Chapter 12
Towards Collaborative Forensics

Mike Mabey and Gail-Joon Ahn

Abstract Digital forensic analysis techniques have been significantly improved and evolved in past decade but we still face a lack of effective forensic analysis tools to tackle diverse incidents caused by emerging technologies and the advances in cyber crime. In this paper, we propose a comprehensive framework to address the efficacious deficiencies of current practices in digital forensics. Our framework, called Collaborative Forensic Framework (CUFF), provides scalable forensic services for practitioners who are from different organizations and have diverse forensic skills. In other words, our framework helps forensic practitioners collaborate with each other, instead of learning and struggling with new forensic techniques. In addition, we describe fundamental building blocks for our framework and corresponding system requirements.

Introduction

Computer crime has swiftly evolved into organized, and in some cases state sponsored, cyber warfare. The tools digital forensic examiners currently use are too limited to take on the challenges that are rapidly approaching their forensic cases. Before long, fundamental changes in the industry will make many of the forensic techniques used today obsolete [15]. Although many contributing elements can be identified, the heart of the problem is that current digital forensic examinations are

A preliminary version of this paper appeared under the title "Towards collaborative forensics: Preliminary framework" in Proc. of the IEEE International Conference on Information Reuse and Integration (IRI), 2011. All correspondences should be addressed to Dr. Gail-Joon Ahn at gahn@asu.edu

M. Mabey · G.-J. Ahn (✉)
Laboratory of Security Engineering for Future Computing (SEFCOM), Arizona State University, Tempe, AZ 85281, USA
e-mail: mmabey@asu.edu; gahn@asu.edu

T. Özyer et al. (eds.), *Information Reuse and Integration in Academia and Industry*,
DOI 10.1007/978-3-7091-1538-1_12, © Springer-Verlag Wien 2013

too time-inefficient. The three principal causes of this inefficiency are summarized as follows:

Software Limitations: Single workstation computers have served as the primary tool of our society's computing needs for a long time. With the evidence data sets being as large as they are, a single computer simply does not have the resources to deliver sophisticated analysis results in a timely manner.

Size of Evidence Data: Today a 1 TB hard drive can be purchased for about US$60 and the average hard drive cost per GB is less than US$0.10 [6]. Such low cost makes terabyte-sized systems commonplace among even non-tech-savvy consumers. With such a proliferation of huge storage systems filled with user data, examiners are confronted with a mountain of stored data to work through [31]. The problem is compounded when the situation involves a redundant array of independent disks (RAID) [34] or network attached storage (NAS) unit shared among individuals or employees.

Increased Examiner Workload: As if insufficient tools and large datasets were not enough, digital crime continues to increase in popularity [17, 24, 25], naturally resulting in more investigations. Furthermore, state-sponsored cyberwar promotes the development of increasingly sophisticated software. Simply trying to keep up with the latest methods of penetration, exfiltration, and attack is insufficient to accommodate the pace of digital crime.

In addition, when cases become backlogged, only those designated as more urgent are worked on, potentially leaving suspects' co-conspirators at large and capable of making more victims out of innocent people.

Motivation

The challenges above can be greatly reduced by a secure and robust infrastructure that facilitates *collaborative forensics* [18, 27], which we define as the willful cooperation between two or more forensic examiners during any step in the forensics process, for the benefit of sharing specialized knowledge, insight, experience, or tools. By this we mean to indicate a process and system through which multiple examiners perform their work, using a common interface that provides the means for carrying out all steps of the examination process as well as providing mechanisms for collaboration between the users.

Two advantages of collaboration are of particular interest to us. First, collaboration allows people to draw from others' expertise, which is invaluable when working on problems of a diverse nature or when the problem set of a job constantly changes. Second, collaboration is a method of spreading a workload, which results in less time needed for the job to be completed.

Consider the following hypothetical scenario which illustrates a need for a better method of collaboration. While investigating a case with multiple computational and storage devices that are uncommon, Bob, who is the lead examiner, determines that by soliciting the aid of two subject matter experts that he trusts, the devices could be successfully examined for evidence of interest. However, when Bob makes the request to his supervisor to obtain assistance from these experts, she informs him that the compensation expenses of the experts' consultation fees plus travel costs is too great to justify with the current budget. Bob must find a way to either make do with only one of the experts or to eliminate the travel expenses. Bob needs an effective means of collaborating with these experts remotely.

Similar to the above scenario, it is already quite common for evidence seizure to yield a variety of digital evidence, such as a mix of Windows, Linux, and Mac computers, as well as cell phones, GPS devices, gaming consoles, etc. Since examiners must be certified to work on a particular type of evidence (depending on the investigating agency), such a workload must be split up among personnel. Since there is no tool which can accommodate all evidence types, the evidence presentation lacks uniformity in format and structure.

While many generic collaboration solutions exist today, none of them have been crafted specifically for the needs of the digital forensics industry. To be truly effective, a collaborative forensics infrastructure should maintain the strict privacy and integrity principles the discipline demands, while also giving examiners the flexibility to communicate however is best for the situation. This demands a level of robustness that is simply not offered by collaboration tools at present.

Beyond just communication, collaboration also implies a sharing of resources. For a proper exchange of data (whether it be files needing to be analyzed or the results of an analysis), there must first be a uniform representation of that data, and then a common storage space solution where all collaborators can keep their resources secure. This will require the establishment of standards to ensure that all parties can access and interpret the data. Means to efficiently manage resources will also be needed.

If examiners are to collaborate on a large scale, it will also be crucial for this infrastructure to provide vast amounts of computing power, which is best accomplished through some distributed processing method. Ideally, a distributed processing solution would also include scalable resources. Because there is not a single technological solution that will properly meet this need for all organizations, there must be a generic way to interface for such processing resources.

To best facilitate collaboration among examiners, a collaborative forensics solution should not be limited to supporting its use on a small number of operating systems. This would hinder the collaboration process and may exclude experts who could offer potentially crucial insight.

The rest of this paper is organized as follows. We first discuss the progress made by others in related fields in section "Related Work". In section "CUFF: Collaborative Forensic Framework" we provide the architecture of our solution, which is an abstraction of the most essential components. We then introduce all other necessary components and provide details on how to realize our framework in section "Realization of CUFF". Section "Conclusion" concludes this paper by summarizing our contributions and discussing our future work.

Related Work

The nature of our work is such that it brings together aspects of other, previously completed works which we discuss here by topic.

General Digital Forensics: Two challenging types of evidence that forensics examiners need to be able to analyze at times are Redundant Array of Independent Disks (RAID) storage systems and drives protected with encryption. In [34], Urias and Liebrock attempted to use a parallel analysis system on RAID storage systems, and documented the issues and challenges they faced with that approach. Similarly, multiple methods of properly handling the challenges presented by encrypted drives have been presented by Casey and Stellatos in [5] and by Altheide et al. in [2].

Distributed Processing for Forensics: With distributed processing in use so much today and in so many distinct settings, it is natural to think of using it to divide the workload of digital forensics processing. Several years ago, when the use of distributed processing was not yet as common as it is today, Roussev and Richard proposed a method for moving away from single workstation processing for forensic examination to a distributed environment [31]. A few years later, Liebrock et al. proposed improvements upon Roussev and Richard's system in [21], which introduced a decoupled front-end to a parallel analysis machine.

In [32], Scanlon and Kechadi introduced a method for remotely acquiring forensic copies of suspect evidence which transfers the contents of a drive over a secure Internet connection to a central evidence server. While this effort is a step for the better in terms of making evidence centrally accessible, it is difficult to see the direct utility of such an approach without accompanying software or analysis techniques to take advantage of storing the evidence on a server. Furthermore, the presented approach relies on either using the suspect's Internet connection to upload the image or images, or the use of a mobile broadband connection. Given the relatively abysmal upload speeds for current mobile broadband when dealing with data sets that are hundreds of gigabytes or even a few terabytes large, this approach will continue to be prohibitively inadequate.

Forensics Standardizations: Garfinkel has made great efforts to create standards to improve the overall digital forensic examination process. Garfinkel et al. presented the details of a forensic corpora in [16] with the purpose of giving researchers a systematic way to measure and test their tools. Garfinkel took this a step further in [13] with his work to represent file system metadata with XML. Finally, in [15] Garfinkel put forth a challenge to researchers and developers everywhere to take note of the current industry trends and take them head on with innovative forensic solutions that match the properties of emerging technologies.

Storage: Since our realization of our framework is built upon a cloud, we also consider work done by researchers to address some of the issues related to shared storage in a cloud. Du et al. proposed an availability prediction scheme for sharable objects, such as data files or software components, for multi-tenanted systems in [8]. In [36], Wang et al. introduced a middleware solution to improve shared IO performance with Amazon Web Services [3]. Increasing the security of the data stored in a cloud has been improved upon by Liu et al. in [22] and by Zhao et al. in [38].

In addition to the above subject areas, there also appears to be a trend toward supporting collaboration mechanisms in digital forensics tools such as FTK 3 [11]. But, to the best of our knowledge, there has yet to be a single system which can satisfy all the functionalities set forth in section "Introduction" in a truly robust manner.

CUFF: Collaborative Forensic Framework

Based on the features and requirements necessary to achieve collaborative forensics as enumerated in section "Introduction" and the work presented in [23], this section describes our framework, called Collaborative Forensic Framework (CUFF), and elaborates what mechanisms are needed to facilitate these features. As illustrated in Fig. 12.1, our framework consists of four core components (i) to mediate communication between components in the system, (ii) to coordinate the distributed analysis processing, (iii) to maintain the shared storage space, and (iv) to provide a basic interface to the system for the user interface. While a precise set of APIs for these four components may vary for the deployment setting, they should always fulfill specific foundational operations and always have the same basic interactions with the other components. We now discuss these two points in context of each component:

Analysis Block: This component is the workhorse of the system, and in truth all other components are simply in place to either provide an interface to it, or to facilitate its proper function. The Analysis Block is composed of a controller as well as all processing resources. Ideally, the processing resources would be quite substantial and capable of handling a continuous inflow of analysis jobs of significant size. The controller will receive a large number of analysis requests, and is expected to enqueue and dequeue each job request in an organized and efficient manner, which should also be fault-tolerant and maintain a high level of responsiveness. Because it is in charge of maintaining the queue of jobs, the controller oversees the processing resources and ensures that they are used properly according to a selected method of prioritizing the jobs.

Storage: This component keeps track of all acquired disk images, the analyses of their contents, comments and notes from users, and related information all need to be kept for performing forensic tasks. To do this, it must accept incoming data streams of acquired disk images, and strictly maintain the integrity of the data through validation of the original checksums. Requests for getting and putting data to and from the storage component will come at a high rate, particularly

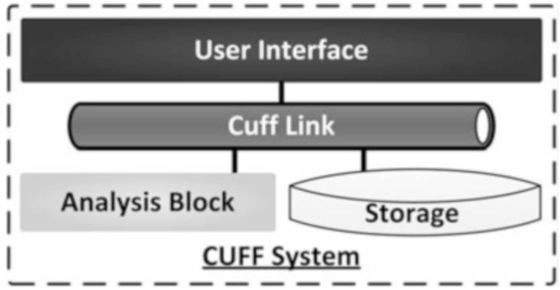

Fig. 12.1 Each of the components in CUFF fulfill one of the four main objectives of the system. All inter-system communication passes through the Cuff Link, and the end user only interacts with the user interface component

from the processing resources in the Analysis Block, so the storage component's response time needs to be controlled. To maximize reusability, a generic method of transferring data to and from the storage unit should be used such that distinct data types (such as analysis results, user comments, and communications between users) will not need any specialization made to the system. In coordination with whatever access control mechanism is implemented, the storage component also maintains strict confidentiality of the data it stores. The storage component must also be flexible enough to allow temporary and/or limited access to case data for subject matter experts conducting consultation work, allowing them to collaborate with those directly responsible for the case.

User Interface: This is the access portal through which all the system's features are made available. More specifically, the user interface supports evidence acquisition, allows users to view the structure and contents of files, accepts requests for specific analyses to be performed on files or groups of files, and provides a means for users to communicate and share data and information with each other.

Cuff Link: This component mediates communication between all other components in the system. It validates parameter input and stores location information for each of the other components. Also, since it is the component that manages the forensic process, it is responsible for assigning examiners jobs and notifying supervisors when the work on a case has been completed. The Cuff Link maintains order in the system by dictating the available APIs for each of the other components. It also simplifies the implementation of other components by reducing the number of connections they must make down to one.

Realization of CUFF

In this section, we describe how to realize the CUFF framework using commercial off-the-shelf (COTS) software and open source tools. For each component, we address the desired functionality, some of the challenges associated with achieving such functionality, and what tools and software meet these challenges and why.

It is highly desirable for an implementation of CUFF to be easily accessible by the users that will collaborate through it, to have scalable resources, and to have built-in redundancy for fault-tolerance. While it would be possible to implement CUFF as a set of desktop applications that communicate with other remote installations through a peer-to-peer networking architecture, such an approach would be difficult to monitor and assure that all connected parties strictly abide by the rules of evidence.

To achieve the system-wide features we desire, we have selected to build upon a cloud-based infrastructure, deploying each of the components as virtual machine (VM) instances. Using a cloud architecture has many obvious advantages. One advantage is that VM instances can be spawned quite easily, improving the scalability as well as the reliability and recovery time of all the components of the system. A second advantage is derived from the fact that each of CUFF's features are made accessible through various web services, including a web interface that can be accessed and used by all authenticated users. Although most of the web services in the system are only accessible internally, the use of web standards increases the composability of the system.

While several cloud architectures exist, OpenStack [29] stands out as one built for a high level of flexibility and scalability while also exporting an API compatible with Amazon EC2 and S3 services [3], hence allowing the use of the widely-used euca2ools [10] set of cloud administration tools.

We now elaborate on our implementation of each of CUFF's components. Although much of this section is dedicated to discussing a messaging protocol (section "Scheduling Analysis Jobs"), we would propose that efficient collaboration among forensic examiners depends heavily on the intelligent appropriation of the analysis resources, which begins with the scheduling of their use. Hence, this is a core component to address properly as we work towards our goal of facilitating collaboration.

Cuff Link

As stated earlier, the Cuff Link provides a couple of key functions for the overall system. It mediates communication, manages the forensics examination flow, validates input, and exposes an API for the other components.

One thing that must be taken into consideration is that not all communication types used in CUFF have the same behavior. Some types of Internet traffic are difficult to stop and process before sending it on to its intended destination, such as uploading or downloading files. However, with other types of traffic there is no difficulty in intercepting, processing, and then forwarding the messages being sent, such as requests for certain evidence files to be analyzed.

With this in mind, it would not be wise to impose a single method of handling communication. Rather, the Cuff Link uses multiple technologies appropriate for the type of messages being handled. Because of this, it was necessary to divide the functionality of the Cuff Link such that it acts as a layer of abstraction in

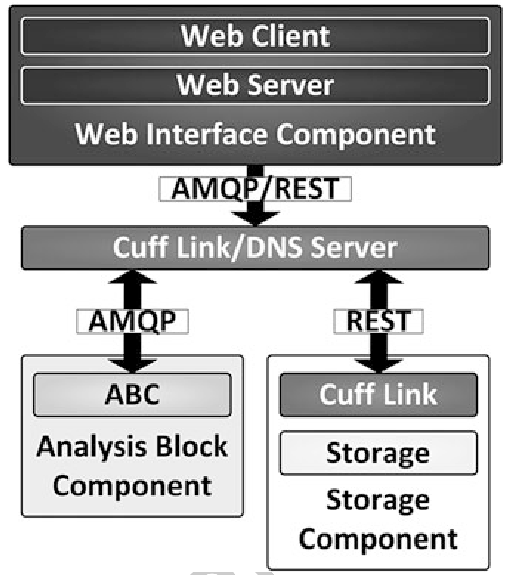

Fig. 12.2 Inter-component communication passes through the Cuff Link, which is deployed in multiple locations to provide the layer of abstraction necessary for the various components

multiple locations. This is illustrated in Fig. 12.2 where an element of the Cuff Link is running on the Storage component.

To accommodate the communication types that cannot be interrupted (which is limited to traffic to and from the Storage component), the Cuff Link first provides a domain resolution through Domain Name System (DNS) server, which translates URLs into IP addresses. This makes it easy for other CUFF components to send traffic to a destination without knowing its exact location. The component making a request only needs to specify the generic name for the destination server, such as http://cuff.storage.example. The Cuff Link DNS server can then resolve the name to the appropriate server.

Typically, this kind of action would require that the URL http://cuff.storage.example be registered with some authoritative entity that stores all official URLs. However, because we have configured the Cuff Link as a primary master name server for the domains used within CUFF and specified that the system's components should query the Cuff Link before any other servers, such URLs are resolved within the system without making an external DNS query. This does require that any CUFF components that need to be accessed by this means have their IP address associated with their appropriate domain by the Cuff Link. Once this has been done, however, the DNS server can also potentially perform some level of load balancing among available servers by rotating which server's IP address it uses as the resolution of the URL.

The second thing the Cuff Link does to accommodate communication with the Storage component is to leverage a Representational State Transfer (REST) web service on the Storage component. REST web services allow for certain actions (in this case uploading and downloading files) to be specified in the URL of the web request, using the type of web request (POST, GET, DELETE, etc.) as one factor for interpreting what action should be taken. For example, a GET request in the form http://cuff.storage.example/listing/2398-56-1-9125 is interpreted by first removing the base URL, leaving listing/2398-56-1-9125, which is a request for the evidence listing for the case number 2398-56-1-9125.

Each different type of web request is interpreted in a specific way. Within the logic of these web services, we are able to perform the input validation and mediation necessary for other components to access storage resources. Additionally, by adding the appropriate filters, we can manage the forensics examination flow by noting when certain events take place and taking action. For example, when a new device image is being uploaded, the Cuff Link can easily recognize this event and perform a predefined action, such as notify the appropriate supervisor that the new case needs to be assigned to an examiner.

The communication in CUFF that is of the type that can be easily interrupted before reaching its destination is typically being sent to or from the Analysis Block (traffic of this type are discussed in more detail in section "Scheduling Analysis Jobs"). As messages are sent to the Analysis Block, they are first sent to the Cuff Link and checked to ensure that the analysis request is well-formed and that the specified Analysis Block is within a reachable domain. This would be the mechanism whereby multiple deployments of CUFF could share analysis resources.

Storage

Similar to the communication in the system, there are two types of storage needs in CUFF. First, because it is built on a cloud, there is a need for some way to store the VM images that run in the system. This storage need is distinct because VM images are large, rarely change, but also may be needed to start up an instance very quickly.

Second, because cloud instances cannot store any persistent data within the image itself, all data must be stored in a container suited for the particular purpose of being temporarily attached to an instance and storing any data that needs to be preserved. Examples of this type of data includes evidence images, evidence analysis results, and database files. Evidence images will need to always be accessed in a read-only mode to preserve their integrity. Furthermore, the rules of evidence dictate that the system have a means of conducting logging and auditing on the access of any stored data in the system.

To accommodate these features, we take advantage of the two storage facilities available from the OpenStack architecture. As shown in Fig. 12.3, these facilities are distinct, but together they satisfy the needs of our framework. The first is Swift, an object storage component that, when used in connection with Glance OpenStack's image service, can provide discovery, registration, and delivery services for virtual disk images through a REST web interface. As such, Glance will act as the image registry for the system.

The second storage facility we use is volumes, which are similar in functionality to Amazon's Elastic Block Storage [3]. Each volume is labelled with a universally

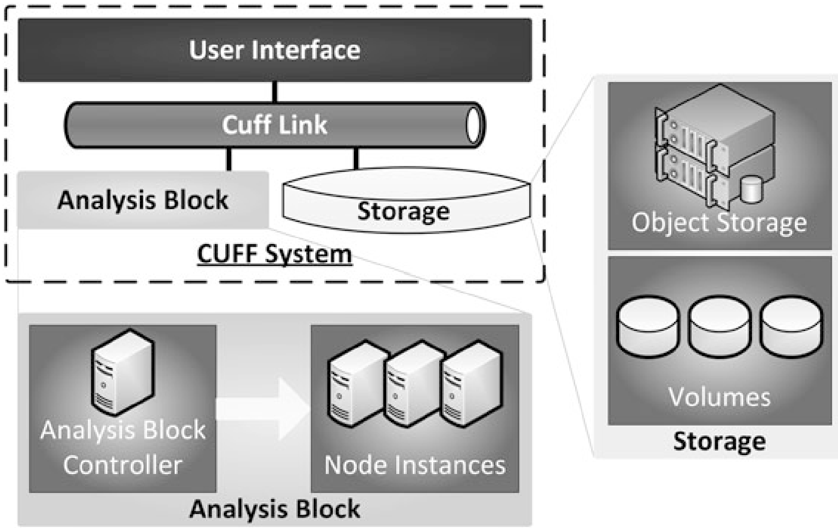

Fig. 12.3 The Analysis Block and Storage components are composed of multiple parts, each filling a separate role

unique identifier (UUID)[1] that distinguishes it from other volumes. The primary use of these volumes will be to store evidence images and analysis results. Typically, a separate volume will be created for each device uploaded to the system. If a hard drive is seized that contains multiple partitions, each partition may also be stored on a separate volume, to be determined by the examiner when uploading the evidence to the system. Analysis results will be stored on volumes separate from the evidence to which they pertain, but will store the UUID of the corresponding evidence volume to keep the two connected.

In order to maintain the integrity of evidence images stored in volumes, a snapshot is taken of each volume immediately after the upload is complete. This essentially makes the original volume read-only because, although changes are technically allowed to the volume, all write operations are saved in a "child volume" that is separate from the original and can be easily discarded when the volume is no longer being accessed by an instance.

One challenge that arises from doing automated, distributed analysis is finding an efficient means of referring to and transferring portions of evidence images (e.g. files or file segments). This is inherently a storage issue, because it is the Storage component that provides access to this data. And while it is true that because we have made snapshots of the evidence image volumes, we can technically attach the root volume to multiple instances in a read-only fashion, this still requires that

[1] UUIDs are 128-bit numbers that are used in distributed systems to uniquely identify information. The assurance that a UUID is in fact unique is derived from the number of theoretically possible numbers, which is about 3×10^{38}. Because of this, UUIDs are used to identify the volumes in the system.

```
<fileobject>
  <filename>README.txt</filename>
  <id>2</id>
  <filesize>43</filesize>
  <partition>1</partition>
  <alloc>1</alloc>
  <used>1</used>
  <inode>6</inode>
  <type>1</type>
  <mode>511</mode>
  <nlink>1</nlink>
  <uid>0</uid>
  <gid>0</gid>
  <mtime>1258916904</mtime>
  <atime>1258876800</atime>
  <crtime>1258916900</crtime>
  <byte_runs>
   <run file_offset='0' fs_offset='37376' img_offset='37888' len='43'/>
  </byte_runs>
  <hashdigest type='md5'>2bbe5c3b554b14ff710a0a2e77ce8c4d</hashdigest>
  <hashdigest type='sha1'>b3ccdbe2db1c568e817c25bf516e3bf976a1dea6</hashdigest>
</fileobject>
```

Fig. 12.4 This portion of a DFXML file shows how a single file object is stored with all its metadata. In the case where a file is fragmented in the disk image, multiple <run> tags will be contained under the <byte_runs> section

the instance have access to the entire image. An approach that allows for concise data transfer and thorough data representation would be better.

Garfinkel's Digital Forensics XML (DFXML) representation for file system metadata [13, 14] is just such an approach. We employ DFXML in CUFF to aid in improving the efficiency and standardization of how CUFF stores and transmits data. DFXML provides a standard way of representing and accessing the contents of an imaged drive by using an XML file to store the offsets and lengths of all "byte runs" (file fragments) on the disk, thereby acting as an index for the image. Using this DFXML file, an entity can access specific files by simply specifying the byte runs of the file and concatenating the returned results. An example of how a file's information is stored in a DFXML file is shown in Fig. 12.4.

One issue we discovered in working with DFXML is that Garfinkel's tool [12] for creating DFXML files from acquired disk images was that the tool currently only produces a simple Document Type Definition (DTD) specification for each DFXML document, which doesn't allow for type validation. To help encourage the adoption of DFXML as a standard, we have created an XML schema detailing tag hierarchy and complex data types. Using this schema to validate an image's file system representation, any digital forensic tool can reliably use this standard in its interactions with the disk image. We believe such a schema will help developers of forensic tools to be more willing to adopt this data format as a standard, because they have the assurance of precise data types with any DFXML file.

Analysis Block

As stated previously and as depicted in Fig. 12.3, there are two types of components that make up the Analysis Block, the Controller and the Nodes. The Analysis Nodes act as a network of VM instances that can be used in two different ways. First, nodes can be used as a means to perform distributed, automated analysis. In this case, a node is sent files to be analyzed, and the tool is executed from the command line, performing the requested analysis in an automated fashion. Because analysis node images are virtual machines, they can be configured to run a wide selection of operating systems, maximizing platform compatibility.

The other way of using the nodes is to manually interact with them and exercise fine-grained control over the work performed. A node is sent the files and the specified program begins execution, but the node does not indicate to the program to begin analyzing the files. Instead, the node enters a waiting state until the user accesses it remotely from the system's dashboard. At this point, the user is in complete control of the node and can perform whatever functions are necessary.

Since it is required that all inter-component communication in the system go through the Cuff Link, communication is to be standardized and regulated through the use of agents which run on all nodes in the system. While all node agents will be programmed with a standard set of communication protocols, each distinct analysis node's agent will be customized to the analysis programs being hosted on that image. This allows the agent to store whatever parameters necessary to interface with the programs as well as retrieve the analysis results. Because much of the implementation for these nodes will be the same for all node types, this improves the ability to flexibly support new file systems, operating systems, analysis algorithms, and so forth.

Forensic Flow

The most important feature of the entire system is the fact that it accommodates the main tasks of any digital forensic investigation. Since most of these tasks involve the Analysis Block in some way, we discuss the connection of each task to the Analysis Block.

1. *Acquisition*: When a user is uploading a new evidence image to the system, the destination volume for that image is mounted to a VM instance crafted specifically for handling this task. During the upload, the instance generates checksums of the image, which are validated against the checksums of the original device, and then stored for later use during evidence validation after operations are performed on the image. Finally, the instance takes a snapshot of the volume to effectively seal it from further changes.
2. *Validation*: In order to guarantee the integrity of images and files in the system, Analysis Nodes are utilized to calculate and validate the checksums before and after every data transmission.

3. *Discrimination*: By creating a VM image that can use sets of checksums of known good files (such as the Reference Data Set provided by the National Institute of Standards and Technology [28]), the system can highlight those files which are unknown for the examiner, effectively eliminating an extensive number of files they need to look at.

To support such an event-based forensic workflow and accommodate relevant workflow management features in CUFF, we consider existing approaches on a web-based workflow systems [7, 20, 26]. Especially, we believe workflow modeling approaches [9, 33] would help design and govern forensic flow and related tasks in CUFF. In addition, BPEL (business process execution language) would be another candidate to articulate a particular forensic flow for facilitating web-based events [19].

Scheduling Analysis Jobs

One of the merits of the Analysis Block's design is that it not only provides a collection of resources for analyzing evidence, but also does so in a very generic fashion, making it very reusable. The Analysis Block Controller (ABC) is indifferent to both the nature of the analyses to be performed as well as what the requirements are for the operating system or software used to perform the work.

While constructing a generic controller, we realized that the system had a great need for scheduling automated analysis jobs, so we created a set of utilities suited to accommodate this need. Our scheduling utilities have three specific purposes. The first purpose is to package information regarding analysis jobs for the entities in the system that will carry out the work. The information packaged includes the type of analysis to be carried out, the location of the subject of the analysis (i.e. the files to be analyzed), and some sort of ordering or priority information for the analysis subject.

The second purpose of the utilities is to create a channel by which the information described above can travel from the user who inputs it to its end destination, meaning the Analysis Node that will perform the analysis specified in the job information. Creating this channel implies that there is a path defined for the job that passes through multiple components of the CUFF system.

The third and final purpose of the utilities is to queue jobs according to the ordering or priority specified in the job information, and then to distribute jobs to available nodes upon receipt of an assignment request.

A few significant properties of these objectives emerge upon examination that should be highlighted. As job information is packaged, a data encapsulation method must be chosen that is standard and efficient. The efficiency of both passing the data along the defined path through the system and interpreting it is important for preventing the utilities from becoming a bottleneck, especially since there is a one-to-many relationship between the Analysis Block Controller and the Analysis Nodes. No upper bound is imposed on the number of nodes in the system, nor is there any restriction on the configurations of the nodes, which again emphasizes the importance of using standard data formats. Since the nodes submit requests for job assignments, the scheduling utilities do not preempt the work of nodes. Finally, as the utilities distribute jobs, care must be taken to prevent starvation of jobs with lower ordering or prioritization.

Multiple approaches for ordering or prioritizing can be adopted depending on what is most important for the deployment domain. One approach would be to calculate the resource demands of each job on the system using the order of complexity for each of the available analysis tools, an estimated turnaround time for a small baseline job, and the size proportion difference between the baseline job and that of a queued job request. With these values, CUFF could prioritize jobs so that those jobs with a significant workload on the system are either spread out among jobs with a lesser demand or are delayed until off-peak hours.

A second approach would be to assign priority based on the importance of the case of which it is a part. At times, one investigation will be more pressing than all others currently being worked on. In such a scenario, the case that has a higher level of criticality should be given priority over other analysis work being done in the system. By giving the user the ability to specify what criticality level a certain case has been given, jobs related to that case can be allotted more time for being analyzed by the system resources.

In our deployment of CUFF, we have implemented the second prioritization approach. We reiterate that this scheduling scheme is specifically for automated analysis. In other words, it is purposed for when users submit batches of requests to analyze data segments which will then be carried out without any further input from users. We anticipate examiners will interact more closely with VM instances in the cloud on occasion, but that is a use case distinct from the one we address here.

To begin describing the behavior of the utilities, we first identify how we have satisfied the purposes of the utilities as set forth earlier. First, the container format used for all messages is JavaScript Object Notation (JSON), which is both easily transmitted and easily interpreted from this notation to programming objects and vice versa. Second, to satisfy the needs for a path through the system components, a queuing mechanism, and a distribution method, we use RabbitMQ [30], an implementation of the Advanced Message Queuing Protocol (AMQP) [1]. RabbitMQ is a messaging broker, which allows for a common yet generic method for passing messages between components by creating queues for messages, producers that put messages into queues, and consumers that take messages out of the queues.

Figure 12.5 shows the sequence of how messages travel through CUFF using RabbitMQ. The process is initiated at step 1, when the user submits a batch of analysis requests to be done for a case. In step 2, each request is processed by the web server and is reformed to the format understood by RabbitMQ, such that the list of files for a single analysis request is stored in the body of the message, as are the criticality of the request and the tool or algorithm to be used on the files. If the tool or algorithm needs specific command line parameters, these are also stored in the body of the message. After the message has been properly crafted, it is passed

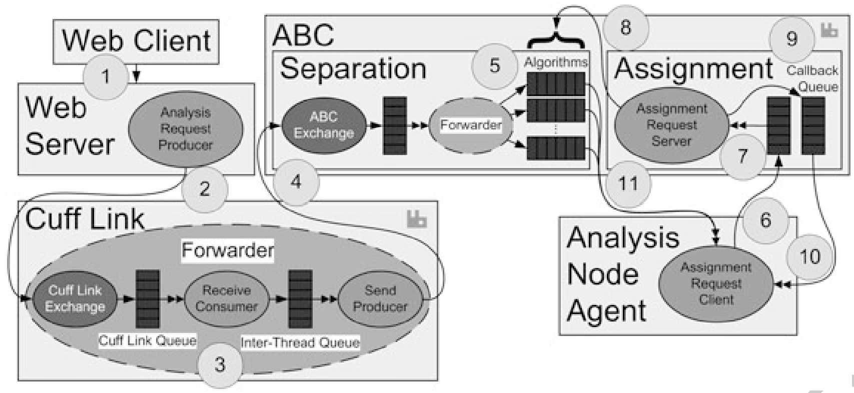

Fig. 12.5 The path of job request messages being passed through CUFF. *Single-headed arrows* indicate a message being sent to the broker, and *double-headed arrows* indicate a message being retrieved from the broker. Components with the *RabbitMQ logo* in the *top right* corner are running the messaging server

to the Analysis Request Producer. From this point on the message is in the care of the broker until it is delivered to an Analysis Node.

In step 3, the message is sent to the Cuff Link that is running a forwarder utility. In a deployment of CUFF that does not interact with other deployments, this utility is fairly insignificant because it simply passes messages on to the Analysis Block Controller. However, for deployments that interact with each other and allow jobs from one organization to be analyzed on another deployment, this is the mechanism that would be responsible for directing messages to the other deployment's Cuff Link. The forwarding of messages in this manner would depend on the name server of the first deployment knowing how to resolve the domain names of any connected deployments.

After a message has been received by the appropriate Cuff Link, it is forwarded to the Analysis Block Controller in step 4. In step 5 each message is put into a queue with messages that have both an equivalent criticality level as well as the same analysis type. Messages remain in these queues until retrieved by an Analysis Node Agent.

Before we proceed, it is appropriate that we discuss the behavior of Analysis Node Agents. As illustrated in Fig. 12.6, each Analysis Node follows a specific set of state transitions which are governed by the agent running on the node. During most of a node's time running in the cloud, it will be working on an analysis task for a job that it has been assigned. When the job is done, the agent sends the results to the Storage component and submits a job assignment request. An assignment indicates to the agent from which queue to take a job. Upon requesting a job from a particular queue, the agent will either be informed the queue is empty, in which case a subsequent assignment request would be made, or it will have the necessary information to retrieve the files from the Storage component and begin the analysis.

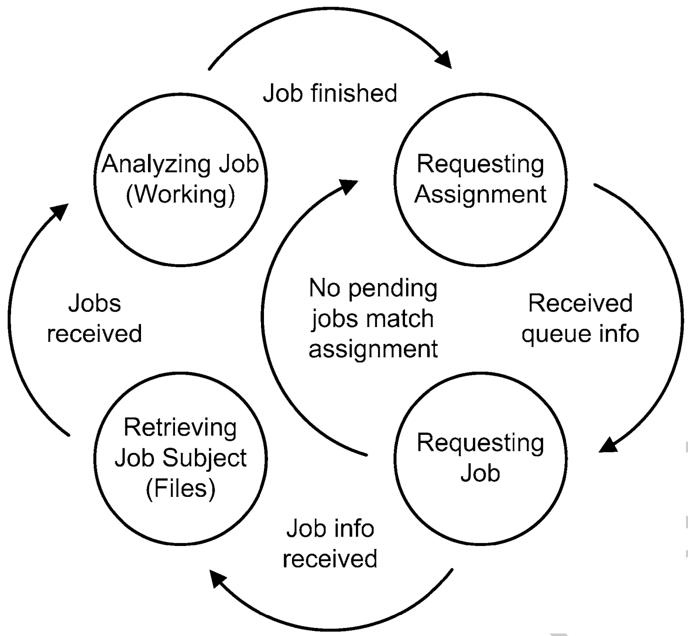

Fig. 12.6 The state transitions for Analysis Node Agents

In step 6, the Analysis Request Client segment of the node agent sends a job request message to the Analysis Request Server on the ABC. During a course of step 7, all assignment request messages are stored in a queue and are processed in the order they were received. The Analysis Request Server then utilizes a read-only interface to the job queues to evaluate the best candidate for the node in step 8.

Next, in step 9, the Analysis Request Server puts this information into the callback queue that was specified as part of the message body from step 6. It is noted that although it technically fulfills the purpose of a queue, we have designed our use of the callback queue to store at most one element. The reason for this is that job description messages sent to it will only be in response to an assignment request message, which will only be sent to the ABC when the node has completed a previous analysis job, at which point it takes step 10 and consumes the contents of the callback queue. Hence, the callback queue is only used because it is required by the broker.

Finally, having received an assignment, the node agent takes one of the jobs from a queue in step 11. At this point, the agent can get the files from the Storage component and begin the analysis.

In addition, for handling the assignment request messages, each message contains the necessary information for the ABC to make an appropriate job assignment, namely, the types of analyses the node can perform as well as the desired level of criticality. This turns out to be quite a critical element in this scheme, because it is the node agent that requests what criticality level the job should have that is assigned to it. This means that the anti-starvation requirement is satisfied in the implementation of the node agent, which keeps track of the quantities of jobs *completed* for each distinct criticality level. Then, before the agent submits an assignment request, it first compares the ratios of completed jobs for each criticality level with the ratio specified by the system administrator and selects the most outlying level to include in the request.

Evaluation of Scheduling Utilities

The overall viability of CUFF as a digital forensics analysis framework depends on its ability to distribute the work of analyzing evidence by scheduling nodes to be responsible for smaller atomic portions of the analysis work. Therefore, to demonstrate that our method of scheduling jobs is also viable, and to support our claim that the Analysis Block Controller functions efficiently and is scalable enough to facilitate collaboration between examiners, we present our set of tests which simulate real analysis jobs being assigned to nodes and executed.

In terms of execution profile, the procedure of accepting and separating job request messages (steps 1–5 in Fig. 12.5) is remarkably different from the procedure of assigning jobs to nodes (steps 6–11 in Fig. 12.5). The former will occur in bursts of batches as practitioners submit groups of jobs to the system and will not have a continual inflow. The latter will be a steady disbursement of jobs one at a time to nodes as they become ready. Because of such a difference, it is less important that the separation procedure be time-efficient and more important that it provide reliable delivery of every single message to the Analysis Block, whereas the assignment procedure should be likewise reliable but also expeditious to manage queued jobs and respond to each assignment request from nodes so as to minimize their wait time between analyses for the greatest possible throughput.

One challenge in evaluating the performance of the scheduling utilities is the fact that processes are running on completely separate VM instances in the cloud and hence have separate system clocks. For the separation procedure, this proved to make it prohibitively difficult to produce reliable travel times since the path of a single message is linear and does not return to its origin. Even with a Network Time Protocol (NTP) service running on the Cuff Link to try to keep all the components' times synchronized, impossible (i.e. negative) travel times continued to plague our results which are on the order of thousandths of seconds.

The assignment procedure is different, however, because the Analysis Node initiates the request *and* is the final destination, allowing for very reliable measurements that are obtained from the same system clock. Because of this, we will only present the empirical results of the assignment process.

To test how well the assignment process facilitates collaboration, we will equate certain actions of practitioners to the process of automatically retrieving, analyzing, and sharing the analysis results of a task as they are carried out by an Analysis Node. We affirm that this comparison is acceptable for the reason that, whenever the occasion calls for it, practitioners will manually perform these same operations by taking control of an Analysis Node. One difference between the two scenarios is that nodes only retrieve and work on one analysis task at a time. To accommodate

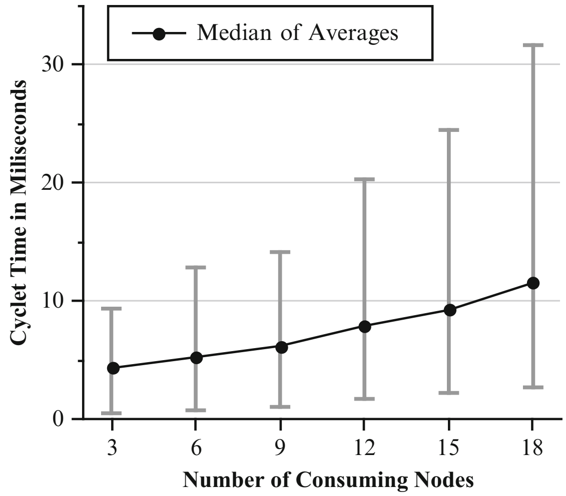

Fig. 12.7 The minimum, maximum, and median values for sets of averages where each set contains average execution times for a number of concurrently running nodes

Table 12.1 Precise numbers for the evaluations illustrated in Fig. 12.7. Numbers are given in milliseconds

Consumers in group	Minimum average	Median average	Maximum average
3	3.8295	4.31941	5.0408
6	4.5067	5.24674	7.6002
9	5.1649	6.19420	7.9581
12	6.1907	7.89824	12.4222
15	7.0317	9.24481	15.2574
18	8.8711	11.55450	20.1397

this difference, we consider the behavior of groups of nodes that may be coupled with an organization or team of practitioners. In this way, the nodes are not limited in their capacity to collaborate since they can logically pool their resources and results.

To help make the nodes' behavior more realistic, each of the job analysis messages has been given between 5 and 15 fake file descriptors. When an Analysis Node retrieves a job, it simulates the load of processing each file by sleeping for 5 ms per file before continuing with its normal execution.

In our test, we created groups of Analysis Node consumers in multiples of three. Timing mechanisms were implemented to measure the completion of steps 6–11 from Fig. 12.5, which we call a "request cycle." Each consumer made 2,000 synchronous requests[2] with an average cycle time calculated for each set of 100 requests. Minimum, maximum, and median values of all nodes in the group were then calculated as illustrated in Fig. 12.7 and as detailed in Table 12.1.

[2]Here we mean that each node made synchronous calls while all the nodes ran asynchronously.

We recognize that the evaluations we have presented here focus on a single component in a complex system comprised of many other elements that could have a significant effect on the scalability of our framework. However, with many of these other elements of CUFF still in development, we chose to demonstrate that the most fundamental of all the components is an appropriate method that will facilitate other components' with a high level of reliability and reuse. In doing so, we believe CUFF's abilities can be further extended to take on increasingly realistic analysis tasks.

User Interface

The main purpose of the user interface is to provide a means for the users to take advantage of all of CUFF's features. These features fall into three main categories: evidence browsing and communication, analysis management, and storage management.

Evidence browsing (extraction) is the task that takes more of an examiner's time during the forensic process than any other task. During this stage of work, the examiner looks through the file system of the acquired evidence, identifies files to be analyzed, studies the results of analyses, and makes decisions based on those results. This is the phase when collaborating with colleagues and subject matter experts is the most beneficial, so it is logical to combine the browsing tools with the collaboration tools into one interface.

We have created a simple web interface with the Google Web Toolkit that demonstrates one way in which these tools can be combined. As depicted in Fig. 12.8, the left pane provides means for navigating evidence, the user's contacts, and the communication files connected with the case that is currently open. The evidence navigation section is populated by deriving the original file system structure from the DFXML file of the evidence image.

The right side of the interface shows the contents of the selected file, a detailed listing of the currently selected directory, and a space for adding comments about the evidence. At this time, this approach of adding comments is the primary means of collaboration that we have implemented into our system. These comments are stored with the case metadata and analysis results.

We initially considered implementing a more sophisticated, near-real-time communication mechanism by adopting the Google Wave Operational Transformation algorithm [35]. However, due to the complexity of implementing this algorithm outside its intended use for a deployment of "Wave in a Box" [37] and because of its recent transitory state to become an Apache incubation project [4], that implementation effort still needs further investigation.

During the extraction phase, examiners also need a way to specify what forensic analysis needs to be performed on which files. To do this, the examiner needs access to something that presents the available analysis tools and algorithms for the files that have been selected. Figure 12.9 is gives an idea of what such an interface would

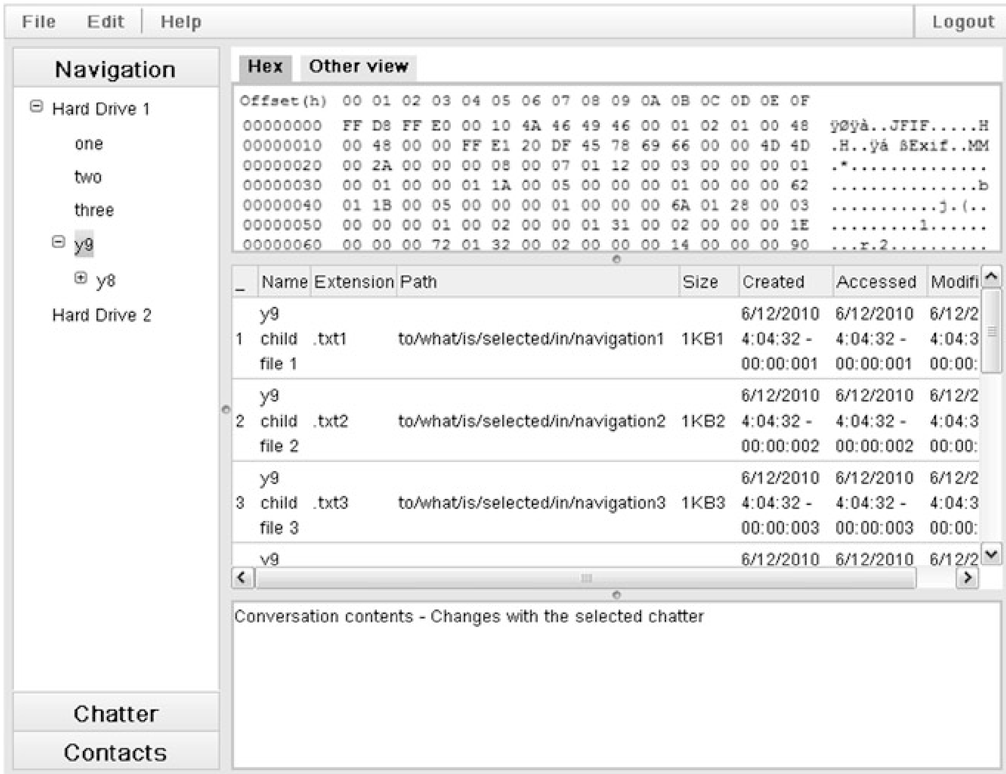

Fig. 12.8 This simple web interface allows users to browse the contents of evidence and communicate with each other

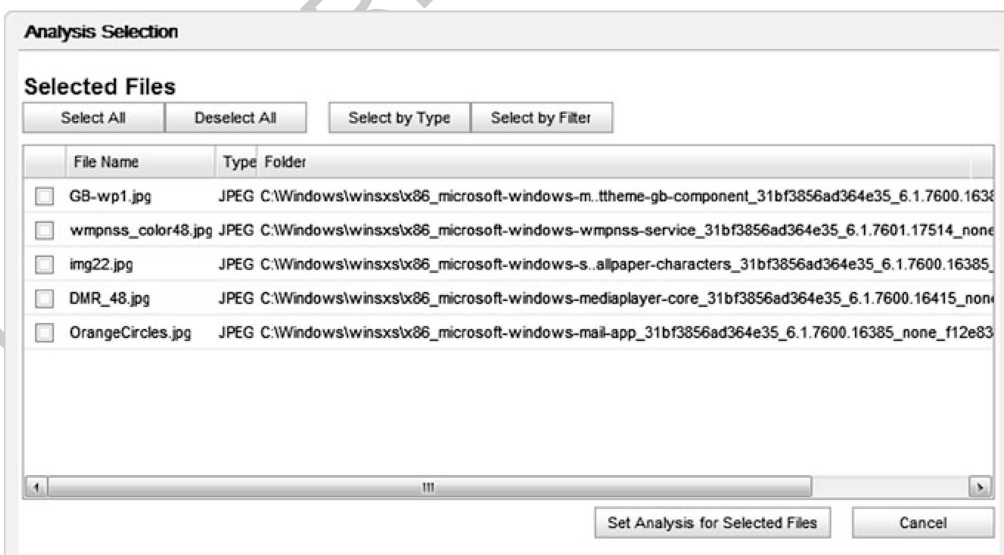

Fig. 12.9 Using this simple tool in CUFF, users will be able to specify evidence to be analyzed and check on its progress

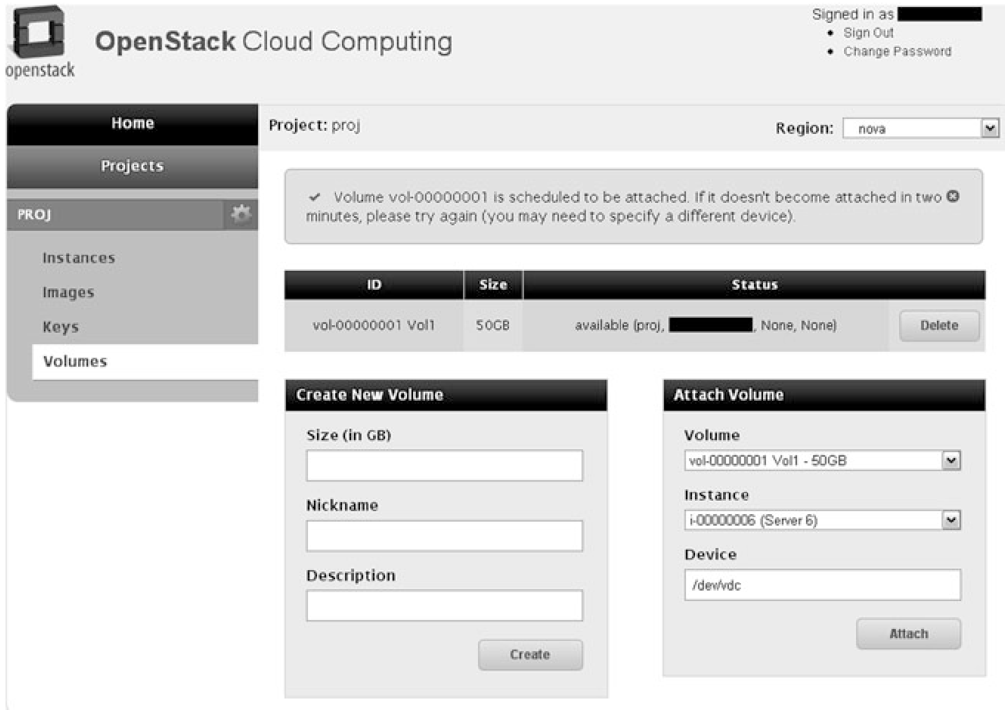

Fig. 12.10 A screen shot of the OpenStack dashboard immediately after connecting a volume to a running instance as /dev/vdc

look like. In the dialog presented to the examiner, the files that have been previously been marked to be analyzed in the evidence browsing window are listed. From this list the examiner can manually select files, select all the files, deselect all the files, select by file type, or select by a regular expression filter. Once the desired files have been selected, the examiner can then click on "Set Analysis for Selected Files" to choose from a list of available analysis algorithms and tools, after which the analysis jobs will be queued into the system.

Because storage volumes are needed for new evidence images and evidence analysis results, there must be some way for the user to execute the operations of creating volumes and attaching them to instances, at least until such processes can be fully automated by the system. One simple solution to performing these tasks is to utilize the web dashboard provided by OpenStack, which is shown in Fig. 12.10. While using this dashboard for all volume operations will be a bit tedious for jobs of any substantial size, it does provide the needed functionality.

Conclusion

In this paper we have discussed the trends of computer crime and the tools to combat those crimes. From these trends we have determined that collaboration among examiners through a secure and robust system would give them a significant advantage to successfully identify both inculpatory and exculpatory evidence in a timely manner. We set forth our requirements for such a system in a framework based on principles of scalability and interoperability. We then provided details for an implementation of the framework and the additional components that are necessary for the basic operations of a live deployment of CUFF.

For this extended work, we have implemented CUFF on the OpenStack cloud architecture, which provides many needed functions for the system. We also described in detail how the Cuff Link mediates communication between components and how the Storage component leverages the strengths of OpenStack's storage features. We also presented a potential use of the DFXML data representation format and introduced our XML schema for DFXML to enhance reliability of data types within a DFXML file. In addition, we proposed our approach to scheduling the use of the system's resources through an efficient messaging protocol.

As we continue to improve upon our implementation of all the components in CUFF, we will perform evaluations and usability testing on our system. As part of this effort, we are currently in correspondence with law enforcement agents in multiple locations to ensure that our research is in alignment with the needs and specifications of those for whom these tools are intended.

Another aspect we will consider as we continue our work on this framework is issues dealing with multi-cloud scenarios. We will be exploring means of securely connecting multiple deployments together so as to allow for sharing of resources and analysis tools to a much higher level without compromising the system's compliance with the rules of evidence.

References

AQ1 1. Advanced message queuing protocol (amqp) project home. http://www.amqp.org/
2. Altheide C, Merloni C, Zanero S (2008) A methodology for the repeatable forensic analysis of encrypted drives. In: EUROSEC '08: proceedings of the 1st European workshop on system security, Glasgow. ACM, New York, pp 22–26. doi:http://doi.acm.org/10.1145/1355284.
AQ2 1355289
3. Amazon web services. http://aws.amazon.com/
4. Apache wave incubating project home. http://incubator.apache.org/wave/
5. Casey E, Stellatos GJ (2008) The impact of full disk encryption on digital forensics. SIGOPS Oper Syst Rev 42(3):93–98. doi:http://doi.acm.org/10.1145/1368506.1368519
6. Cost of hard drive storage space. http://ns1758.ca/winch/winchest.html
7. Denning PJ (1996) Workflow in the WEB. In: Fischer L (ed) New tools for new times:
AQ3 electronic commerce. Future Strategies, Inc., Alameda
8. Du J, Gu X, Reeves DS (2010) Highly available component sharing in large-scale multi-tenant cloud systems. In: Proceedings of the 19th ACM international symposium on high performance distributed computing, HPDC '10, Chicago. ACM, New York, pp 85–94. doi:http://doi.acm.org/10.1145/1851476.1851487. http://doi.acm.org/10.1145/1851476.1851487
9. Dumas M, Hofstede AHMt (2001) Uml activity diagrams as a workflow specification language. In: Proceedings of the 4th international conference on the unified modeling language, modeling languages, concepts, and tools, Toronto. Springer, London, pp 76–90. http://dl.acm.org/citation.cfm?id=647245.719456

10. Euca2ools user guide. http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3
11. Forensic toolkit (ftk). http://accessdata.com
12. Garfinkel SL Afflib.org open source computer forensics software – fiwalk. http://afflib.org/software/fiwalk
13. Garfinkel S (2009) Automating disk forensic processing with Sleuthkit, XML and Python. In: IEEE systematic approaches to digital forensics engineering, Oakland, pp 73–84. doi:10.1109/SADFE.2009.12
14. Garfinkel S (2010) Aff and aff4: where we are, where we are going, and why it matters to you. In: Sleuth kit and open source digital forensics conference, Chantilly
15. Garfinkel SL (2010) Digital forensics research: the next 10 years. Digit Investig 7(Suppl 1): S64–S73. The proceedings of the tenth annual DFRWS conference doi:10.1016/j.diin.2010.05.009. http://www.sciencedirect.com/science/article/B7CW4-50NX65H-B/2/19b42d7f2ccc4be6794c5a1330a551bb
16. Garfinkel S, Farrell P, Roussev V, Dinolt G (2009) Bringing science to digital forensics with standardized forensic corpora. Digit Investig 6(Suppl 1):S2–S11. The proceedings of the ninth annual DFRWS conference. doi:10.1016/j.diin.2009.06.016. http://www.sciencedirect.com/science/article/B7CW4-4X1HY5C-3/2/090ebc16025d598c775d87c8abbb7ae5
17. Higgins KJ (2010) Zeus attackers deploy honeypot against researchers, competitors. DarkReading. http://www.darkreading.com/insiderthreat/security/attacks/showArticle.jhtml?articleID=228200070
18. (ISC)^2 US Government Advisory Board Executive Writer's Bureau (2010) Do punishments fit the cybercrime? Infosecurity. http://www.infosecurity-us.com/view/12029/do-punishments-fit-the-cybercrime-/
19. Juric MB (2010) Wsdl and bpel extensions for event driven architecture. Inf Softw Technol 52:1023–1043. doi:http://dx.doi.org/10.1016/j.infsof.2010.04.005. http://dx.doi.org/10.1016/j.infsof.2010.04.005
20. Krishnakumar N, Aheth A (1995) Managing heterogeneous multi-system tasks to support enterprose-wide operations. Distrib Parallel Databases 3(2):155–186
21. Liebrock LM, Marrero N, Burton DP, Prine R, Cornelius E, Shakamuri M, Urias V (2007) A preliminary design for digital forensics analysis of terabyte size data sets. In: SAC '07: proceedings of the 2007 ACM symposium on applied computing, Seoul. ACM, New York, pp 190–191. doi:http://doi.acm.org/10.1145/1244002.1244052
22. Liu Q, Wang G, Wu J (2010) Efficient sharing of secure cloud storage services. In: 2010 IEEE 10th international conference on computer and information technology (CIT), Bradford, pp 922–929. doi:10.1109/CIT.2010.171
23. Mabey M, Ahn GJ (2011) Towards collaborative forensics: preliminary framework. In: 2011 IEEE international conference on information reuse and integration (IRI), Las Vegas, pp 94–99. doi:10.1109/IRI.2011.6009527
24. Menn J (2010) Fatal system error: the hunt for the new crime lords who are bringing down the internet, 1st edn. PublicAffairs, New York
25. Menn J (2010) US experts close in on google hackers. http://www.cnn.com/2010/BUSINESS/02/21/google.hackers/index.html
26. Miller JA, Palaniswami D, Sheth AP, Kochut KJ, Singh H (1998) Webwork: meteor's web-based workflow management system. J Intell Inf Syst 10:185–215. doi:10.1023/A:1008660827609. http://dl.acm.org/citation.cfm?id=290056.290067
27. Moraski L (2011) Cybercrime knows no borders. Infosecurity. http://www.infosecurity-us.com/view/18074/cybercrime-knows-no-borders-/
28. National software reference library. http://www.nsrl.nist.gov/Downloads.htm
29. Openstack project home. http://www.openstack.org
30. Rabbitmq project home. http://www.rabbitmq.com
31. Roussev V, Richard GG III (2004) Breaking the performance wall: the case for distributed digital forensics. In: The proceedings of the fourth annual DFRWS conference

32. Scanlon M, Kechadi MT (2009) Online acquisition of digital forensic evidence. Lecture notes of the institute for computer sciences, Social informatics and telecommunications engineering, vol 31, pp 122–131. Springer, Berlin/Heidelberg
33. Sharp A, McDermott P (2001) Workflow modeling: tools for process improvement and application development, 1st edn. Artech House, Inc., Norwood
34. Urias V, Hash C, Liebrock LM (2008) Consideration of issues for parallel digital forensics of raid systems. J Digit Forensic Pract 2:196–208. http://www.informaworld.com/10.1080/15567280903140953
35. Wang D, Mah A, Lassen S (2010) Google wave operational transformation. Version 1.1. http://wave-protocol.googlecode.com/hg/whitepapers/operational-transform/operational-transform.html
36. Wang J, Varman P, Xie C (2010) Middleware enabled data sharing on cloud storage services. In: Proceedings of the 5th international workshop on middleware for service oriented computing, MW4SOC '10, Bangalore. ACM, New York, pp 33–38. doi:http://doi.acm.org/10.1145/1890912.1890918. http://doi.acm.org/10.1145/1890912.1890918
37. Wave in a box announcement. http://googlewavedev.blogspot.com/2010/09/wave-open-source-next-steps-wave-in-box.html
38. Zhao G, Rong C, Li J, Zhang F, Tang Y (2010) Trusted data sharing over untrusted cloud storage providers. In: 2010 IEEE second international conference on cloud computing technology and science (CloudCom), Indianapolis, pp 97–103. doi:10.1109/CloudCom.2010.36