Native Actors: How to Scale Network Forensics

Matthias Vallentin
UC Berkeley
vallentin@cs.berkeley.edu

Dominik Charousset
HAW Hamburg
dcharousset@acm.org

Thomas C. Schmidt
HAW Hamburg
t.schmidt@ieee.org

Vern Paxson
ICSI / UC Berkeley
vern@icir.org

Matthias Wählisch
FU Berlin
waehlisch@ieee.org

ABSTRACT

When an organization detects a security breach, it undertakes a forensic analysis to figure out what happened. This investigation involves inspecting a wide range of heterogeneous data sources spanning over a long period of time. The iterative nature of the analysis procedure requires an interactive experience with the data. However, the distributed processing paradigms we find in practice today fail to provide this requirement: the batch-oriented nature of MapReduce cannot deliver sub-second round-trip times, and distributed in-memory processing cannot store the terabytes of activity logs needed to inspect during an incident.

We present the design and implementation of Visibility Across Space and Time (VAST), a distributed database to support interactive network forensics, and `libcppa`, its exceptionally scalable messaging core. The extended actor framework `libcppa` enables VAST to distribute lightweight tasks at negligible overhead. In our live demo, we showcase how VAST enables security analysts to grapple with the huge amounts of data often associated with incident investigations.

Keywords

Security, Network Forensics, Message-oriented Middleware

1. INTRODUCTION

Network forensics and incident response require prompt investigation and reaction to minimize the damage incurred by a security breach. When security analysts explore an incident, they typically need to access data from a large time span to determine scope and breadth of the attack.

The sheer volume of data to investigate during this process largely exceeds both storage and processing capacities of single-machine system architectures. In addition, analysts conducting forensics explore data in an extensive, iterative fashion, dynamically generating numerous queries as they work to pinpoint the salient evidence. Such an interactive process fits poorly to distributed approaches such as MapReduce architected around batch-oriented processing. Distributed in-memory cluster computing, which can provide quick response times, requires the entire data set to fit in memory—likewise infeasible when faced with the terabytes of data that logging at a large site produces every day.

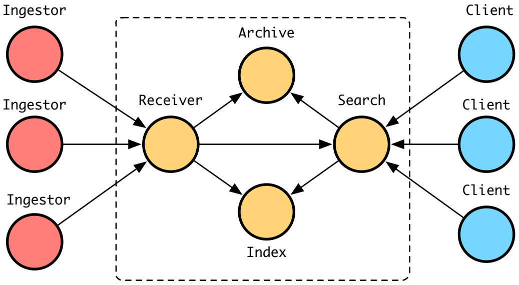

Figure 1: VAST architecture based on actors (circles). Data enters the system through Ingestors and clients send queries to retrieve activities. Processing happens in the VAST core (dashed box) and scales over a cluster of commodity machines.

Because no existing architecture suits the needs of this domain, we set out to design and implement a scalable, interactive platform. We compose our fully distributed system of lightweight, native actors embodied in the `libcppa` programming environment. The system, *visibility across space and time* (VAST), unifies in a single framework retrospective data analysis and proactive measures to automatically apply previously developed queries to events that may occur in the future [1].

Our demo showcases how VAST achieves the required performance and interactivity with real-world data. We give visitors the opportunity to see and interact with the system to better understand the nature of forensic network security analysis.

2. SCALABLE FORENSICS

Scaling network forensics to terabyte volumes while retaining an interactive user experience requires a robust software infrastructure that proves highly scalable.

2.1 An Actor-based Architecture

Current heterogeneous compute environments can be fully exploited with a runtime abstraction that *(i)* harnesses available parallelism within a single machine, *(ii)* facilitates clus-

SIGCOMM ’14 Chicago, Illinois USA
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631471.

ter deployments while providing a type-safe messaging layer with minimal memory usage, and *(iii)* can offload expensive computation to specialized hardware (such as GPUs). With `libcppa`,[1] we contribute an actor library that satisfies these needs. This C++ distribution layer is tailored to high-performance applications and has demonstrated superior performance [2]. The environment provides a programming paradigm based on the actor model [3]. In this model, concurrent entities—actors—execute independently and in parallel. Using unique, location-independent addresses, actors communicate asynchronously via message passing, and do not share state. The `libcppa` runtime system can distribute actors dynamically—even on GPUs—to balance work load. The actor model—originated in 1973—has gained momentum with the advent of multicore machines and cloud computing. Today, actor systems can be found in back-end software of large service providers such as eBay.[2] `libcppa` is the first native, open source framework that scales up to GPGPU-enhanced clusters and down to the IoT.

2.2 The VAST System

VAST employs `libcppa` as a central part of its design, and structures its logic entirely in terms of actors. Figure 1 shows a high-level overview of the system. Data arrives from a variety of different sources. The INGESTOR actor parses the input and transforms the data into VAST's internal event model before compressing it into *segments* and shipping it to VAST's core over the network. To allow for flexible deployment scenarios, the core may run on a single box as well as on a cluster of commodity machines, enabled by `libcppa`'s network-transparent messaging layer. The data segments arrive at the RECEIVER actor, which assigns the segment a unique identifier from a 64-bit ID space and then forwards it to ARCHIVE (a key-value store of compressed data segments) and INDEX (a horizontally partitioned array of type-specific bitmap indexes [4]). After decompressing the segment, the indexing process starts multiple worker actors, each of which process a slice of the contained events. Because `libcppa` uses copy-on-write semantics for intra-process messages, the workers can share the same (immutable) data safely. The high degree of concurrency enables single-machine ingestion rates of 100K events/sec and index construction rates of 50-200K events/sec, varying based on the data type to index (e.g., booleans vs. containers of values).

Clients request specific events by formulating boolean expressions. For example, the query

```
:addr in 10.0.0.0/8 && conn.proto == "udp"
```

would match all events containing an IP address from the subnet 10.0.0.0/8 with transport protocol UDP. Both predicates in this conjunction are strongly typed and the LHS and RHS must have compatible types. In the example above, the LHS of the first predicate represents a *type query* that applies to all events containing one or more values of type `address`. The LHS of the second predicate represents a *schema query* and refers to specific value (or set of values) that resolve according to `conn.proto`, where `conn` represents the event type and `proto` the name of the data value. VAST autogenerates schemas during data ingestion. In the future, we will allow operators to manually define their own schemas to perform explicit type conversions (e.g., to parse string values as IP addresses) and augment values with domain-specific semantics (e.g., label IP address values as connection originator and responder).

When a query enters the core, SEARCH performs syntactic and semantic checks against the schema and then forwards the parsed query AST to INDEX, which performs the query upon the relevant partitions. Each predicate yields a separate index hit, which INDEX combines asynchronously as they arrive. The index hits serve as pointers into the events located at ARCHIVE. A dedicated QUERY actor started by SEARCH receives hits, asks ARCHIVE for the corresponding segments, decompresses the received segments into events, checks for candidate matches (if the indexing cannot provide guaranteed matches for a given operator), and sends the matches back to CLIENT.

VAST currently comprises 25K lines code, available open-source under a BSD-style license.[3] In the near-term future, we plan to deploy and use VAST operationally at the Lawrence Berkeley National Laboratory.

3. DEMO

In our demonstration, we showcase how operators investigate security incidents in real time. Moreover, we allow attendees to interact with the system and ask their own queries.

Specifically, we illustrate typical workflows of security analysts. These often begin with a piece of intelligence (e.g., a sensitive URI, an IP address of a compromised machine, an MD5 hash of a file), which serves as starting point for an interactive, iterative search. Through refinement of search criteria, the analyst converges on a relevant set of events that isolate the activity germane to the incident.

Our VAST instance runs remotely and contains hundred of millions of events produced by the Bro network monitor [5], covering a time span of several months. We will show that interactive analysis with distributed actors is feasible for such numbers.

4. REFERENCES

[1] ALLMAN, M., KREIBICH, C., PAXSON, V., SOMMER, R., AND WEAVER, N. Principles for developing comprehensive network visibility. In *Proc. of Workshop on Hot Topics in Security (HotSec)* (July 2008).

[2] CHAROUSSET, D., SCHMIDT, T. C., HIESGEN, R., AND WÄHLISCH, M. Native Actors – A Scalable Software Platform for Distributed, Heterogeneous Environments. In *Proc. 4rd ACM SIGPLAN Conf. on Systems, Programming, and Applications (SPLASH '13), WS AGERE!* (Oct. 2013), ACM.

[3] HEWITT, C., BISHOP, P., AND STEIGER, R. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd IJCAI* (San Francisco, CA, USA, 1973), pp. 235–245.

[4] O'NEIL, P. E. Model 204 Architecture and Performance. In *Proceedings of the 2nd International Workshop on High Performance Transaction Systems* (London, UK, 1987), Springer-Verlag, pp. 40–59.

[5] PAXSON, V. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks 31*, 23–24 (1999), 2435–2463.

[1] `http://libcppa.org`

[2] `http://www.ebaytechblog.com/2014/03/11`

[3] `https://www.github.com/mavam/vast`