

Effective Monitoring of a Survivable Distributed Networked Information System*

Paul Rubel, Michael Atighetchi,
Partha Pal
BBN Technologies
{prubel,matighet, ppal}@bbn.com

Martin Fong
SRI International
mwfong@sri.com

Richard O'Brien
Adventium Labs
richard.obrien@adventiumlabs.org

Abstract

In 2002, DARPA put together a challenging proposition to the research community: demonstrate using an existing information system and available DARPA developed and other COTS technologies that a very high level of survivability against unconstrained attack by a nation-state-level red team is achievable. This report describes the monitoring, intrusion detection, and reporting infrastructure of the resulting system, highlighting the design principles and lessons learned that are generally applicable to survivable information systems.

1. Introduction

Over the past decade, substantial resources have been invested in the defense of sensitive information systems against the threat of cyber-attacks. Initially, system security focused on attack prevention. However, it soon became obvious that it is impossible to build a system that is invulnerable to every new attack, but yet is usable, interoperates with others and is economically viable. These realizations lead to a focus on detection. If attacks cannot be prevented, the thought was, they must be quickly detected so that human operators can take remedial actions. However, intrusion detection systems (IDSes) do not identify all “zero-day” attacks and can produce a large volume of low-level alerts, including many false positives. From the realization that not all attacks can be prevented, and some may not even be detected accurately and early enough, came the newest focus: tolerate or survive the undesirable effects caused by (partially) successful attacks.

Each new realization increased the necessity, importance, and complexity of monitoring. Initially, logging was less critical because all attacks were “prevented”. As detection became necessary, monitors were placed at the hosts and along network

flows. In order to survive rather than just detect attacks, additional monitoring of the network, within hosts, and within applications is needed.

Implementing the required level of monitoring is non-trivial and involves finding solutions to a set of challenges. Where should one place sensors so that sensor coverage and fidelity are maximized? How can one mitigate the tension between protection and detection (e.g., how to encrypt all traffic without blinding sensors)? How does one deal with information overload and resource over-utilization while enabling the processing of very large amounts of monitoring events?

In this paper we describe how we dealt with these monitoring challenges in the context of designing the survivable version [1] of a military information management system. The design, by selectively combining aspects of protection, detection, and adaptive response, provides a high barrier to entry from outside the system and also makes it difficult to spread attacks within the system; increases the likelihood of detecting attack activity as early as possible, even if the attack source cannot be determined exactly; and copes with attacks by adapting to the changes they cause. We give an overview of the survivable system in Section 2, and then describe sensor placement and alert generation in Section 3. Section 4 notes the techniques used to boost alert fidelity. Section 5 focuses on correlation techniques in alert processing. We summarize lessons learned from Red Team Exercises in Section 6, present related work in Section 7 and conclude with a summary and future work.

2. Background – The defense-enabled JBI

A number of efforts have been under way to enhance the reach and capability of information management systems to meet the DoD’s need for network centric warfare. The Joint Battlespace Infosphere (JBI) is one such effort. The JBI is a

* This work was supported by DARPA and AFRL under Contract no. F30602-02-0134

framework that allows diverse clients to interact with each other in a decoupled manner.

An exemplar JBI was the starting point for our efforts to establish the new high-water mark in survivability. We first designed and validated a survivability architecture (henceforth called the “DPASA¹ architecture”), and then used it to create a survivable version of the exemplar JBI. We called our new JBI version the “defense-enabled JBI”, which was subjected to multiple rounds of Red Team Exercises.

One of the key survivability requirements was that the defense-enabled system must survive 12 hours of sustained attacks by a nation-state level adversary with fairly unrestrictive rules of engagement. Therefore, the survivability architecture adopted a stringent view of “trust”, requiring proofs and independent corroboration before granting a service request or executing a command. This independent corroboration, along with traditional detection, is one of the main motivators of the monitoring infrastructure included in the architecture.

The architecture has an elaborate configuration of observers ranging from traditional Intrusion Detection Systems (IDSs) to application embedded sensors and policy enforcement engines. These capture information and monitor for abnormalities throughout the system.

Figure 1 shows the notional representation of the defense-enabled JBI. The clients are organized in 4 different LANs, shown at the bottom. The core services are organized in 4 redundant LANs, also known as quads, shown at the top. Each of the 8 LANs includes an EMERALD NIDS (Network Intrusion Detection System) and a Cisco PIX VPN (Virtual Private Network) router as the perimeter gateway. Every host is equipped with an Autonomic Distributed Firewall NIC card (ADF) [2] that restricts communication to virtual private groups of hosts and is controlled by a policy server (PS) located, redundantly, in the quads. Each quad also contains a Correlator (qXcor), which runs the EMERALD Correlator [3] and an Advisor component. Further details of the architecture can be found in [1].

The next section explains how we monitor the various complex components mentioned above, and how we combine various detection capabilities.

3. Sensor Placement and Alert Generation

Effective monitoring of a complex system requires designing for sensing and interfacing with existing

¹ DPASA is the project acronym, which stands for Designing Protection and Adaptation into a Survivability Architecture.

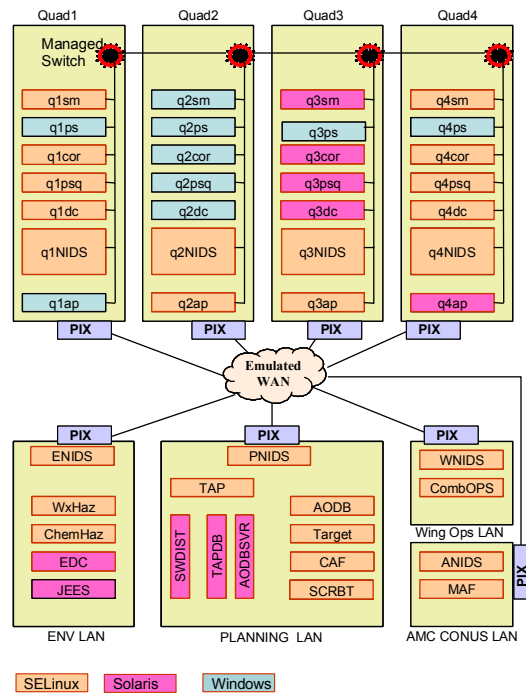


Figure 1: The defense-enabled JBI Exemplar System

reporting facilities. The goal is to detect attacker presence as early as possible, even if it is not possible to identify the precise location of the attacker. The general principle we followed toward that end is to cover each subsystem with sensors distributed across various system layers (network, host, OS, middleware etc.). Each sensor has its own focused point of view and watches either entry/exit of data and control messages, or application behavior. Collectively the sensors should provide an overlapped coverage of the attributes we are interested in monitoring. We also attempted to minimize monitoring that is inline with application functionality and instead pushed monitoring to offboard processors or processes that are not part of the end-to-end application functionality.

This overlap in coverage goes hand in hand with the Defense in Depth (DiD) features of the DPASA architecture, which implies that in order to reach a high-value target an attack needs to penetrate or bypass a number of defenses. Each of these defenses has its own understanding of what is permissible and provides an opportunity for monitoring: anything that is not allowed by a layer of defense is cause for an alert. These alerts can initially be generated with very little analysis as the very existence of traffic is indicative of a problem. Sometimes a more in-depth analysis of alerts at a higher level is also necessary

(e.g., Byzantine behavior may not be notable as such at the packet level but may manifest itself at the application level).

Once a sensor has detected a malicious event, the next step is to report it. Contrary to sensor coverage, diversity in reporting format and mechanism is undesirable. For DPASA we encoded all alerts as EMERALD alerts. These alerts were then funneled to Correlator processes running in the core and to EMERALD's eAMI processes.

The rest of this section provides details about the mechanisms that produce alerts, and improvements to enable earlier and more accurate detection.

3.1. Network-Level Alerts

3.1.1. Emerald NIDS. The Emerald NIDS is a COTS product for detecting abnormal network traffic on a protected network. It uses pattern-based anomaly detection and performs local correlation that combines alerts according to source and target IP addresses.

For use in DPASA we made three modifications to the base NIDS that are discussed here. First, we enabled the NIDS to take advantage of properties provided by our ADF NIC cards. Second, we made the NIDS aware of differences between encrypted and non-encrypted traffic. Finally, we modified the NIDS to protect multiple LANS. The NIC configuration of the NIDS is shown in Figure 2; A, B, and C are ADF NICs where A is used for reporting while B and C are used for sensing.

The first NIDS modification dealt with encrypted traffic and the ADF cards. Recall that an ADF encrypts traffic as it leaves the NIC using a key specific to a VPG. This is problematic for the NIDS, which needs to inspect the packets for many of its checks. In order to make the packet content visible to the NIDS, the NIDS' ADF was made a member of each VPG. This allowed inspection of VPG traffic but also made the second change necessary.

In a normal configuration, ADFs pass decrypted version of traffic from the VPGs it is a member of to the host and drop all other traffic including non-encrypted traffic. In this configuration a host only sees the "legal" traffic from its network interface. However, one needs to monitor all network traffic, legal or illegal, encrypted or unencrypted. The ADF audit (described next) covers encrypted but illegal traffic (i.e., traffic from VPGs that should not come to this NIC). But without the addition of an additional, non-ADF NIC (for each sensing NICs i.e., B and C in Figure 2), the NIDS will not have visibility of Non VPG traffic that was sent in clear and might contain attack packets and might have spoofed addresses (the ADFs prevent spoofing in

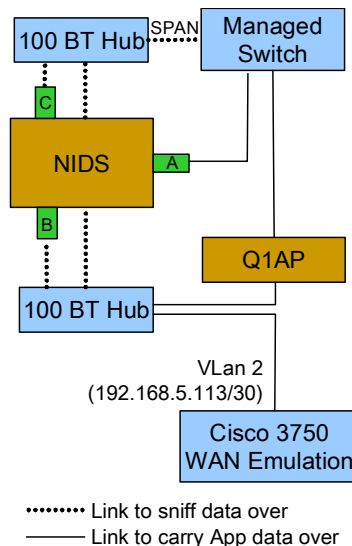


Figure 2- Network Topology Diagram for the NIDS of Quad1

VPG traffic). Our second change was to add this non-ADF NIC and the logic to take the source NIC into account. This configuration can be seen in Figure 2 where each ADF NIC, (A,B,C), has an additional non-ADF network connection to the same network.

Finally, as seen in Figure 1, there is an emulated WAN between the quads and the clients. If the NIDS devices only watched traffic on the LAN they are on, incidents emanating from outside the LAN, which could potentially be blocked at the border firewall (PIX, ADF, etc.), would not be visible. These external incidents, such as port-scans, could be precursors to more effective attacks that that penetrate external boundaries, and it would be beneficial if these are visible. For this reason, our NIDSs sense traffic from both inside the LAN and outside. Note that the introduction of a VPN via PIX routers has an effect on this monitoring. The same argument for putting the NIDS probes outside the LAN but within the VPN could also be made for moving the probes outside the VPN. The NIDS needs to be able to see VPN traffic unencrypted on the inside of the VPN but also needs visibility outside the PIX for attack traffic that does not make it into the VPN. This lesson was learned the hard way via red-team experimentation. We note more in the Lessons Learned section at the end of the paper.

3.1.2. ADF Audits. In addition to performing VPG encryption and packet filtering, ADF NICs can generate an audit alert if a packet does not match any of the rules, if the packet has a spoofed source address, or if an attempt was made to spoof an audit

packet. Because generating an ADF audit required sending a packet over the network and because the ADF cards do not have a mechanism for throttling traffic, our ADFs were configured to only audit outgoing traffic. This prevents an attacker from using the ADF audit as a denial of service multiplier by sending bad packets to the card that would cause the card to generate additional network traffic to report the bad packet.

The collection of ADF alerts is centralized at the PS, which receives audit traffic from all ADF cards and generates alerts when appropriate. The PS can be configured to consolidate audit events so that an alert is not generated until a specific number of audit events occur, or once a particular type of alert is generated another will not be generated for a given time period. In addition to audit-based alerts, the PS also generates application-level alerts, i.e. upon distributing policy changes to ADFs and receiving conflicting commands from SMs. In order to not have the PS be a single point of failure and allow continued audit traffic analysis, in the face of a PS failure or attack, operators could switch over between primary and backup PSs when necessary.

3.2. Host-Based Alerts

3.2.1. Local Controllers. Each DPASA host runs at least one instance of the *Local Controller* (LC) which extracts and correlates local alerts generated due to policy violations, and sends alert summaries to the correlator process in the core. In addition to generating alerts for SELinux and CSA policy violations, the LC also monitors system integrity and performance, and generates alerts when anomalies are detected. In effect, the LC is responsible for consolidating many alerts on a host.

Core hosts each have a single LC while each client host, viewed as more susceptible to attack, has three LCs that watch one another for signs of compromise. The LC drives sensors, collates host events, and does some rate limiting and aggregation to prevent flooding of the correlation components with reports.

The LC furthermore monitors the system through periodic file system integrity scans. This enables detection of attacks that attempt to replace commonly used executables with malicious ones. When a modification is detected, the file is restored or removed and an alert is generated and sent to the core. In the case where many files have been modified, each modification is noted in the same aggregate alert to preserve bandwidth and processing time. The LC also monitors the CPU load, available memory, and free disk space. It compares this information against a statistical profile, computed during attack-free runs of the system. If a large

deviation is detected from the profile, a report is generated and sent to the core. In addition, on Linux and Solaris hosts, the LC monitors process creation and death, and traces the lineage of new processes. If a new process has an unknown parent, then it is marked as suspect and an alert is sent to the core.

3.2.2. Process Domain Enforcement and Host Intrusion Detection. DPASA uses the concept of protection domains to isolate components in the system and to prevent corrupted processes from accessing critical resources in an inappropriate manner. The intent of a protection domain is to allow the components within the domain a sufficient set of privileges to perform their normal functions while at the same time limiting additional privileges that might be exploited by a corrupted process. In addition, protection domains can be used to protect application specific resources from attack. Any actions that exceed a process's privileges are disallowed and audited. Any audit event indicating disallowed actions raises an alert.

At the OS and process level, Security Enhanced Linux (SELinux) provides process-level protection mechanism for Linux systems and the Cisco Security Agent (CSA) provides the protection mechanisms for Windows and Solaris systems. These allow fine-grained policies to be defined that specify exactly what type of access each process should have to files, network connections, processes and other system resources. Both SELinux and CSA audit any policy violations into a local file and, for DPASA, these files are processed by the local controller, which can turn them into alerts. Types of policy violations include attempts by a process to access a file or directory that was not allowed, attempts to make a network connection that was not allowed, and attempts to change security policy or security credentials. Additionally, on Solaris we deployed the EMERALD [3] Host Intrusion Detection System (HIDS) for additional monitoring and alerts generation.

3.3. Application-Level Alerts

While network, host, and policy alerts are very useful for detecting anomalous or malicious behavior, applications are ultimately the final word on the correctness of the system. While it may be more difficult, hence the success of DiD, it is quite possible for an application to stay within the constraints of host or network policies and take malicious actions. *Ultimately applications need to be responsible for noting and reporting problems.* In DPASA this was accomplished via specific application-level anomaly alerts and more generic Java exception alerts.

3.3.1. Application-Level Anomaly Alerts. There are a variety of checks included within DPASA that look for anomalous application behavior. In general, on account of the additional context available to the applications, these alerts are more serious and important than generic Java exception alerts and prompt quick action from the operators. These checks include:

- Alerts about unexpected or out-of-session clients attempting to publish data.
- Alerts about inappropriate actions by System Managers detected by a Byzantine agreement protocol.
- Alerts about a component that has stopped sending heartbeats to the core.
- Alerts about bad data being published due to size and rate constraints.

3.3.2. Java Exception Alerts. DPASA generated alerts whenever error-level log messages were generated. In some cases it was possible to generate an appropriate and specific EMERALD alert. Other times a generic report was created. When exceptions caused alerts, the generated alert included the stack trace and exception message, as well as the local source of the source of the alert.

One very useful piece of information in an alert is the subject of the alert. For Java exceptions this attribution is not always easy. The sender of an alert may be working correctly and is reporting a problem on the other end of a network connection. Hence, generic Java exceptions are given low confidence and more specific alerts are used if possible (e.g., a JVM security violation may produce a specific policy violation alert).

The lower importance given to generic Java exceptions does not mean that the information is not useful. Java exceptions proved valuable in detecting application-level attacks during the red team exercise.

3.3.2. Rate Limit Alerts. Another interesting problem deals with rate-limiting of alerts. One of the alerts sent by our network code is a rate-limit alert. If too many messages are received in a short period of time: an alert is generated in the receiver, an exception is thrown, and the message is dropped. This poses particular problems to the alert generation code in the logger. If the traffic generating the alert is itself alert traffic, care needs to be taken to stop the system from flooding itself. To get around this problem, the alert code has slightly higher limits than the other code. When a non-alert message is rate-limited, this higher rate threshold allows an alert to be sent out reporting the original alert.

4. Boosting Alert Fidelity

Any method that increases confidence in alert content, be it the source or any other aspect of the alert, makes analysis more straightforward. In DPASA we leveraged a number of technologies to increase our belief in the contents of alerts.

The most effective technique to increase confidence was using the ADFs to make network packets unspoofable. Because the hosts only have limited control over them, packets with spoofed address cannot be sent through the ADF NICs. This means that when an alert is received the source of the network packet can be used to corroborate the contents of the data. If there is a mismatch between the contents of an alert and the network-level sender the alert can be disregarded and replaced with an alert about a sender trying to send false information.

In addition, the defense mechanisms that had a well defined policy provided very high fidelity alerts because any violation or attempted violation of the policies they were enforcing is a serious concern. Note that such policies can appear at any system level—the VPG policies are an example of network level policies, the SELinux and CSA policies are example of host and OS level policies, and JVM security policies and fault tolerant protocols (e.g., detection of a Byzantine behavior) are examples of application level policies.

Finally, ensuring consistent policies across hosts and ADF-supported VPGs also increases confidence. By generating policies from templates and variables shared between multiple policies, commonalities, such as those between quads, can be factored out. This allows specific policies to be generated automatically [4], with fewer errors. With this framework alert receiving components need to do less disambiguation to get to the ground truth of a number of alerts.

5. Alert Processing

Once alerts are generated the next step is to make sense of them in the system. The guiding principle in this regard is to utilize multiple correlation techniques leveraging near-sensor correlation when possible; and to augment traditional correlation with dynamic assessment of trust in system components. This facilitates filtering of the false positives or non-useful alerts, making more critical and trusted alerts more visible to the operators.

Application-level sensors (e.g., rate limiting), the local controllers, the EMERALD NIDS and HIDS, and sensors embedded in policy enforcement mechanisms perform quite a bit of "near sensor" analysis and correlation. DPASA used the

EMERALD correlator [3], which combines probabilistic and mission-based correlation to consolidate and rank a stream of security incidents relative to the needs of the analyst, given the topology and operational objectives of the protected network. After eliminating low interest alerts (defined by operators) the alerts are vetted against the known network topology. A relevance score is produced based on the alert target (where it is situated in the topology) and the vulnerability implications of the incident type. Next, alerts are prioritized depending on the degree to which an alert is targeting a critical asset or resource, and the amount of interest the user has registered for this incident class. Finally, an overall incident rank is assigned to each alert, which provides a combined assessment of the likelihood of success of the reported activity and its impact on the overall mission. The correlator next combines related alerts using an attribute-based clustering algorithm. This results in a filtered, lower-volume, content rich security-incident stream, with an incident-ranking scheme that allows analysts to identify incidents that pose the greatest risk to the mission objectives. To consolidate logically identical alerts (e.g., alerts associated with a TCP port scan from a specific originating host), the EMERALD correlator merges alerts into threads and issue updates to them. When disparate, (i.e. logically non-identical) alerts or threads are correlated, the resultant alert is a meta-alert. Meta-alerts summarize and encapsulate alerts. In turn, meta-alerts, along with threads and individual alerts, may be correlated into other meta-alerts.

In addition to the EMERALD correlator we also made use of an Advisor. The Advisor views the alerts as accusations made by one component against others and uses that view to dynamically update the trustworthiness of the system components using algorithms described in [5]. When the trustworthiness of a component diminishes below a threshold, the Advisor generates "advice" for the system operators such as "reboot host X" or "block host Y's NIC".

6. Lessons Learned from the Red Team Exercise

Previous sections have primarily focused on design aspects—how to build an effective distributed infrastructure for monitoring security incidents. This section deals with lessons learned, and in some cases relearned during red-team evaluations.

In the introduction we discussed the problem of false positives and our goal to reduce them. Using the sensors discussed in Section 3, combined with the fidelity boosting mechanisms in Section 4 and the corroboration mechanisms from Section 5 we

reduced them to a manageable level. This reduction was achieved using a combination of alert reduction and alert calibration.

The White Team analysis, delivered to DARPA, of the exercise showed that in the run with the least amount of alert reduction, 20663 low-level incidents were reported by sensors but only 545 of them were turned into alerts. This represents a reduction of greater than 37 times. Other runs had even greater reduction factors. A large fraction of the sensed incidents that were eliminated in this process were false positives eliminated due to our aggregation and cross checking algorithms, buttressed by our policy enforcement and anti-spoofing measures.

During testing, involving vast numbers of non-attack runs, we gained a high level of confidence that some alerts were in fact benign. Though these false positives were presented to operators, they could be safely ignored. Time pressure kept this process from being automated, though as a lesson-learned we should have made time for an automated fix. Even with a low rate of spurious alerts, some of which were expected by users beforehand, it was non-trivial to quickly check if anything anomalous was going on, particularly during an actual attack.

Finally, as noted in Section 3.1, we learned a lesson when our VPN routers were taken down by the Red Team. When deploying our VPN we thought that since we were using a well-tested COTS product we could get away without monitoring outside the VPN. This was a costly mistake as it rendered us unable to detect an attack that stopped the system. Though this visibility would have meant additional events and alerts to process, the alternative, complete blindness, is definitely worse.

7. Related Work

Monitoring is an active area of research in both academic and commercial areas. Network administrators use a large variety of COTS, open-source, and research monitoring products. SNMP-based tools, such as Nagios, enable continuous network monitoring via polling as well as on demand notification via traps. Conversely, end system mechanisms such as virus checkers and automatic updates, provide similar capabilities to end-systems. Ganglia [6] efficiently aggregates data about clusters in a trusted environment. Astrolabe [7] focuses on aggregating large-scale system state into summaries.

The main differentiation of our work is how the monitoring solution deals with coverage of sensor placement. We focused on designing overlapping sensor regions into a survivability architecture to boost alert fidelity while maintaining strong security guarantees, even at the cost of performance.

Previous papers on DPASA have focused on the mechanism for trust assessment [5], policy generation [4], and an overview [1]. This paper takes these as given and presents how the events are gathered and an architecture for their combination.

8. Conclusions and Future Work

Knowledge about the state of the system is essential for survival. Without such knowledge effective realignment of the system under attack is impossible. Choosing how and where to place sensors, as well as how to transport and distill their reports is a critical aspect of a survivability architecture. This paper has noted a number of techniques for placing sensors, making them more effective, and for aggregation and analysis of sensor output. During the red-team exercises, the sensing and reporting infrastructure successfully detected attack symptoms and provided gross localization contributing to about 75% success rate against an adversary with considerable access and privilege within the system.

However, interpreting and responding to the alerts required expert level understanding of the system by the human operators (in the exercises the developers played the role of system operators). This kind of dependency on experts to reason about alerts is costly, and not scalable. Moving forward we are working on automated mechanisms that minimize the expert involvement in interpreting alerts and selecting responses.

9. References

[1] J. Chong, P. Pal, M. Atighetchi, P. Rubel, F. Webber, "Survivability Architecture of a Mission Critical System: The DPASA Example", in *Proc. of the 21st Annual Computer Security Applications Conference (ACSAC), December 2005*.

[2] T. Markham, L. Meredith, and C. Payne, "Distributed Embedded Firewalls with Virtual Private Groups", in *Proc. of DARPA Information Survivability Conf. - Volume II, April 2003*

[3] P. A. Porras and P. G. Neumann. "EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances", in *Proc. of the 20th National Information Systems Security Conf, NIST, October 1997*.

[4] P. Rubel, M. Ihde, S. Harp, and C. Payne, "Generating Policies for Defense in Depth", in *Proc. of the 21st Annual Computer Security Applications Conference (ACSAC), December 2005*.

[5] P. Pal, F. Webber, M. Atighetchi, and N. Combs, "Trust Assessment from Observed Behavior: Toward and Essential Service for Trusted Network Computing", in *Proc*

of IEEE Network Computing and Applications (NCA) July 2006.

[6] M. L. Massie, B. N. Chun, and D. E. Culler, "The GAnglia Distributed Monitoring System: Design, Implementation, and Experience", *Parallel Computing, Volume 30, Issue 7, 2004*, pp. 817-840.

[7] R. Van Renesse, K. P. Birman, and W. Vogels. "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining", *ACM Transactions on Computer Systems, Volume 21, Issue 2, 2003*, pp 164-206.