Contents lists available at ScienceDirect

Digital Investigation

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

DFRWS 2015 Europe

A scalable file based data store for forensic analysis

Flavio Cruz [a,*], Andreas Moser [b], Michael Cohen [b]

[a] Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, USA
[b] Google Inc., Brandschenkestrasse 110, Zurich, Switzerland

A B S T R A C T

Keywords:
Distributed database
Incident response
Sqlite
Evidence analysis
Distributed computing

In the field of remote forensics, the GRR Response Rig has been used to access and store data from thousands of enterprise machines. Handling large numbers of machines requires efficient and scalable storage mechanisms that allow concurrent data operations and efficient data access, independent of the size of the stored data and the number of machines in the network. We studied the available GRR storage mechanisms and found them lacking in both speed and scalability. In this paper, we propose a new distributed data store that partitions data into database files that can be accessed independently so that distributed forensic analysis can be done in a scalable fashion. We also show how to use the NSRL software reference database in our scalable data store to avoid wasting resources when collecting harmless files from enterprise machines.

Introduction

Digital Forensics necessarily deals with the storage, manipulation and exchange of large quantities of data, from disk images, memory images, to logical objects such as files, and analysis results (Garfinkel, 2010). In addition, practitioners do not only need to store large quantities of data, but they also need to be able to analyze it and ensure it can be easily exchanged.

Traditionally, proprietary storage formats such as the Eye Witness Format (EWF) have been developed to store evidence in structured containers (Guidance Software, 2014). Other proposals facilitate the free interchange of data, one example is DFXML which stores digital forensic information within an XML schema (Garfinkel, 2012).

The Advanced Forensic Format 4 (AFF4) was initially proposed as an interchange format for digital evidence (Cohen et al., 2009). The AFF4 proposal is essentially an object data store – objects are defined with appropriate behaviors and these are stored in the evidence file. The original AFF4 paper describes a data-at-rest file format centered around the Zip archive format and a number of objects with predefined behaviors (such a Containers, Streams etc). These objects are instantiated through a central *Resolver* which abstracts file storage details from the application.

The GRR Rapid Response (GRR) framework is a live forensic and incident response framework constructed using the AFF4 technology (Cohen et al., 2011). Rather than operating on static evidence files, the Resolver in GRR is implemented as an abstraction to a NoSQL data store. The application then uses the Resolver to permanently store AFF4 objects inside a NoSQL data store, while the rest of the application only deals with high level objects. NoSQL technologies are becoming increasingly popular in forensic analysis (Wen et al., 2013) since they offer more flexibility and scalability than relational databases (Parker et al., 2013).

The initial implementation of GRR was based around the proprietary BigTable technology (Chang et al., 2008) and demonstrates impressive scalability in remote response of very large numbers of machines. In the open source version of GRR that has since been released, the framework supports a number of interchangeable data store backends. By

* Corresponding author.
E-mail address: flaviocruz@gmail.com (F. Cruz).

http://dx.doi.org/10.1016/j.diin.2015.01.016

default, GRR uses a backend based on MongoDB (MongoDB, 2014). Other options include for example a MySQL (MySQL, 2014) backend. The scalability of the GRR system heavily depends on the performance of the data store technology, so choosing the underlying technology is extremely important.

In this paper we present a new data store backend that can be used as a storage layer for the AFF4 Resolver. We analyze the access patterns of AFF4 objects focusing specifically on the way that the GRR system utilizes the AFF4 space. By tailoring the data storage to the specific use case presented by GRR and AFF4, we implement a data storage layer that significantly improves the overall scalability of the GRR system in general.

This paper is organized as follows: First, we present the AFF4 object model and specifically examine how the GRR system utilizes the AFF4 abstraction. By analyzing the specific access pattern we propose a novel implementation of a NoSQL data store engine based on the SQLite database technology. We then evaluate the new data store in comparison to previous data stores. Finally, we utilize the new data store to perform a typical forensic analysis step – collect all the executable files on a Windows system which are not already known by the NSRL software reference database (NSRL, 2014b). The use of NSRL and other hash deduplication techniques has been demonstrated in the past to dramatically increase the efficiency of evidence collection and analysis, particularly for remote forensic applications (Rowe, 2012; Fisher, 2001; Watkins et al., 2009).

The AFF4 object model

The Advanced Forensic Format 4 (AFF4) was initially proposed as an interchange format for digital evidence that stores forensic data in object abstractions. All AFF4 Objects have a type, which specifies their behavior (e.g. An object of type AFF4Stream can be used to present an abstract stream interface), and a number of data attributes that contain additional information about the object (Cohen et al., 2009).

Every AFF4 object is identified by a Universal Resource Name (URN) which specifies an object uniquely within the AFF4 namespace. A URN is globally unique within the AFF4 universe and all access to AFF4 objects occurs via the *AFF4 Resolver* – a central logical factory for AFF4 objects. One can open, create and store AFF4 objects through the resolver, without consideration to their actual persistent serialization.

An important property of the AFF4 design is that the AFF4 namespace universe is assumed to be incomplete at any specific time. For example, when one obtains an AFF4 volume containing a number of AFF4 objects, it does not imply that we know the complete subset of the AFF4 universe. For example, an AFF4 object may refer to other AFF4 objects which are not necessary stored in that specific volume (i.e., there may be unresolved external references). This property allows merging different AFF4 volumes containing overlapping parts of the AFF4 namespace. Similarly, it does not make sense to directly enumerate any parts of the AFF4 namespace (since any specific implementation can not know the complete space). All AFF4 objects are related via semantic relations and therefore the AFF4 subsystem does not directly enumerate names, but must follow existing semantic links.

The following example illustrates this important point. Consider the logical collection of files on one machine's filesystem. The container aff4:/C.12345/fs/os/c:/Windows refers to the Windows directory of that filesystem. If we want to list the files contained within the Windows directory, we can not simply query the AFF4 subsystem directly to enumerate all URNs (e.g. with a wild card of aff4:/C.12345/fs/os/c:/Windows/*.*). Instead, we must explicitly store references to all children inside the AFF4-Volume object aff4:/C.12345/fs/os/c:/Windows itself, which are then used to retrieve the children of the directory.

The overall effect is that the data store must only support access to AFF4 URNs by exact name, rather than provide enumeration strategies. For the use of AFF4 in the GRR application this means that the application itself must maintain internal indexes to support object enumeration in cases where this is needed. For these reasons, modern key-value store NoSQL databases are a particularly good fit for serving the needs of the AFF4 data model (Grolinger et al., 2013).

The GRR rapid response framework

The GRR Rapid Response (GRR) Framework is a modern incident response and remote forensic tool designed to perform live forensics on a large number of systems. The GRR framework is outlined in Fig. 1. Although the details of the system are specified elsewhere (Cohen et al., 2011), the most pertinent point of this architecture is that GRR is constructed over the AFF4 subsystem. In practice, this means that all data stored in the GRR data store consists of serialized AFF4 objects. The AFF4 Resolver which allows

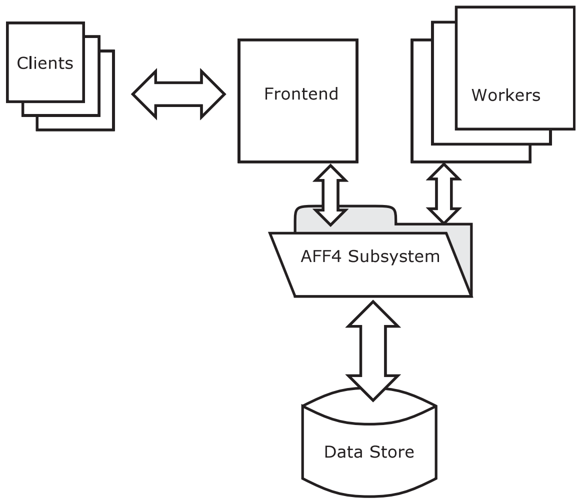

Fig. 1. The GRR Architecture. Clients use the HTTP protocol to exchange messages with the Frontend servers. Frontend servers in turn communicate with the AFF4 subsystem to queue messages in the data store. Workers communicate with the AFF4 subsystem in order to perform analysis tasks and schedule new operations on the clients. Note that all parts of the GRR framework interact with the AFF4 subsystem, which in turn abstracts access to the data store.