Hunting in the Enterprise: Forensic Triage and Incident Response

Redacted for Blind review

Abstract

In enterprise environments, digital forensic analysis generates data volumes that traditional forensic methods are no longer prepared to handle. Triaging has been proposed as a solution to systematically prioritize the acquisition and analysis of digital evidence. We explore the application of automated triaging processes in such settings, where reliability and customizability are crucial for a successful deployment.

We specifically examine the use of GRR Rapid Response (GRR) - an advanced open source distributed enterprise forensics system - in the triaging stage of common incident response investigations. We show how this system can be leveraged for automated prioritization of evidence across the whole enterprise fleet and describe the implementation details required to obtain sufficient robustness for large scale enterprise deployment. We analyze the performance of the system by simulating several realistic incidents and discuss some of the limitations of distributed agent based systems for enterprise triaging.

Triage is a process commonly applied in the medical field in order to ration limited resources and to maximize their overall effectiveness [14]. In the medical arena, responders follow a systemic approach to assess the severity of injuries and the likelihood of successful treatments for medical casualties.

Digital forensics is increasingly used in more diverse contexts, such as criminal and civil cases as well as in incident response. Increase in the utilization of digital forensic capabilities has lead to larger case load and longer backlogs of evidence which must be analysed by a limited number of highly trained investigators [19]. The increased data volume places challenges on traditional forensic methods and has led to several proposals for updating best practice techniques in order to cope with the work load [15].

1. Triage in Digital Forensics

Triaging has been proposed for managing long case backlogs [20]. Drawing inspiration from medical triage techniques [11], the goal of digital forensic triaging is to prioritize evidence for acquisition and analysis in order to maximize case throughput.

Particularly, many digital forensic procedures are designed to discover if evidence exists at all on a computer which was potentially relevant to the case. Drawing an analogy from medical triaging [14], one can define triaging as a process which classifies digital evidence into three groups:

- The system is likely to contain crucial evidence but this evidence is unlikely to be destroyed in the near future.
- The system is likely to contain crucial evidence which may be imminently destroyed.
- The system is not likely to contain relevant evidence and therefore should not be acquired or processed.

Classifying potential systems into these categories allows to prioritize evidence acquisition and analysis. Digital evidence collection must follow the Order Of Volatility (OOV) [10, 1], in that some sources of evidence are more volatile and likely to change, hence should be collected sooner. For example, memory images are more volatile than disk images, since system state is likely to change quickly [21], or be completely lost if the system is powered down. Yet the contents of memory contain extremely valuable evidence in many investigations [27]. In this context, it is critical to obtain the memory image as fast as possible.

It is important to contrast the aims of triaging analysis with traditional digital forensic analysis. While traditional digital forensic analysis aims to establish irrefutable findings upon which a solid case may be built, triage analysis has a much lower evidentiary burden of proof. The triaging step is merely trying to establish whether the system is likely to be involved with the case. This lower burden of proof opens the possibility for wider automation in the triaging process, with a higher acceptable false positive rate. However, the danger with automated triaging is that subtle evidence might be missed.

For example, consider a case where the investigation requires analysis of the URLs the suspect accessed using a browser. A full forensic analysis might require the cache material to be examined, browsing history reconstructed and time lines created. On the other hand, the triaging step merely discovers whether the browser cache contains any references to the user name, or web site in question. Thus, the triaging step can be implemented as a simple keyword search, where a significant number of hits result in classifying the system as potentially containing evidence, leading to acquisition and further analysis of the system.

While triaging analysis is less rigorous than a full digital forensic examination, it must necessarily be applied to many more systems. This can be achieved by recruiting less trained investigators to perform the analysis using standard tools (e.g. recruiting police officers, provided with minimal digital forensic training and using commercial tools). Alternatively, specialized tools may be developed to ensure that triaging analysis is as automated as possible, for example the FBI's image scan tool - a law enforcement only tool used to triage for contraband images [2]. Ideally, the triaging strategy is tailored to the specifics of each case.

1.1. Privacy considerations

Another complication with applying the triaging process is the effect it has on the privacy of the system's owner. In traditional digital forensic analysis, the systems acquired and inspected have a very high probability of being relevant to the case. In criminal matters, these systems must fall under the terms of the relevant warrant before they can be examined at all. Usually, the warrant lists all systems that are to be examined in advance, in order to minimize the chances of examining unrelated systems.

In contrast, triaging affects a wider selection of systems, some of which may turn out to be irrelevant to the case, or unrelated to the suspect. A triage step using too general search criteria may therefore reveal private information irrelevant to the case, as well as select unrelated systems for further inspection with traditional forensic analysis procedures, thus inadvertently violating the owners' privacy. Carefully tailored search criteria lower the false positive rate, resulting in a more focused and effective triage, while simultaneously protecting the privacy of system owners. For example, consider again a search for the presence of specific keywords in the user's web history. A specific and unique term such as an email address or domain name is likely to produce a lower false positive rate than a general term which is likely to occur in unrelated web history.

2. Triaging and Incident Response

Aside from traditional forensic investigations accompanying criminal cases, digital forensics is increasingly employed as part of enterprise incident response procedures. Forensic readiness is defined as the procedures that an organization can take in advance of an intrusion in order to expedite the incident response process [9, 18, 22].

Enterprise incident response is typically time constrained, requiring rapid collection and analysis of digital evidence [3]. For example, when responding to a potential security compromise, the need for acquisition of forensically sound evidence must be balanced with rapid disruption and neutralization of the attacker threat and minimizing the resulting economic loss [9].

In this enterprise context, applying a systemic triaging process is crucial [17]. Not only does triaging reduce the number of systems which must be manually examined to a manageable level, but triaging also ensures that investigators have the opportunity to acquire forensically sound evidence of systems of value, while maintaining the Order Of Volatility - thus ensuring the possibility for post-incident legal proceedings.

For example, consider a typical network forensic investigation [3]. Often, there are several compromised systems under the control of the attacker, all using the same kind of Remote Attack Tool (RAT). A triaging procedure might require searching for unique evidence of the RAT in memory, on disk or in the Windows Registry. This might include specific registry keys used by the tool or binary strings unique to the RAT's executable which may be found in memory.

If evidence of compromise is found in memory, the machine is then chosen for immediate response and a copy of memory is acquired immediately. However, if the RAT has been uninstalled such that it exists only on disk but is not currently running, the machine's hard disk can be acquired at a later time. This determination is performed according to the Order Of Volatility - it is crucial to obtain memory images as soon as possible in order to freeze the state of RAM for further analysis.

In the initial parts of an incident response investigation, it is not known how many systems are likely compromised using the same RAT. Therefore triaging must be applied to all systems rapidly in order to narrow down the list of possibly compromized ones. In an enterprise, this requires rapid triaging of many hundreds if not thousands of machines.

Once systems have been triaged, evidence must be acquired. Forensic acquisition systems fall into two broad categories. The first requires the investigator to have physical access to the system and boot the system into a mode which allows low level access to the physical disks (e.g. a boot disk such as Encase Portable [13]). This approach is more resilient against potential malware on the target system. However, it is more time consuming since it requires the system to be taken off line, and the investigator to be physically present to acquire the image. This mode of operation is generally not usable for obtaining a memory image without installing specialized hardware in advance [26].

Another approach is to deploy an agent on the target system and perform analysis remotely (e.g. Encase Enterprise [12], GRR [6]). This approach minimizes the time required to respond (since physical access is not required). However, it may be susceptible to interference from malware, since the agent relies on the integrity of the operating system to operate.

3. The GRR Rapid Response System

GRR Rapid Response (GRR) is an open source agent based scalable enterprise forensic platform [6, 23]. The

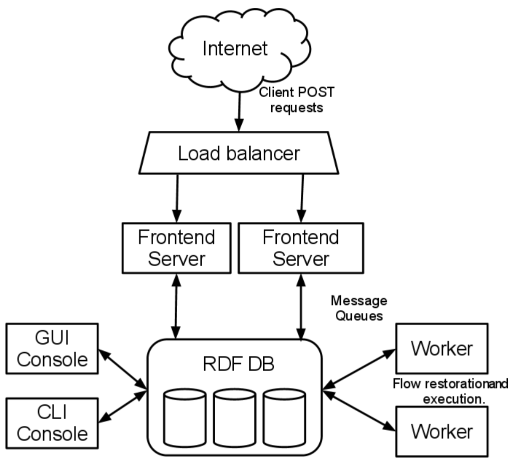


Figure 1: GRR Component overview. The client agent communicated over the internet with the front end servers. These manage message queues mediating communication between the agent and the workers and console. The workers execute flows - which are predetermined modules of analysis.

goal of GRR is to lower the total cost of triage analysis by distributing that analysis to the system agents. This allows the system to triage a large number of systems quickly, and allows investigators to focus their efforts on those systems which best progress the investigation.

GRR is one of several agent based remote forensic platforms [12, 16]. The following discussion focuses on some implementation challenges addressed within the GRR system, but these challenges are also faced by other platforms. It is the discussion and analysis of these issues that the authors hope will advance the current state of the art in enterprise forensic triaging.

Figure 1 shows an overview of the GRR system. The GRR Agent is deployed on enterprise assets and reports back to the central server. Message queues on the frontend are able to queue messages to the agent - termed *requests* - and process messages from the agent - termed *responses*.

A sequence of requests and responses to the agent which achieve a single analysis goal are termed *Flows*. Flows can be suspended indefinitely by being serialized to the data store. This technique allows the GRR Framework to achieve the scalability required to simultaneously interact with many thousands of agents. The details of how flows are implemented in the GRR framework are described elsewhere [6], however for the purpose of this discussion, a flow can be thought of as a single analysis operation performed on the agent, for example fetching a file, performing a keyword search on a directory or running a memory analysis module.

The following sections describe some of the innovative features of GRR and how these features work to implement the Triaging process described in Section 1.

3.1. AFF4 and data modeling

The Advanced Forensic Format v4 (AFF4) was conceived as a means of modeling and interchanging forensic data in a consistent way between different systems [7]. In order to facilitate an extensible and scalable interchange mechanism, an object oriented data model was proposed based on the RDF relational model.

GRR uses the AFF4 data modeling system to model the information returned from the agents in a persistent data store. The system maintains a persistent view of the entire enterprise fleet within the AFF4 space and all agent interactions can simply be viewed as updates to the AFF4 data model to reflect samples of system states on the agents.

While the detailed AFF4 data model description can be found elsewhere [7, 5], the following description highlights some of the important concepts as they are implemented in the GRR system.

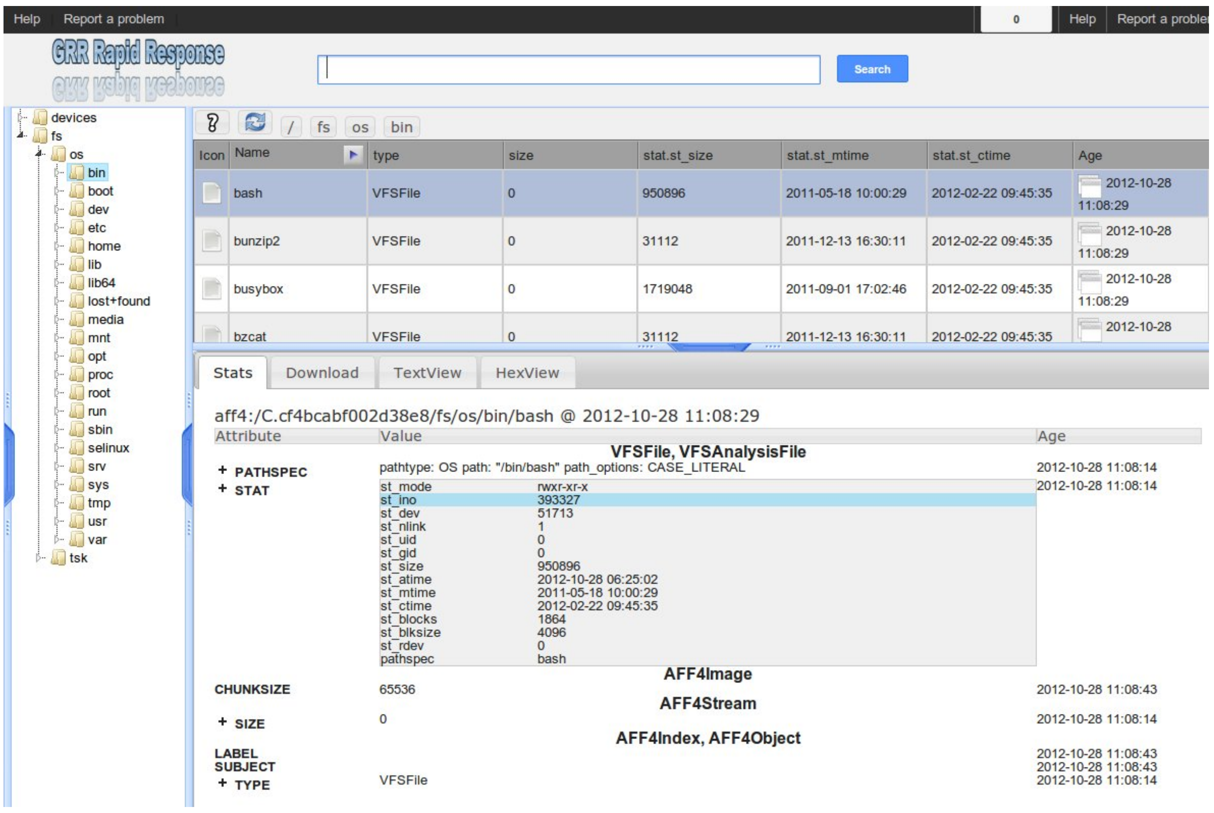
At its heart, the AFF4 data model defines abstract *AFF4 Objects*, each having a specific *Type*. These are merely collections of *Attributes* and type specific methods. An AFF4 object is known by its *Uniform Resource Name* (URN), allowing the object to uniquely exist in the AFF4 namespace. For example, an AFF4 Object which has a type of *AFF4Stream* provides the *read()* , *seek()* and *tell()* methods. GRR implements a number of specific AFF4 Object types, as well as a number of Attribute types to represent remote files on the GRR agent system.

AFF4 objects are abstract objects, responsible for their own serialization to a data store. The data store simply provides persistent storage for AFF4 objects, and the specific implementation of the data store is not important. For example, for persistent interchange format, a data store may be implemented using ZIP files [7]. The GRR framework utilizes an interchangeable no-sql persistent data store [24, 4].

The original AFF4 scheme did not allow for versioned attributes. This is however required in a live remote forensic system such as GRR to represent constantly changing forensic artifacts. For example, if we retrieve a file from the agent, we know the file contents at that instant. However, at a later time, the file may have changed, so repeating the file download will require a new version to be stored in the AFF4 model.

This creates a time smear of data collected from the client. Every piece of data has a timestamp associated with it and is referred to as the *Age* of the data.

Figure 2 illustrates a file examined using the GRR GUI. The file is known by its URN (e.g. *aff4:/C.cf4bcabf002d38e8/fs/os/bin/bash*) as a unique reference within the global AFF4 name space. The URN itself can be broken down into a client component, followed by a Virtual Filesystem component, followed by the path of the file on the agent's system. Note that the GUI refers to the version of the file at a specific time and the attributes are also listed with their own timestamps. By clicking the



Icon	Name	type	size	stat.st_size	stat.st_mtime	stat.st_ctime	Age
	bash	VFSFile	0	950896	2011-05-18 10:00:29	2012-02-22 09:45:35	2012-10-28 11:08:29
	bunzip2	VFSFile	0	31112	2011-12-13 16:30:11	2012-02-22 09:45:35	2012-10-28 11:08:29
	busybox	VFSFile	0	1719048	2011-09-01 17:02:46	2012-02-22 09:45:35	2012-10-28 11:08:29
	bzcat	VFSFile	0	31112	2011-12-13 16:30:11	2012-02-22 09:45:35	2012-10-28

aff4:/C.cf4bcabf002d38e8/fs/os/bin/bash @ 2012-10-28 11:08:29

Attribute	Value	Age
	VFSFile, VFSAnalysisFile	
+ PATHSPEC	pathtype: OS path: "/bin/bash" path_options: CASE_LITERAL	2012-10-28 11:08:14
+ STAT	st_mode rwxr-xr-x st_ino 393327 st_dev 51713 st_nlink 1 st_uid 0 st_gid 0 st_size 950896 st_atime 2012-10-28 06:25:02 st_mtime 2011-05-18 10:00:29 st_ctime 2012-02-22 09:45:35 st_blocks 1864 st_blksize 4096 st_rdev 0 pathspec bash	2012-10-28 11:08:14
	AFF4Image	
CHUNKSIZE	65536	2012-10-28 11:08:43
	AFF4Stream	
+ SIZE	0	2012-10-28 11:08:14
	AFF4Index, AFF4Object	
LABEL		2012-10-28 11:08:43
SUBJECT		2012-10-28 11:08:43
+ TYPE	VFSFile	2012-10-28 11:08:14

Figure 2: Navigating the GRR VFS using the GUI.

selectors in the "Age" column, the user can easily switch between viewing different versions of this file.

From a triaging point of view, the system must be capable of handling multiple versions for each piece of data since the same machine may be examined multiple times for different investigations at different times. The triage process must therefore be able to account for older versions of the same file - for example, a specific cache history artifact was found at one time instance but not at a later time instance.

3.2. Agent Reliability and Stability

The GRR system depends on the agent being available and trusted on all enterprise assets in order to schedule large scale triage hunts. Like all enterprise software, there are non-security considerations for wide deployment, such as system stability, resource consumption and licensing costs.

It is therefore extremely important to ensure that agents use as few resources and are as reliable as possible when running on enterprise systems. While a significant level of testing has gone into the agent, there are always situations which could cause unexpected extreme resource consumption. For example, a keyword term containing a regular expression may have exponential running time on the agent [8], leading to uncontrolled resource usage. It is therefore imperative to provide guaranteed bounds on the agent's resource consumption, before it can be safely installed on all corporate computing assets.

GRR uses a *Nanny* process to control agent memory footprint, and provides assurances about maximum agent resource utilization. The Nanny is a very small process (memory footprint around 200 kilobytes) which is registered as a Windows Service. The Nanny is responsible for launching and monitoring the main GRR Agent. Once the agent is spawned, the Nanny constantly monitors for a heartbeat signal. If a heartbeat is not received during a configured time span, the Nanny kills the agent and restarts it after a small resurrection time.

The GRR Agent is specifically designed to expect every action to cause a crash. Therefore, the agent maintains a transaction log of actions it is currently performing. If the crash occurs during the execution of an agent action, the agent is restarted by the Nanny. On start-up, the agent examines the transaction log, and notices that the last action resulted in a crash. The agent then sends a failure message to the server to inform it about the crash.

This arrangement drastically improves the stability of the GRR system and its resilience to a number of error conditions as described in the following sections.

3.2.1. Handling GRR Agent Crashes

Even though the GRR Agent is very thoroughly tested, sometimes crashes may still occur in the agent or perhaps

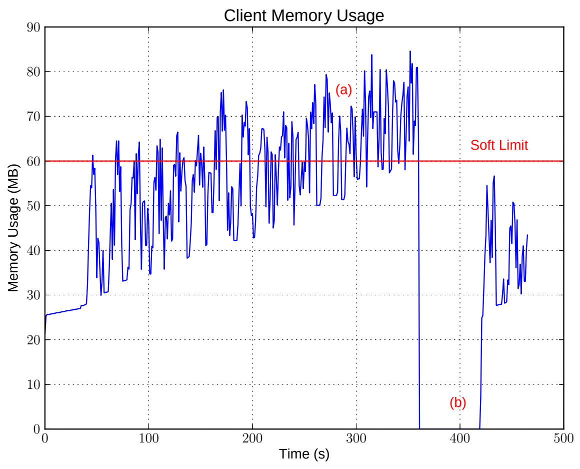


Figure 3: A footprint of an agent modified to leak memory.

in a third party library. A vivid example of this condition is the case of loading a memory driver for memory analysis or acquisition. Although the memory driver is well tested, there are cases where memory imaging can lead to a blue screen of death crash (BSOD). If the machine is rebooted, the agent will have to detect that a crash occurred, fail the memory analysis request, and inform the server (and ultimately the investigator) about the crash.

As a remote forensics tool, GRR interacts heavily with low level third party libraries. For example, consider a GRR server request for parsing the registry. If the third party registry parsing library contains a bug and crashes when attempting to parse the registry file, the recovery process is as follows:

1. The server issues a request for the Agent to parse the registry file.
2. The agent writes a transaction log with the currently running request.
3. The agent then calls the registry parsing library attempting to parse the file, but the agent crashes as a result.
4. The Nanny notices the child agent exited and restarts it after a short time.
5. On startup, the agent examines the transaction log and discovers that it exited during a request. It then sends a failure message to the server for that request.
6. The server simply records the error message in the relevant flow and informs the user that parsing this file failed. However, the agent remains available for further analysis.

3.2.2. Agent Memory Footprint

The GRR Agent is a long running process which uses many third party libraries. It is possible that some of these libraries leak memory. However, as mentioned previously, it is crucial to ensure that the agent memory footprint on the system is small.

The Agent maintains queues of incoming requests and outgoing responses. As requests are queued up, the agent's memory usage increases, however, when its memory usage rises above the soft limit, the agent refuses to accept new requests. While processing the remainder of the incoming queue might increase client memory for a short while, eventually client memory will begin to drop as its queues empty.

Once the agent's memory falls below the soft memory limit, it can begin accepting new requests. This causes the client to essentially track the soft memory limit level - going above it for a time, and falling below it for a while. If there is a persistent memory leak however, the agent will consume more and more memory and the memory usage will eventually, even when queues are all empty, remain above the soft limit. In this case, the agent will simply voluntarily exit. The Nanny will then restart the agent. Note that in this case, no requests are lost and the agent simply continues where it left off. Since the agent maintains no local state, the agent restart does not affect any current analysis.

To demonstrate this in practice, we have modified one of the GRR agents to leak memory at a constant rate and have monitored its resource consumption during a large file transfer. Figure 3 illustrates how this modified agent manages its own memory footprint.

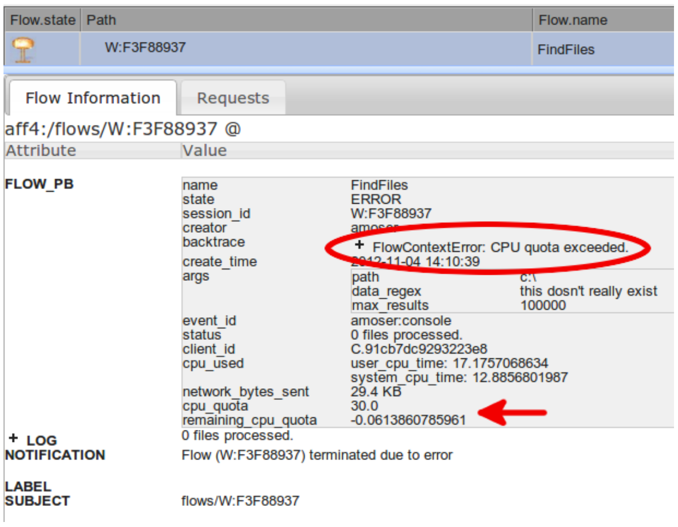
On the left side of the graph, the agent uses memory as it processes requests. The memory consumption spikes up as new requests are processed but stops shortly below the 60 MB soft limit we have set for this experiment. However, as more and more memory is leaked, the soft limit is exceeded more frequently (around label (a)). As the agent sends more data to the server, some of this memory is returned to the operating system and the agent falls below the soft limit.

At some point, the memory usage does not drop below the soft limit, even when there is nothing to send in the output queue. At this point, the agent simply exits. After a resurrection period (b), the nanny process restarts the agent, which immediately starts transferring the file again.

In addition to the soft memory limit, there is also a hard memory limit configured. If an agent uses more than this amount of memory, the Nanny process can forcibly terminate the agent. Those two limits together effectively guarantee bounds on total memory consumption of the agent.

3.2.3. Excessive CPU Utilization and Hangs

Sometimes, unavoidably, the agent becomes unresponsive while processing some actions. For example, attempting to access an NFS share which is unavailable might block the agent in an IO call indefinitely. The GRR system is also capable of recovering from this condition as the hung agent stops sending the Nanny the heartbeat signals. After a while, the nanny process will kill and restart the



Flow.state	Path	Flow.name
	W:F3F88937	FindFiles

Flow Information | Requests

aff4:/flows/W:F3F88937 @

Attribute	Value		
FLOW_PB	name	FindFiles	
	state	ERROR	
	session_id	W:F3F88937	
	creator	amoser	
	backtrace	+ FlowContextError: CPU quota exceeded.	
	create_time	2012-11-04 14:10:39	
	args	path	c:\
		data_regex	this dosn't really exist
		max_results	100000
	event_id	amoser:console	
	status	0 files processed.	
	client_id	C.91cb7dc9293223e8	
	cpu_used	user_cpu_time: 17.1757068634	
		system_cpu_time: 12.8856801987	
	network_bytes_sent	29.4 KB	
	cpu_quota	30.0	
	remaining_cpu_quota	-0.0613860785961	
+ LOG	0 files processed.		
NOTIFICATION	Flow (W:F3F88937) terminated due to error		
LABEL			
SUBJECT	flows/W:F3F88937		

Figure 4: A *Find* flow has terminated due to CPU quota exhaustion. The specified CPU quota was 30 seconds, and the flow has run out of available CPU quota. It is killed with the error message CPU Quota Exceeded.

agent, which will then send a failure response to the GRR server's request upon checking its transaction log.

Another problem is the total amount of system resources utilized during the execution of an analysis flow. For example, a keyword search might result in an unexpected level of CPU usage if the files we intend to examine end up being large. So despite the search proceeding as normal (i.e. the agent is continuing to send heartbeat signals to the Nanny), the total resource utilization is still unexpectedly large.

To control agent system resource usage, GRR implements per flow resource quotas. As a flow progresses by sending requests to the agent, the agent reports the amount of resources utilized for each request (e.g. CPU utilization, bytes transmitted). The flow then tallies these resources and shows the total amount of system resources used. When total resources utilized exceeds the allotted quota, the flow is forcibly terminated.

Figure 4 shows an example of a flow that has been terminated because it has exhausted its CPU quota while running on an agent.

3.3. Hunts

A GRR hunt is a mechanism for controlling the launching, monitoring and aggregating of results from running a specific flow on a large subset of agents registered in the GRR system. A hunt is the primary method for performing enterprise wide triaging.

The launching of a hunt is illustrated in Figure 5. Deployed agents periodically check in with the GRR system, where they are examined by the *Foreman* component. The Foreman has a set of rules which are compared against the agent's known profile. For example, a rule can specify a

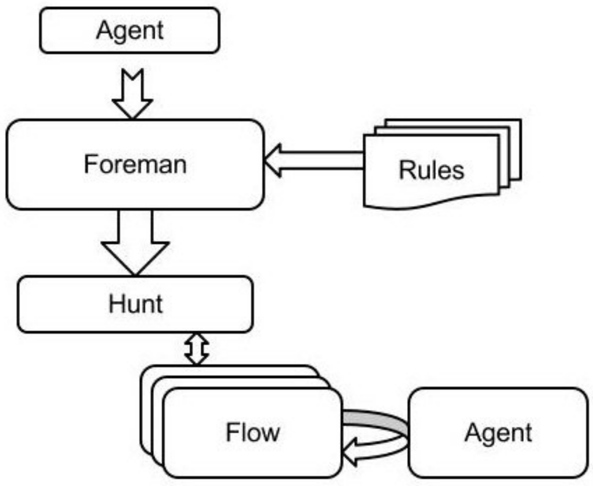


Figure 5: Scheduling Hunts.

required operating system, usernames or hostname for the agent.

If a rule is found to have matched, the agent ID is passed to a Hunt flow for processing. The hunt then launches the required analysis flow and tracks its progress. When the child flow completes, its results are collected and recorded in a central location.

Investigators follow the following three phase process when issuing new hunts:

1. *The planning phase*: In this phase, the user designs the type of analysis required to triage the system according to the Triage process described in Section 1. An example for this phase would be finding suitable keywords or searching for specific files. The analysis is then designed to single out those systems which might contain important evidence or be relevant to the investigation.
   An important consideration here is to decide if immediate action is required in the event the system is found to contain relevant evidence. Immediate actions might include the collection and preservation of volatile evidence such as certain files or memory images, or the automatic notification to responders for urgent attention (e.g. to perform manual forensic disk acquisition).
   The user must also decide if the hunt should run on all systems or only on systems matching a particular profile. For example, the investigation may only be relevant to certain operating systems or even operating system versions. The user can then design a suitable Foreman rule to only trigger hunt collection on the systems of interest.
2. *The collection phase*: In this phase, the hunt and its relevant Foreman rules are active. As clients check into the server, the Foreman compares their profile against the rule set and those agents which match

the criteria are directed to the hunt. The hunt then launches flows on these agents and maintains statistics on each client, as well as calculates the total resource usage of the entire hunt.

3. *The analysis phase*: In this phase, the user gathers data from the hunt flow and examines the results from the collection phase. The user is then able to escalate further analysis of triaged systems.

This general triaging procedure may be applied to a wide range of typical investigation scenarios. The following sections present some typical examples of applying this general methodology in practice.

4. Experiments

The following experiments were designed to show the capabilities of GRR as well as understand the limitations of enterprise wide triaging. Although we use the current implementation of GRR for these experiments, we expect similar general observations to be applicable to other enterprise triaging tools. Indeed, as the GRR framework matures and improves, we expect the absolute performance numbers to improve somewhat, but we feel the following discussion to remain relevant.

In order to emulate a typical GRR installation in a corporate environment we have installed the GRR agent on a large number of corporate workstations and laptops used by enterprise employees (The experimental group). Specifically, we show the usefulness of GRR for targeted triaging, anomaly detection through aggregate statistics, post-incident artifact detection, as well as the easy extensibility of triaging flows. We also measure server and agent resource consumption and show the effect of agent availability on triaging hunts.

4.1. Drive by Download

Consider a hunt designed to find potential downloaders of a malicious exploit from the Internet (This type of attack is often known as *drive by download*). For this example, we assume that analysis of the initial exploit indicates the exploit is only effective against Internet Explorer. We wish to find which corporate assets have been compromised, disrupt the attack and assess the overall damage.

Following the triaging procedure in Section 1, we can design a hunt:

- Targets only Windows systems running Internet Explorer - this is specified in the foreman rule set.

- Search for the malware URL in the browser history caches. We decide to collect the browser cache and history files for triaged systems immediately, as there is a risk that the URL is flushed from the browser cache by the time a responder is able to forensically image the system.

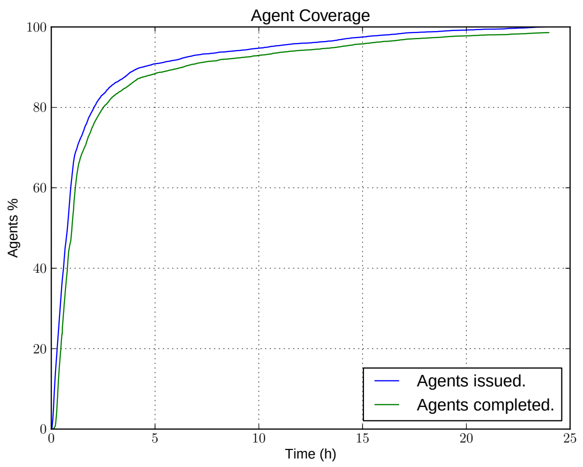
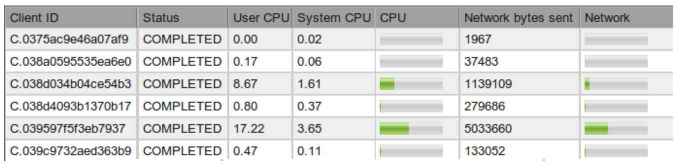


Figure 6: Fraction of agents in the experimental group running the triage flow with time. Initially the rate at which active agents process the triage hunt depends on system capacity, but after a while, the number of new agents participating in the hunt depends on the rate at which these become available and online.



Client ID	Status	User CPU	System CPU	CPU	Network bytes sent	Network
C.0375ac9e46a07af9	COMPLETED	0.00	0.02		1967	
C.038a0595535ea6e0	COMPLETED	0.17	0.06		37483	
C.038d034b04ce54b3	COMPLETED	8.67	1.61		1139109	
C.038d4093b1370b17	COMPLETED	0.80	0.37		279686	
C.039597f5f3eb7937	COMPLETED	17.22	3.65		5033660	
C.039c9732aed363b9	COMPLETED	0.47	0.11		133052	

Figure 7: Variation of CPU load in the driveby hunt.

- We expect the search not to take longer than 1 CPU minute on the agent and transfer less than 100Mb from each agent.

For this experiment, we launched the hunt on all Windows systems in the experimental group and recorded the number of systems completing the triage process. The results are shown in Figure 6.

As can be seen in the figure, the majority of the agents pick up the hunt in the first few minutes after it has been started and after a run time of around 150 minutes the progression of the hunt slows down considerably. The reason for this is that all the agents that were online at the time the hunt was started have all completed the hunt by that time. After this point, only clients that were offline before (they might just be in a different time zone and therefore just starting up as users start the workday) and just come back online pick up the hunt.

Note that we stopped the hunt after 24 hours. In a real world scenario it might take much longer to reach 100% coverage since some machines might not be online for an extended period of time (e.g. laptops with users on vacation).

Figure 7 shows the GRR GUI view of the total CPU and bandwidth transferred for each agent during this hunt. The variation in resource requirement can be attributed to

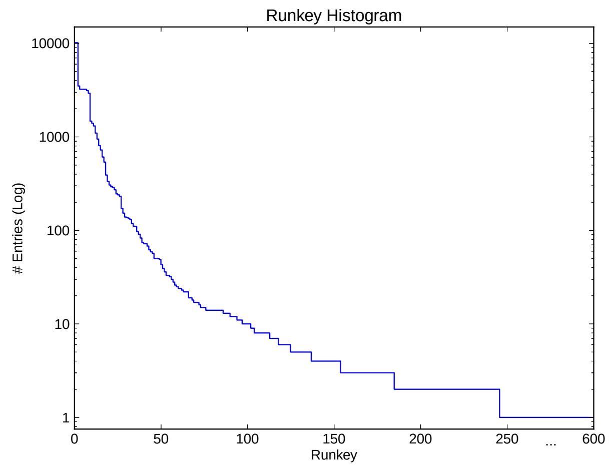


Figure 8: A Histogram of Run keys present on the client systems.

variation in Internet Explorer cache sizes between agent systems. During this experiment, the average amount of CPU seconds used by the agents was 0.70. However, 1.8% of the clients used more than 5 CPU seconds, and a few even used more than 20 CPU seconds.

4.2. Autoruns

Wide scale triaging brings the possibility of cross sectional analysis on enterprise systems. This type of analysis aims to detect anomalies in some systems by comparing them to the entire population of similar systems, and qualifying their variance from the norm (This anomaly may be attributed to a potential compromise).

As an example of this use case, we collected the Run/RunOnce registry keys from the entire Windows experimental group. Although not a complete location of malware startup locations, this is a common mechanism for malware to persist across reboots. Typically, when inspecting a single system it is difficult to ascertain if a specific Run key starts a malicious binary without further analysis. However, by comparing the total number of occurrences across the entire fleet, we can isolate those keys which are less commonly used. We postulate that anomalous Run keys often result from malware infection.

Figure 8 shows a histogram of all the different Run keys we encountered during the experiment. Note, that to generate this histogram we ignored system specific values such as usernames and CIDs. As the figure illustrates, there are many Run keys that launch legitimate binaries and are present on a high percentage of the investigated systems. One example for those common entries is the key "%programfiles%\windows sidebar\sidebar.exe /autorun" which was found on nearly every system we investigated. Other examples are common system drivers for printers or popular software packages.

However, the figure also shows that there is a very long tail of keys that are only present on very few systems or entirely unique. Specifically, 60% of the collected Run keys were only present on a single machine. In our experiment, a quick manual investigation of just those keys identified most of them as legitimate. Nevertheless, we also encountered interesting keys like "[. . .]\local\temp\goodluck (2).exe" that seem anomalous and warrant further investigation.

4.3. Artifact Detection

Commonly after an intrusion, a similar backdoor is installed on a large number of systems to provide the attacker with the means to re-compromise the system in the event it was cleaned up. Typically, this backdoor can be achieved by changing the system's configuration or installing a Remote Access Tool (RAT).

In this experiment, we simulated a security incident that affected a small number of our experimental testbed machines. Specifically, we changed a configuration file for the cron service on a random sample of around 1% of the systems in a way malicious programs would. We then used the GRR system to check all the machines for the contents of this specific file using a "Find" flow using a regular expression to search for the modification.

The goal of this experiment was to assess how quickly simulated malicious modifications can be detected in the experimental setup and the total time it takes to find all such instances.

Figure 9 shows the results of this experiment. As in previous experiments, a large fraction of the artifacts are discovered in the first hour after scheduling the hunt. After that the detection rate slows and gradually approaches 100%. In a real world incident, some machines might not be online for prolonged periods so a 100% detection rate might not be achieved for a long time. The GRR hunt will, however, just continue running (given a high enough expiration time) so if those missing machines come back online at a much later time, the artifacts will still be detected and an alert can be raised.

The resource usage statistics give a clear view of the agent impact for this hunt. On average, each agent used less than 0.01 CPU seconds to complete the action for this experiment (with a standard deviation of only 0.008 CPU seconds) and sent less than 100 bytes over the network. This more uniform level of resource usage can be attributed to smaller variance in the sizes of system configuration files.

4.4. A Simple GRR Triaging Flow

Memory analysis represents one of the most volatile sources of evidence. In this example, we simulate a hunt to search agent memory for a known unique string. If the string is found, we take the memory image using Volatility's crash dump writing support[25], and subsequently copy the image to the server for evidentiary preservation.

To achieve this, we wrote a customized flow reusing existing components. Figure 10 shows a code snippet of this

```
 1  @flow.StateHandler(next_state=["ExamineResults"])
 2  def Start(self, responses):
 3    pathspec = PathSpec(path=r"\\.\pmem", pathtype=Path.MEMORY)
 4    self.CallFlow("Grep", pathspec=pathspec,
 5                  grep_regex="notepad.exe",
 6                  next_state="ExamineResults")
 7
 8  @flow.StateHandler(next_state=["DownloadImage"])
 9  def ExamineResults(self, responses):
10    if responses and responses.success:
11      # Each response is a hit on the signature so we create a crash dump.
12      self.destination = r"C:\Windows\Temp\crash.dmp"
13      args = dict(destination=self.destination)
14      self.CallFlow("VolatilityPlugins",
15                    plugins="raw2dmp",
16                    args=args,
17                    next_state="DownloadImage")
18    else:
19      # Signature was not found in memory.
20      self.Log("Grep did not yield any results.")
21
22  @flow.StateHandler()
23  def DownloadImage(self, responses):
24    # Now download the crash dump.
25    self.CallFlow("GetFile", path=self.destination)
```

Figure 10: A GRR flow to search for memory signatures and take memory images.

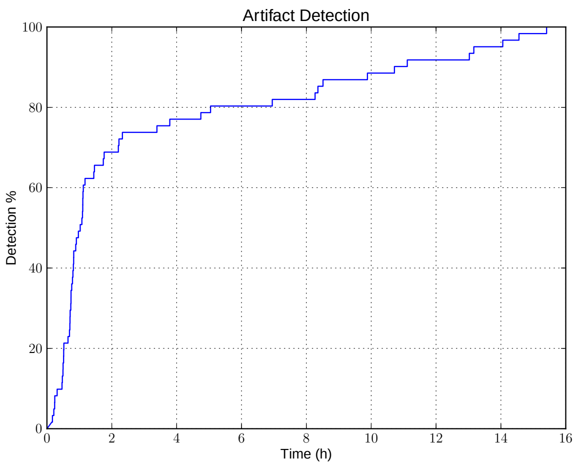


Figure 9: Percentage of artifacts detected over time.

flow. The flow consists of 3 distinct states. While the agent is performing the work, the flow can be serialized on the server, not consuming server resources.

The Start state (Line 2) is the initial state of the flow. It issues a *Grep* subflow to detect the pattern in the physical memory of the system. The results of this flow are then examined in the *ExamineResults* state (Line 11). If the pattern is found, the Volatility *raw2dmp* plugin is invoked to quickly save a copy of the live memory to the local disk, and then a *GetFile* flow is issued to download this file to the server and preserve it, forming a chain of custody. Alternatively, the *GetFile* flow can be used to copy memory over the network without locally storing it, but this might take longer and thus result in a more smeared image.

5. Discussion

We have chosen the experimental conditions to be representative of a typical enterprise hunt for a pro-active incident response team. As illustrated in Figure 6, the response time of the system depends heavily on the usage pattern of the systems. For agents running on laptops and other roaming systems, usage patterns show clear times where the system is not available. Beyond the scalability of the server itself, this non-availabily is a factor delaying the speed of response. A significant number of systems are not available at any particular time (e.g., users in a different time zone) and some systems remain unavailable for an extended period (e.g., users on vacation).

The response curve in Figure 6 can be divided into three phases. Within approximately an hour from the start of the hunt the response curve is very steep, as the active agents are added to the flow. However, after that time, the response curve becomes much flatter, as the only new systems added to the hunt are those which were not initially active. Finally the curve flattens off, as most systems in the fleet have participated in the hunt and new systems become rarer.

Since systems are added to each hunt as they become available, increasing system capacity only has the effect of compressing the first phase (i.e. handling the initial load presented by currently available agents). Once the initial load is handled, we must wait for systems to become available, and an increase of the GRR system capacity is not expected to speed up the hunt progression.

When considering the amount of resource use the drive by download hunt elicited in Figure 7, we note that most agents did not spend much time searching the Internet Explorer cache. However a few agents spent a significant amount of time performing this action, perhaps due to unexpectedly large cache configurations. This variation highlights the need for guaranteed bounds on resource utilization as even simple actions that usually do not result in excessive load utilization can cause unexpected load on some systems.

Section 4.2 explores some new possibilities which become available when employing a wide reaching enterprise triaging system, such as GRR. The ability to compare the running configurations of similar systems concurrently promises to reveal anomalous configurations. Indeed the cross sectional correlation anomaly detection is effective in isolating configuration changes which are not common, and therefore potentially suspicious. However, the experiment also highlights inherent difficulties with this approach - namely that the variations form a very long tailed distribution. The manual examination of every system with a potentially unique autorun key is still time consuming. Potential variations of this approach might involve retrieving the executable file with GRR and post processing it using an up to date virus scanner or other binary analysis engine.

6. Conclusions

Triaging is essentially an optimization using scarce resources more efficiently for the analysis of systems which are more likely to be interesting for an investigation. The goal of a triaging procedure is simply to ascertain whether the system is likely to be of interest, and how urgently traditional forensic acquisition techniques must be applied in order to maximize the evidentiary value of the system.

Since the burden of proof is far lower for a triage analysis, it is possible to automate this analysis heavily. Automation allows us to apply triaging to many more systems, resulting is a wider net. Automation also assists in protecting the privacy of individuals who are found to not be relevant for the investigation.

GRR is an enterprise remote forensic platform which is concentrated on forensic analysis and acquisition. By lowering the cost of forensic analysis on each system, we can use GRR to perform enterprise wide triaging procedures.

GRR is suited to run widely on enterprise systems due to safety mechanisms built into the agent, allowing analysis to be run confidently on many systems, without the risk of adversely affecting operational systems.

References

[1] Brezinski, D., Killalea, T., 2002. Rfc 3227: Guidelines for evidence collection and archiving. Internet Engineering Task Force.

[2] Cantrell, G., Dampier, D., Dandass, Y., Niu, N., Bogen, C., 2012. Research toward a partially-automated, and crime specific digital triage process model. Computer and Information Science 5 (2), p29.

[3] Casey, E., 2006. Investigating sophisticated security breaches. Communications of the ACM 49 (2), 48–55.

[4] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., Gruber, R., 2008. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) 26 (2), 4.

[5] Chow, K., Shenoi, S., 2010. Advances in Digital Forensics VI: Sixth IFIP WG 11.9 International Conference on Digital Forensics, Hong Kong, China, January 4-6, 2010, Revised Selected Papers. Vol. 337. Springer.

[6] Cohen, M., Bilby, D., Caronni, G., 2011. Distributed forensics and incident response in the enterprise. digital investigation 8, S101–S110.

[7] Cohen, M., Garfinkel, S., Schatz, B., 2009. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. digital investigation 6, S57–S68.

[8] Cox, R., 2007. Regular expression matching can be simple and fast (but is slow in java, perl, php, python, ruby,...).
URL `URL:http://swtch.com/~rsc/regexp/regexp1.html`

[9] Endicott-Popovsky, B., Frincke, D., Taylor, C., 2007. A theoretical framework for organizational network forensic readiness. Journal of Computers 2 (3), 1–11.

[10] Farmer, D., Venema, W., 2005. Forensic discovery. Vol. 18. Addison-Wesley.

[11] Hogan, D., Burstein, J., 2007. Disaster medicine. Lippincott Williams & Wilkins.

[12] Inc., G. S., Oct. 2012. Encase enterprise.
URL `http://www.guidancesoftware.com/encase-enterprise.htm`

[13] Inc., G. S., Oct. 2012. Encase portable kit.
URL `http://www.guidancesoftware.com/encase-portable.htm`

[14] Iserson, K., Moskop, J., 2007. Triage in medicine, part i: concept, history, and types. Annals of emergency medicine 49 (3), 275–281.

[15] Jones, B., Pleno, S., Wilkinson, M., 2012. The use of random sampling in investigations involving child abuse material. Digital Investigation 9, S99–S107.

[16] Khurana, H., Basney, J., Bakht, M., Freemon, M., Welch, V., Butler, R., 2009. Palantir: a framework for collaborative incident response and investigation. In: Proceedings of the 8th Symposium on Identity and Trust on the internet. ACM, pp. 38–51.

[17] Lim, K., Lee, S., Lee, S., 2009. Applying a stepwise forensic approach to incident response and computer usage analysis. In: Computer Science and its Applications, 2009. CSA'09. 2nd International Conference on. IEEE, pp. 1–6.

[18] Mitropoulos, S., Patsos, D., Douligeris, C., 2006. On incident handling and response: A state-of-the-art approach. Computers & Security 25 (5), 351–370.

[19] Richard III, G., Roussev, V., 2006. Next-generation digital forensics. Communications of the ACM 49 (2), 76–80.

[20] Rogers, M., Goldman, J., Mislan, R., Wedge, T., Debrota, S., 2006. Computer forensics field triage process model. In: Proceeding of the Conference on Digital Forensics Security and Law. pp. 27–40.

[21] Schuster, A., 2008. The impact of microsoft windows pool allocation strategies on memory forensics. digital investigation 5, S58–S64.

[22] Tan, J., 2001. Forensic readiness. Cambridge, MA:@ Stake.

[23] Various, Oct. 2012. Grr rapid response is an incident response framework.
URL `https://code.google.com/p/grr/`

[24] Various, 2012. Mongodb.
URL `http://www.mongodb.org/`

[25] Various, 2012. The volatility framework.

URL `https://code.google.com/p/volatility/`

[26] Vömel, S., Freiling, F., 2011. A survey of main memory acquisition and analysis techniques for the windows operating system. Digital Investigation 8 (1), 3–22.

[27] Walters, A., Petroni, N., 2007. Volatools: Integrating volatile memory into the digital investigation process. Black Hat DC 2007, 1–18.