available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Distributed forensics and incident response in the enterprise

M.I. Cohen, D. Bilby, G. Caronni*

Google Inc, Switzerland

Keywords:
Remote forensics
Live forensics
Digital forensics
Incident response
Information security
Malware
Memory forensics
Distributed computing

ABSTRACT

Remote live forensics has recently been increasingly used in order to facilitate rapid remote access to enterprise machines. We present the GRR Rapid Response Framework (GRR), a new multi-platform, open source tool for enterprise forensic investigations enabling remote raw disk and memory access. GRR is designed to be scalable, opening the door for continuous enterprise wide forensic analysis. This paper describes the architecture used by GRR and illustrates how it is used routinely to expedite enterprise forensic investigations.

1. Introduction

Digital forensics is an established field in civil and criminal investigations. In the enterprise, digital investigations are often associated with incident response – the detection and investigation of system compromises and targeted attacks. Within the corporate sphere, investigations typically focus on timely response and damage assessment, in addition to maintaining evidentiary standards. The typical enterprise owns and deploys many machines serving a multitude of roles – for example, workstations, laptops and servers. All these machines can be used as launch points for internal attacks and may become involved in forensic investigations. Digital readiness has been previously defined as "the ability of an organization to maximize its potential to use digital evidence whilst minimizing the costs of an investigation" (Rowlingson, 2004). A distributed forensic investigation platform therefore serves to increase digital readiness by lowering the investigative cost and increasing the quality of digital evidence obtainable.

Traditional forensic acquisition consists of shutting the target system down, removing its disk and acquiring a bit for bit copy of the drive, followed by a manual analysis of the drive image. Forensically sound write blocker hardware is often employed during the acquisition step – often requiring the physical removal of the system's hard disk (Carrier and Spafford, 2003). Advantages of this technique include the preservation of digital evidence, and court validated procedures potentially enhancing the admissibility of evidence (Scientific Working Group on Digital Evidence, 2006).

This process is time consuming, requiring a trained forensic investigator to be physically present for the acquisition. This increases the cost of imaging and response time and reduces the number of machines that can be imaged in an investigation. Shutting down a running system may also lead to the loss of important volatile evidence (Walters et al., 2007). Additionally, without knowledge of all assets involved in an incident, even the best evidence preservation techniques fail to give an accurate picture. Timeliness of response becomes more significant to preservation of evidence, than the potential admissibility of the evidence.

Live response has been used as a critical part of the investigative process in order to capture and document volatile system state prior to hard disk imaging (Walters et al., 2007). A number of toolkits are available for automating live evidence capture prior to the seizing of computers (e.g., Microsoft COFFEE – a law enforcement only tool (Microsoft Corporation, 2011)). Volatile evidence can be used after

* *Corresponding author*.
E-mail address: scudette@google.com (M.I. Cohen).
1742-2876/$ – see front matter
doi:10.1016/j.diin.2011.05.012

acquisition to support the investigation in addition to a disk image. Arguably, the complete acquisition of physical memory is sufficient to enable most volatile system state to be subsequently deduced (Suiche, 2011; MANDIANT, 2011).

Triaging has been proposed as way of reducing the time required to image many systems (Garfinkel, 2010). The goal is to both identify relevant evidence quickly and guide the investigative process by directing human resources to reduce the overall response time (Rogers et al., 2006). Triaging often employs less thorough forensic techniques. For example, rather than analyzing the disk by keyword searching across all files, undeleting files, or unpacking container files, an investigator might choose to only search for keywords in documents located in a certain directory. This might be considered a quick way of determining the relevance of this specific disk to the case. Similarly rather than a bit for bit image of the target media, an investigator may selectively image only the data which is likely to be evidentiarily relevant (Termed selective imaging (Turner, 2006)).

Remote live forensic analysis has been proposed as a solution to reducing the time required for response. Some simple tasks can be performed using inbuilt operating system facilities, e.g., examining files via domain file sharing or standard remote administration tools. Typically however, remote system level access to deployed systems is provided via a pre-installed agent servlet. The remote agent allows an investigator to directly connect to the system, and perform rapid analysis or triage the system for subsequent acquisition (Guidance Software Inc., 2011; Various, 2011a). Care must be taken in order to protect the communications between the agent and the management console, typically this communication is encrypted and secured using sophisticated protocols (Mccreight et al., September 2004; Various, 2011a).

Since the system under investigation itself is used for running the remote agent, data obtained from the live system might be subject to subversion by locally active malware, or an attacker. Although case law is rare in this area, legal analysis suggests that authenticity challenges could be successfully defended by thorough tool testing (Kenneally, 2005), although some additional legal risk is afforded by remote live forensics (Kenneally and Brown, 2005).

1.1. *Privacy*

Having a remotely accessible and often silent enterprise management agent installed can pose serious concerns for privacy of the users. Many enterprise management tools give the investigator complete access to the underlying filesystem and memory. This power can be abused to seriously compromise a user's privacy since personally identifiable information (such as credit card numbers, credentials or encryption keys) can be gleaned from memory images and browser cache objects (Kornblum, 2009).

With tighter regulations regarding privacy it is becoming imperative to protect the user's information (Lasprogata et al., 2004; Kim, 2006). This requirement is in direct competition with the need for expedited and thorough analysis for locating compromises (Kim, 2006). Typically in a large organization, routine investigative tasks (e.g., malware infections) must be delegated quickly to a large number of operators, while sensitive investigations which might require access to private information must be legally managed within a small number of operators (e.g., those granted access under a legal warrant).

Unfortunately, current products do not provide fine grained access controls or the auditing flexibility to tailor access to user data (Casey and Stanley, 2004). Due to the interactive nature of most tools, any user granted access to its management console, automatically provides that user with complete access to all files on the remote system. Furthermore, auditing the files that the user accessed is not usually implemented. This may give the investigator complete, unfettered and unaudited access to all files.

On the other hand, by automating pre-devised analysis tasks, our system allows only specific, audited tasks to be initiated on the remote system. For example, searching for files matching a particular keyword can be initiated automatically without the investigator seeing what files actually exist in the user's home directory. These files may then be quarantined for subsequent review by an authorized person.

1.2. *Correlation*

Deployed systems have a variety of software installed (often by users) presenting different kinds of risks for the security of the system and the network. When investigations are conducted on an enterprise scale, the ability to query the entire fleet of deployed systems can in itself provide useful signals for compromise. For example, if a new vulnerability is discovered in a particular version of a product, up to date statistics of all systems in inventory running the vulnerable version can be used to rapidly determine risk exposure and formulate a response plan. This can occur even when the vulnerable software is not part of the standard build environment, but was installed by a user.

As malware is developed to appear more innocuous, it attempts to blend in with other system artifacts (Peron and Legary, 2005). For example, malware commonly uses the same process name as common system processes or use similar registry keys or files as legitimately installed software. Detection of the malware is thus more difficult when analyzing the system in isolation. However, when considering the correlation of artifacts across a large fleet of machines, many running the same install base, indicators of compromise are easily seen. For example, if a commonly installed executable has a particular hash value on most systems, but a different hash value on a few systems in the fleet – this might indicate that this binary was maliciously replaced.

In addition, correlating information across time may also provide valuable indication of compromise – as in the case of new and unusual artifacts suddenly appearing. For example, assume a series of snapshots in time of autorun processes (processes which start when the system is booted) is available for each system. If subsequently a malware is detected on a particular system, inspecting these snapshots allows us to establish a timeline of compromise more reliably (e.g., without relying on potentially manipulated timestamps on the compromised system).

1.3. Current state of the art

The task of forensically analyzing enterprise systems is challenging on many levels. Primarily the enterprise posses a large number of often geographically distributed machines. Scalability may be expressed on a number of dimensions:

- Number of monitored assets
- Number of operators that can simultaneously investigate a single system.
- Number of objects (files, processes) that can be tracked.
- Average time required to respond. Particularly in geographically remote locations deploying of trained investigators requires long lead times.
- Average time to search all assets for indications of compromise.
- Variety and types of investigative operations that can be performed (e.g., keyword search, memory imaging, process enumeration).

The Integrated Digital Investigative Process (IDIP) (Carrier and Spafford, 2003) initiates the investigation during the Notification phase. In the enterprise, notification mechanisms include many indicators of compromise, such as Network IDSs, Host IDS and automatic log analysis systems. Once an indicator of compromise is identified, the IDIP calls for the preservation, survey and search for evidence.

Existing remote live forensic approaches consist of a management console which can connect to a remote agent. For example, over secure shell (Various, 2011a). This approach may not work in every case, especially if the target system is connecting to the Internet over a Network Address Translation device, or behind a firewall.

In addition, typically the number of simultaneous agent connections might be limited, either by license restrictions or by technical limitations. It is therefore not typically possible to contact every asset in an organization to follow up on potential notification. Due to this limitation, investigators are often faced with the need to narrow down searches to a small number of machines very early in an investigation, often causing them to miss potential leads.

2. The GRR rapid response framework

GRR (GRR Rapid Response) was born out of our desire to implement an open source tool which can scale to many thousands of machines, be managed collaboratively through a web interface, and support all major platforms.[1]

GRR advances the field of remote live forensics, by tackling auditing, privacy issues, and being scalable. It provides a secure and scalable platform to facilitate a rich set of forensic analysis solutions. The scalability which the framework provides allows deviation from the traditional "one system at a time" approach into "continuous forensic analysis" model, applicable to the entire asset fleet. Automated analysis can be rapidly developed and dynamically dispatched to all assets simultaneously with minimal response time and maximal reach.

Typically the GRR client agent is distributed pre-installed in the corporate System Operating Environment (SOE), or pushed to clients via standard enterprise software distribution mechanisms. In the following discussion we describe how the GRR architecture achieves its security and scalability goals. We then proceed to describe how it is used in practice to assist in typical scenarios requiring corporate-wide forensic analysis.

2.1. Communications

As described previously, communication between the client agent and the server is critical for the security and reliability of the system. The GRR Server communicates with the clients (agents running on managed assets) via Messages. We term the messages sent from the server to the client "Requests" and the messages sent from the client to the server as "Responses". Generally a single request may elicit multiple responses. Fig. 1 gives an overview of how messages are relayed. The precise network architecture of the GRR system is described later, in Section 6.

The client communicates with the server using the HTTP protocol, receiving batched requests and sending batched responses over periodic HTTP POST requests. Each client has its own message queue (based on the client's Common Name), from which requests directed to that client are drawn. Workers are specialized processes which handle responses by drawing them from specified queues. Clients can only receive requests intended for them, but can send responses to any worker queue.

All communications between the server and client are always encrypted using AES256 with a random session key and Initialization Vector (IV), including the initial Certificate Signing Request exchange. The session key is in turn encrypted using the RSA public key of the remote end, while messages are also signed using the private key of the sender.

Should the server certificate (using an internal Certification Authority whose public key is hardcoded on all clients) get compromised for any reason, a new one can be rapidly deployed. Each client periodically checks for a "latest server

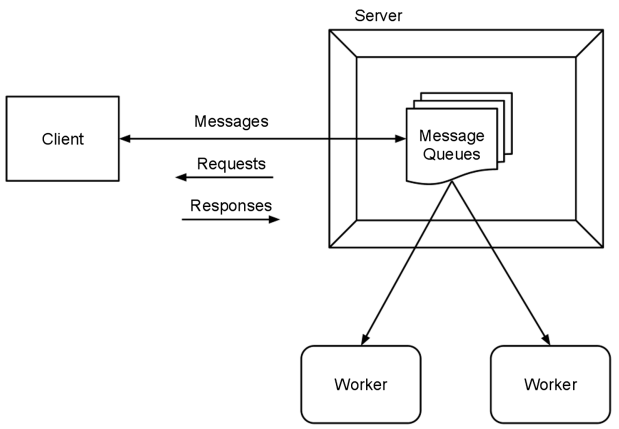

Fig. 1 – An overview of communications paths in the GRR architecture.

[1] The tool is available at http://code.google.com/p/grr/ under an open source license.

certificate" over the network, and upgrades its locally stored copy if needed.

Each request message issued causes the client to execute a client action - a specific routine that performs a single primitive action. For example, the client has a ReadBuffer (path, offset, length) primitive which reads up to a given amount of data from a file at a specified offset. These simple primitives are combined into arbitrarily complex operations by the server.

Replay attacks can be problematic in a remote forensic system. If an attacker is able to capture client responses to a routine request and replay them, the server might record old or out of date data about this client – potentially displacing recent data. In the GRR framework however, replaying POST messages is ineffective since each response is sequenced – hence the system will simply drop replayed messages as retransmissions. In addition, the server timestamp of a request is included within the encrypted and signed payload of each POST response. The client requires an incrementing timestamp between POST requests. If timestamps do not agree, the server requests are simply ignored.

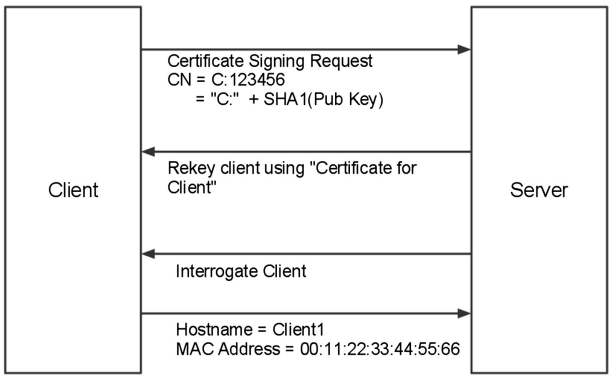

Fig. 2 – Enrollment process for a new client. The client prepares a CSR with a CN composed of its public key, and requests it to be signed by the server. The server issues the new certificate and then initiates an Interrogation Flow, to retrieve more information about the client. The more information the client reveals about itself, the more confident we can be of the identity of the client.

2.2. *Client enrolment*

The GRR client agent is typically distributed via standard enterprise software management mechanisms. Since all client communication is encrypted, clients must obtain their own X.509 certificate prior to the initial server communication. In an enterprise environment, central certificate management is avoided, else SOEs need to be customized for each machine. It is more desirable to allow automatic client enrollment, where a client can obtain its own certificate upon first invocation. In the GRR framework, clients enroll by automatically creating a Certificate Signing Request (CSR) with a unique client Common Name (CN). The CSR is then encrypted with the server public key, and sent to the server for signing. The server signs this CSR and returns an X.509 certificate for the client. When the server generates a certificate for a client, it also stores it in the database. All future communications with this client will utilize this stored certificate. To revoke a client, this key simply gets deleted from the database.

The key enrollment process must be robust to client impersonation. For example, the Encase Enterprise remote collection agent, despite using a complex cryptographic protocol, did not authenticate the client agent, other than by its IP address. This potentially allowed rogue agents to impersonate another system using standard network attack tools (Newsham et al., 2007). In GRR, since the client creates the CSR and chooses the CN, it would be possible for a rogue client to impersonate any other client by simply creating a new CSR containing the other clients CN, and communicating with the server using this new cert. This would allow for impersonation attacks, and potentially compromise the integrity of forensic evidence.

We address this threat in GRR by enforcing the CN to be dependent on the hash of the certificate public key. This in turn ties the CN to the private key, preventing impersonation attacks. A rogue client can not create a valid CSR having a predetermined CN of another client, with a meaningful key pair, without also knowing the other client's RSA private key.

The enrollment process is illustrated in Fig. 2. After enrollment, the server initiates a discovery flow, querying the client for common identifying attributes. The client IDs are uniquely associated with a knowledge base about the client (e.g., host name, IP address, MAC address etc).

Although we can not make guarantees about the authenticity of the information the client returns during the discovery phase, the GRR framework guarantees that a rogue agent can not impersonate another agent. The integrity of each client is therefore independent. GRR uses the client CN as a unique ID in all information kept about the client.

2.3. *Flows*

GRR Supports simultaneous deployment on many thousands of machines, only a fraction of them may be accessible at any given moment. The traditional model has a management console gaining interactive control of the client (e.g., Secure Shell). This requires the investigator to be present when the target system is available. This is less than ideal in a global enterprise where the investigator may not even share the same timezone as the target system. It is also infeasible to have many thousands of administrative console processes waiting for connections to remote systems which may only become accessible after weeks or months.

Automated analysis may require executing many sequential steps. For example, listing a directory, identifying the executables, then calculating each file's hash. Current solutions create a dedicated console process that waits for the client to complete each step before issuing the next step. This limits scalability as the server needs to allocate resources for each client and wait for potentially a long period until the entire analysis is complete.

The key to our solution is the use of state serialization to suspend execution for analysis processes for each client. These serialized forms are then stored dormant on disk until the client responds, posing no resource drain on servers. We term these constructions a Flow. Flows are simply state

machines with well defined serialization points, where it is possible to suspend their execution. We implement our flows using the Python programming language (Various, 2011b), and serialize them using the standard Pickle facility.

Fig. 3 illustrates a typical flow designed to copy a large file. We begin by issuing a message to the client, requesting the hash of a particular file. After sending this request, the flow is suspended and serialized to disk. When the client becomes available, the request is issued and the responses are returned to the server. The server can then resume the flow and push the responses to the next state. If the hash already exists in our file corpus, there is no need to fetch it, and the flow terminates.

The file may be very large (e.g., an ISO image), so it must be read in blocks, each block read with a distinct request. We issue the request for the first block and suspend. A while later, the client will return data for these requests, which we pass into the WriteBlock state. As long as the file download is not completed, we issue another request to read the next block from the client, until the entire file is transferred. Downloading a large file from a live system might lead to an incoherent file if the file is simultaneously modified. The GRR *GetFile* flow requests the client to calculate a hash of the file prior to downloading, and then compares the hash once the file has completed to ensure a successful transfer.

Note that the flow object is only present in memory during the state execution, where it receives responses as parameters. During this execution the flow may issue new requests, their responses targeting a different state. While the requests are pending for the client, the flow is serialized to disk and no memory is used. The client can disappear partway through copying the file and the copy will resume at a later time when the client returns.[2]

Flows essentially maintain context state for a particular interaction with the client. This is analogous to web applications maintaining context using a session abstraction. Similarly, flows are identified using a randomly selected session ID. All responses from the client are addressed to a specific flow using this random session ID.

2.4. *Transmission of messages*

Responses to a request are all destined to a particular state in the corresponding Flow object. For example a request might be ListDirectory, which may elicit many Responses, one for each file in the directory. These responses are all passed to the flow at a particular state.

In order to indicate when all responses have been sent to a particular request, the client sends a STATUS message with a return code for the executed action. This message contains details of the error in the event that the Client Action failed to run, or an OK status if all is well.

This general pattern is shown in the Fig. 4. Note that in practice requests from multiple concurrent flows are batched together without waiting for responses (i.e. in the example below request 1 and 2 would be sent together, then all the responses might arrive back in any order).

[2] However, since there is no permanent state stored on the client, responses to a single request can not be split across reboots. For example, a process listing is coherent, but a file copy might not be, since it consists of multiple requests for consecutive blocks of data.

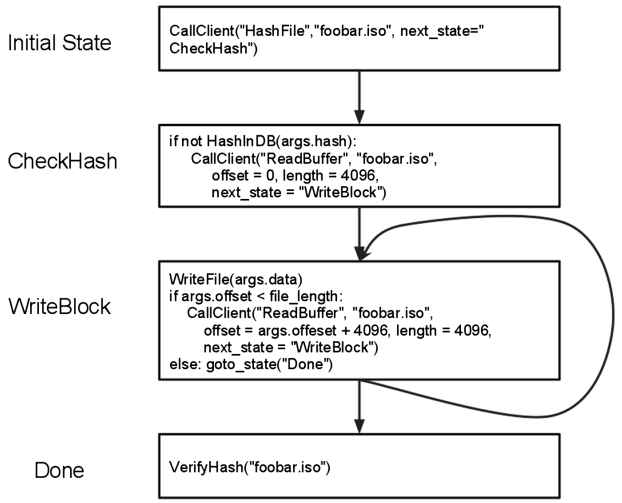

Fig. 3 – A flow to copy a file from the client. The flow begins by asking the client for a hash of the file. If the file does not exist in our corpus, we ask for the first block, and then the next block until the entire file is retrieved. Finally we verify the hash to ensure the file was transferred correctly.

Because the client may disappear at any time (e.g., due to a reboot), the framework must be capable of resuming the execution of the flow at a later time, when the client returns. GRR implements automatic message queuing and retransmission in the event that the client reboots part way through executing a client action. For example, if the ListDirectory action was requested, it might return a partial list of filenames before the client reboots. Since the client does not maintain permanent state, the client can not resume the action. When the client becomes available again, the request is retransmitted and the action is re-executed.

2.5. *Processing queues*

GRR supports multiple processing queues on the server. This allows specialized workers to be used for specific tasks, automatically routing all messages to these workers.

The flow session ID indicates which queue it belongs to. E.g., "W:ABCDEF" indicates that the message should be routed to queue "W" and addresses flow "ABCDEF". Thus a specific flow is always communicated over a particular queue.

The ability to maintain separate queues for flows allows us to implement several types of workers, each listening to different queues. For example, arguably the most security sensitive operation in the GRR architecture is the signing of new clients' CSRs. This requires the worker to have access to the GRR Certification Authority private keys. The principle of least privilege requires that the process which has access to private keys not perform any other task - reducing its attack surface (Viega, 2002). In the GRR architecture, the CA worker is a specialized process with access to the CA private keys. It only listens on a specific CA queue, and is unable to run any other worker flow. Fig. 5 illustrates several message queues simultaneously communicating with the client.

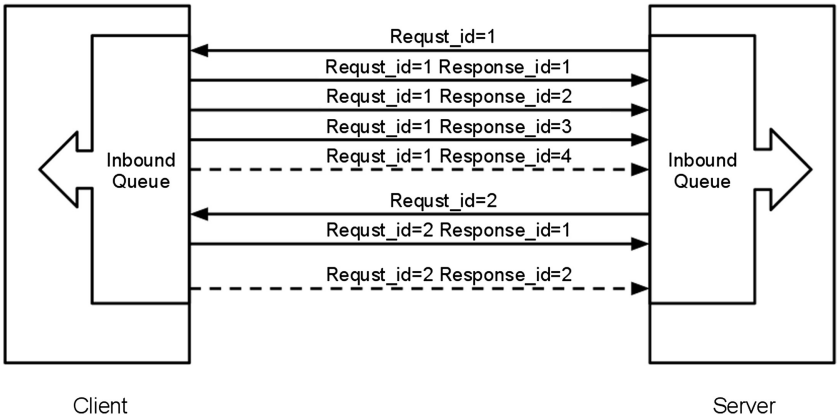

Fig. 4 – Message transmission between client and server. Both client and server maintain a queue of inbound messages ordered by an incrementing ID. Each request can elicit multiple responses, each referencing the request which generated it. The final response is a Status message (depicted by a dashed line) indicating the success of the request execution.

2.6. *Network architecture*

An important requirement of an enterprise forensics platform is to be able to scale to many thousands of machines analyzed concurrently. GRR is designed to be scalable, i.e., deploying more machines allows us to handle more clients simultaneously. In this section we describe the distinct server components, and illustrate how they can be deployed on a server cluster. Clearly for smaller installations, all processes can be run on the same machine.

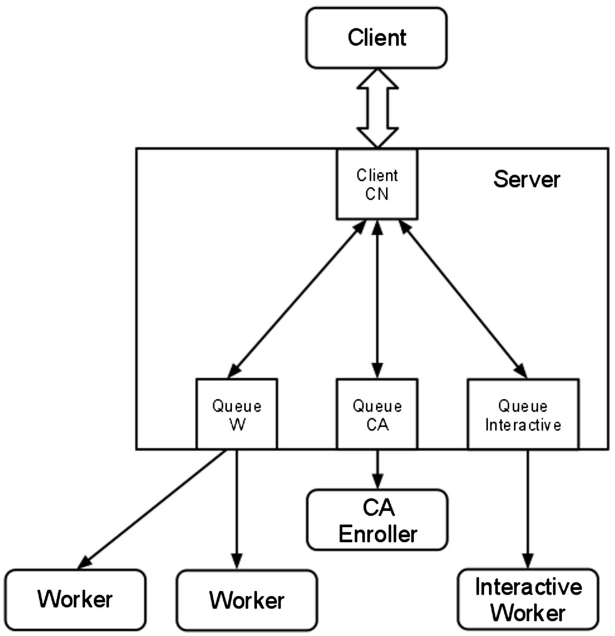

Fig. 5 – Processing queues in the GRR architecture illustrating three distinct types of workers. The general purpose workers retrieve messages from the general queue named "W". A specialized worker responsible for CA enrollments communicates to the client on the queue "CA" while an interactive worker has its own queue for the client.

Fig. 6 illustrates a typical GRR installation. Client POST requests arrive from the Internet and are distributed through a load-balancer to a bank of front end servers (FE). The FE's task is to decrypt the client responses and queue them until a final STATUS response is received for each request. When all responses are received, the worker is messaged to process this flow. The FE also checks for pending requests to this client on the relevant client message queue. Any pending requests for the client are encrypted and sent back in the same POST transaction.

Workers periodically check their message queues for completed requests. When a completed request is received, the worker retrieves the serialized flow from the database (after exclusively locking it). The flow's execution state is then restored by un-serializing it (using the pickle module). The responses are then passed into the relevant state method in the flow. While executing, the flow may issue further requests, which will be placed on the clients queue in the database. Once the state method has completed executing, the flow is re-serialized and placed back on the database. It is

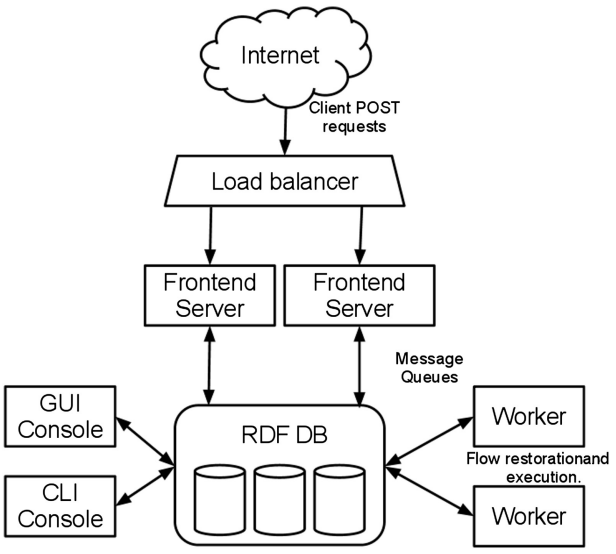

Fig. 6 – GRR backend components.

important to emphasize that the same flow can run on any random worker in the worker pool. The flow object contains all of its state and does not depend on global state.

The database model used in GRR is a Resource Description Framework (RDF) datastore. RDF stores information triples in the form (subject, predicate, Object) (Manola et al., 2004). GRR subject names begin with the common name of the client - a randomly distributed name (due to the hash properties). This naturally leads to a scalable database architecture, since the RDF information space can be easily sharded between subjects (Bell et al., 2010), allowing dedicated database servers to independently manage an even portion of the space. For example, we could choose to run 4 RDF databases, the first managing clients with the name C:111111 to C:3333333, the second C:3333334 to C:666666 etc. While large scale testing has not yet been undertaken, studying similar systems we believe we can linearly scale this system to many thousands of active clients.

Finally Fig. 6 shows the Console process interacting with the database by starting new flows and displaying the result of completed flows. GRR currently has a command line based console, and a web based GUI.

2.7. *Client agent capabilities*

The GRR client agent is implemented on multiple platforms. On Microsoft Windows™ it runs as a system service, while on Linux™ and Mac OSX™ it runs as a root level process. GRR also installs signed kernel drivers to provide access to physical memory.

The agent is able to satisfy requests from the server by running various Client Actions - code routines on the client which receive a request and generate one or more responses. Client actions can provide access to the operating system's APIs, such as reading a file, or listing a directory. This allows for fast and efficient examination, however this is most vulnerable to subversion from malware through API hooking techniques (Peron and Legary, 2005).

Simple actions, such as reading a buffer from a file, raw disk or memory, can be combined on the server to produce more complex forensic analysis. For example, a flow utilizes the Sleuthkit on the server to issue buffer read requests for the raw disk, allowing reading of raw filesystem data structures and recovery of deleted files. Similarly, randomly reading buffers from raw memory can be combined on the server, to analyze the live memory image using the Volatility memory analysis scheme.

Low level remote access to devices is an effective and widely utilized technique (Cohen, 2005; Guidance Software Inc., 2011). The advantage is that the client is very simple, and complex forensic analysis is done server side. The server can employ many novel analysis techniques without needing to upgrade the deployed client (e.g., upgrade the version of the Sleuthkit).

Unfortunately, for remote systems with significant network latency, exporting the raw device in this way can be very slow due to the large number of round trip request/response messages. It is far more efficient to use the Sleuthkit library on the client itself to read the raw device.

For this reason, the GRR agent itself incorporates a number of open source capabilities. The Sleuthkit library is incorporated for access to raw disk filesystem analysis (Carrier, 2011). The REGFI library (Morgan, 2011) is also incorporated to enable the agent to forensically analyze registry files on the running system, with reduced susceptibility to API hooking subversion. Registry files must typically be read using the Sleuthkit on a running system, because they are normally locked by the system. GRR also incorporates the Volatility Memory analysis framework to be able to analyze the raw system memory on the client if the operating system is supported by Volatility (Walters et al., 2011).

3. Implementation challenges

In traditional forensic analysis, we collect information about a system in a frozen state. For example, a directory listing in an image does not yield different results at different times. However, when GRR collects information remotely, that information represents a point-in-time snapshot of system state. This unique aspect of remote live forensics presents a paradigm shift in forensic analysis system design, as every piece of information must have a timestamp of when it was acquired.

We have adopted AFF4 as the data model for GRR (Cohen et al., 2009; Schatz and Cohen, 2010). AFF4 defines abstract objects representing entities uniquely identified by a universally unique URL. Information is organized into RDF triples, making statements about the objects.

The traditional AFF4 model assumes objects are static, and has no concept of statement age. For example, the following statement makes a claim about the hash of a file on the client:

```
aff4:/34a62f06/boot.ini aff4:sha256 "udajC5…BVi7psU"
```

This statement assumes the hash for this file is always the same. However, when GRR retrieves the hash of this file at different points in time, the hash may be different if the file has changed. Thus AFF4 statements must be qualified with the time at which the statement was made. We have extended AFF4 values to have an Age - the timestamp when the attribute was acquired. For example, it is possible to have multiple different hashes for the same file at different times.

```
aff4:/34a62f06/boot.ini aff4:sha256 "uajC5…7Xp == @130
3384321"^^<grr:hash>, "cRmYX…kYQ == @1303386902"
^^<grr:hash>
```

The GUI needs to convey and reconcile this paradigm for the investigator. As can be seen in Fig. 7, the GRR GUI shows the age of each AFF4 statement and allows the user to specify a mechanism to refresh this statement, as well as navigate through older snapshots.

4. GRR case studies

The following section describes some scenarios of how GRR is currently used in our enterprise to assist with large scale forensic analysis and incident response. Although GRR is still in early development phase, it is useful for these scenarios. As the tool evolves we hope to apply it to more cases. We

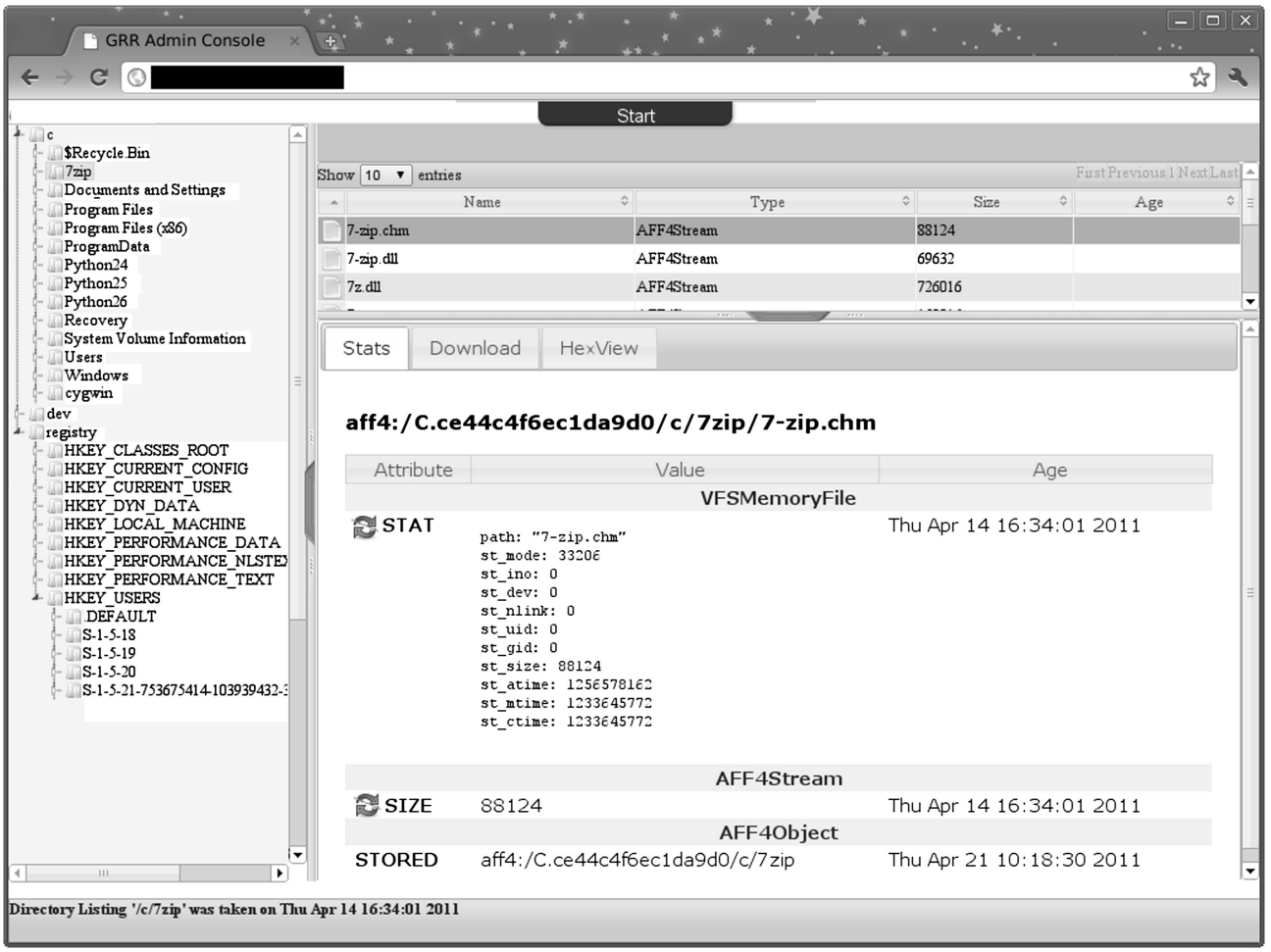

Fig. 7 – The GRR Admin GUI browsing a remote Windows™ system. The registry and file system are available within the same Virtual Filesystem (VFS), and are represented using AFF4 Objects. The AFF4 *Stats* view displays all the attributes related to the object, each of which has an age. The user may choose through the GUI to update the property - creating a flow for the target system. Note the */dev/* virtual filesystem which provides access to the raw disk of the client, including reading files using the Sleuthkit.

currently have two ways of scheduling flows on clients - the first is to target a specific client, while the second allows us to target a specific class of clients (e.g., Windows 7 machines within the EU).

4.1. *Investigate leakage of intellectual property*

In this scenario, restricted information has been leaked and the aim of the investigation is to discover people who had access to this information.

- Flows are started for all corporate assets belonging to users with a particular level of access, searching for specific keywords indicative of the information leaked. The search can be restricted to certain file types or directories in order to increase efficiency or comply with privacy limitations. Unallocated and deleted files can optionally be searched.
- As assets come online, they begin to process their flow and eventually most of the fleet has completed the requested search. Note that it is possible to queue an unlimited number of flows for an indefinite length of time with no resource loading, since flows are simply serialized data stored on disk on the server.
- Now we have a list of machines across the enterprise which have shown a positive hit. Further investigation can be made to isolate users of interest. Note that user privacy has not been compromised since user data was never copied to the server. Only the fact that a user possessed files containing a keyword is established.

4.2. *Isolate targeted malware attack*

A piece of malware is detected on a corporate asset and we wish to know if the attackers have compromised other hosts.

- Flows can be started to look for specific attributes of the found malware - e.g., an MD5 hash, file type or typical file name. We can also search for specific registry keys or even patterns in system memory particular to the running malware.

- As assets come online, they each begin searching for the malware infection, reporting to the central sever potential compromises. This will work for systems on or off the corporate network as GRR is designed to work across the Internet and over slow connections.
- For those machines which are determined to be compromised, flows can be scheduled to force a system shutdown, or pop a message box requesting the user to take their machine to system support services for a system reinstall. This rapid response can quickly incapacitate an attacker who has many systems under their control.

4.3. *Comply with discovery requests*

In this scenario, a discovery order requests a copy of a specific user's machine. However, the machine may be remote and inaccessible, or the user should not be alerted.

- A flow is scheduled for this user's machine which remotely images the drive if network bandwidth is determined to be sufficient.
- It is possible to use hash based imaging (Cohen and Schatz, 2010) to minimize the volume of data that needs to be transmitted over the network, or perform automatic redaction to remove certain files from the image (e.g., those files containing intellectual property unrelated to the discovery request).

4.4. *Periodic health check snapshots*

In this scenario we wish to collect periodic snapshots of automated analysis of each system's state. The snapshot includes the analysis of autorun executables and their hashes (executables automatically started at system reboot), the kernel's SSDT, a list of loaded kernel drivers.

- Flows are scheduled for all systems in the enterprise to run this snapshot every week.
- Snapshots are stored in the database.
- If a system is found to have been compromised, we can initiate a search through the snapshots to determine when the system was compromised.
- As hashes of executables are collected we can check current and past hashes against current malware databases. This allows us to determine that a system was previously compromised in the past, using a previously unknown malware which is now known - even if the malware has already been upgraded by the attacker.

5. Conclusions

Real-time live analysis on a large scale opens up many possible avenues for powerful forensic and incident response techniques. The ability to run automated analysis over most corporate assets simultaneously lowers the investigative effort required per machine extending the investigative reach. Using these abilities, investigators are able to ask complex questions about the state of their environment and get the answers within minutes instead of days.

GRR is an open source, innovative enterprise distributed forensic and incident response system supporting the major client platforms. GRR is designed to be scalable and enables continuous forensic analysis on all corporate assets simultaneously. The scalability afforded by the GRR architecture innovates current enterprise incident response procedures by enabling wide reaching analysis. This increases forensic readiness by lowering the cost of response and increasing the quality of evidence obtained.

At the same time, GRR maintains users' privacy by allowing for non-intrusive automated analysis, with audited access to retrieved data. GRR strikes a balance between protecting access to user data and warranted forensically sound analysis.

Although GRR is still in early stages of development, it is clear that already the tool is useful for managing large scale investigations in the enterprise. The field of live forensic analysis presents new challenges, such as data snapshots and non-interactive analysis, requiring the development of new data models and user interface designs.

REFERENCES

Bell C, Kindahl M, Thalmann L. MySQL High Availability: tools for robust data Centers. Oreilly & Associates Inc; 2010.

Carrier B, Spafford E. Getting physical with the digital investigation process. International Journal of Digital Evidence 2003;2(2):1–20, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.757.

Carrier B. The Sleuth Kit, http://www.sleuthkit.org/; 2011.

Casey E, Stanley A. Tool review-remote forensic preservation and examination tools. Digital Investigation 2004;1(4):284–97, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.6733.

Cohen M, Schatz B. Hash based disk imaging using AFF4. Digital Investigation 2010;7:S121–8.

Cohen M, Garfinkel S, Schatz B. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. Digital Investigation 2009;6:S57–68.

Cohen M. Hooking IO calls for multi-format image support, http://www.sleuthkit.org/informer/sleuthkit-informer-19.txt; 2005.

Garfinkel S. Digital forensics research: the next 10 years. Digital Investigation 2010;7:S64–73, http://www.dfrws.org/2010/proceedings/2010-308.pdf.

Guidance Software, Inc.. EnCase enterprise, http://www.guidancesoftware.com/computer-forensics-fraud-investigation-software.htm; 2011.

Kenneally E, Brown C. Risk sensitive digital evidence collection. Digital Investigation 2005;2(2):101–19.

Kenneally E. Confluence of digital evidence and the law: on the forensic Soundness of live-remote digital evidence collection. UCLA Journal of Law and Technology 2005;5, http://www.lawtechjournal.com/articles/2005/05/_051201/_Kenneally.pdf.

Kim E. The new electronic discovery rules: a place for employee privacy? The Yale Law Journal 2006;115(6):1481–90.

Kornblum J. Implementing bitlocker drive encryption for forensic analysis. Journal of Digital Investigation 2009;5(3–4):75–84.

Lasprogata G, King N, Pillay S. Regulation of electronic employee monitoring: identifying fundamental principles of employee privacy through a comparative study of data privacy legislation in the European Union, United States and Canada. Stanford Technology Law Review 2004;4:24.

MANDIANT. MANDIANT Memoryze, http://www.mandiant.com/products/free_software/memoryze/; 2011.
Manola F, Miller E, McBride B. RDF primer. W3C recommendation 10; 2004.
Mccreight S, Weber D, Garrett M. Patent: Enterprise computer investigation system, http://www.freepatentsonline.com/6792545.html; September 2004.
Microsoft Corporation. Computer online forensic evidence Extractor (COFEE), http://www.microsoft.com/industry/government/solutions/cofee/default.aspx; 2011.
Morgan TD. RegLookup, http://code.google.com/p/reglookup/; 2011.
Newsham T, Palmer C, Stamos A, Burns J. Breaking forensics software: Weaknesses in critical evidence collection. In: Proceedings of the 2007 Black Hat Conference, https://www.isecpartners.com/files/iSEC-Breaking/_Forensics/_Software-Paper.v1/_1.BH2007.pdf; 2007.
Peron C, Legary M. Digital anti-forensics: emerging trends in data transformation techniques. In: Proceedings of 2005 E-Crime and computer evidence Conference; 2005.
Rogers M, Goldman J, Mislan R, Wedge T, Debrota S. Computer forensics field triage process model. In: Proceeding of the Conference on digital forensics security and law, pp. Citeseer; 2006. p. 27–40.
Rowlingson R. A ten step process for forensic readiness. International Journal of Digital Evidence 2004;2(3):1–28, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.6706.
Schatz B, Cohen M. Refining evidence containers for Provenance and accurate data Representation. Advances in Digital Forensics VI; 2010:227–42.
Scientific Working Group on Digital Evidence. SWGDE best practices for computer forensics v2.1, http://www.swgde.org/documents/current-documents/2006-07-19%20SWGDE%20Best%20Practices%20for%20Computer%20Forensics%20v2.1.pdf; 2006.
Suiche M. MoonSols Windows memory Toolkit, http://www.moonsols.com/windows-memory-toolkit/; 2011.
Turner P. Selective and intelligent imaging using digital evidence bags. Journal of Digital Investigation 2006;3:59–64, https://www.dfrws.org/2006/proceedings/8-Turner.pdf.
Various. OpenSSH, http://www.openssh.com/; 2011a.
Various. Python programming language Official Website, http://www.python.org/; 2011b.
Viega J. Building secure software: how to avoid security problems the right way. MA: Addison-Wesley Reading; 2002.
Walters A, Petroni Jr N. Volatools: Integrating volatile memory forensics into the digital investigation process. Black Hat DC 2007, http://www.blackhat.com/presentations/bh-dc-07/Walters/Paper/bh-dc-07-Walters-WP.pdf; 2007.
Walters A, Dolan-Gavitt B, Various. Volatility - An advanced memory forensics framework, http://code.google.com/p/volatility/; 2011.