MemTri: A Memory Forensics Triage Tool using Bayesian Network and Volatility

Michalas, A. and Murray, R.

MemTri: A Memory Forensics Triage Tool using Bayesian Network and Volatility

Antonis Michalas
Cyber Security Group
Department of Computer Science
University of Westminster
London, UK
a.michalas@westminster.ac.uk

Rohan Murray
Cyber Security Group
Department of Computer Science
University of Westminster
London, UK
rohanlmurray@gmail.com

ABSTRACT

This work explores the development of MemTri. A memory forensics triage tool that can assess the likelihood of criminal activity in a memory image, based on evidence data artefacts generated by several applications. Fictitious illegal suspect activity scenarios were performed on virtual machines to generate 60 test memory images for input into MemTri. Four categories of applications (i.e. Internet Browsers, Instant Messengers, FTP Client and Document Processors) are examined for data artefacts located through the use of regular expressions. These identified data artefacts are then analysed using a Bayesian Network, to assess the likelihood that a seized memory image contained evidence of illegal firearms trading activity. MemTri's normal mode of operation achieved a high artefact identification accuracy performance of 95.7% when the applications' processes were running. However, this fell significantly to 60% as applications processes' were terminated. To explore improving MemTri's accuracy performance, a second mode was developed, which achieved more stable results of around 80% accuracy, even after applications processes' were terminated.

CCS Concepts

•**Applied computing** → **Computer forensics; Investigation techniques; Evidence collection, storage and analysis; System forensics;**

Keywords

Digital Forensics; Triage; Cyber Crime; Digital Evidence; Random Access Memory

1. INTRODUCTION

With the current advances in digital forensics, it is becoming more common for law enforcement personnel to encounter digital devices as part of seized evidence to be examined. This list of digital devices include various machines with different architectures and specifications (e.g. desktops, laptops, mobile phones, tablets etc). The growing influx of seized digital devices has generated a backlog of court case evidence to be forensically examined [9]. A proposed solution for alleviating this evidence backlog is to develop triage execution tools that incorporate data mining techniques [23]. The main aim of such triage tools is to quickly assess whether a digital device contains relevant case evidence or not, and how much priority should be placed on fully analyzing the device.

Even though there are many crime classification triage tools for disk and mobile forensics, there is a clear lack of any such similar triage tool for memory forensics. The absence of such memory forensics tools is considered as an obstacle that prevents investigators from effectively analyzing digital devices. This is mainly due to the fact that various research has shown that memory can contain critical evidence such as internet browsing data, network traffic, malware, passwords, cryptographic keys and decrypted content, some of which may never be stored to disk [8, 10]. A possible reason for the apparent low research in developing crime classification triage tools for memory forensics is due to the complexity in analyzing operating system (OS) memory structures, which is still a fairly adolescent area of research. The open-source tools Volatility [31] and Rekall [29] have aided in simplifying the analysis of such OS memory structures by incorporating the academic research done by various authors in reverse engineering these structures. In this paper, we leverage from the various research incorporated into the Volatility framework [31] and we propose MemTri – a memory triage application that analyzes OS memory structures. It was simply decided to utilize the Volatility framework for this project, due to it being the most widely utilized and tested memory analysis tool in the academic community. Another factor that may have contributed to the apparent research in developing crime classification triage tools for memory forensics, is due to the fact that acquiring memory requires careful planning and skill in order to collect a 'forensically sound' [34] memory image, which in-turn has led to the slow adoption of performing memory image acquisitions by law enforcement departments.

Another challenge in memory forensics is that, if the user terminates the application process used to perform an illegal activity then the freed virtual address space is often quickly overwritten by other activity within the operating system. However, Garfinkel et al. [6] showed that portions of unallocated memory can remain unchanged for up to 14 days –

MIST'17, October 30, 2017, Dallas, TX, USA

DOI: https://doi.org/10.1145/3139923.3139926

even when the system is actively being utilized. Therefore, since some data artefacts may not be overwritten in unallocated memory space by the OS, it is still possible to extract such data artefacts for memory analysis, similar to carving for files in a file system.

MemTri offers a way to quantitatively measure the likelihood that a specific criminal offence was committed. The results are based on an extended analysis of test evidence data artefacts that were found in Random Access Memory (RAM), and help us to determine the priority that should be placed on fully examining a set of memory images. MemTri is built on top of a Bayesian Network and the Volatility Framework. Furthermore, in order to successfully achieve our goal, we developed a certain set of algorithms through which we can assess the effectiveness of locating data artefacts in RAM, after the process that generated the artefact has terminated. This is of paramount importance since a forensics analysis is likely to be held after the termination of the application that used to create private information of the "corrupted" parties.

1.1 Summary of Contributions

The contribution of this work is twofold. First, we contribute to the field of memory artefact identification by providing regular expression patterns that can be used to identify different types of memory artefacts generated by various applications, namely, Chrome, Tor, Filezilla, Skype, Wickr, Libre Writer and Microsoft's Notepad. MemTri, also confirms the regular expressions patterns designed by [27, 10] to locate browser memory artefacts generated by visiting websites and performing Google search engine queries. Furthermore, we developed our regular expression patterns in such a way that will enable us to successfully capture other kinds of browser artefacts such as those generated when a file is downloaded. In addition to that, M. Simon and J. Slay [28] noted that Skype contact information and communication content can be extracted from physical memory, however did not provide regular expressions patterns to capture this data. In our work, we confirm the existence of such Skype information in memory and we explicitly develop regular expressions to capture these Skype memory artefacts.

Second, we contribute to the memory forensics triage field. MemTri analyzes a memory image and provides an output rating which measures the likelihood level of a specific criminal activity. This can be considered as the fundamental contribution of this work since, to the best of our knowledge, there is a complete lack of published research where authors attempt to develop a digital forensics triage tool aimed specifically at analyzing criminal activity found in a memory image. This work, examines the effectiveness of two designed approaches for locating criminal evidence in memory. The first approach, involves using the Volatility Framework to dump the memory of target applications which are then searched for evidence. The second approach, involves scanning the entire memory for evidence. The provided results give valuable insights regarding which of these approaches is generally better suited for a memory forensics triage environment.

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we describe related works regarding existing triage solutions in the field of digital forensics. In Section 3, we introduce the system model, as well as the preliminaries that MemTri is built on. In Section 4, we introduce how MemTri works and in Section 5 we describe the development phase. In Section 6 we present implementation and performance evaluation results while in Section 7 we discuss some anomalies that were noted throughout the evaluation. Finally, in Section 8 we conclude the paper.

2. RELATED WORK

In this section we review the most important works that have been published in the field of digital forensics and we specifically focus on existing triage solutions.

According to [26], the definition of triage in regards to digital forensics is "a process in which digital evidence is ranked in terms of importance or priority". There have been proposed various methodologies for developing triage tools for the main branches of digital forensics, i.e. disk forensics, memory forensics, mobile phone forensics and network forensics.

Bogen et al. [1] developed Redeye – a disk document triage tool. Redeye, utilizes a corpus-based term weighting scheme (TF-ICF) and semi-supervised machine learning to triage identification of documents that relate to a specific case. The corpus-based term weighting scheme mainly assesses the similarity between documents based on the frequency of a word and its position in relation to other keywords. Document analysts are then able to identify documents that are most likely similar/related to certain key documents they have marked as relevant to an investigation. The system further monitors the tags and comments made by analysts in order to 'learn' which type of documents are of particular importance to an investigation. Moreover, Redeye successfully aided to significantly reduce the completion time of a forensic analysis. Even though Redeye focuses on a different field of forensics than MemTri, it demonstrates the ability of supervised machine learning techniques (similarly utilized by MemTri's Bayesian Network) to successfully triage tasks in an investigation.

Li et al. [13] developed a memory triage tool that uses fuzzy hashing to intuitively identify malware by detecting common pieces of malicious code found within a process. Authors, identified a limitation with the asymmetric distance computation of existing fuzzy hashing algorithms and assess four key insights, based on precision and recall, which can improve the fuzzy hashing algorithms' performance. The improvement of such fuzzy hashing algorithms aids investigators to more quickly and accurately determine whether a machine has been affected by malware before attempting a full investigation. MemTri's performance is similarly tested using such performance measures which can reveal key areas of triage-related improvements.

Walls et al. [35] developed DEC0DE, a mobile phone forensics triage tool. DEC0DE, uses block hash filtering (BHF), Viterbi's algorithm and Decision Tree inference. During BHF, similar byte streams between mobile phone models, which most likely will contain operating system data that is not relevant to the investigator, are removed. Hence, the remaining data is likely to be user' data such as call logs and address book information. This data is further processed using Viterbi's algorithm and Decision Tree inference to improve the recall and precision of the filtered data. Authors, highlighted that mobile phone forensics triage can help to gather key information upfront for use in suspect

interviews, before the full analysis is performed which can take months to complete due to backlog of devices to be analyzed. Similarly, MemTri provides the digital investigator with a quick assessment of key evidence artefacts found in a memory image which can then be used as persuasive evidence in a suspect interview.

In [12], authors developed a network triage application that uses a client-server model in order to search multiple client machines for evidence. An automated network triage (ANT) server that hosts various services is used to configure and boot PXE enabled clients. When the client machine boots, a batch script is simply ran to search for keywords, patterns and file hashes on the client machine's disk. This network forensics triage tool can essentially help to locate a machine within a network that was most likely involved in the crime being investigated. Thus the identified machine can be seized/prioritized for further investigation. Without such a triage tools an investigator would have to analyze all the machines in the network individually which is impractical/time-consuming.

The aforementioned works, show that triage tools have proven to be a valuable solution to the 'data volume challenge' [23]. Generally, these triage solutions offer a quick way of narrowing down the devices to those that contain critical data before a full digital forensics analysis is performed.

3. SYSTEM MODEL AND PRELIMINARIES

In this section we describe the system model as well as some basic concepts that will be used in the rest of the paper.

Suspect Machine (SM): For the needs of our work, four types of software applications, namely Internet Browser (Tor, Chrome), Instant Messenger (Wickr, Skype), Document Processor (Windows Notepad, Libre Writer) and FTP Client (Filezilla), are examined. Therefore, we created several Windows 7 virtual machine instances where we preinstalled the various software applications listed earlier. These applications are also referred to as the *'target applications'*. Each virtual machine is then shutdown and a copy of the virtual machine files is made. These copied files are referred to as the base virtual machine image which is used as the starting point for performing the suspect activity analysis.

Evidence Search Engine (ESE): The Evidence Search Engine component is responsible for extracting evidence artefacts from the 'suspect' memory image and translating them into features that can be used by a bayesian network analyser. This approach was mainly inspired by [10, 27, 28] which showed that intuitive evidence artefacts can be retrieved by simply searching for ASCII/Unicode data patterns generated by specific applications. The first step in the operation of the ESE is to identify the running processes within the memory image that match the target applications mentioned earlier. The 'committed' virtual address space of the target application processes are then dumped into a file referred to as the '`procdump`' file. The ASCII and Unicode content of the '`procdump`' file are then extracted out into another file, referred to as the '`proctext`' file, in preparation for filtering. The next step of filtering out the evidence artefacts within the application's process is done through the use of regular expressions. These regular expressions are selected during the execution of training phase which is later discussed in Section 4.2. This regular expressions approach is also flexible in that it can locate evidence artefacts in a memory image regardless of the OS environment in which the artefacts were generated. Additionally, regular expressions can be executed fairly quickly to locate evidence within large datasets. This intuitiveness, flexibility and speed offered by regular expression evidence searching methods, are essential traits for the development of an effective digital forensics triage tool. These regular expressions are selected during the execution of a training phase that is discussed later in Section 4. When an evidence artefact is found, the final step is to associate the artefact with a feature(s). In this case, a feature is synonymous with a scenario id shown in Table 2. The entire ESE process of locating the evidence artefacts in a process' virtual address space and converting them to features is illustrated in Figure 1. The features are then entered into the BNA component to assess the level of illegal firearms criminal activity found.

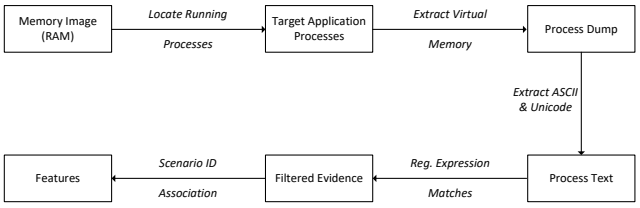

Figure 1: Designed steps for MemTri's Evidence Search Engine component

Bayesian Network Analyser (BNA): The second main component of the MemTri application is the Bayesian Network Analyser (BNA). The BNA is responsible for analysing the evidence found by the ESE in the 'suspect' memory image, in order to produce an output rating of the likelihood the suspect was involved in illegal firearms trading. The BNA consists of a Bayesian Network Model (BNM) that is designed with three layers, i.e. the hypothesis layer, sub-hypothesis layer and evidence layer as shown in Figure 2. Each node in the Bayesian Network is designed to capture three probability states; '*Yes*', '*No*' and '*Uncertain*'.

The hypothesis layer consists of one node (H_1) that represents the overall conclusion about whether the suspect used the seized computer for illegal firearms trading (see Figure 2). Therefore, the hypothesis layer stores the final numeric output rating result of the MemTri application, which is essentially the posterior probability of the '*Yes*' state of the hypothesis node.

The sub-hypotheses layer represents the various types of applications that the suspect was likely to use in performing an illegal firearms trading activity (see Figure 2). As such, the sub-hypothesis layer is modelled to contain four nodes (H_2 – H_5), one for each application type that is examined. The MemTri application also displays an output rating (i.e. the '*Yes*' posterior probability state) for each of the sub-hypothesis nodes, so that the digital investigator can easily analyze which application types contain the most relevant evidence in the investigation.

The evidence layer represents the artefacts of evidence that are likely to be found, given that the suspect conducted an illegal firearms trading activity with any of the application types modeled in the sub-hypotheses layer. For example, evidence node E_1 (see Table 1) represents the occurrence of any relevant web engine search evidence, given that the

suspect used an Internet Browser application type (represented by the sub-hypothesis node H_2) to perform an illegal firearms trading activity. If an evidence artefact is found, as indicated by the outputted features of MemTri's ESE component, the state of the corresponding evidence node is set as 'observed' (i.e. its '*Yes*' posterior probability state is set to 100%). The evidence nodes are the only nodes directly updated in the Bayesian Network. The state of all other nodes are automatically updated through the performance of Bayesian Inference, which ultimately calculates the final output rating at the hypothesis layer.

The Bayesian Inference process relies on digital forensics expert knowledge, which is encoded into the node edges of the Bayesian Network, in the form of 'Likelihood' Joint Probability Tables. The calculations are performed in a way that the 'Prior' probability values for three possible states of all nodes, are equally set to 33.33%. Hence, there are no initial biases about the illegal activity found in the 'suspect' memory images.

An overview of the entire design for the BNM, along with the relevant meaning of all the nodes, can be found in Figure 2 and Table 1 respectively. This designed BNM is generally easy to interpret, in that the linking of the nodes shows a logical analysis/reasoning between the evidence observed and the hypothesis answers being tested for. For example, nodes (E_1 – E_4) all represent the evidence that relate directly to the sub hypothesis H_1 being tested, which in turn is a part of the overall main hypothesis H being tested. This ease of interpretation of the BNM supports a timely decision-making process, which is a favourable feature when seeking to triage a criminal investigation.

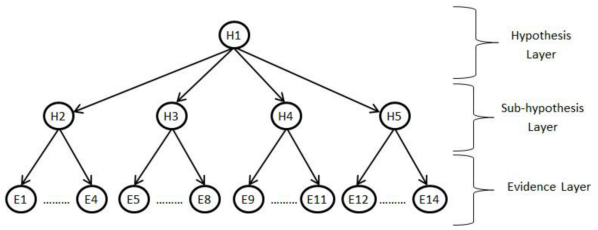

Figure 2: Designed Bayesian Network Model for the MemTri Application

Bayesian Network: In digital forensics triage, law enforcement personnel often has to make quick decisions based on evidence found on a crime scene. Thus, a soundly built Bayesian Network can efficiently aid in determining the best course of action to be taken based on the evidence found. The Bayesian Network model is an acyclic graph that encodes the conditional independence relationship of the graph nodes. We decided to build the Bayesian Network based on the model proposed in [24], since it is simple to interpret and has proven successful in correctly analyzing real-life criminal investigations. Comparative studies have also analyzed that Bayesian approaches, have on average the best accuracy performance (88.5%) [15] compared to other supervised machine learning (SML) techniques such as Support Vector Machines, Decision Trees and K-Nearest Neighbour. This combination of accuracy and ease of interpretation supported by Bayesian Network approaches, are favourable traits when seeking to triage a criminal investigation. Additionally, Bayesian Networks handles missing evidence most elegantly, since it is naturally incorporated into its design. Handling missing evidence is of paramount importance for successful forensics investigations, since evidence can often be missing due to it being destroyed or not yet discovered. For the needs of our work, we used the Netica [18] software to set up our Bayesian Network.

Type	Nodes	Description
Hyptothesis	H1	The suspect performed illegal firearms trading activity on the seized computer
	H2	The suspect utilised an Internet Browser to perform illegal firearms trading
	H3	The suspect utilised an Instant Messenger to perform illegal firearms trading
	H4	The suspect utilised a Document Processor to perform illegal firearms trading
	H5	The suspect utilised a FTP Client to perform illegal firearms trading
IBE*	E1	Web engine search for illegal firearms trading related content
	E2	Visited a website known to contain illegal firearms trading content
	E3	Downloaded file suspected of containing illegal firearms trading content
	E4	An anonymous type Internet Browser was used
IME*	E5	Has contact information for a known illegal firearms dealer
	E6	Sent message suspected to be related to illegal firearms trading
	E7	Transferred file suspected of containing illegal firearms trading content
	E8	An anti-disk forensics Instant Messenger was used
DPE*	E9	Typed content related to illegal firearms trading
	E10	Disk location of file suspected of containing illegal firearms content
	E11	Document was password protected
FTPE*	E12	FTP Connection to Server IP suspected to be used for illegal firearms trading
	E13	FTP Client user credentials used to connect to illegal firearms trading server
	E14	Transferred file suspected of containing illegal firearms trading content

Table 1: Symbolised meaning of the nodes in MemTri's BNM; Internet Browser Evidence, Instant Messenger Evidence, Document Processor Evidence, FTP Client Evidence

4. MEMTRI

Having described the system model, we are now ready to proceed with the actual description of MemTri.

4.1 Suspect Machine Preparation

The suspect memory images are collected through a virtualisation memory acquisition technique. VMware's hypervisor handles the virtual machine's memory management and facilitates the dumping of the virtual machine's memory contents into the '`.vmem`' file, when it enters a suspended state. According to Gruhn et al. [7], virtualisation memory acquisition techniques generally produce the best quality memory images in terms of *atomicity* and *integrity*. This is due to the fact that virtual machines can be immediately suspended which halts all activity in the virtual machine. With VMware, the entire main memory of the virtual machine is dumped in a '`.vmem`' file when suspended. Therefore, virtualisation memory acquisition methods provide perfect atomicity [7]. There is also negligible change to the memory of the running virtual machine from the time the digital investigator decides to suspend the machine to the collection of the '`.vmem`' memory image file. Hence, virtualisation memory acquisition methods also exhibit perfect integrity [7].

Suspect Activity Scenarios. The first step in the generation of the suspect activity scenarios (SAS) is determining the set of operations that can be performed with a specific application type. For example, internet browsers can perform download operations while instant messengers can send and receive messages. The next step involves the identification of a set of words, website URLs and contact names, collectively referred to as '*case words*', that are particular interesting to an illegal firearms trading investigation. The final step is designing the actual SASs, based on the combi-

nation of an application operation with certain '*case words*'. This, will help us to simulate the performance of an illegal firearms trading activity. The list of developed SASs are shown in Table 2.

App. Type	Alias	Scenario ID	Scenario Description
Internet Browser	WEB	W1	Perform a google search for content relating to illegal firearms trading
		W2	Visit a website that is flagged as containing content related to illegal firearms trading
		W3	Download a file that is suspected to contain illegal firearms trading content
		W4	Utilise the Tor browser
Instant Messenger	MSG	M1	Add a suspected illegal firearms dealer to messenger contact list
		M2	Send a message containing language relating to illegal firearms dealership
		M3	Transfer a file that is suspected to contain illegal firearms trading content
		M4	Utilise the Wickr messenger application
Document Processor	DOC	D1	Type content into a document related to illegal firearms dealership
		D2	Save/Open a file that is suspected to contain illegal firearms trading content
		D3	Open a password protected document
FTP Client	FTP	F1	Connect to FTP Server (enter the server's IP address)
		F2	Connect to FTP Server (enter the user credentials)
		F3	Transfer a file that is suspected to contain illegal firearms trading content

Table 2: List of developed suspect activity scenarios

Based on the designed SASs, we gathered two sets of memory images. The first set, is used to train MemTri to locate data artefacts generated by each of the SASs. These memory images are referred to as '*training images*'. To generate the training images, each of the SASs is performed on a separate base virtual machine image. Then, the memory collected is examined for data artefact patterns (see Section Section 4.2).

The second set, is used to test MemTri's ability to successfully identify multiple SASs performed in a simulated criminal activity environment. These memory images are referred to as the '*test images*'. To generate the test images, a set of 20 experiments were performed involving multiple scenario ids. The scenario ids were selected based on a sampling criteria aimed to cover a reasonable spread of the possible sample space in the Bayesian Network. A better method for selecting the scenario ids would be to implement a Bayesian Network random sampler such as Gibbs sampler [11]. However, this is one of the future directions we plan to follow.

4.2 Collection of the Memory Images

For the needs of this work, we consider two kinds of memory images, namely training and test memory images.

Generating the Training Memory Images. Each of the SASs listed in Table 2 was performed on a copy of the base virtual machine image and a memory image was collected for each. This resulted in a total of 14 memory '*training images*'. The training images are essentially used to teach MemTri how to identify the data artefacts generated when a SAS has been performed. In order to accomplish this, the ASCII and Unicode text of each training image was searched manually for notable patterns that held the evidence data being sought. The ASCII and Unicode text were extracted from the training images using the Linux strings utility and the evidence data was examined in a typical text editor. Several data artefact patterns were manually identified for each SAS performed in the training images and regular expressions are programmed into MemTri's ESE to identify these patterns.

Generating the Test Memory Images. The test images were generated based on 20 designed experiments. Each experiment was conducted on a copy of the base virtual machine image, after which a memory image was collected at three different points. The first point of memory collection was done while the target applications were still running. This is referred to as the '*Running Phase*' test images. The virtual machine was then resumed and the target applications were terminated. Immediately after terminating the target applications, the virtual machine was suspended and a second memory image was collected. This second point of memory collection is referred to as the '*Stopped Phase*'. The virtual machine was then resumed for the final time and left to run idly for 5 minutes. Then, the virtual machine was suspended and a third memory image was collected. This third point of memory collection is referred to as the '*Delayed Phase*'. At the end, a total of 60 test images ($\approx$ 4GB each) were collected to test MemTri's ability to locate and analyse evidence artefacts after completing its training phase.

5. MEMTRI APPLICATION DEVELOPMENT

MemTri was built in the C++ programming language. In the following paragraphs we discuss how MemTri's mode of operations, ESE and BNA were developed.

5.1 Modes of Operation

MemTri was initially developed with one mode of operation, referred to as '*normal mode*'. Normal mode uses Volatility's `pslist` plugin to locate the Windows 7 processes using the link-list enumeration method. The committed virtual addresses space for each target application process is then dumped into a file using Volatility's '`memdump`' plugin, which is later searched for evidence artefacts. It was later observed that the Windows `_EPROCESS` structure is usually unlinked from the active process list immediately after the process is terminated; thus, MemTri's normal mode of operation would not locate the process to dump its contents. Therefore, another method named '*scan mode*' was introduced, in order to improve MemTri's ability to locate evidence artefacts even after the process has been terminated.

With scan mode, MemTri essentially scans the entire physical address space of the memory image for processes and evidence artefacts. In this case, Windows 7 processes are located using the pool scanning technique implemented by Volatility's '`psscan`' plugin. As we will see later, there are also minor differences in the regular expressions utilised to discover evidence artefacts. Since scan mode searches evidence artefacts independently from locating `_EPROCESS` structures, it is able to locate evidence artefacts even after the target application processes have been terminated. However, a possible disadvantage of scan mode is that it may not be able to locate an evidence artefact that spans across a page boundary since it searches the unordered physical address space of the memory image rather than the ordered virtual address space of the application's process, as done in

normal mode.

5.2 Evidence Search Engine Implementation

The ESE contains the following four main steps:

1. Locate the target applications process structures in memory.
2. Extract the unicode and ASCII text from memory.
3. Search for evidence artefacts that occurred as a result of SASs performed.
4. Generate features that can be assessed by the Bayesian Network.

The implementation of the ESE for MemTri running in normal and scan mode is slightly different. The differences will be explained along the way in the following paragraphs.

Locating the Target Application Processes.

In normal mode, MemTri locates processes by link list enumeration using the '`pslist`' plug-in, while in scan mode by pool scanning using the '`psscan`' plug-in. Both plug-ins output a list of the processes found which is parsed by the process_filter() function. This function, searches for the names of the target application processes which are examined for evidence artefacts and associates it with the relevant application type.

Extracting ASCII and Unicode Text.

To extract ASCII and Unicode text from memory, the Volatility plug-in '`memdump`' is first used to dump the target processes' committed virtual address space to a file. The process memory dump files are named after their respective process id (PID) and stored in a folder called 'procdump'. Dumping the process' virtual address space to a file is handled by the process_dump() function. The ASCII and Unicode text of the process memory dump files are then extracted and stored in a folder called 'proctext'. In scan mode, the ASCII and Unicode texts are extracted directly from the actual memory image file. A function process_text() is responsible for extracting the ASCII and Unicode when MemTri is executed in either mode. In order to extract this information from either the process memory dump file or the memory image file itself, MemTri performs a command-line execution of the 'strings2.exe' [16] utility with the respective file as an input parameter.

Evidence Filtering and Feature Generation.

Evidence is filtered out from the extracted ASCII and Unicode text using regular expressions. Advantages of this regular expressions implementation is that it is simple to implement and can recall high volumes of evidence. Additionally, this method is flexible in that it can locate evidence artefacts regardless of the type of the OS the memory image was captured from. However, this implementation ignores non-ASCII and non-Unicode data that can contain evidence. Another evidence searching implementation could have been to navigate the process memory structures in order to access data, for example in the process' heap. This was similarly done by Okolica and Peterson [19] to access the contents of the windows clipboard through examining process heap data. This method, theoretically should allow all evidence relating to a process to be examined. However, it may not be a flexible approach since it commonly requires constant reverse engineering of memory structures every time a new Windows OS is launched. The '`yarascan`' plug-in by Volatility essentially implements both the navigation of process memory structures (more specifically the VAD Tree structure) and regular expressions, to perform contiguous searching of memory. However, it requires a long time to return search results, which is not a suitable for a triage environment.

MemTri utilises the '`grep`' [30] utility to perform the regular expression pattern searches. It was decided to use the '`grep`' utility since it was found to be significantly faster than a previous implementation that utilised the standard C++ `<regex>` library to search for regular expressions.

The set of regular expressions actually utilised by MemTri, differ based on the operation mode executed. In normal mode all of the regular expressions are available for use. The rationale is that since the regular expressions are applied within the constrained context of the process' memory, it is generally safe to apply both '*strong*' and '*weak*' regular expressions with little risk of identifying false positives. On the other hand, with scan mode, a limited number of regular expressions are actually utilised. More precisely, some of the 'weak' regular expressions are excluded. The rationale for this implementation is that since scan mode searches the entire memory image, the straightforward use of 'weak' regular expressions is likely to identify many false positive results. It was therefore decided that in order to use some of the 'weak' regular expressions, an 'application launch' verification process first had to be passed. The 'application launch' verification process, simply involves searching for unique regular expression patterns that are generated when an application is launched. These 'application launch' patterns were identified by simply comparing the text in a memory image which the application was launched from another image in which it was not launched. Certain patterns that were found to be unique in the memory image that contained the launched application were then selected for use in the verification process.

To identify artefacts that are particularly relevant to an investigation, a set of 'case words' are referenced from text files. Then, the function load_case_words() is responsible for reading the corresponding 'case_database' text files into the MemTri application. The regular expressions incorporate these 'case words' to provide a sort of semantic searching feature to locate relevant evidence artefacts. This implementation approach allows a digital investigator to flexibly use the MemTri application for different kinds of criminal investigations, by simply updating the 'case_database' text files with 'case words' that are particularly important to that investigation.

The counts of the regular expressions in the ESE that successfully located evidence artefacts in the memory image are stored in feature variables. These feature variables are the input data that the Bayesian Network uses to identify which evidence nodes should be set as '*observed*'.

5.3 Bayesian Network Analyser Implementation

The Bayesian Network Model for MemTri's BNA component is implemented using the '`dlib`' library [11]. This library was chosen simply because it contains all the necessary structures required for building a Bayesian Network Model.

BNA implementation has three main stages:

1. Building the Bayesian Network Model.
2. Entering the Joint Probability Table values based on Expert's Knowledge.
3. Perform Bayesian Network Inference based on evidence observed.

Building the Bayesian Network Model.

The Bayesian Network Model (BNM) utilised by MemTri is implemented design shown in Figure 2. There is 1 main hypothesis node, 4 sub hypothesis nodes and 14 evidence nodes. The linking of the node's edges to form the Bayesian Network, is done exactly according to the designed model illustrated in Figure 2. This BNM is utilised for both normal mode and scan mode.

Joint Probability Tables Setup.

The next stage in building the BNA is entering the probability values for the 'Likelihood' Joint Probability Tables (JPT), which are referenced during the Bayesian Inference process. These 'Likelihood' JPT values are gathered through a memory forensics expert questionnaire. We used this online questionnaire to collect responses anonymously from several digital forensics companies and private digital forensics expert practitioners. A weighted average of the response results are then entered as the 'Likelihood' JPT's values.

Bayesian Inference on Evidence.

The final stage is to analyse the evidence found via Bayesian Inference, which produces the investigation triage output ratings. Initially, a Bayesian Inference process is performed in order to update all the 'Prior' probability nodes' states to equally be 33.333%. The function mark_observed_evidence() then assesses the feature array variables generated by the ESE and marks the state of the respective evidence nodes as 'observed', based on the evidence found. After marking relevant evidence nodes' states as 'observed', Bayesian Inference is performed using the *join_tree* algorithm [11]. The final output rating results are then displayed by printing the '*Yes*' state values of the main hypothesis node H and the sub-hypothesis nodes H_2 to H_5. It was decided not only to print the final output rating of the main hypothesis node but also to include the ratings of sub-hypothesis nodes, so that the digital investigator can clearly see which type of target applications most likely contained evidence relevant to the actual investigation.

6. IMPLEMENTATION AND RESULTS

In this section, MemTri's performance is analysed based on *accuracy*, *precision* and *recall* These are common performance measurements to assess similar tools and were also utilised in [14, 13, 5, 22, 35]. The formulas for the aforementioned performance measurements are all based on variables that count the number of true positive (TP), true negative (TN), false positive (FP) and false negative (FN) results produced.

6.1 Performance

The performance results, measures MemTri's ability to detect certain events performed by a suspect with the targeted applications, based on the regular expressions developed during the training phase.

6.1.1 *Accuracy*

The accuracy, measures how close the actual results produced by MemTri were to the expected results. MemTri's normal and scan mode accuracy results for three collection phases (*running*, *stopped* and *delayed*) are calculated using the following formulae:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

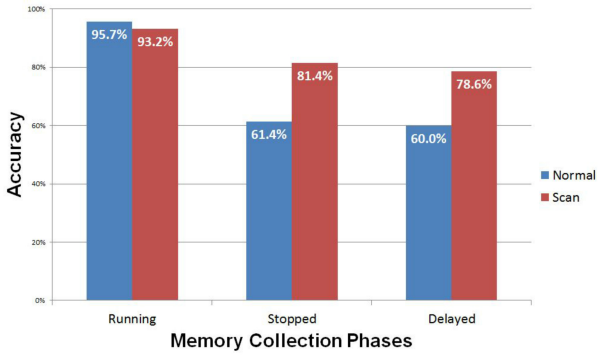

Figure 3: MemTri's average accuracy results for executions in normal and scan mode across the three phase sets of test images

An average of MemTri's accuracy performance for each phase set of test images is illustrated in Figure 3. As can be seen, MemTri's normal mode accuracy performance of 95.7% was 2.5% better than scan mode's accuracy performance (93.2%), for the 'Running' phase test images. Therefore, when the target application processes are still running in memory, MemTri's normal mode is better able to differentiate which SASs were performed, compared to scan mode. However, for the 'Stopped' and 'Delayed' phase test images, MemTri's normal mode accuracy performance dropped over 34% to 61.4% and 60.0% respectively. MemTri's scan mode on the other hand, experienced a smaller $\approx 12\%$ drop in accuracy performance for the 'Stopped' and 'Delayed' phase test images, to 81.4% and 78.6% respectively. Therefore, when the target application processes are terminated, MemTri's scan mode approach identifies more correctly the SASs that were performed in such cases, compared to normal mode.

One factor that negatively impacted on both normal and scan mode's accuracy performance, was that MemTri failed to locate evidence artefacts with *misspelt* words. However, this is something to be expected since the dictionary we used was only intended for testing and research purposes and it is not considered as complete.

The other factors that affected normal and scan mode's accuracy performance, pertained specifically to the implementation differences of each mode. These factors are discussed later under the precision and recall respectively. Essentially, since precision and recall are more targeted subset

measurements of accuracy performance, a fall in precision or recall results in a fall in accuracy and vice versa.

6.1.2 Precision

Precision, measures the probability that a predicted positive result reported by MemTri is actually correct based on the SASs that were performed on the test image. The precision results for MemTri's execution on each of the test images, are calculated using the following formulae:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

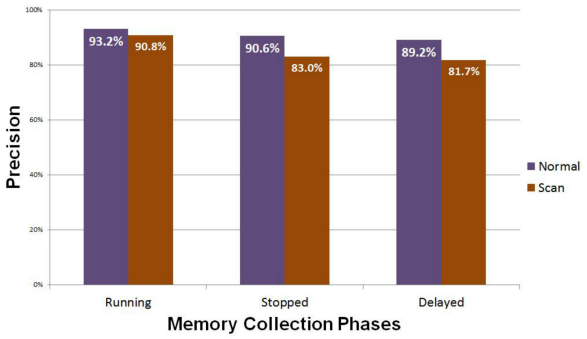

Figure 4: MemTri's average precision results for executions in normal and scan mode across the three phase sets of test images

An average of MemTri's precision performance for each phase set of test images are illustrated in Figure 4. As can be seen, MemTri's normal mode maintained higher precision performances of 93.2%, 90.6% and 89.2%, for the 'Running', 'Stopped' and 'Test' phases respectively, compared to scan mode which had precision performances of 90.8%, 83% and 81.7% respectively. Therefore, in the cases where MemTri predicts that it has positively identified a specific scenario as being performed, normal mode is more likely to have predicted correctly compared to scan mode.

The main assessed reason for normal mode's better precision performance is that when MemTri is executed in normal mode, only the process' committed virtual address space region is searched. Hence, noise patterns that may similarly exist elsewhere in the memory image are filtered out. Scan mode on the other hand, searches the entire memory image for data artefact patterns. This, resulted in a regular expression, which is designed to locate a specific data artefact pattern for a target application, matching another similar data artefact pattern generated by some other application. Such cases mainly occurred with 'weak' regular expression. Nonetheless, both normal and scan mode were able to maintain a reasonably high precision performance (above 80%) for all three phase sets of test images. The higher the precision performance of a digital forensics triage tool, the more evidential supports it lends for the issuance of a search warrant [35].

6.1.3 Recall

Recall, measures MemTri's ability to correctly identify all the memory evidence artefacts that were generated by performing the SASs. The recall results for MemTri's execution on each of the test images are calculated using the following formulae:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

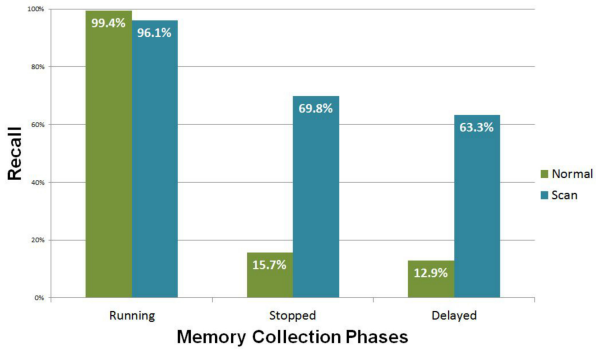

Figure 5: MemTri's average recall results for executions in normal and scan mode across the three sets of phase test images

An average of MemTri's recall performance for each phase set of test images is illustrated in Figure 5. As can be seen, MemTri's normal mode recall performance of 99.4% was 3.3% better than scan mode's recall performance of 96.1%, for the 'Running' phase test images. Therefore, when the target application processes are still running in memory, MemTri's normal mode is able to identify correctly a greater number of SASs that were preformed, compared to scan mode. However, for the 'Stopped' and 'Delayed' phase test images, MemTri's normal mode recall performance dropped significantly by over 83% to 15.7% and 12.9% respectively. MemTri's scan mode on the other hand, experienced a much smaller drop in recall performance of $\approx$ 28% for the 'Stopped' and 'Delayed' phase test images, to 69.8% and 63.3% respectively. This drop in recall for both normal and scan mode is consistent with the observations made in Garfinkel et al.'s [6] research, which illustrated that memory addresses freed immediately after terminating a process, is initially quickly overwritten then more gradually by the OS when in an idle state. As such, it is expected that after a target application process terminates, MemTri is likely to recall less data artefacts due to some of the artefacts being overwritten by the system's activity.

MemTri has a higher recall performance in normal mode than with scan mode for the 'Running' phase test images. The main reason for this is that normal mode uses all the available regular expressions while scan mode uses a limited set of regular expressions to search for evidence. More specifically, scan mode excludes some 'weak' regular expressions. Furthermore, for the 'Stopped' and 'Delayed' phase test images, MemTri's normal mode experienced a significant drop in recall of $\approx$ 83% compared to scan mode which only dropped by $\approx$ 28%. The main reason for this is that when a process is terminated, its `_EPROCESS` structure is usually immediately unlinked from the OS's list of active processes. Normal mode relies on enumerating the list of active processes to locate the target process's VAD Tree structure, which in-turn is needed to dump the process' virtual memory address space before searching for evidence. Therefore,

if the process has been unlinked from the active process list by the Window OS, MemTri's normal mode will not locate any evidence. A few instances were observed where a closed application process was simply marked as terminated and remained attached to the OS's active list of processes.

Overall, the results of this work indicate that the evidence recall performance of a Memory Forensics Triage tool is likely to diminish, if the suspect terminates the application used to commit the criminal activity. In such cases, it is better to use scan mode to uncover leads in an investigation, since it has a better stable recall performance of over 60% compared to normal mode which is under 16%.

6.1.4 Overall Performance

Generally the accuracy, precision and recall performance of MemTri indicates that if the targeted application processes are running, the better approach for locating evidence in memory is to search the committed virtual address space of the running process as done with normal mode. However, if the targeted application processes have been exited, the scan mode approach implemented by MemTri of simply searching the entire memory image, is better in such cases.

Overall, MemTri's scan mode implementation produces more stable results across all three phases during which the test images were collected. The normal mode implementation however, better identifies the relevant evidence artefacts in a memory image when the targeted application processes are still running.

7. OBSERVATION AND ANOMALIES

This section discusses any observations and anomalies that were noted throughout the evaluation of MemTri. Apart from that, the description of the observations and anomalies is coupled with a discussion regarding their impact on the generated results.

We start by discussing a common anomaly observed within various test images generated for this project. The observation was that the virtual memory addresses of a target application process dump at times contained data that was generated by another target application. For example, data generated from performing a scenario with the Tor browser application, was found within the process dump for the Skype messenger application. This data was then identified by MemTri as being generated by the scenario with different id.This anomaly could have simply occurred due to freed memory addresses from the Tor application being acquired by the Skype application. When memory is freed in a Windows OS environment, its content remains intact until overwritten by the activity of another application process.

Another reason for this anomaly could be due to a phenomenon that occasionally occurs during memory acquisition which is referred to as 'page smear' [2]. Page smear occurs when a memory image contains pages of data that is associated with varying times of system activity. Therefore, it is possible that at the time Skype's (`_EPROCESS`) structure was captured, it's currently referenced virtual memory space could have been later updated with pages resulting from concurrent activity performed by the Tor application process.

8. CONCLUSION

In this paper, we presented MemTri. A Memory Forensics triage tool that identifies data artefacts in memory for certain Internet Browsers, Instant Messengers, Document Processors and FTP Client applications, using regular expressions. This work, demonstrated that even after a targeted application process was terminated, some data artefacts could still be extracted from unallocated regions of memory. MemTri was implemented with two modes of operation, i.e normal mode and scan mode. MemTri's normal mode implementation produced more accurate results in the cases where the targeted processes were still running, since it effectively filtered out noise patterns. However, its performance was severely impacted when processes were terminated. Scan mode on the other hand, maintained a fairly high and stable performance even after processes were terminated. Overall, this work demonstrated that a significant amount of data artefacts can be found in main memory, which can offer valuable information to a law enforcement personnel seeking to triage an investigation.

We hope that this project will inspire further research into developing digital forensics triage tools, that will focus on assessing criminal activity found in main memory. Furthermore, this work only utilised a limited set of case-specific words to locate evidence. The next stage is to implement a Knowledge-based Natural Language Processing (NLP) system into MemTri's Evidence Search Engine which utilizes a domain-specific dictionary [25]. This upgrade will allow MemTri to effectively locate evidence in the context of any specified criminal investigation, thus making it practical for use in a real-life environment. In addition to that, we plan to integrate our tool in a trusted cloud environment such as the one presented in [20, 21, 32, 33] and allow users from different geographical locations to enhance the dictionary based on their experience. Finally, adding a weight on each of the existing words has the potential to improve the effectiveness of our tool since it will be easier for a digital investigator to identify real criminal activities and categorize them by their importance. This can be done by utilizing a reputation system that will collect users' votes in a privacy-preserving way [3, 4, 17].

9. REFERENCES

[1] P. L. Bogen, A. McKenzie, and R. Gillen. Redeye: A Digital Library for Forensic Document Triage. In *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries - JCDL '13*, page 181, New York, USA, jul 2013. ACM Press.

[2] M. Cohen. Forensic analysis of windows user space applications through heap allocations. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 237–244. IEEE, jul 2015.

[3] T. Dimitriou and A. Michalas. Multi-party trust computation in decentralized environments. In *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, May 2012.

[4] T. Dimitriou and A. Michalas. Multi-party trust computation in decentralized environments in the presence of malicious adversaries. *Ad Hoc Networks*, 15:53–66, Apr. 2014.

[5] Q. Feng, A. Prakash, H. Yin, and Z. Lin. MACE: High-Coverage and Robust Memory Analysis for Commodity Operating Systems. In *Proceedings of the 30th Annual Computer Security Applications*

Conference on - ACSAC '14, pages 196–205, New York, USA, dec 2014. ACM Press.

[6] T. Garfinkel, B. Pfaff, J. Chow, and M. Rosenblum. Data lifetime is a systems problem. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop: beyond the PC - EW11*, page 10, New York, USA, sep 2004. ACM Press.

[7] M. Gruhn and F. C. Freiling. Evaluating atomicity, and integrity of correct memory acquisition methods. *Digital Investigation*, 16:S1–S10, 2016.

[8] K. Hausknecht, D. Foit, and J. Buric. RAM data significance in digital forensics. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1372–1375. IEEE, may 2015.

[9] B. Hitchcock, N.-A. Le-Khac, and M. Scanlon. Tiered forensic methodology model for Digital Field Triage by non-digital evidence specialists. *Digital Investigation*, 16:S75–S85, mar 2016.

[10] N. Joseph, S. Sunny, S. Dija, and K. L. Thomas. Volatile Internet Evidence Extraction from Windows Systems. In *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5. IEEE, dec 2014.

[11] D. King. dlib C++ Library, 2016.

[12] M. B. Koopmans and J. I. James. Automated network triage. *Digital Investigation*, 10(2):129–137, sep 2013.

[13] Y. Li, S. C. Sundaramurthy, A. G. Bardas, X. Ou, D. Caragea, X. Hu, and J. Jang. Experimental Study of Fuzzy Hashing in Malware Clustering Analysis. In *8th Workshop on Cyber Security Experimentation and Test (CSET 15)*, 2015.

[14] F. Marturana and S. Tacconi. A Machine Learning-based Triage methodology for automated categorization of digital media. *Digital Investigation*, 10(2):193–204, sep 2013.

[15] D. McClelland and F. Marturana. A Digital Forensics Triage Methodology based on Feature Manipulation Techniques. In *2014 IEEE International Conference on Communications Workshops (ICC)*, pages 676–681. IEEE, jun 2014.

[16] G. McDonald. strings2: An improved strings extraction tool, 2013.

[17] A. Michalas and N. Komninos. The lord of the sense: A privacy preserving reputation system for participatory sensing applications. In *Computers and Communication (ISCC), 2014 IEEE Symposium*, pages 1–6. IEEE, 2014.

[18] Norsys Software Corp. Netica, 2016.

[19] J. Okolica and G. L. Peterson. Extracting the windows clipboard from physical memory. *Digital Investigation*, 8:S118–S124, aug 2011.

[20] N. Paladi, C. Gehrmann, and A. Michalas. Providing user security guarantees in public infrastructure clouds. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2016.

[21] N. Paladi, A. Michalas, and C. Gehrmann. Domain based storage protection with secure access control for the cloud. In *Proceedings of the 2014 International Workshop on Security in Cloud Computing*, ASIACCS '14, New York, NY, USA, 2014. ACM.

[22] C. Platzer, M. Stuetz, and M. Lindorfer. Skin Sheriff: A Machine Learning Solution for Detecting Explicit Images. In *Proceedings of the 2nd international workshop on Security and forensics in communication systems - SFCS '14*, pages 45–56, New York, USA, jun 2014. ACM Press.

[23] D. Quick and K.-K. R. Choo. Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273–294, dec 2014.

[24] I. Ray and S. Shenoi. Reasoning about Evidence using Bayesian Networks. In *Advances in Digital Forensics IV*, pages 275–289. Springer, New York, USA, 2008.

[25] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816. AAAI Press, jul 1993.

[26] M. K. Rogers, J. Goldman, R. Mislan, T. Wedge, and S. Debrota. Computer Forensics Field Triage Process Model. In *Proceedings of the Conference on Digital Forensics, Security and Law*, volume 1, pages 27–40, 2006.

[27] H. Said, N. Al Mutawa, I. Al Awadhi, and M. Guimaraes. Forensic analysis of private browsing artifacts. In *2011 International Conference on Innovations in Information Technology*, pages 197–202. IEEE, apr 2011.

[28] M. Simon and J. Slay. Recovery of Skype Application Activity Data from Physical Memory. In *2010 International Conference on Availability, Reliability and Security*, pages 283–288. IEEE, feb 2010.

[29] A. Sindelar, A. Moser, J. Stuettgen, J. Sanchez, M. Cohen, and B. Misha. Rekall Memory Forensic Framework, 2016.

[30] K. M. Syring. GNU utilities for Win32, 2014.

[31] The Volatility Foundation. Volatility 2.4 (Art of Memory Forensics), 2014.

[32] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. HÃijbsch, and I. Paraskakis. Paasword: A holistic data privacy and security by design framework for cloud services. In *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, pages 206–213, 2015.

[33] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. Hübsch, and I. Paraskakis. Paasword: A holistic data privacy and security by design framework for cloud services. *Journal of Grid Computing*, 15(2):219–234, Jun 2017.

[34] S. Vömel and F. C. Freiling. Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digital Investigation*, 9(2):125–137, 2012.

[35] R. J. Walls, E. Learned-Miller, and B. N. Levine. Forensic triage for mobile phones with DEC0DE. In *SEC'11 Proceedings of the 20th USENIX conference on Security*, page 7. USENIX Association, aug 2011.