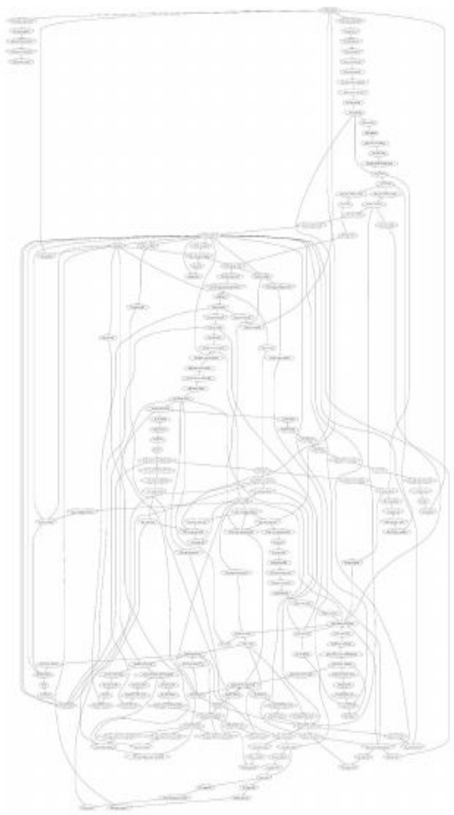
FusionProbe: A Look Into The Utility Of Fusing Static & Dynamic Analysis

Rodrigo Branco, Imani Palmer, Rob Adams and Edmond Rogers
Information Trust Institute

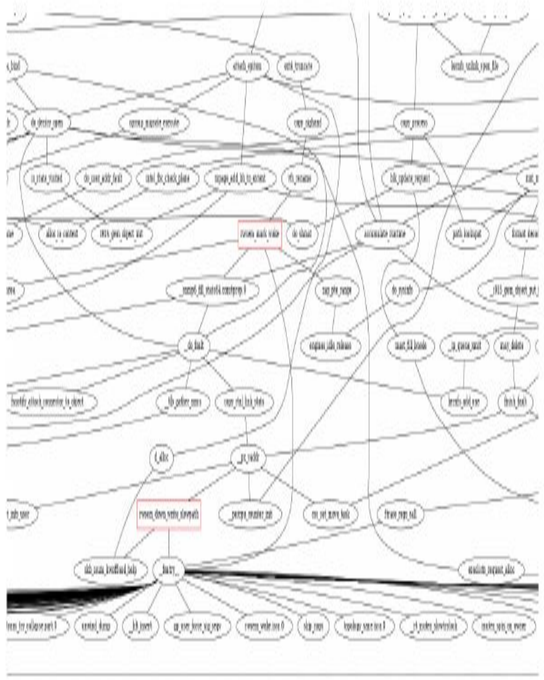
Introduction

There is no way to prevent all the bugs and/or vulnerabilities. Many researchers rely on static analysis, while its fundamental limits restrict it to yield only approximations of program behavior. To counter this, dynamic analysis ascertains concrete information regarding the layout and position of the data in the memory of the running program. Our goal is to provide further insight by fusing both analyses. We rely on an open-source tool called **Memorizer [1]** to obtain dynamic data access tracing. We rely on **Ghidra** to obtain static system call data. We then combine the two datasets. A representation of this combined dataset is shown below.



The Enhancement of Reverse Engineering Toolkits

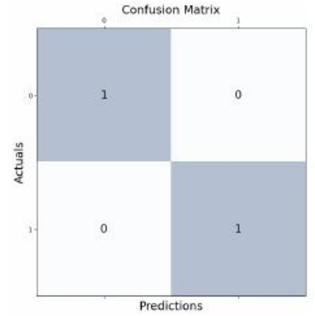
FusionProbe is a Ghidra plugin that obtains on the automated static analysis performed by Ghidra, and retrieves a static call graph, as well as, the dynamic analysis performed by Memorizer [1], and builds a dynamic call graph. These graphs are then connected to determine where intersections exist. The goal of this is to provide insight on how software actually runs. An initial look into this hybrid approach is shown below.



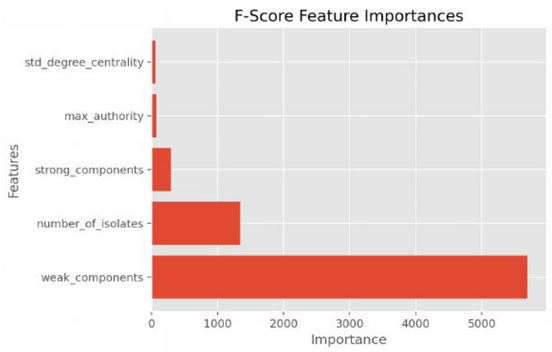
The red rectangle boxes are shown to be the two main points of intersection between the static analysis graph and the dynamic analysis graph. Here we can see that both rwsem_marke_wake and rswsem_down_write_slowpath are connectors.

The Classification of Anomalous Computer Software & Software Chains

These system call datasets are really the first of its kind in terms of data provided about computer software from the Linux kernel [1]. We know we can differentiate computer software with this low-level data. Our initial study was conducted on a proof-of-concept dataset performing binary classification. The input dataset was built on CAPMAPs of PoCs for real CVEs (CVE-2016-7042, CVE-2014-0196, CVE_2015-3290, CVE-2016-100044) and traditional Linux commands (ar, cp, cpio). The feature set was derived from graph metrics and then classified with a decision tree classifier. The initial resulting accuracy is 1.0, precision is 1.0, recall is 1.0 and F1-score is 1.0. The resulting confusion matrix is shown below.



The best features based on the F-score are weak components, number of isolates, strong components, max authority and standard degree centrality.



Conclusions

While this initial exploration is not conclusive evidence. It is an initial promise of the ability to perform classification on this dataset. Our goal is to continue this work and dive even further into graph-related features. The current feature set solely looks at each computer software as a graph. Here we use binary classification to label computer software as bad or good. However, we can extend this towards anomalous files, functions, call chains, and software versions. The current works most similar to this work are in 3 areas: malware detection, classifying unknown malware into families, and auto-extract program or protocol specifications.

While classification and clustering are our current methodologies being used, we do foresee the possible need for Graph Neural Networks (GNN) and its variants. The previous work of GNNs and its variants on classifying malware files show strong evidence that using vectorization can improve feature engineering in malware analysis, resulting in improved classification performance. Vectorization allows us to take our graphs and transform them into d-dimensional vectors of real numbers. With Node2vec we can learn a representation that organizes nodes based on their network roles and/or communities they belong to. In our case, we can understand what function and file play a certain role in computer software. While the benefits of using vectorization in the security space are still in their infancy, we have some promising examples.

REFERENCES

[1] Nick Roessler, Yi Chien, Lucas Atayde, Peiru Yang, Imani Palmer, Lily Gray, and Nathan Dautenhahn. 2021. Lossless instruction-to-object memory tracing in the Linux kernel. In Proceedings of the 14th ACM International Conference on Systems and Storage (SYSTOR '21). Association for Computing Machinery, New York, NY, USA, Article 2, 1–12. https://doi.org/10.1145/3456727.3463767