Digital Forensics Event Graph Reconstruction

Daniel J. Schelkoph(✉), Gilbert L. Peterson, and James S. Okolica

Air Force Institute of Technology (AFIT), 2950 Hobson Way, Wright-Patterson AFB, OH 45433, USA
{daniel.schelkoph,gilbert.peterson,james.okolica}@afit.edu

Abstract. Ontological data representation and data normalization can provide a structured way to correlate digital artifacts and reduce the amount of data that a forensics investigator needs to process in order to understand the sequence of events that happened on a system. However, ontology processing suffers from large disk consumption and a high computational cost. This paper presents Property Graph Event Reconstruction (PGER), a data normalization and event correlation system that utilizes a native graph database to store event data. This storage method leverages zero index traversals. PGER reduces the processing time of event correlation grammars by up to a factor of 9.9 times over a system that uses a relational database based approach.

Keywords: Graph database · Digital forensics · Property graph Ontology · Event reconstruction

1 Introduction

With society's ever-increasing reliance on technology, the demand for digital forensics has risen significantly. This is noted in the 2017 Bureau of Labor Statistics ten-year job outlook figures for the related fields of Forensic Technicians (up 17%) [2] and Information Security Analysts (up 28%) [3]. Part of this need is driven by the time-consuming task of manual data correlation required for digital forensics investigations [7].

One potential approach for automating data correlation is to leverage an ontology [5]. An ontology creates standardized data structures for events, allowing for relationships between heterogeneous data sources and creates data that is highly connected, representing a graph-like structure. Using expert rules, certain patterns (or subgraphs) can be identified to combine data into higher-level events [7]. Current approaches leverage the Resource Description Framework (RDF) from the Ontological Web Language (OWL) as a storage medium [6]. OWL formatted data is stored as tables in a relational database. However, identifying patterns in a graph using a relational database can be slow; finding an adjacent node in the graph representation of an ontology is an $O(log_2 n)$ problem [17].

The presented Property Graph Event Reconstruction (PGER) tool provides fast correlations by leveraging a property graph database to speed graph traversals. Utilizing the native graph property of zero-index traversal, finding an adjacent node can be accomplished in constant time. This provides a substantial speed increase. PGER has been programmed to abstract forensics events from expert generated inference rules or machine generated rules provided by the Temporal Event Abstraction and Reconstruction (TEAR) tool [16].

In testing, PGER was able to correctly identify 65/67 startup and shutdown events based on an 8 statement expert ruleset. Additionally, PGER automated the correlation of web history and downloaded data (18 statements) to determine the entry point of the file on the system and any activity involving the file on the host system. Finally, PGER was compared to a relational database approach by testing much longer (86 statements) machine generated ruleset that detect the opening of Microsoft Word. In this test, PGER performed 9.98 times faster.

2 Related Work

2.1 Event Reconstruction Techniques

Due to the ever-increasing amounts of data forensic examiners have to analyze, a focus of digital forensics research is to try and create an automated way to consolidate data and reconstruct events [15]. In addition, many forensic investigations are for legal proceedings with strict rules regarding evidence, so it is desirable that event reconstruction be the result of a formal theory [9]. Previous research has tried different methods to establish such a theory: finite state machines [9], machine learning [14], and inference rules [12,19].

Finite State Machines. Finite state machines provide a mathematical foundation for forensic events, providing rigor to findings [9]. Finite state machines are constructed by working backwards from the final state and using expert knowledge to make transitions and states, eventually ending with the events that need to happen before the final state can be accomplished [13]. Unfortunately, the number of possible variations from each final state produce very large finite state machines and are hard to create by experts [6]. At times, the finite state machines may be shrunk by previous events or other evidential information [13]; however, this is currently a manual task and limits the usefulness of this process [6].

Machine Learning. Expert-created event patterns can be complex and time-consuming to create. Some researchers have tried using machine learning to automatically find patterns in data. In 2006, researchers proposed a neural network that found the execution of various sequences in Internet Explorer [14]. This neural network was able to reconstruct events with an accuracy of 90%. However, neural network techniques do not show how low-level actions are associated with other events to infer a high-level activity [6]. Since examiners were not able to explain why the neural network created a certain series of events, this technique is hard to use in evidential situations [7].

Temporal Event Abstraction and Reconstruction (TEAR) is another attempt at using machine learning [16]. Unlike neural networks, this method of pattern matching allows humans to confirm the identified patterns and trace the high-level events to the individual low-level events. Its algorithms create a hierarchy of events using pattern matching in order to represent a high-level event (See Fig. 1) [16]. At the lowest level are terms which represent an atomic event such as a registry key modification. Each term has an action, a type, and a regular expression to determine what events get a specific term label. A term representing a created file in a user's document directory would have an action of 'Created', a 'file' type, and a regular expression of '^.*/Documents/.*$'. The next level on the hierarchy is strings. Strings are composed of other strings and/or terms. Next, production rules consist of both terms and strings, and provide a successful pattern for a high-level event. This helps represent multiple paths to the same high-level event. If a production rule with no parent finds a match in the data, then the high-level event occurred.

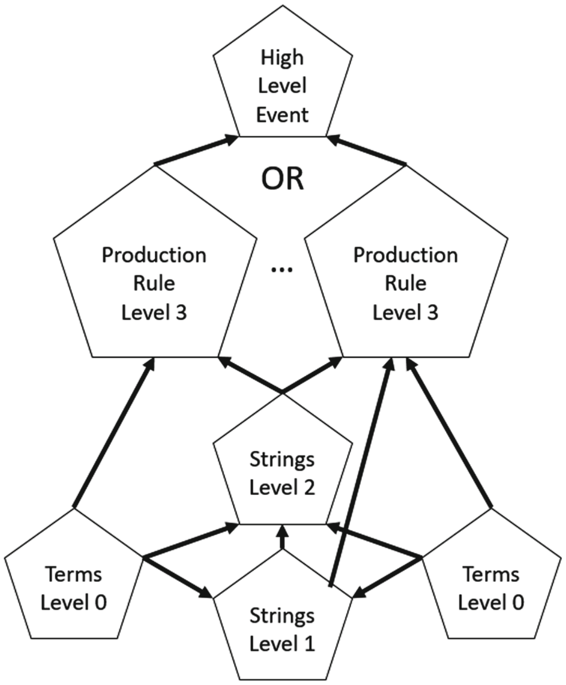

Fig. 1. Temporal event abstraction & reconstruction hierarchy.

2.2 Inference Rules

Inference rules allow forensic tools to reconstruct events by applying rules to existing information from the device image. This method performs similarly to Expert Systems; some tools even use an Expert System directly [19]. The rules are used to find patterns in existing data and, if a rule matches a pattern, a reconstructed event is created. For instance, an inference rule could look for events that contributed to the insertion of a USB device [12]. If the rule triggers on the appropriate registry entries, it could create a new reconstructed event stating that the USB device was inserted. This technique requires well-structured ontological data because the applied rules rely on the semantics of the data [6]. In research, the FORE tool uses the expert system in the Jena Apache Framework [19]. The Semantic Analysis of Digital Forensic Cases (SADFC) uses the ORD2I ontology and queries on the dataset as its rule set. If a query matches a set of data, they are combined to form a reconstructed event [6]. A tool called Parallax Forensics (ParFor) uses its own tool to query data and functions similarly to SADFC [20]. PGER utilizes inference rules for its expertly generated ruleset.

2.3 Graph Databases

A graph database allows the user to interact with data as nodes and edges. This is much different from a traditional relational database that represents data as tables. Using a graph representation can result in performance benefits for connected data, but it is highly dependent on how the graph database model is constructed [17]. Each graph database model is on two spectrums: the data format and the processing method [17].

Although all graph databases represent a graph, each graph model has a data format on a spectrum between non-native and native [17]. Non-native storage converts nodes and edges to relational database tables or another format (e.g. document-based). This can be useful if the database is large, allowing for a straightforward way to shard the data over multiple servers [17]. For native storage, the graph is the storage mechanism. This can provide performance benefits for certain queries, as the database does not have to construct the graph before processing [17].

Graph databases are also distinguished by their processing method on the spectrum between non-native and native. This is how the graph handles all Create, Read, Update, Delete (CRUD) operations [17]. Non-native processing does not use a graph to conduct an operation. Instead, it deals directly with how the data is stored. This style can utilize performance benefits like indexes from relational or document databases [17]. Native processing utilizes a graph to perform CRUD operations, providing performance benefits unique to graph databases [18]. However, not all query operations are faster in a native graph database. Care must be taken to ensure that queries take advantage of the performance benefits.

One of the biggest performance benefits for native graph processing is the ability to perform an index free traversal [18]. In highly connected data, it is often useful to examine the relationships between different nodes. To determine if two pieces of data share a relationship, a traditional database (non-native processing) would need to perform a join on multiple tables. This would require at least two index-based searches with a runtime of $O(\log_2 n)$ [18]. To determine more complex relationships, like finding if a particular walk on a graph exists, the queries become even more complex and time-consuming. Native graph processing allows a search on related data (data incident to another piece of data) in constant time, otherwise known as an index free traversal [18]. Searching a graph for a particular walk (subgraph) can be much faster by processing in a native format.

Most ontological digital forensics research uses RDF and OWL to store event data. Forensic of Rich Events (FORE) [19], Digital Forensics Analysis Expression (DFAX) [5], and Ontology for the Representation of Digital Incidents and Investigations (ORD2I) [7] are all stored using these standards. OWL is typically stored as a non-native storage model; the previous examples use relational database tables to store all triples. This means that these databases do not have the ability for index-free traversal, making queries regarding graph structure time consuming [18].

neo4j is a labeled property graph that is stored in a native graph format [8]. One major difference between this model and RDF is the ability to store data in nodes, allowing for more compact graphs in certain instances [1]. neo4j also allows for index-free traversal of the graph, allowing queries to take advantage of relationships between nodes for rapid queries [18]. In fact, the main use of labeled property graphs is for rapid transactions [17].

3 Property Graph Event Reconstruction (PGER)

Property Graph Event Reconstruction (PGER) performs abstraction of user actions from digital media. It utilizes a labeled property graph to store ontological event data, leveraging the speed of index free traversals. Events are represented in terms of artifacts and relationships, making queries easier to understand and create. It also allows users to leverage quick path searching to quickly find subgraph patterns in event data. PGER is a combination of several tools, some were created specifically for PGER and others were re-purposed for PGER as shown in Table 1.

PGER accomplishes event reconstruction in four processing layers (Fig. 2). The first layer takes a device image and extracts events. The second layer converts the extracted events into ontological subgraphs stored in neo4j. The normalization layer ensures identical objects are represented by the same object. Finally, the abstraction layer uses either expert rules or a machine generated ruleset to extract higher-level events.

Each layer is designed as an atomic entity, allowing for independent operation. This design isolates versions of a database during testing or multiple cases so they can be processed in a pipeline-like fashion. Each forensics image is represented by a separate neo4j database and is run in a separate instance of a docker container. The next sections discuss the details of each processing layer.

Table 1. PGER Tool Origins.

PGER Step	Tool Name	Existing Tool	Created
Data Extraction	PLASO	X	
	TEAR Event Extractor	X	
Graph Conversion	Logstash	X	
	Logstash Parsers		X
	Python Script		X
Normalization	Normalizer		X
Abstraction	Expert Rules		X
	Application of TEAR Ruleset		X
	TEAR Ruleset	X	

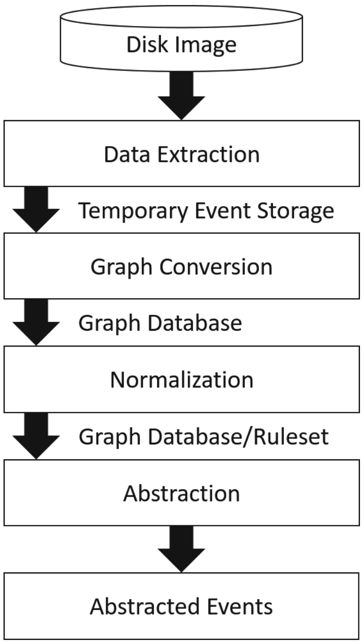

Fig. 2. PGER processing layers.

3.1 Data Extraction

The first processing layer takes a device image and converts it into an intermediate format. This format can then be converted to a graph database in another processing layer. There are two ways PGER can create this intermediate format: PLASO and TEAR event extraction.

PLASO (Plaso Langar Aő Safna Öllu) converts a device image into a super-timeline and outputs the resulting data to an elastic database [11]. The other method of data extraction is utilizing the TEAR event extraction. It is a C++ program that takes the device image and creates a series of event files. File table, registry, and Windows events are among the data sources that are stored as CSV files. A Sqlite file for each browser is also extracted for web history.

3.2 Graph Conversion

Different event artifacts, such as registry keys or prefetch information, contain unique details that provide additional insight on the forensics image. Examples include the values of a registry key or the time an executable was run. The processing steps required are specific for each type of artifact.

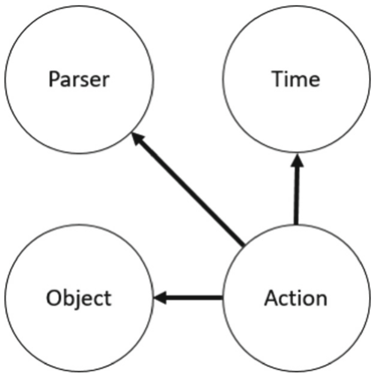

Fig. 3. Core subgraph.

The main purpose of the graph conversion processing layer is to convert heterogeneous event artifacts into a semantic, graph-based format. The base of the semantic format is the core subgraph (Fig. 3). This is based on the standard format found in SOSLA [16]. Every artifact is converted into a core subgraph and contains an action, an object, a parser, and a time. The time is a Unix timestamp and is unique in the database. This represents the time that an action occurred. Multiple artifacts that occur simultaneously are linked to the same time node. The action contains a description of an action that affects a digital object. The object is the digital object that is affected by the action and contains an identifying name of an object such as a URL, registry key, or file path. The identifying name is unique in the database. Since objects are unique, they are also indexed by neo4j, providing speed improvements during queries. If there are different objects with the same name, such as registry keys in user hives, the username is appended to the beginning of the identifying name. Each artifact subgraph is combined with the existing subgraphs in the neo4j database.

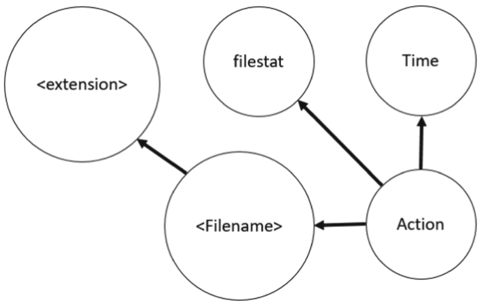

Fig. 4. File table subgraph.

File Table. The File Table parser produces at least one subgraph for each item in a system's file table. This subgraph adds an additional node to the core subgraph that represents the extension of the file object. The extension node is unique in the database. To help reduce congestion in the database, file table times (created, changed, modified, accessed) are combined into the same action if they occurred at the same time. The action description lists all the times that have changed during that timestamp ('Modified, Created Time Altered') and a new field is added to the action where each changed time type is an item in a list ("['Modified', 'Created']").

Web History. Each web browser has its own parser due to differences in history recording. In general, a parser records three different events: history, downloads, and keyword searches. Figure 5 details subgraph examples. History events add a visit ID to the core subgraph. Visit IDs link to other visit IDs to indicate a sequence of events in a browser. Downloaded events show the location of the downloaded object and the URL source. Keyword search subgraphs add a field in the action node that indicates the words used in a search.

Registry Keys. All parsed registry keys that are unique for each user (user-class.dat and ntuser.dat files) contain the core subgraph with the key value as the object and a node with an edge to the key indicating the owner. All other registry keys omit the user node in their subgraph. The action node contains a field that holds the value of the registry key. Registry keys that provide more information have an expanded subgraph.

Recent Apps (`SOFTWARE/Microsoft/Windows/CurrentVersion/Search/RecentApps/`) and User Assist (`SOFTWARE/Microsoft/Windows/CurrentVersion/Explorer/UserAssist`) registry keys help provide evidence of program execution [4]. Both keys are updated when a program is run to populate recently used programs lists in Windows. These entries use the registry subgraph but add an additional object node with an edge to the action, indicating the program specific program that was executed.

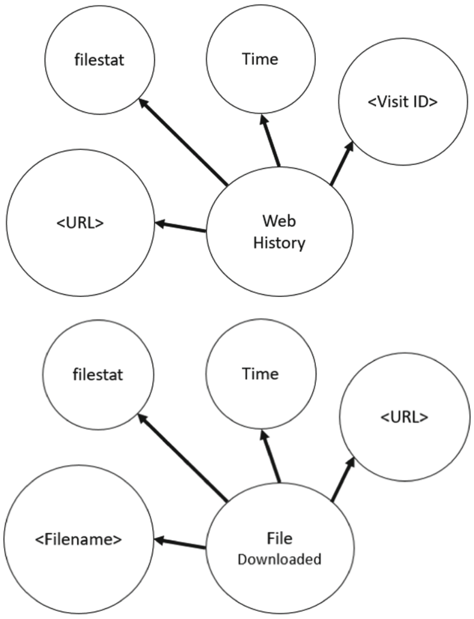

Fig. 5. Web history and download subgraphs.

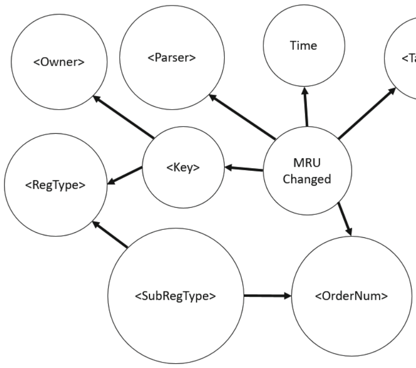

Fig. 6. MRU registry subgraph.

Most Recently Used (MRU) registry entries (`SOFTWARE/Microsoft/Windows/CurrentVersion/Explorer/RecentDocs/`) retain a list of recently used files for each extension (.doc, .jpg, etc) [4]. The recently used files for each extension are listed in ascending numerical order where 0 represents the most recently used file. The MRU subgraph (Fig. 6) captures this data by creating a sub-registry type node that represents the most recently used files of a particular extension. In Fig. 6, this is .ppt. The numerical order of an extension's MRU files are created as separate nodes to allow easy traversal of the files in order of use. There is also an additional object attached to the action node that was extracted from the value of the correct registry key and indicates the used file. The rest of the subgraph follows the standard registry subgraph.

Additional registry entries that can be processed are appcompatcache and shellbag keys. Like recent apps and users assist entries, appcompatcache helps provide evidence of program execution, updating registry values when a program is executed [4]. Shellbag entries maintain UI information for folders viewed in the Windows File Explorer [4]. These values can remain in the registry if the original directory is deleted, possibly providing additional information on the file system.

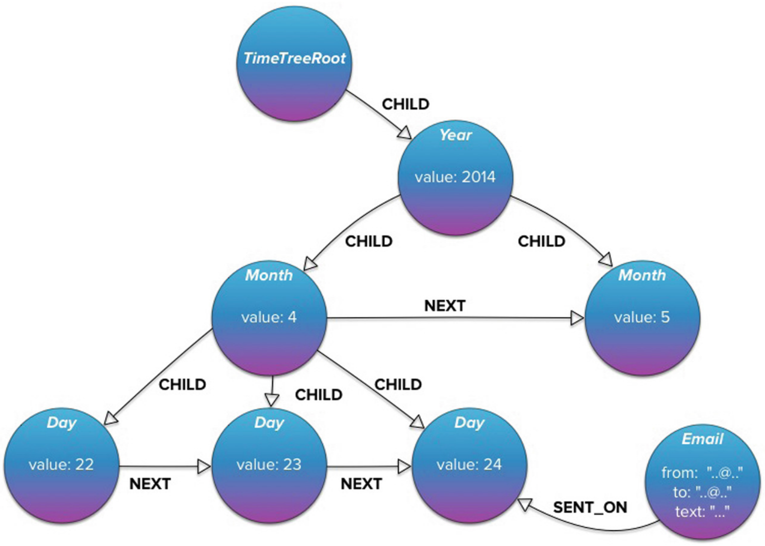

Fig. 7. Time tree format [10].

After an artifact has been transformed into a subgraph, neo4j performs a transformation of the subgraph centered on the time node. Using the timetree plugin from GraphAware, these time events are converted into a time tree (Fig. 7) [10]. The timestamp is divided into year, month, day, hour, minute, and second levels, and all actions are attached to the second nodes. This transformation allows queries to search for temporally adjacent events by following the next relationship.

3.3 Normalization

The next processing layer normalizes the neo4j database by combining objects that are the same but have a different identifying name. For example, MRU Registry Values only list the filename of the used file

(`Snowball_Fighting(1).doc`) in an object node where the file table entry for the same file will be listed in an object node using the whole path (`C:/Users/user/Downloads/Snowball_Fighting(1).doc`). In order to properly see all the actions that happen on an object, these nodes need to be combined. The normalization processing layer combines nodes if there is only one match. If there are multiple matches, the node is linked to the possible matches with the relationship called 'POSSIBLE_REAL_PATH'.

3.4 Abstraction

Abstracting low-level event data into higher-level events is the final processing layer. Abstraction in PGER uses two different sources: expert rules and data mined rulesets.

Expert Rules. Expert forensics knowledge rules extract high-level events from a sequence of low-level events. For example, if there is evidence of a program execution, and a file is accessed with an extension associated with the executable within a certain time window, it is likely that the two actions are related. These known sequences can be quantified into rules to find specific subgraphs. Examples of these patterns can be found in the subsequent paragraphs.

History of a File. A simple example of extracting information from the graph database is determining a file's history. Since every file is unique in the database, the specific node can be expanded to list all the actions that affect the file. This can be done with the following query:

```
1 MATCH (obj:object)
2 WHERE obj.filename = "<filename>"
3 MATCH (sec:Second)<-[:AT_TIME]-(act:action)--(obj)
4 OPTIONAL MATCH (obj)<-[:LINK_TARGET]-(lnkObj)--(
      lnkAct:action)
5 RETURN sec.time, collect(act.action), collect(lnkAct
      .action)
```

The first two lines find the desired object. The third line finds all the actions and times that affect the desired object. Line four gives the history of link files that are associated with the desired object.

Power Events. Using a combination of objects from the Windows event logs, power events can be determined (shutdown, startup, sleep). If enabled, the Windows Customer Experience Improvement (CEI) service will start and stop just before power events. The Windows Event Log service does the same. The Event Log also records the Windows Version and uptime when Windows powers on. Each of these events appear at every power event and serves as the baseline for power events for a given time window. Additional events may appear in the shutdown/startup process: Microsoft Windows Power Troubleshooter - System

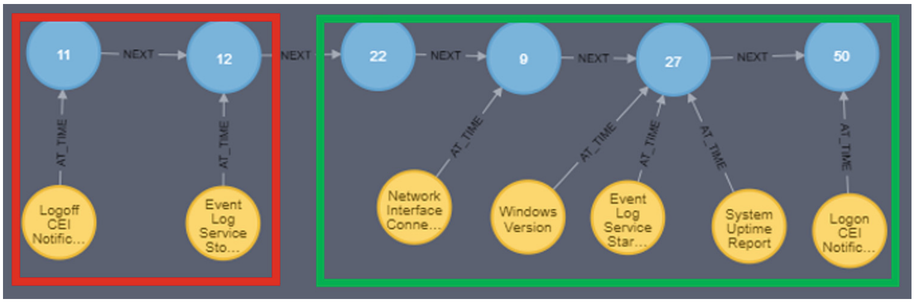

Fig. 8. Shutdown event (Left Box), startup event (Right Box).

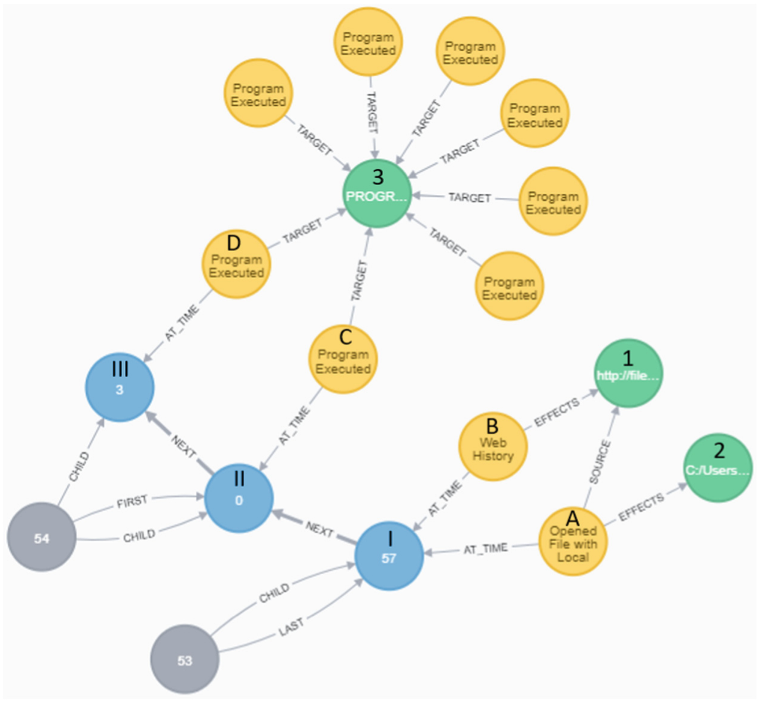

Fig. 9. Download/File system integration.

Returned from a Low Power State; Network Interface Connected/Disconnected; Microsoft Windows RestartManager - Starting/Ending Session. These optional events help increase the likelihood of a power event occurring.

Web History. Combining web history entries can show complex activities. As mentioned previously, there are three main types of data recorded in the graph: history, downloaded files, and keyword values. By chaining consecutive web visits using the visit IDs and time, a sequence of visits can be obtained. Furthermore, downloaded files can be tracked to the destination on the host file system where file history can be obtained (Fig. 9). This figure shows the web history entry (1) and the downloaded location of the file (2). The subgraph also shows that Microsoft Word opened using a prefetch file (3).

Combining Expert Rules. The previous abstractions can also combine together to create new layers of abstraction. Figure 9 is a basic concept of this idea. Both web history and file table information are combined to enrich an examiner's understanding of events that occurred. All actions nodes (A-D) describe the action the user took on artifacts 1–3: opened a file directly from the Firefox browser (A) from a specific URL (B), executing a program (C-D). The graph database is a great way to structure abstraction using a tree-like structure as a new node could be created, linking all actions A-D to this narrative.

3.5 Data Mined Ruleset

The second method of abstraction is to use machine generated rulesets created by Temporal Event Abstraction and Reconstruction (TEAR). These rulesets are much larger and more detailed than the rules generated by experts. For example, TEAR found 86 distinct patterns for the various ways Microsoft Word opens as accessing the program through a jump list or from a web browser affects different artifacts. While possible for an expert to categorize and detail every variation, it is extremely time consuming. To utilize a TEAR rulset, PGER performs a conversion of the strings and terms from TEAR relational database into a tree structure (See Fig. 1) that can then be converted into expert rules. These rules are then applied to the graph in order to produce the same results as TEAR.

4 Results

PGER provides a speedup for digital forensics artifact correlation and event reconstruction. The following tests compare PGER's ability to find events based on expert and machine generated rules compared to truth data and a performance comparison of PGER and a relational database implementation using a machine generated ruleset to find when Microsoft Word opens.

The test image was a 65 GB Window 10 image with sample activities that included: Web Browsing/Downloading Files from Microsoft Edge, Mozilla Firefox, and Google Chrome; Microsoft Office: Creation/Manipulation of Word, Excel and PowerPoint Files; Viewing Downloaded PDFs; Sleep, Startup, and Shutdown Sequences; Viewing image files; Manipulating files in Windows Explorer.

All tests were conducted on a machine with the following specifications: CPU - i3-6100U (2 Cores, 4 Threads), RAM - 12 GB, HDD - 250 GB Samsung 840 EVO SSD, OS - Ubuntu 16.04 LTS, Docker Version - 17.09, Python Version - 3.6, Elastic Stack Version - 5.5.2, neo4j Version - 3.2.2, PLASO Version - 1.5.1.

Table 2 lists details on the size of the resultant PGER graph.

Table 2. neo4j Database Parameters After Graph Conversion.

neo4j Database Size	656.18 MB
Node Count	1,069,671
Relationship Count	2,587,503
Processing Time	00:16:28

4.1 Expert Rule Results

The following tests evaluate PGER's ability to accurately abstract events using expert generated rules compared to the known activity on the device image. The first test finds all startup/shutdown events and the second test finds all files downloaded from a web browser.

Power Events. A power event represents a shutdown or startup on the forensics machine. To find the power events on the device image, a power event is primarily determined by the status of the CEI and Windows Log Services. If they are shutdown, it indicates a shutdown event; the opposite is also true. Startup also can contain optional Windows log entries to further bolster the evidence of a startup event.

Table 3. Power Event Sequence (UNIX Timestamps).

Start Time	End Time	Event
1491971379	1491971414	Startup
1491972643	1491972656	Shutdown
1491972859	1491972888	Startup
1492112687	1492112712	Startup
1492113771	1492114425	Shutdown

After applying rules to the dataset, a total of 65 shutdown events and 67 startup events were detected. There was apparently an error in the rules as two shutdown events were missing. After examination, there were two sequences that contained consecutive startup events. One such sequence is in Table 3.

After reviewing the actions between the two consecutive startup events using the query below, a Windows Update seemed to occur.

```
MATCH (sec:Second {time: 1491972888})
MATCH p = (sec)-[:NEXT*..500]->(:Second {time:
    1492112687})
UNWIND nodes(p) as secNode
MATCH (secNode)<-[:AT_TIME]-(act:action)-[:EFFECTS
    ]->(obj:object)
```

```
5 RETURN secNode.time, collect(act.action), collect(
      obj.filename)
```

Of the 5706 objects that had altered timestamps, 4830 matched the pattern C:/ Windows/WinSxS/*; these files are known to be related to Windows updates. The shutdown sequence during Windows updates differs from other shutdowns and do not include additions to the event logs.

Updates might also explain the large gap in the start and end times for some shutdown events. For example, the last event in the table has a difference of 654 s between its start and end times. Upon examination, several files are changed 361 s after the start of the event. These files match the pattern C:/Windows/WinSxS/ amd64_windows-defender-am-sigs or C:/ProgramData/Microsoft/Windows Defender /Definition Updates/. As a result, it appears that Windows Defender Definitions are updated before a shutdown. In future iterations, this needs to be captured as additions to the expert rules.

Table 4. Downloaded File Expert Rule Result.

Download Information	
Username	user
Download Time	1497574737
Filename	C:/Users/user/Downloads/Snowball_Fighting(2).doc
Shortened URL	http://files.geekdo.com/geekfile_download.php?

Web History	
Time	URL
1497574730	/filepage/28906/snowball-fighting-rules-word-doc
1497574732	/file/download/2hkk77tped/Snowball_Fighting.doc
All URLs start with https://www.boardgamegeek.com	

File History	
Time	Action
1497574765	Accessed, Created Time Altered
1497574767	Modified Time Altered
1497574768	Changed Time Altered
1499350036	Recent .doc Changed
1499350036	Recent Docs Changed

Combined Expert Rules. The downloaded files expert rule is a great example of utilizing many different low-level event types to create a complex abstraction. The expert rules combine the following: previous web history, URL source for the downloaded file, location of the download file on the forensics image, file history of the downloaded file and username responsible for modified registry keys. The desired subgraph is similar to Fig. 9.

Firefox history shows nine files were download from the browser in the image and that the rules found all nine entries. One example entry is contained in Table 4.

These tables highlight the insights gained through connected data and provides multiple corroborating artifacts. An examiner can see recent web history, the URL source, the file system destination for the downloaded file, and what actions took place on the downloaded file. PGER was able to perform this relationship-heavy query for all nine objects in 18 ms.

4.2 Machine Generated Application Results

Applying expert rules, as evidenced by the results above, can be an effective way to abstract data. However, the rules in the previous section are short, only matching 8–18 different events in a time window. Some user actions are incredibly hard to capture using expert knowledge alone due to the immense number of objects and events that affect the outcome. TEAR tackles this problem by using a machine to generate patterns for complex events. For example, TEAR extracted a combination of 86 different artifacts that can occur when Microsoft Word opens. Incorporating the TEAR method of finding high-level events into PGER is important to evaluate PGER's ability to abstract complex events.

The final evaluation tested PGER's ability to replicate the results produced by TEAR, which uses a relational database on the same dataset. Both processing time and accuracy in replicating TEAR's results were the criteria for evaluating PGERs performance.

Testing for both PGER and TEAR used the same device image, event extractor, and pre-processed ruleset. The ruleset was limited to opening Microsoft Word as the lone top-level rule. Terms that represented accessed prefetch files appeared in all term sequences. Since both tools used the same data extraction method, this step was ignored in both testing categories. Timing results were an average of three runs.

Processing Time. Performance of PGER is compared to the processing time of TEAR due to their differing storage methods for their ontologies; TEAR uses a relational database and PGER utilizes a property graph database. This test shows how a property graph database can have a significant effect on performance on forensics queries. To help further understand the comparison, the processing steps of TEAR and PGER are compared in next paragraph.

TEAR matches high-level events utilizing following steps:

1. Load event data from a SQLite database to memory
2. Apply terms to events, abstract terms to strings
3. Process the generated term list in chronological order. Find high-level events by matching a variant with the term list of the current time window.

To accomplish the same feat, PGER uses the graph conversion, normalization, and abstraction processing layers. Graph conversion and normalization

Table 5. PGER and TEAR Runtime Comparison (h:mm:ss).

TEAR Step	TEAR Time	PGER Step	PGER Time
1	0:29:52	Graph Conversion	0:16:28
		Normalization	
2	3:02:06	Build Ruleset Tree/Apply Terms	0:04:32
		Create Time Windows	
3	0:09:34	Find High-Level Events	0:01:12
Total Time	3:41:32		0:22:12

matches with TEAR's first step. PGER's abstraction layer accomplishes both steps 2 and 3. To provide a direct comparison, the PGER abstraction layer splits into two categories to better match TEAR's steps. The first group, corresponding to TEAR's second processing step, contains building the ruleset tree, term application, and time window creation. Finding high-level events corresponds with TEAR's third processing step. In both programs, steps 1 and 2 are only accomplished once per image as long as the same ruleset is applied. Table 5 compares TEAR and PGER runtimes.

Both PGER and TEAR found 12 instances of the event. PGER applies the machine generated ruleset to a forensics image in less time than TEAR. This advantage is shown the best in step 2 where the TEAR ruleset is constructed into a tree and used to find the objects that apply to each term. One significant factor in the processing time difference is that PGER does not abstract sets of terms into strings; PGER finds all high-level rules only as different compositions of terms. This results in longer comparisons between time windows and rulesets, but PGER filters time windows in step 3 that do not include terms that are in every variant. In this case, the difference is significant, eliminating nearly 80% of all time windows (From 10,178 to 2,444). If there were rules that did not filter out as many time windows, the PGER runtime to find high-level events (TEAR step 3) could increase by a factor of five. However, PGER would still provide a performance advantage over TEAR.

5 Conclusion and Future Work

PGER reduces the processing time of event correlation grammars by up to a factor of 9.98 over a relational database based approach by using a native graph processing and storage format. This allows users to leverage the advantages of a graph database without the cost of just-in-time assembly of a graph or the completion of many join statements. Utilizing a graph database also allows for more natural queries of ontological data, affording users the ability to find subgraphs by searching for path patterns. If an ontological language uses a non-native processing method, it may be beneficial to convert into a native graph before analysis. PGER can also match the accuracy of TEAR and provide an increase in processing speed by filtering time windows and not abstracting terms into strings. Finally, expert rules were applied to the database, providing useful information for the examiner. However, longer rules might be better created by machine pattern matching, like TEAR, due to the difficulty in codifying the execution of a large set of events and objects.

Future work includes evaluating PGER performance compared to a RDF datastore, test PGER at scale on larger device images, expanding the expert ruleset, and applying curation rules to TEAR rulesets.

Acknowledgments. The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

References

1. Angles, R.: A comparison of current graph database models. In: Proceedings of IEEE 28th International Conference on Data Engineering Workshops, ICDEW 2012, pp. 171–177. IEEE (2012). https://doi.org/10.1109/ICDEW.2012.31
2. Bureau of Labor Statistics: Occupational Outlook Handbook: Forensic Science Technicians (2017). https://www.bls.gov/ooh/life-physical-and-social-science/forensic-science-technicians.htm
3. Bureau of Labor Statistics: Occupational Outlook Handbook: Information Security Analysts (2017). https://www.bls.gov/ooh/computer-and-information-technology/information-security-analysts.htm
4. Carvey, H., Hull, D.: Windows Registry Forensics, 2nd edn. Elsevier, Cambridge (2016). https://doi.org/10.1016/C2009-0-63856-3
5. Casey, E., Back, G., Barnum, S.: Leveraging CybOX^{TM} to standardize representation and exchange of digital forensic information. Digit. Investig. **12**(S1), S102–S110 (2015). https://doi.org/10.1016/j.diin.2015.01.014
6. Chabot, Y., Bertaux, A., Nicolle, C., Kechadi, M.T.: A complete formalized knowledge representation model for advanced digital forensics timeline analysis. Digit. Investig. **11**, S95–S105 (2014). https://doi.org/10.1016/j.diin.2014.05.009. http://www.sciencedirect.com/science/article/pii/S1742287614000528
7. Chabot, Y., Bertaux, A., Nicolle, C., Kechadi, T.: An ontology-based approach for the reconstruction and analysis of digital incidents timelines. Digit. Investig. **15**, 83–100 (2015). https://doi.org/10.1016/j.diin.2015.07.005
8. Chao, J., Graphista, N.: Graph Databases for Beginners: Native vs. Non-Native Graph Technology (2016). https://neo4j.com/blog/native-vs-non-native-graph-technology/
9. Gladyshev, P., Patel, A.: Finite state machine approach to digital event reconstruction. Digit. Investig. **1**(2), 130–149 (2004). https://doi.org/10.1016/j.diin.2004.03.001
10. GraphAware: GraphAware Neo4j TimeTree (2018). https://github.com/graphaware/neo4j-timetree
11. Guðjonssón, K.: Mastering the Super Timeline With log2timeline (2010). https://www.sans.org/reading-room/whitepapers/logging/mastering-super-timeline-log2timeline-33438

12. Hargreaves, C., Patterson, J.: An automated timeline reconstruction approach for digital forensic investigations. Digit. Investig. **9(Suppl.)**, S69–S79 (2012). https://doi.org/10.1016/j.diin.2012.05.006
13. James, J., Gladyshev, P., Abdullah, M., Zhu, Y.: Analysis of evidence using formal event reconstruction. Digit. Forensics Cyber Crime **31**, 85–98 (2010). https://doi.org/10.1007/978-3-642-11534-9
14. Khan, M.N., Mnakhansussexacuk, E., Wakeman, I.: Machine Learning for Post-Event Timeline Reconstruction. PGnet (January 2006), 1–4 (2006)
15. Marrington, A., Mohay, G., Clark, A., Morarji, H.: Event-based computer profiling for the forensic reconstruction of computer activity. AusCERT2007 R&D Stream **71**, 71–87 (2007). http://eprints.qut.edu.au/15579
16. Okolica, J.S.: Temporal Event Abstraction and Reconstruction. Ph.D. thesis, AFIT (2017)
17. Robinson, I., Webber, J., Eifrem, E.: Graph Databases, 2nd edn. O'Reilly Media Inc., Sebastopol (2015)
18. Rodriguez, M.A., Neubauer, P.: The graph traversal pattern. Computing Re-search Repository, pp. 1–18 (2010). https://doi.org/10.4018/978-1-61350-053-8, http://arxiv.org/abs/1004.1001
19. Schatz, B., Mohay, G., Clark, A.: Rich Event Representation for Computer Forensics. In: Asia Pacific Industrial Engineering and Management Systems APIEMS 2004, pp. 1–16 (2004)
20. Turnbull, B., Randhawa, S.: Automated event and social network extraction from digital evidence sources with ontological mapping. Digit. Investig. **13**, 94–106 (2015). https://doi.org/10.1016/j.diin.2015.04.004