

dimensional (three-dimensional) space. Then we sort the malware samples by the value each respective sample has by the first principal component, and use that to order them on a space-filling Hilbert curve. Having computed the position of each node, we then color each node using the first three principal components to determine the red, green and blue color values of the color. Thus similar samples appear close together and take on similar colors. Currently, the nodes can be either circles or squares; circles represent "known" samples (samples that we have ground-truth for) whereas squares represent unknown samples (samples that the Kaspersky anti-virus engine could not label).

We chose to use a 2d-grid layout and to use the color channels to encode additional similarity information to deal with two problems that result from more traditional, scatterplot multi-dimensional scaling techniques. First, grid layouts avoid the problem of overlapping nodes. We felt that the visualization would be more effective if users did not have to visually parse overlapping nodes. Second, grid layouts maximize the usage of available pixels on the screen. This additional screen real-estate per node will allow us, in future work, to explore using node shape to encode additional node similarity data.

4.2 Sequence Visualization

Our sequence visualization reveals similarities and differences between malware samples. A depicted in figures 8-11, we render a sample corpus' subsequences as follows. We assign each unique sequence a unique color. Then to render an individual sample, we order its sequences by a global, unique identifier (in this case, the database ID for the sequence), and render them as sequential, colored blocks, with their widths scaled as the natural logarithm of their length. This logarithmic scaling of each sequence allows us to use screen real estate more efficiently.

Figure 6 depicts our approach to interactivity with these visual representations: brushing a sequence in one malware sample highlights identical sequences occurring in the same and other samples, and also causes a detail panel to render, which offers descriptive information about that sequence.

We chose to render our sequence visualization as a set of each sample's sequences ordered by unique identifier so as to support visual comparison of the sequences that appear in the execution trace of each sample. Figure 8 illustrates the way in which such an ordering supports such comparisons. Unfortunately, rendering the data in this way does not reveal cases in which sequences are repeated, and does not reveal cases in which they appear in different orders depending on the sample. In future work we hope to explore ways to address this issue.

4.3 Filter Panels

On the left of the display we render the names of behavioral traits we have extracted from the malware corpus. Currently, we support trait types such as registry key modifications, network communications, and file drops. By brushing an action-type the malware samples that exhibit that action highlight. This linking between filters and the code sharing map supports user insights into the nature of the various regions of the map, as well as insights into the similarities and differences between regions. Figure 12 illustrates this feature.

5. USE CASES

5.1 Visually Discovering and Investigating Cluster Structure in Malware

Our similarity map visualization reveals cluster structure, or lack thereof, within malware corpora. In Figure 7 a number of malware groupings can clearly be discerned: separate red, yellow, green and blue groupings, for example. On the other hand, the lower-right quadrant of the map appears muddy and diffuse, indicating the lack of clear cluster structure among those malware samples. Because this map is a projection of a much higher-dimensional space onto a few dimensions, there is inevitably some information loss. Allowing the user to highlight samples and inspect the similarities in their sequence sets, which we support, is

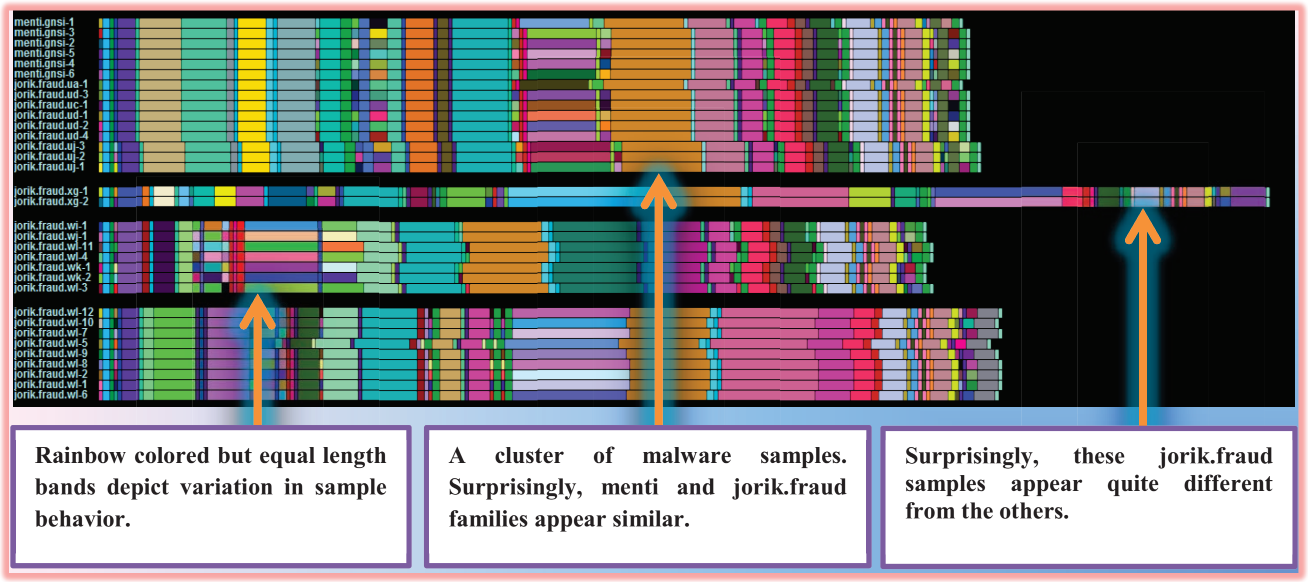


Figure 8. The user brushes an IP address and port on the left, and the samples that connect out to that IP address and port highlight on the right.

crucial to allowing users to fully take advantage of the map.

5.2 Exploring Behavioral Trait Relationships in Relation to a Corpus

The filter panels on the left of the interface allow the user to discover how various behavioral traits distribute over the malware corpus. In Figure 12 the user brushes an IP address and TCP port and the samples that connect out to that address and port highlight; it is clear that the cluster of malware in the lower left of the similarity map is characterized by the fact that it connects out to that address and port. The ability to engage filters and discover such relationships between cluster and feature is helpful in allowing analysts to gain insight into the meaning of the similarity map.

5.3 Relating Unknown Malware Samples to Known Malware Samples

We have found our system to be useful in relating novel samples to known malware samples. Figure 9 demonstrates observation of a group of malware samples that are close together on the sample similarity map. Some of these samples (indicated by their labels, on the left) are unknown, and others are known. What our system reveals is that the unknown samples (which the Kaspersky anti-virus engine was unable to classify) are behaviorally quite similar to the adjacent xtoober samples. To get more information about the samples in the sequence inspector, the user can mouse over the sequences that appear in the sequence inspector at the top of the screen (demonstrated in Figure 6).

5.4 Exploring Sequence-Sharing Relationships Between Malware Samples

Our system provides users with visual insight into which samples and clusters of samples have executed a focal system call sequence. Figure 6 demonstrates the actions our visualization performs when the user selects a group of samples from the sample similarity map, and then brushes over one of the sequences in the sequence inspector. All of the occurrences of that sequence in the sequence inspector highlight, demonstrating that the sequence is shared by all of the samples currently under inspection. Additionally, all of the samples in the corpus highlight in the sample similarity map. In this case, the focal sequence seems to occur mainly in a dense cluster of otherwise highly similar samples.

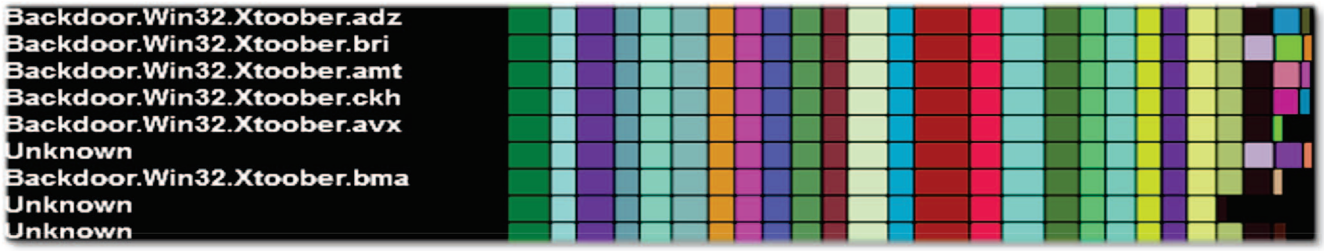

Backdoor.Win32.Xtoober.adz
Backdoor.Win32.Xtoober.bri
Backdoor.Win32.Xtoober.amt
Backdoor.Win32.Xtoober.ckh
Backdoor.Win32.Xtoober.avx
Unknown
Backdoor.Win32.Xtoober.bma
Unknown
Unknown

Figure 9. Visually clustering unknown malware samples with known malware samples. The unknown samples appear almost identical to the labeled samples in terms of their behavioral sequences.

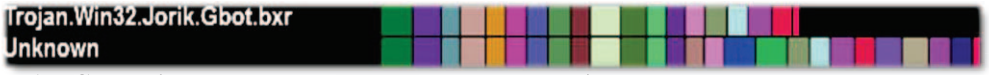


Figure 10. Comparing unknown and known samples. There is clearly overlap between the two samples.

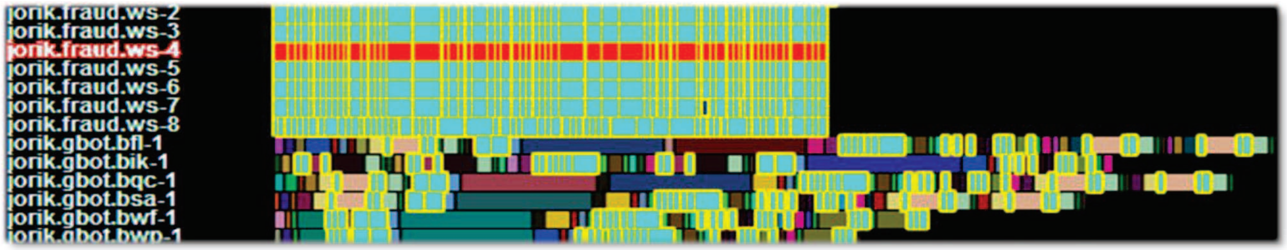
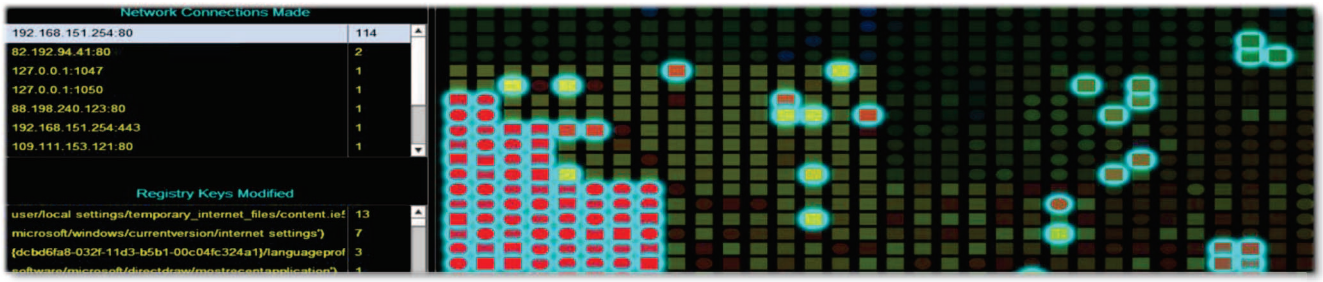


Figure 11. Comparing one sample with many. The user brushes jorik.fraud.ws-4 and it is highlighted in red. As a result, all of the sequences that other samples have in common with jorik.fraud.ws-4 are highlighted.



Network Connections Made

192.168.151.254:80	114
82.192.94.41:80	2
127.0.0.1:1047	1
127.0.0.1:1050	1
88.198.240.123:80	1
192.168.151.254:443	1
109.111.153.121:80	1

Registry Keys Modified

user/local settings/temporary_internet_files/content.ie5	13
microsoft/windows/currentversion/internet settings')	7
{dcbd6fa8-032f-11d3-b5b1-00c04fc324a1}/languageprof	3
software/microsoft/directdraw/mostrecentapplication')	1

Figure 12. The user brushes an IP address and port on the left, and the samples that connect out to that IP address and port highlight on the right.

5.5 Supporting Analysis Inheritance and Reuse

When the user brushes over a sequence, a tooltip comes up with a label describing at least some of what the system calls in that sequence do. There is also a field for analyst notes. Currently this is unused, but in future work this could support analyst annotation of sequences such that when new samples appear that contain these sequences analysts of these new samples could see that these sequences had already been “analyzed,” and could benefit from this previous analysis work.

6. RELATED WORK

To the best of our knowledge, our work is the first to explore interactive visualization of large malware corpora. In the machine learning area, Storlie, et al. have explored using Markovian stochastic adjacency matrices derived from malware dynamic execution traces [8]. Our work, on the other hand, focuses on partitioning system call sequences with Markovian methods.

In related visualization work, Conti et al. introduced visualization techniques to support analyst comprehension of the structure of binary files [5]. Conti’s work is premised on exposing a lower level of abstraction than ours, which focuses on system call sequences. Trinius et al. have explored applying treemaps and thread graphs to comparative analysis of malware [6]. Whereas they focus on the relative quantities of system call categories executed by malware under dynamic analysis, we focus on shared system calls sequence relationships. And whereas they focus on supporting exploration of the similarities between small groups of malware samples, we focus on an interface that provides an overview of similarity between large numbers of malware samples that then allows a user to drill into a selected group of samples and compare them.

Quist et al. have created a system for interactively visualizing malware execution traces and execution trace metadata at the CPU instruction level [7]. Whereas this work focuses on supporting reverse engineering work on individual samples at the assembly language level, our work supports reverse engineering work across many samples at the system call level.

7. FUTURE WORK

To extend the work we have done we plan to evaluate a number of our methods and improve them where possible. Currently we are using a Hilbert curve to lay out our sample similarity grid, but this may not be the optimal grid layout for our data, and we plan to explore the existing literature on multi-dimensional scaling and grid layouts. We would also like to explore incorporating multiple similarity metrics into our system, such that a user could, for example, toggle between sequence similarity, call graph edit distance similarity, and printable strings similarity. Finally, we plan to take our system from prototype to a state that is more robust and capable of handling an extended series of use cases.

8. ACKNOWLEDGMENTS

We would like to acknowledge the Defense Advanced Research Projects Agency (DARPA) for providing the funding that made this work possible.

9. REFERENCES

[1] A.-T. Institute, "AV-Test Statistics Report," 2012. [Online]. Available: http://www.av-test.org/en/statistics/malware/.

[2] "QEMU," 1 June 2012. [Online]. Available: http://wiki.qemu.org/Main_Page. [Accessed 1 June 2012].

[3] M. Russinovich and B. Cogswell, "Microsoft TechNet," Microsoft, 1 6 2012. [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx. [Accessed 1 6 2012].

[4] P. R. H. S. Christopher D. Manning, 2008. *Introduction to Information Retrieval*, Cambridge University Press, 2008.

[5] G. Conti, E. Dean, M. Sinda, B. Sangster and J. Goodall, 2008. Visual Reverse Engineering of Binary and Data Files. *Visualization for Computer Security*, Springer Berlin / Heidelberg, 2008, pp. 1-17.

[6] P. H. Trinius and J. a. F. T. Gobel, 2009. Visual analysis of malware behavior using treemaps and thread graphs. In *Proceedings of the International Workshop on Visualization for Cyber Security (VizSec 2009)*.

[7] D. Quist, 2009. Visualizing compiled executables for malware analysis. In *Proceedings of the International Workshop on Visualization for Cyber Security (VizSec 2009)*.

[8] C. Storlie, S. V. Weil, D. Quist, B. Anderson, 2012. Stochastic Identification and Clustering of Malware with Dynamic Traces. *Malware Technical Exchange Meeting 2012*.