An Architecture for Scalable Network Defense

Tim Strayer, Walter Milliken, Ronald Watro,
Walt Heimerdinger[†], Steve Harp[††], Robert P. Goldman[†††], Dustin Spicuzza,
Beverly Schwartz, David Mankins, Derrick Kong, Peiter Mudge Zatko
BBN Technologies *[†]Honeywell* *[††]Adventium* *[†††]SIFT, LLC*

{*strayer|milliken|rwatro|dspicuzz|bschwart|dm|dkong|mudge*}*@bbn.com*
walt.heimerdinger@honeywell.com; steven.harp@adventiumlabs.org; rpgoldman@sift.info

***Abstract*—We describe a novel architecture for network defense designed for scaling to very high data rates (100 Gb/s) and very large user populations. Scaling requires both efficient attack detection algorithms as well as appropriate an execution environment. Our architecture considers the time budget of traffic data extraction and algorithmic processing, provides a suite of detection algorithms—each designed to present different and complementary views of the data—that generate many "traffic events," and reduces false positives by correlating these traffic events into benign or malicious hypotheses.**

I. Scalable Architectures

One of the most pressing challenges imposed on network defense mechanisms is the significant increase in network speeds. While the well-known Moore's Law states that compute power doubles every eighteen months, Gilder's Law [1] states that communication power doubles every six, suggesting that bandwidth grows at least three times faster than computer power. This is a harsh reality for computer network defense; the implication is that defensive strategies must be inherently scalable, or they become moment-in-time solutions.

Scalable attack detection algorithms must operate efficiently and effectively without regard to the bandwidth of the input. Since bandwidth triples with computation power, it is impossible to consider "scalable" algorithms without also considering the scalability of their execution environments. The increasing volume of input also implies that there is less time available to investigate each alert issued by the algorithms, precipitating the need to have fewer, higher valued alerts. A truly scalable network monitoring solution, therefore requires innovation in the scalable algorithms themselves, the ability to extract and process traffic features at line speed, and reduction and aggregation of the output into high value alarms.

In the abstract, algorithms take some input, process it, and then generate some output. Network defense algorithms take as input features extracted from the network traffic. There are physical limits to hardware, and the most important one is the speed of accessing memory. As the network line rate increases, DRAM memory access latency dominates many network processing algorithms. While algorithms may have inherent theoretical properties that govern how well they scale, until those algorithms are applied to a real execution environment, their scalability remains only theoretic.

An effective monitoring solution must also consider how to handle the output of the algorithms. Attack detection algorithms with a single, myopic view of the network tend to issue many more false alarms than true ones, causing the operators to either ignore the alarms (guaranteeing that all attacks evade detection), or raise the alarm threshold (allowing some attacks to pass that would have otherwise been caught).

An attack detection algorithm's output—the alarms it generates—is a product of the algorithm's internal model of the network and actions of the attack. Most of the time these models are implicit in the algorithm itself; when the algorithm makes a choice about the maliciousness of network activity, it is applying its internal models, and therefore the results are limited to the model's accuracy and coverage. A better approach is to employ explicit network and attack models at a correlation point, where *events* (not yet alarms) from multiple algorithms can be reinforced to provide high value alarms, or explained away to reduce false positives.

Current network monitoring algorithms face strong challenges when their use is intended for very high speeds. Several approaches have attempted to address these challenges. Broder and Mitzenmacher [2] describe the use of Bloom filters to speed up the accounting of network data traffic; Estan and Varghese [3] describe research directions in the context of gathering information from high speed net-works; Iannaccone *et al.* [4] describe an effort for basic measurements of relative high-speed links (e.g., OC-12 and OC-48) employing special network interface cards (NICs).

II. SMITE

With research funding from the US Government, we are developing a network monitoring solution called *SMITE: Scalable Monitoring in the Extreme*. The SMITE architecture design reflects three key observations on building for scale. First, very high data rates necessarily cause changes in the way traffic features are extracted from the network. Second, the features that are extracted must be appropriate to support a broad range of attack detection algorithms. Third, the algorithms should be viewed as contributing evidence about attacks, where the evidence is correlated to prove (or disprove) an attack hypothesis.

A. Layered Approach

Figure 1 shows the three levels of the SMITE architecture: Sensors, Algorithms, and the Correlator. The sensors extract simple traffic features and other information from the network

This material is based upon work supported by DARPA under contract N66001-08-C-2050 with SPAWAR. Approved for Public Release, distribution unlimited.

traffic and place that data into temporary data structures. Here, very close to the actual traffic, there is no time budget for complex analysis or careful culling of the alerts generated. Instead, the sensors perform very fast processing to build the data structures that act as a first pass aggregation of the traffic features.

Data collected at one or more sensors is passed to the Algorithms Layer, where various algorithms do more complex processing of the extracted traffic features and build more complex data structures. The time budget at this layer is not as constrained because the algorithms are operating on aggregated data periodically collected from the Sensors Layer.

The top layer in the architecture, the Correlation Layer, holds the Event Correlation Analyzer. It takes as input the events generated by the various algorithms and, using models of both attack profiles and normal network behavior kept in the correlation knowledge base, explores a space of hypotheses to determine which events are truly high value alerts.

Note that most intrusion detection systems do not explicitly separate the Sensors and Algorithms Layers because both functions—feature extraction and feature analysis—are performed on the same computational platform. However, the two layers require very different execution environments, especially at high data rates. The number of memory accesses is the key driver for the design of a sensor, since memory is by far the slowest aspect of processing the traffic, while the time budget allows algorithms to run on a more conventional computational platform.

Also note that a feature of this architecture is that it supports algorithms that operate on different temporal and spatial scales. Sensors operate on very little context—that of the packet only—and have very little time to process the features. The algorithms operate on features collected from many packets, allowing detection of events that occur within flows and even over several flows. Since data can be stored much longer at the Algorithms Layer, algorithms that employ long-term trend analysis can be used here as well.

This layered architecture provides an extensible framework for adding new algorithms and expanding the deployment footprint, supporting both singular and distributed deployment.

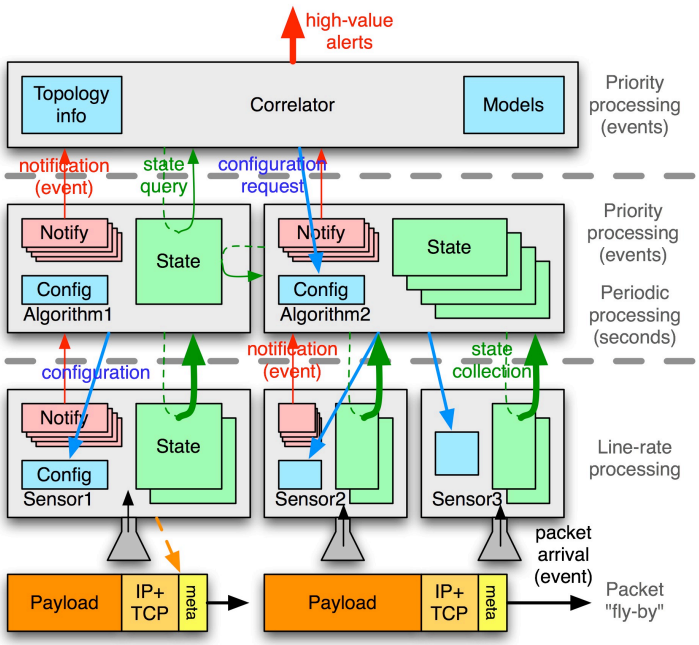


Figure 1—SMITE Layered Architecture

B. Scaling Issues

The primary place to address scaling is at the Sensor Layer, where the sensors meet the traffic, and where lessons from router design can be applied. The key driver in high-speed network processing system design is the duration of a minimum-size packet at link speed; any operation that takes longer than this time must be parallelized either by breaking the operation down into smaller steps (e.g., pipelining), or by spreading the load over multiple processing elements (e.g., cluster parallelism).

Successful high-speed router designs generally employ pipelining rather than load splitting because pipelining rarely changes the behavior of the processing, while load-splitting can introduce ordering and state-sharing complications. Load-splitting designs usually depend on flow bandwidth being small relative to a single processing element, and on passing all packets of all flows that share state through a single processing element. These characteristics do not necessarily hold for the target environment for a network monitor—single flows can be multiple gigabits in bandwidth, and many detection algorithms of interest must examine traffic across multiple flows.

The minimum packet duration (T_{min}) constrains the design choices for hardware implementation at multi-gigabit speeds. At 1 Gb/s, T_{min} is about 500 ns, depending on the link layer used. A T_{min} of 500 ns allows thousands of instructions per packet in a single conventional 3 GHz CPU core, but only about 10 random memory references to main (DRAM) memory. Six of these 10 memory accesses are used just to read the packet into memory and selected fields into the CPU registers. This leaves a scant 4 memory accesses per packet for sensor data structures. These large structures usually will not fit into CPU L1 or L2 caches, and exhibit no locality of reference, so access to main memory is typical. It is worse, of course, at higher line rates.

Thus, conventional processors are marginal at processing packets at 1 Gb/s line rate, so load-splitting parallelism would be required. Since this restricts algorithm choice, may require additional bookkeeping about packet ordering, and scales poorly to 100 Gb/s (requiring hundreds of CPUs, each with its own main memory), we use high-performance pipelined

hardware, the approach that has proven itself successful in router applications [5].

Despite the advances in the area of network monitoring, the current state of the art does not achieve the monitoring at very high speeds. Monitoring network traffic with the intention of feeding network defense algorithms operating at these speeds requires special platform designs, and many approaches are based on research into high-speed routers [6]. Special purpose network processors, such as Intel's IXP line [7], are capable of high speed switching but fail at more complex processing due to memory limitations.

The SMITE platform is therefore a high-performance pipelined FPGA hardware design that can support a wide variety of sensors that examine the traffic stream flowing past each sensor in the processing pipeline. In general, computation is not a limiting factor; the primary limitations on sensors are memory access time and memory table size.

C. Event Correlation

Any system that seeks to provide high value alarms that are actionable and accurate with few false alarms must aggregate and fuse as much detector information as possible. Individual detection algorithms will report suspicious or malicious activity with varying degrees of detail or confidence. Yet data from as many detection alerts as possible must be included in the process of generating high value alarms. The reasoning process must analyze detection events with widely varying scope and credibility in a single framework.

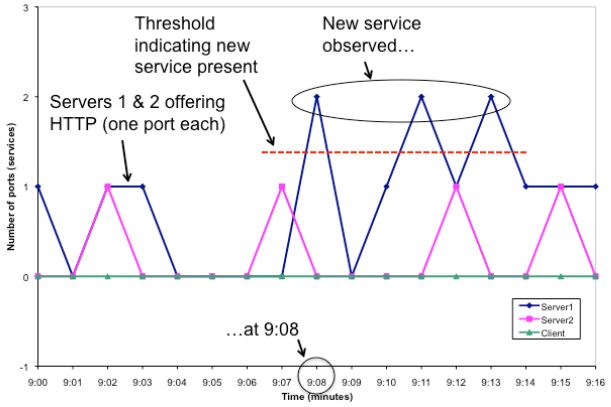
Our approach for building the Event Correlation Analyzer component is to leverage the successful Scyllarus event correlation engine [8]. Scyllarus synthesizes high value alarms from these events in a three-step process: First, events from individual detection algorithms are clustered with events sharing common attributes such as time of arrival or locality in address space. Next, based on these clusters, event explanations are hypothesized, some as malicious, and some as benign. Since some of the events represent competing hypotheses to explain the same reports (e.g., what looks like malware transfer might be simply software update), the final stage of the process weighs evidence for and against different event hypotheses.

D. Attack Detection Algorithms

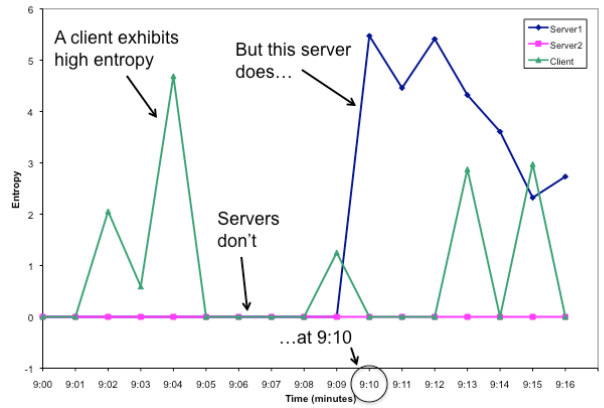
Informed by these architectural considerations, SMITE employs algorithms covering the major forms of attack: malicious behavior, malicious code infections, information gathering, and covert control of assets. These algorithms are not a canonical set of algorithms; while they do cover an interesting set of attacks, the SMITE architecture is designed to be extensible to new sensors and algorithms.

It is important to note that there is some overlap in the attacks these algorithms can detect. This is a feature we exploit in SMITE. Reducing the number of false alarms, and consequently increasing the number of high value alerts, requires corroboration and correlation of data gathered by the various algorithms.

A: New Server Detected



B: Entropy Too High



C: Client Acts Like a Server

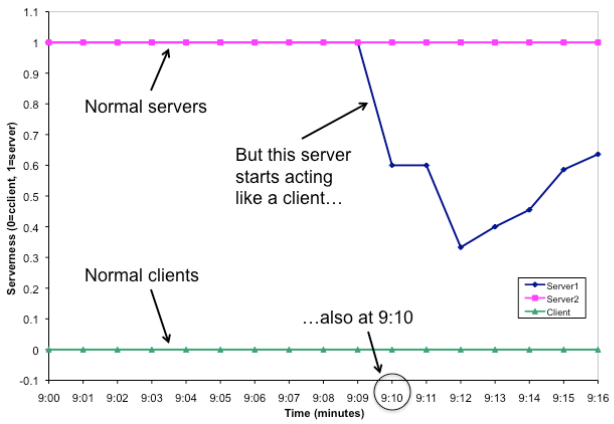


Figure 2—Detection Algorithms

III. Worked Example

Figure 2 shows a worked example of the SMITE detection architecture. The monitor is placed at the gateway of an enterprise network, where there are two well-known web servers. In this attack, one of the internal servers has been compro-

mised and BitTorrent has been installed. BitTorrent is a peer-to-peer network where files are distributed among various hosts, and hosts both get and serve these files, so BitTorrent nodes are simultaneously both clients and servers.

Figure 2a shows the results of the "Host Characterization" algorithm that detects new servers. At 9:08, a new service port is detected on Server 1. Figure 2b shows the "Entropy" algorithm tracking the diversity of connections from a particular host. Clients naturally show higher entropy because they typically initiate many connections, where servers do not. At 9:10, Server 2 starts showing many outgoing connections. Figure 2c measures how much a host acts like a server, which is part of the "Connection Analyzer" algorithm. At 9:10, Server 1 starts acting like a client.

Figure 3 shows how the Event Correlation Analyzer is building evidence towards a hypothesis of attack. When new service ports where added to Server 1, the Correlator made note of that ("Server Adds Ports" became "Observed"). There are two explanations, one benign ("Legit Svc Added") and the other malicious ("Infected"), but there is not enough evidence to pick. Then Server 1 started connecting to many hosts ("High Entropy External"), not what servers do, and it began to act like a client ("Server to Client"), again not what servers do. The plausible explanation for both is that Server 1 now "Initiates Conns". Combined with the possibility that the server is "Infected", enough evidence is present to set the alarm.

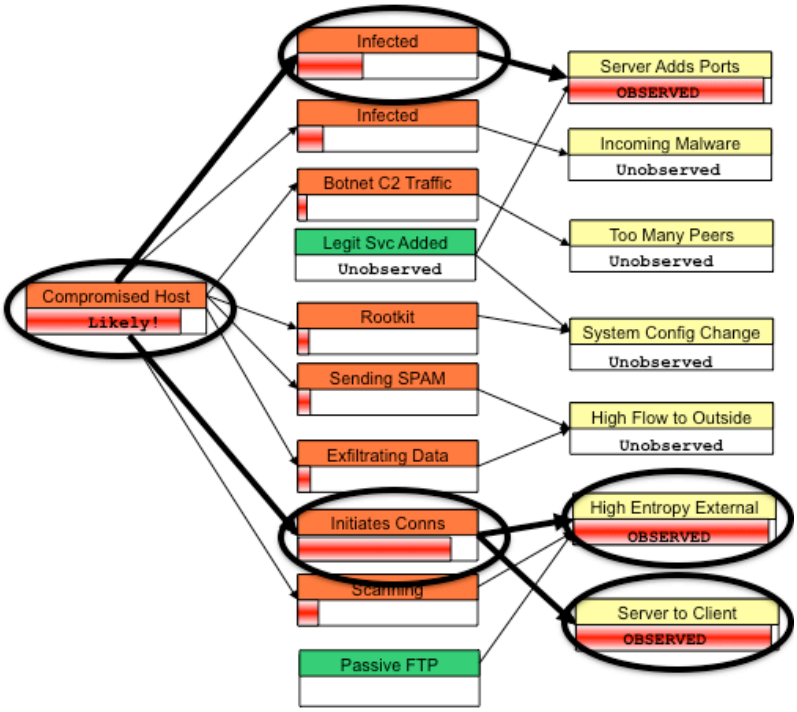


Figure 3—Event Correlation Analyzer

IV. Conclusion

This paper presents the SMITE network monitoring architecture designed specifically to scale in both network line rate and network size. The obvious solution to scalable monitoring is to develop and use algorithms that are intrinsically scalable. This is a good and necessary first step, but the key insight to scaling is not just the algorithms but having an appropriate platform on which to execute the algorithms. Since network line rates are outpacing processing speed, network features must be extracted from the traffic and placed into aggregating data structures as simply as possible. Here, at the network interface, it is memory accesses and not computation that is the limiting factor.

The SMITE architecture separates the feature extraction and processing into a Sensor Layer (running at network line rate) and an Algorithm Layer (processing aggregated data, so running with less time constraints). The architecture provides a suite of algorithms providing multiple views into the data. The algorithms produce "events," or notification of anomalous behavior, and pass these events on to the Correlation Layer. The Correlation Layer affords the algorithms the luxury of being liberal with event notification—less effort is required to cull the events within the algorithms themselves since the Event Correlation Analyzer actually works better with more events. Without the Correlation Layer, the events would be alarms, and number of false positives would be overwhelming. With the Event Correlation Analyzer, the system builds hypotheses to explain the observed behavior, looking for both benign and malicious explanations, until one or the other is proved; only then is an alarm sent to the network operator.

References

[1] G. Gilder, *TELECOSM: How Infinite Bandwidth will Revolutionize Our World*, Free Press/Simon & Shuster, 2000.

[2] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters," *Internet Mathematics*, 2004.

[3] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *SIGCOMM*, Pittsburgh, PA, August 19-23, 2002.

[4] G. Iannaccone, C. Diot, I. Graham, and N. McKe-own, "Monitoring Very High Speed Links," ACM Internet Measurement Workshop, San Francisco, CA, November 1-2, 2001.

[5] C. Partridge, et al., "A 50-Gb/s IP Router," *IEEE/ACM Trans. on Networking*, Vol. 6, No. 3, June 1998.

[6] S. Kumar, *et al*., "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection," *SIGCOMM Computer Communication Review*, August 2006.

[7] Intel IXP2XXX Product Line of Network Processors.

[8] W. Heimerdinger, "Scyllarus Intrusion Detection Report Correlator and Analyzer," *Proceedings of the DISCEX-III Conference*, 2003.