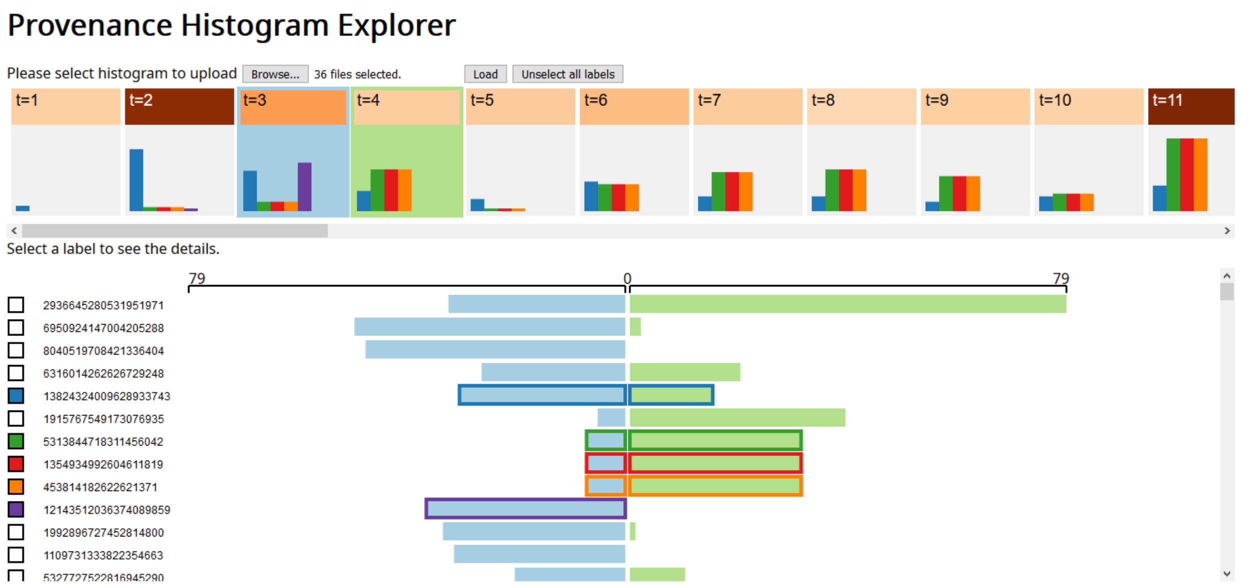
Provenance Histogram Explorer

Michael Kim, Junfeng Xu



Abstract—We present *Provenance Histogram Explorer*, a web based visualization solution for exploring and analyzing provenance histograms generated by UNICORN, an advanced persistent threat (APT) detector. While UNICORN is effective in detecting malicious activities by analyzing provenance log by processing it into efficient yet expressive provenance histograms, it also poses a challenge to security researchers who wish to understand and study the said malicious activities, because UNICORN does not represent the generated histogram in an intuitive and human-friendly way. We tackle this problem by designing and implementing a visualization solution that takes in UNICORN's histogram representations, and displays the histogram in an approachable manner, that facilitates the analysis if malicious activities as reflected in the histograms by researchers, by making it easier to identify anomalies, to study the trend in the program activities, and to understand the histogram in general.

1 Introduction

Data provenance is the metadata that records the history of the creation and operations performed on a data object. This gives people a clear picture of how a piece of data came to exist in its current state. The application of provenance benefits a variety of disparate communities including scientific data processing, cloud computing, databases, software development, and storage [6]. For instance, provenance from machine learning programs can potentially provide users valuable information about how training data influences the decisions made by the program. The management of data provenance has been extensively studied. There are many systems designed to support the analysis of provenance data. [2, 1, 8, 7].

In this project, we focus on provenance data that describes the activities of operating systems. Such data is useful for many security-related purposes, including the detection of system intrusions, as well as the analysis of malicious programs. Namely, we deal with the provenance data collected by the CamFlow framework [8] and processed by UNICORN [4], a novel security application.

Since UNICORN processes provenance data into a textual representation, it is challenging for human researchers to read and understand the process data, especially considering the quantity and signal-to-noise ratio in the said data. In this paper, we propose a solution for visualizing a provenance distribution. The solution, Provenance Histogram Explorer, operates mainly in three phases:

1. Collection of provenance data
2. Preprocessing of provenance data into histograms using a modified version of UNICORN
3. Visualization of provenance histograms

Results from our implementation demonstrate that our solution provides visualization features to assist provenance researchers.

2 Background

We will now give an overview of the structure and meaning of the provenance data we are working with. We will first introduce the definition and format of the system provenance data, and then explain how this data is processed by UNICORN into histograms.

2.1 PROV-DM

Provenance data represents causality between entities in the system. The provenance, in a directed acyclic graph (DAG) form, enables users to audit the history of system entities such as files, sockets by traversing the graph.

In our solution, we use the core structures of PROV-DM to model provenance data [9]. PROV-DM is defined by World Wide Web consortium for client-side manipulation of provenance. The model consists of the following key elements:

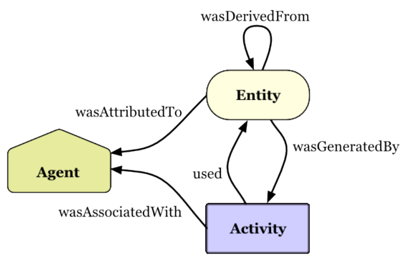


Fig. 1: PROV Core Structure [9]

- *Entity:* An entity can be considered as a generic representation of different types of data objects, such as packets, files, applications, etc.
- *Activity:* An activity represents actions over a period of time performed upon *entities*, such as sending a packet or modifying a file.
- *Agent:* An agent can be a user or a machine that is responsible for *activity* acting on an *entity*. For example, in the system domain, a Linux machine (agent) instructs sending (activity) a packet (entity) via a socket (entity).

A provenance record (PR) expresses the process of generating entities with activities by involved agents. According to PROV-DM, a provenance record includes all the key elements described above along with the relations (e.g., usage, generation, derivation, etc.) between them, an example showed in Figure 1. Depending on real use cases, the attributes of provenance records can vary.

2.2 CamFlow

To collect provenance data from the operating system in the form of PROV-DM, we select CamFlow as our provenance collecting framework [8]. The framework, which has been actively maintained by research labs including the University of Bristol, captures the provenance using a built-in framework in the Linux kernel; LSM (Linux Security Module), and NetFilter. LSM is a framework to implement mandatory access control (MAC), and it hooks every access to internal kernel objects [3]. The NetFilter framework is also a set of hooks to internal kernel objects; such as packet for network operations.

CamFlow uses a subset of types defined by the PROV-DM model: Nodes have three types "agent", "activity", "entity" and edges have five types "used", "wasGeneratedBy", "wasInformedBy", "wasDerivedFrom", "wasAssociatedWith". We relate two nodes using an edge. The following lines, simplified version of CamFlow output represents that an agent and activity are related by "wasAssociatedWith" edge.

```
<wasAssociatedWith>
cf:AQAAAAAABIACAAAAAAAAAAAgAAAA (1)
{
      ``prov:type'':``ran_on'', (2)
      ``prov:agent'':``cf:IAAAAAAAABQUxzJBtiHPfgg'', (3)
      ``prov:activity'':``cf:AQAAAAAAAEBa9wEAAAAAAAg'' (4)
 }
```

Listing 1: CamFlow Log Example

Every JSON component in CamFlow has two namespaces "cf:" for CamFlow defined attributes, "prov:" for W3C PROV-DM defined attributes. The "wasAssociatedWith" edge has an ID (1), two Unique ID references (3) (4) to relate the agent and activity; "prov:agent", "prov:activity" [**CamFlow**]. The "prov:type":"ran_on" (4) represents the relationship that the activity (task) was running on the agent (machine).

2.3 UNICORN

UNICORN is an "anomaly-based" advanced persistent threat (APT) detector that "leverages data provenance analysis" [4]. In an APT, the adversary's goal is to gain control of a specific (network of) system(s) while remaining undetected for an extended period of time. Given the slow-and-low nature of APTs, analyzing individual system call logs to detect point-wise outliers is often fruitless. Provenance connects causally-related events in the graph even when those events are separated by a long time period, thus, the framework use provenance as a data source for APT detection.

2.3.1 Labeling vertex using provenance fields of CamFlow

Firstly, UNICORN takes provenance graph from a provenance collecting framework such as CamFlow as an input. Then it hashes each edge in graph with their given node/edge information. For instance, The following line is one of the edge examples.

```
0 (1) 1 (2) 108147482735 (3) 75477540870 (4) 134094297061 (5)
```

Listing 2: UNICORN Label Example

"0" (1) represents an id of source node, and "1" (2) for a destination node. "10814748273557" (3) is a source node type which is the result of hash function of provenance fields. "7547754087012" (4) is the destination hashed node type and "13409429706136" (5) is the hashed edge type.

Table 1: provenance field examples

Field	Description	Cardinality
prov:type	Provenance Type	96 (Edge), 27 (Node)
cf:secctx	Linux Security Context	11
cf:mode	inode mode	5

UNICORN use above 3 fields, which are a subset of provenance fields, to produce a edge and node type. A cardinality of node in our case, for instance, is 1,485. (Cardinality is a parameter, we can add or delete with 27 provenance fields)

2.3.2 Builds at runtime an in-memory histogram

UNICORN consumes the edge information to build an in-memory histogram. It labels each vertex in graph using aforementioned edge information. For instance, if one vertex has an incoming edge then it uses destination node type as its label and vice versa. Then UNICORN iterates this process for every vertex, and it uses neighborhood information to relabel the node. That same vertex mentioned before, the framework hashes an original label, incoming edge type and neighborhood node type. We can set how many hops we take to search neighborhood information. If hop value, is 3 then, it traverse three consecutive edges to abstract the graph substructure.

```
[58572106876571122]−>1 (1)
[62873799927944020]−>1
[66800991716430997]−>26
```

Listing 3: UNICORN Histogram Example

The above is the example of histogram in a text form that we use for our visualization. "58572106876571122" (1) is the label of a bin in the histogram, and 1 is the size of that bin. Each bin label is the hash value which represents a certain type of graph substructure. And that type of graph substructure, based on UNICORN algorithm, is constructed by abstracting the information of neighborhood information.

3 DATA ABSTRACTION

The data visualized in Provenance Histogram Explorer is a time-series data consisting of many discrete *time frames*, where each time frame is a duration in the time. Each time frame contains a *histogram* that describes the changes made to the provenance graph during the duration of that time frame. In the histogram, each bin corresponds to a certain graph substructure in the provenance graph, and the label of each bin is the hash value generated by UNICORN for this substructure. We consider the labels of the bin meaningless, as we currently do not have the capability to map the labels back to the graph substructure.

The cardinality of the whole data set depends on how long did data collection continue, with no fixed upper limit. There could be thousands of different bin labels for a complex program. The size of each histogram bin in a single time frame may range from less than ten, which is the most common case, to many thousands.

The raw data is collected by us using CamFlow and then processed into histogram using UNICORN in the process. We will describe the workflow in detail in section 6.1. The actual data that our application visualizes is stored in a plain-text, CSV-like format, where each file represents one time frame.

3.1 Example Data

To collect an example data set to be used in the demonstration of our application, we recorded the provenance data generated by running Firefox for a short while. We chose Firefox Because it interacts with a variety of system entities, including processes, files and network sockets. The workload was a minute of opening Firefox, searching on Google, and closing Firefox.

After parsing the CamFlow log into UNICORN form, the number of edges was 3,616.

4 Task Abstraction

We identified three major tasks researchers using UNICORN wish to perform using a visualization solution. These tasks were derived from interviews we had with Professor Margo Seltzer, and Michael Han, a PhD student of Professor Seltzer.

- *Finding and analyzing anomalies that triggered UNICORN:* the final UNICORN output is a binary value that says if there is an intrusion or not. Since it is a machine learning model in the end, it needs to be retrained by human researcher when if the prediction is wrong. However, it iteratively consumes consecutive hash values to build a model, it is not easy to identify the feature that triggered the system for validation purpose. A visualization system would help the researchers in finding the exact reason why UNICORN decides that there is an intrusion.

- *Exploring the 'shape' of the histograms of different programs:* A set of applications which has similar behaviors makes difficult to identify the difference in provenance graphs. Compressor and Ransomware, for instance, they share behaviors like encrypting the file and writing it to disk. Researchers want to see the subtle differences highlighted by visualization channels.

- *Identifying the overall trends in time-series histograms:* Provenance researchers can monitor the overall trend in provenance graph. This feature gives an insight about what kind of graph substructures are prevalent across different time windows. Not only researchers, but system administrator can exploit those distributions to optimize or monitor the system.

Since the UNICORN output is textual, it is not intuitively understandable by researchers.

5 Visualization Solution

Our purposed visualization solution has two views:

A "time axis" on which summaries of the histograms in different time frames can be compared.

A histogram view that shows either the change in substructure distribution in a single time frame, or the comparison of the said distribution between two time frames

5.1 The "Time Axis"

As our system deals with time-series data consisting of many time frames, we believe it is necessary to include a view that summarizes the time frames with regard to its location in the time line. In particular, we wish to somehow encode how "anomalous" each time frame is, since the identification of anomaly is both how UNICORN operates, and what our users are interested in.

The *'time axis'* consists of a sequence of *time frames*. Each time frame is displayed as box with a light grey background and an orange "header", in which the exact time of that time frame is displayed. The time frames are placed on a scrollable horizontal "axis" in chronological order. The brightness of the "header"'s background colour encodes the derived *'importance'* of individual frames. A darker colour means a frame with a higher importance, which could indicate an anomaly in the original provenance graph, which in turn may indicate a malicious action performed by the program. The lower part of the box is the backdrop for the summary of highlighted bins, which we will discuss in section 5.3.

Initially, we considered an alternative solution where the "time axis" would go vertically instead of horizontally, and each time frame would contain a summarized version of the histogram. We eventually decided that this idea is suboptimal for the following reasons:

- It adds visual clutter to the "time axis", which distracts us from identifying "important" time frames.
- It is hard to know *a priori* which bins are the most suitable to be put in the summary.

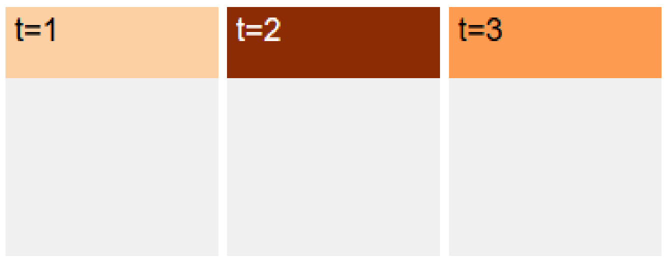


Fig. 3: The first three time frames from the Firefox data. The second frame ($t = 2$) has a very dark colour as this time frame contains an exceptionally high number of changes

Calculation of "Importance"

As previously mentioned, we are interested in identifying anomalous time frames. Therefore, for each time frame, we derive a *'importance'* value that measures how much does this time frame deviates from the rest.

There is no existing measurement for such an "importance" value, so we developed our own algorithm for the calculation of "importance": First, for each histogram bin, we calculate its median size across all time frames, as the "reference" value for each frame. Then, for a single time frame, we calculate the absolute difference between all histogram bin sizes in this time frame and the corresponding reference value. The "importance" of a single time frame would be the sum of all such differences, multiplied by some scaling constant α.

The above can be written as the following formula:

$$I_t = \alpha \sum_{l \in L} |H_t(l) - \text{median}(\{H_{t'}(l') | t' \in [1,n], l' \in L\})| \qquad (1)$$

where

- I_t is the importance of time frame t.
- α is a constant that scales the "importance" value.
- n is the number of time frames.

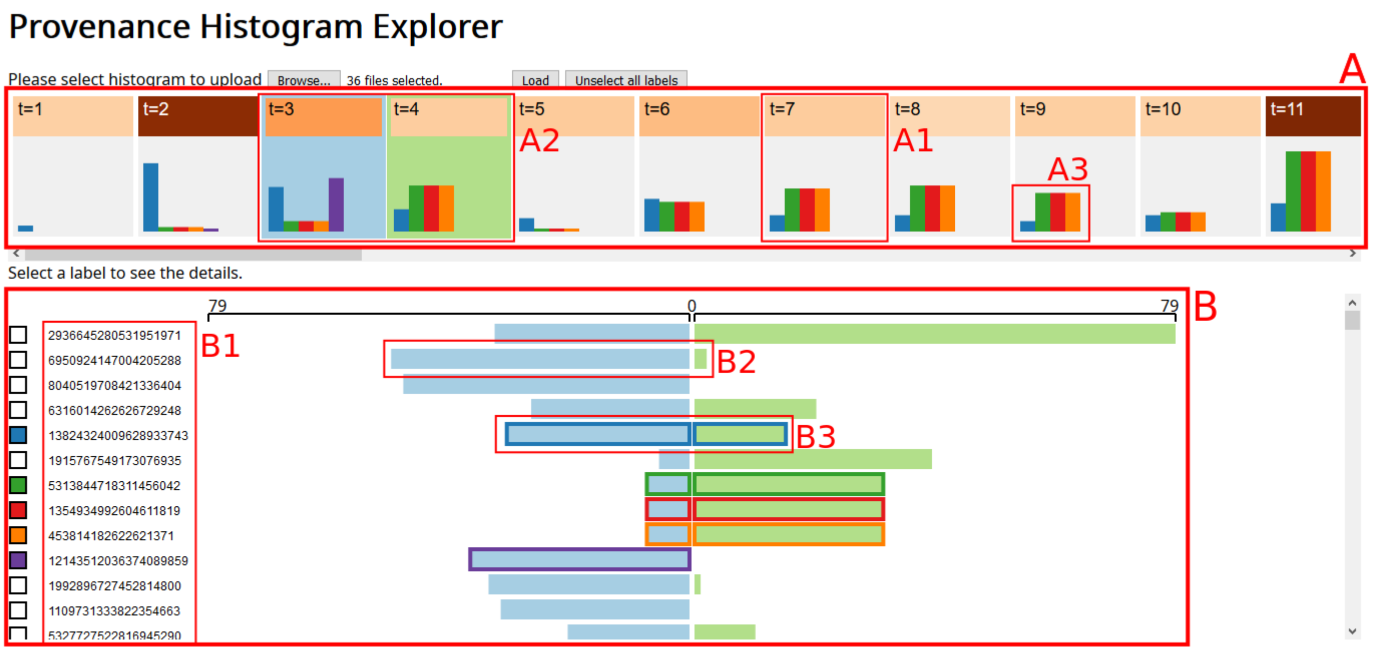


Fig. 2: An annotated overview of the visualization design. A: the “time axis”; A1: a frame on the “time axis”; A2: the two selected frames on the “time axis”; A3: the summary of the highlighted bins; B: the histogram view; B1: the UNICORN hash value associated with each histogram bin; B2: a bin in the histogram; B3: a highlighted bin in the histogram.

- L is the set of hash values assigned to histogram bins.
- $H_t(l)$ is the size of bin l in time frame t.

We multiply the sum of differences by a constant α to control the possible range of derived “importance” values, such that it would not be too large or too small to be used when calculating the colour of the time frame “header”. For the Firefox data set, we set $\alpha = 0.001$. Future developments of our system should look into the possibility of calculating α from the properties of the data set.

Instead of median, we could have used the mean value as reference. However, since we are expecting the presence of anomalous values that deviates from the rest hugely, the mean value would skew upwards. We also considered using total size of all histogram bins at a time frame as the importance (which is essentially using 0 as the reference value), but this would not perform well in cases where, for example, all but the anomalous time frames have large histogram bins, in which case the anomalous time frames would instead be considered not “important”.

Navigation

In addition to giving summary of time frames, the “time axis” also serves as the main navigation control of the application. The user can select a time frame by clicking on the corresponding square on the “time axis”, and the histogram at that time frame will be displayed in the histogram view.

5.2 Histogram View

For each time frame, the user may also be interested in the type of provenance graph substructures that occurred, as well as the number of occurrences, which may indicate the type of actions performed by the program and the number of times the said actions are performed.

The *histogram view* gives a detailed view of the size of histogram bins in selected time frames, represented as an ordinary histograms, where the width of bars encodes the size of corresponding bins, linearly scaled to fit in the size of the histogram.

As mentioned in the previous section, the user can click on a time frame to “select” the time frame. The histogram of the selected time frame will then be displayed in the histogram view. The user can select two different time frames at the same time. In this case, the two histograms to be compared side by side. For the sake of consistency, the histogram belonging to the less recent time frame will always be on the left side, in light blue, while the more recent one will always be on the right, in light green. The background of selected time frames will also change, to correspond to the colour of histogram bins.

The histogram bins in the histogram view are sorted in descending order of total size, that is, the size of this bin in the left histogram plus the size in the right histogram. This is because larger histograms are more likely to be “interesting”, as they are more likely to recur in other time frames. Very large bins may also trigger UNICORN.

We include a scale for the histogram bins on the top of the histogram view, but only label the origin, as well as the bin maximum size in the currently selected time frames, so that the users may gain a rough understanding of the scale of the selected time frames. We believe that for our user, it is more important to perform comparison and to get an understanding of the “shape” of the data, than to get accurate sizes of bins.

5.3 Highlighted Bins

Another important part of the analysis of the histograms is to understand how individual bins in the histogram changes over time, and how they compare to other bins during the course of the change. It would be useful to, for example, to see what is the trend of one individual bin, and identify the association between correlated bins, which could indicate a connection between the program activities that these bins reflect.

The user can *highlight* individual bins by clicking on either the histogram bars, the associated hash value, or the rectangle “checkbox” to the left of the hash value. In the histogram view, selected bins are outlined. The colour of the outline is the colour assigned to the highlighted bin. The highlighted bins will then show up in the “time axis”: each time frame will have a bar whose length encodes the size of the highlighted bin in that time frame.

Each highlighted bin is assigned with a colour. The colour is merely used to associate highlighted bins in the histogram view with bars in the “time axis”. The colours themselves have no meaning, and do not correspond to particular hash values. We put an effort into selecting the suitable colour scheme to ensure that the colour of highlighted bins do not clash with other colours. Eventually, we decided to use a light colour for the histogram bars, as they are big enough to stay

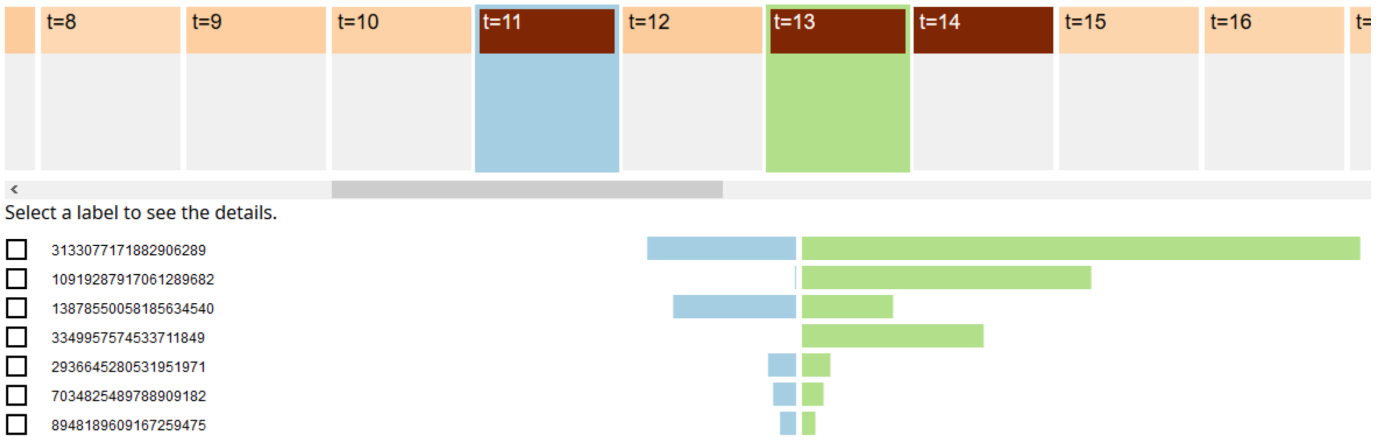


Fig. 4: Two histograms compared against each other.

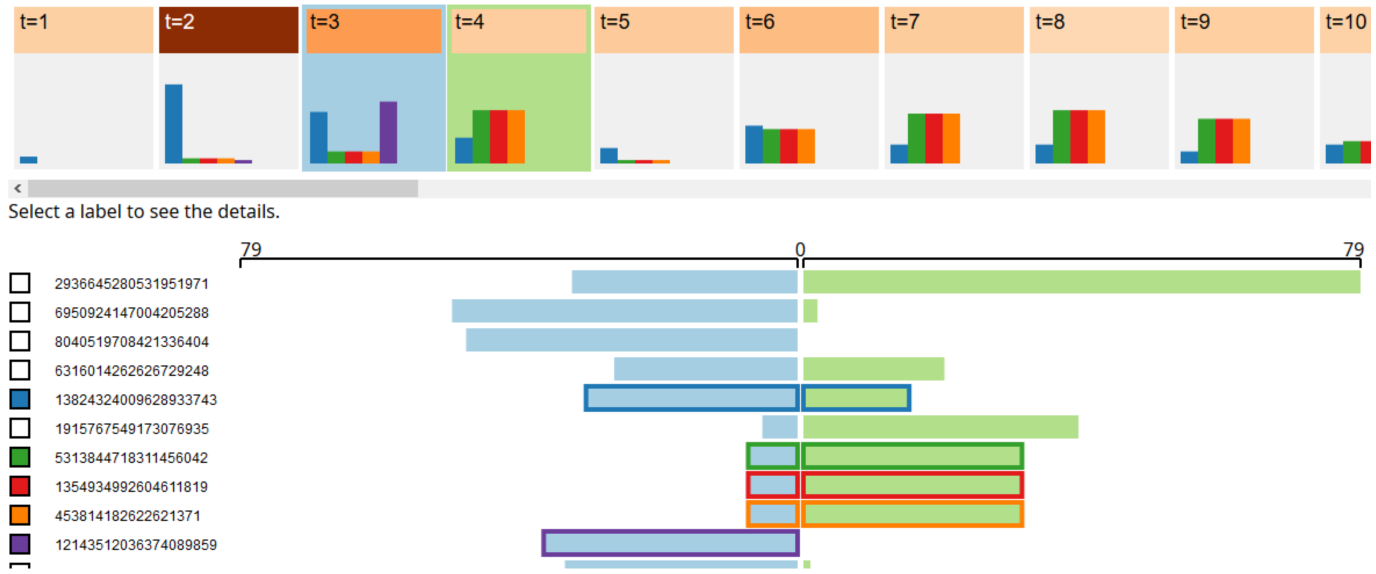


Fig. 5: The Firefox data set with some histogram bins highlighted. A number of patterns and trends can be noticed: the blue bin occurs in all time frames, but its occurrence is especially numerous in the second time frame; the green, red, and orange bins always have the same size; and the purple bin does not occur anywhere other than the third time frame.

distinguishable on a white background even with a light colour, and use the dark version of the same colour scheme as the highlighted colours. In this way, we trivially avoid having indistinguishable colours, and also reduced the total amount of hues in use in our application.

6 Implementation

The technical implementation of this project can be broken down into two components:

The data processing pipeline takes in Camflow log, and outputs time-series histograms which are then used by the application. a modified version of UNICORN that outputs the histograms is used in the process.

The Provenance Histogram Explorer Application is a web application that displays the time-series histograms.

The two components are developed separately. They both function independently without invoking each other.

6.1 Data Collection and Processing

Our application and UNICORN are based on the provenance data collection. We define the process for ease of access to an endpoint; our visualization solution.

Firstly, CamFlow is installed on machine running Fedora. The framework accepts a list of file to track and option for collecting. (e.g. Enable whole system provenance, filter out some specific nodes) Then it produces provenance in the form of binary and the file contains JSON records. Then we copy the Camflow log into the directory of UNICORN.

Secondly, UNICORN translates Camflow log into UNICORN format. Because it hashes a set of selected fields in provenance log, UNICORN and our visualization can be generalized to other provenance framework. Once a list of edges is ready in a form of text file, discussed in detail in Data Abstraction, UNICORN starts to build a in-memory histogram using Graph-chi computation [5]. UNICORN visits every vertex iteratively and relabel each node based on the information of neighbor nodes. With this log, one can use our visualization solution without further manipulation.

We inserted around 10 locations in UNICORN code to log the evolving histograms, saving the internal histograms into plain text files. With this log, one can use our visualization solution without running UNICORN again. Also, future users can map the graph substructure to element of histogram.

6.2 Provenance Histogram Explorer Application

The visualization solution is built as a web application, written in Vanilla JavaScript and plain HTML. It allows the user to upload histogram files in CSV format, where each file corresponds to one time frame, and visualizes the data as described in section 5. The web application is responsible for calculating the importance of time frame. Other than that, the web application does not perform any data processing.

The "time axis" and the histogram view are both rendered as SVG elements on the web page. We used D3.js to handle the binding between SVG DOM elements and the data. `d3-interpolate`, `d3-color`, and `d3-scale-chromatic` was used to handle the colour schemes.

The entire application consists of two files: `vis.html` which is the main web page for the application, and `vis.js` which is the JavaScript that controls the rendering of the visualization views, handles uploads, and deals with navigation. The D3.js and other libraries are imported by loading their code from CDNs. Other than that, the application is entirely self-contained and require no external tools to build or deploy, and can be loaded locally in the browser, making it easy for us to develop, and for future users to set up or integrate into existing systems.

6.3 Contribution Breakdown

The implementation of the data collection and processing pipeline, as well as the collection of example data used to aid development and evaluation, was carried out by Michael Kim.

The web application was written and tested by Junfeng Xu, who also produced all application screenshots included in this report.

7 Result

We will now give an evaluation of the project with regard to the opinion of the future users, and the performance of the application.

7.1 User Interview

We evaluated the application by conducting an interview with Professor Margo Seltzer, a co-author of the UNICORN paper. We loaded the Firefox data set, and gave her a short introduction of the various part of the system, showing the various views, and the highlighting capabilities. We then asked for her opinions about the application, and took notes of her feedback.

She reported that the visualization had successfully made her interested in the data, and wished to "see more". She found the visualization "super interesting", especially because it offers the capability for users to see the "trajectory" of bins over time.

However, she also commented that the example Firefox data set is not ideal. She pointed out that it would be more useful to show a comparison of benign and malicious programs. She also questioned our choice of displaying the change ("delta") of histogram bins, as UNICORN accumulates the size of histogram bins, using a "decay" value to drop older entries in the histograms.

In addition, she suggested that we can define each time frame by the number of provenance graph edges rather than time, which means that each time frame will contain the histogram for, say, one thousand provenance graph edges, instead of all edges that appeared in a one second interval.

7.2 Performance

Initially, we had a concern about the performance of the web application due to the amount of data we were expecting to be displayed. However, by choosing to visualize only the changes ('delta's) of histogram bins and dropping bins that contains only one item, we managed to greatly reduce the number of items to be displayed on the web page.

On a laptop with an Intel Pentium Silver N5000 CPU, which can be considered weak by 2019 standard, the web application runs without issue with the Firefox data set loaded. Navigating and scrolling through the views are smooth. A small but noticeable lag occurs when initially loading the data set, which could be attributed to the calculation of "importance". However, since we do not expect data set to be loaded frequently, we do not consider this to be a major issue in our application.

8 Limitations

In general, due to the limited time available for this project, we had to drop a number of features from our resultant application. This puts some limitations on the use of our system. Nevertheless, we shall discuss in the next section about how these issues can be addressed via future extensions.

Right now, the most significant issue is that the histogram bin labels are not meaningful. They are merely hash values for graph substructures generated by UNICORN. It is impossible to know from these hash values what actual activities were performed. While this does not prevent the users from gaining higher-level understandings of the provenance data of a single program, any attempt to investigate the actual behaviours of the programs will be hindered by the lack of means to interpreted the histogram bins.

Another limitation is that the application is unable to compare the histograms of different programs. This makes certain interesting use cases impossible. For example, a user may be interested in the different between the provenance graph of compression programs and Ransomwares, which cannot be visualized user our current implementation. This also has an impact on our choice of example data.

9 Future Works

We will now list some potential future improvements to Provenance Histogram Explorer. Most of them either address one of the limitations described previously, or implement one of the features that were dropped due to time constraints.

Mapping back to the provenance graph

We originally planned to make it possible to map the histogram bins back to the corresponding graph substructures in the provenance graph. However, we had to drop this feature because of a lack of time. This feature would help tremendously with deeper investigation into the program activities, as the users would then be able to see exactly what program activity does a certain bin in the histogram reflect.

Implementing this feature would require changes to be made to UNICORN, so that each bin label in the output would be associated with the corresponding graph substructure. We also need to consider the visualization solution for such graphs, including its integration into the existing application.

Comparison of multiple histograms

Another extension is to add the capability to load and compare multiple histograms, which would help with comparing malicious and benign programs. While this does not pose any technical difficulties, we believe this would require some significant amount of change to the existing visualization solution, in order to accommodate the increased amount of information on the screen, and to handle the comparison of potentially multiple histograms.

Derivation of different parameters for time frames

While the current algorithm for calculating "importance" is effective in finding time frames that are more likely to be "interesting", we feel that some other measurements, such as the correlation between time frames, may also work. An evaluation of alternative metrics can be a worthwhile future extension.

As mentioned in section 5.1, the α constant in the "importance" formula, is set to $\alpha = 0.001$ for now. This may not work for all data sets, and alternative values, or algorithms to calculate α, may also be developed.

Changing the definition of time frames

As pointed out by Professor Margo Seltzer during the interview, we can choose to define each time frame by the number of provenance graph edges, instead of actual time. Doing so would make histograms more reproducible, as the running time of the programs will no longer have an effect on the histogram generated. This approach also normalizes the size of time frames, making direct comparison more feasible, and reducing the amount of effort required to handle the potentially large difference between the scale of different time frames.

10 Conclusion

We developed *Provenance Histogram Explorer*, a visualization solution for the analysis of provenance histogram data. Our solution is simple and clear, catering for the domain-specific needs of security researchers working with provenance data. The solution succeeds providing sufficient information for users to get a high-level understanding of the provenance data they are analysing. However, there are still room for future improvement, that would enable our users to perform more in-depth analysis using our tool.

References

[1] Sherif Akoush, Ripduman Sohan, and Andy Hopper. "HadoopProv: Towards Provenance as a First Class Citizen in MapReduce". In: *5th Workshop on the Theory and Practice of Provenance, TaPP'13, Lombard, IL, USA, April 2-3, 2013*. Ed. by Alexandra Meliou and Val Tannen. USENIX Association, 2013.

[2] Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya. "DBNotes: a post-it system for relational databases based on provenance". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. Ed. by Fatma Özcan. ACM, 2005, pp. 942–944.

[3] Antony Edwards, Trent Jaeger, and Xiaolan Zhang. "Runtime verification of authorization hook placement for the linux security modules framework". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*. 2002, pp. 225–234.

[4] X. Han et al. "UNICORN: Runtime Provenance-Based Detection for Advanced Persistent Threats". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2020).

[5] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. "GraphChi: Large-Scale Graph Computation on Just a PC". In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX, 2012, pp. 31–46. ISBN: 978-1-931971-96-6. URL: `https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola`.

[6] Kiran-Kumar Muniswamy-Reddy et al. "Provenance-Aware Storage Systems". In: *Proceedings of the 2006 USENIX Annual Technical Conference, Boston, MA, USA, May 30 - June 3, 2006*. Ed. by Atul Adya and Erich M. Nahum. USENIX, 2006, pp. 43–56.

[7] Hyunjung Park, Robert Ikeda, and Jennifer Widom. "RAMP: A System for Capturing and Tracing Provenance in MapReduce Workflows". In: *PVLDB* 4.12 (2011), pp. 1351–1354.

[8] Thomas F. J.-M. Pasquier et al. "Practical whole-system provenance capture". In: *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*. 2017, pp. 405–418.

[9] *PROV-DM Model*. `https://www.w3.org/TR/prov-primer/`. Accessed: 2019-12-10.