RC25172 (W1106-057) June 17, 2011
Computer Science

IBM Research Report

Stream Computing for Large-Scale, Multi-Channel Cyber Threat Analytics: Architecture, Implementation, Deployment, and Lessons Learned

Douglas L. Schales, Mihai Christodorescu, Josyula R. Rao, Reiner Sailer, Marc Ph. Stoecklin, Wietse Venema
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Stream Computing for Large-Scale, Multi-Channel Cyber Threat Analytics

Architecture, Implementation, Deployment, and Lessons Learned

Douglas L. Schales
IBM Research
schales@us.ibm.com

Mihai Christodorescu
IBM Research
mihai@us.ibm.com

Josyula R. Rao
IBM Research
jrrao@us.ibm.com

Reiner Sailer
IBM Research
sailer@us.ibm.com

Marc Ph. Stoecklin
IBM Research
mstoeck@us.ibm.com

Wietse Venema
IBM Research
wietse@us.ibm.com

June 17, 2011

Abstract

The cyber threat landscape, controlled by organized crime and nation states, is evolving rapidly towards evasive, multi-channel attacks, as impressively shown by malicious operations such as GhostNet, Aurora, Stuxnet, or Night Dragon over the past two years. As threats blend across diverse data channels, their detection requires scalable distributed monitoring and cross-correlation with a substantial amount of contextual information. With threats evolving more rapidly, the classical defense lifecycle of post-mortem detection, analysis, and signature creation becomes less effective.

In this paper, we present a highly-scalable, run-time extensible, and dynamic cybersecurity analytics platform. It is specifically designed and implemented to deliver generic capabilities as a basis for future cybersecurity analytics that effectively detect threats across multiple data channels while recording relevant context information, and that support automated learning and mining for new and evolving malware behaviors. Our implementation is based on stream-computing middleware that has proven high scalability, and that enables cross-correlation and analysis of millions of events per second with millisecond latency. We summarize the lessons we have learned over the past three years of applying stream computing to monitoring malicious activity across multiple data channels (e.g., DNS, NetFlow, ARP, DHCP, HTTP) in a production network of about fifteen thousand nodes.

1 Introduction

Modern corporate computer networks have evolved over the past decade from a well-perimeterized layout to a complex of networks that interconnect multiple physical locations, scattered across the Internet. As the productivity of organizations increasingly depends on the availability of their networks, the ability to monitor, analyze, and understand the security impact of network activities has become more important than ever.

Existing monitoring and analytics systems are built on the assumption that all communications and threats can be observed as they enter a network at a small number of vantage points [3, 5, 7, 8, 16, 26–33]. This assumption is no longer satisfied in modern networks. Static, well-defined security perimeters have disappeared due to heterogeneous connectivity methods to the Internet, due to extensive use of tunneling protocols (such as VPNs), and due to growing numbers and increasing diversity of mobile devices (laptops, phones, iPads). At the same time, attackers show increased sophistication and skills in how they combine and evolve existing attacks, in how they employ complex evasive techniques to hide their activities, and

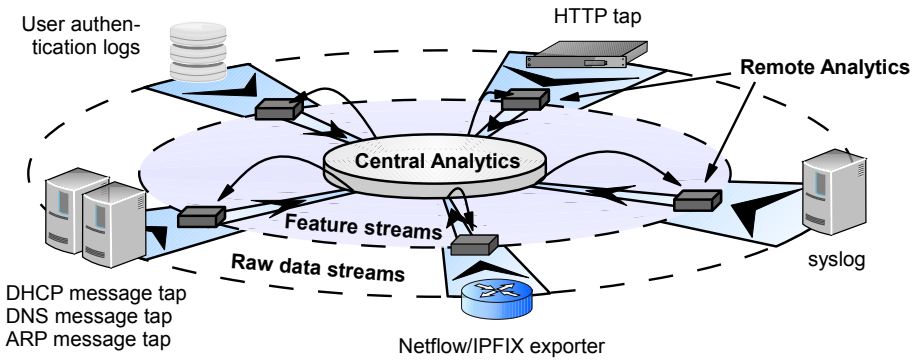

Figure 1: Stream computing for cybersecurity analytics.

in how they use vulnerable systems as intermediate stepping stones to achieve their goals [9]. In parallel, attackers make increasing use of data-driven attacks [4] by exploiting application-level flaws (e.g., web browser vulnerabilities) and targeted social engineering that are invisible to traditional, packet inspection-based systems [24, 25].

In this paper, we present our approach to an extensible, high-performance cybersecurity analytics platform. We turn to the concepts of stream computing as the mechanism for developing and deploying cybersecurity analytics that efficiently cross-correlate multiple data sources, and that detect complex intrusion models in near real-time while retaining the benefit of the recent advances in intrusion modeling, analysis, and performance. Our architecture is able to scale to vast amounts of data in real time, to cross-correlate and aggregate different types of data streams at distributed processing nodes, to handle context-sensitive information centrally while propagating updates to all units, and to incorporate analytics that can be adapted and deployed on the fly. In Figure 1, we schematically depict an overview of our architecture. The core concept in our approach is that of event processing in stream computing (e.g., streams of DNS events, HTTP events, network flows, application logs). Sensors located physically close to the data sources perform an initial processing of incoming raw data streams and forward relevant feature streams into the core of the system. An automated orchestration of data and update streams take care that relevant contextual information is routed to all processing units in a timely manner.

Building on stream-processing middleware, we pursue a novel approach to designing, implementing, and deploying a monitoring and processing infrastructure for cybersecurity analysis. The key insight in our approach is the decoupling of the analysis interface from the deployment mechanism. In other words, a security analyst can write algorithms for analyzing network traffic without depending on physical network topology details, whereas a security administrator can deploy the analytics without having to rewrite the algorithms. Our approach is informed by the lessons we learned over three years from experimenting with stream-computing-based analytics for cybersecurity threat detection and traffic monitoring in dynamic network topologies. The analytics are expressed as algorithms over event streams, and the network is instrumented by sensor elements that consume, combine, and produce event streams. This conceptual modularization and separation of processing promotes parallelizable analytics that allow our approach to scale with the traffic volume, with the network topology, and with the amount of correlations across traffic data features, as event-processing sensors and analytics can be added, moved, and removed in real time. In this paper, we make the following contributions:

- A novel, flexible architecture for cybersecurity monitoring and analysis in highly-dynamic networks that enables multi-channel data analysis and correlation.
- A scalable implementation of the architecture on top of a general-purpose stream-processing engine. We extended the stream-processing engine to locate and combine streams as they become available from the underlying network.

- An evaluation that supports the scalability and real-time goals of the architecture based on our experience with running prototypes, focusing on botnet detection.

2 Background & Requirements

Conventional cybersecurity analytics systems are challenged to detect recent generations of attacks for multiple reasons. First, many systems monitor only a single or few data sources and are therefore limited in the breadth of their view of ongoing activities. For example, many intrusion detection systems deployed today monitor network packet-level interactions at a single vantage point, e.g., the network perimeter [24, 25]. These systems are however oblivious to threats that circumvent the detection, e.g., by entering the network on a mobile device such as a laptop, a USB stick, via social engineering, or through a VPN tunnel. Moreover, existing analytics systems have limited ability to efficiently correlate events across multiple data channels and measurement points in real time. As a consequence, important context information is unavailable to the detection and alerting routines. In computer networks, the lifetime of context information is generally short, and its usefulness degrades rapidly. For example, as the majority of end-user devices become mobile, IP address and switching information may be stale only minutes after it became available to an analytics system: the source IP address of a suspicious network packet loses meaning if it cannot be attributed to a device (or an end user). Accurate attribution of events by combining historical and real-time information is therefore critical when analyzing and responding to ongoing threats. This is a particularly difficult problem when analyzing event streams from multiple channels that have heterogeneous formats and identifiers. Moreover, given that attacks may evolve quickly, new detection algorithms must be implemented and deployed rapidly. Current security analytics systems lack this flexibility and extensibility.

The limitations of existing cybersecurity analytics systems in response to the threat landscape encountered today, have led us to formulate a set of requirements that the next-generation analytics platform should satisfy:

Multi-channel data and context. As the techniques to evade detection become more sophisticated, the ability to combine multiple data channels (or sources) becomes indispensable. The data channels include, e.g., network packet content, flow-level traffic information, address-to-device mappings, network topology information, authentication logs, application logs, or business process dependencies. Such contextual information is critical when attributing a network packet to a specific originating device, user, or process.

Event correlation and history. The analysis of different event channels requires methods to correlate events across multiple data sources efficiently and at high data rates. The combination of the channels enables the construction of the "bigger picture" of activities, which brings an isolated action into the context of a chain of events. While an attack may "fly under the radar" on each individual channel, the correlated view across multiple channels can expose the malicious activities and intents. In addition to the real-time event streams, efficient access to historical data is required to derive models for behavior-based anomaly detectors and to perform forensic analysis.

Scalability and timeliness. To cope with a large set of high-rate event channels, an analytics platform must be able to scale up on the fly. Traditionally, scalability is achieved with two methods: (1) increased processing capacity and (2) concurrent event processing. To sustain a long-term increase in data rates, concurrent processing has proven to be more desirable. In particular, an architecture based on a distributed processing system enables early pre-processing of events at their origin. The results of the analytics should be provided in near real-time to data consumers. This is necessary so as to allow for an immediate alert of administrators and timely response to an ongoing threat (e.g., additional data collection).

Extensibility. As the threat landscape changes, e.g., due to a newly-discovered vulnerability or an evolving attack, new data channels and algorithms need to be dynamically added to the analytics system.

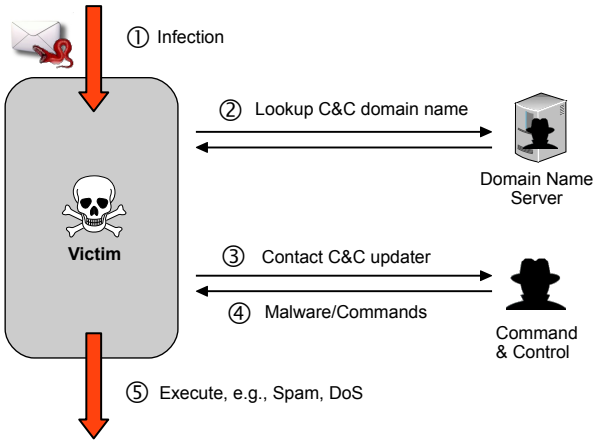

Figure 2: Example of a botnet infection and its command and control (C&C) communication channel.

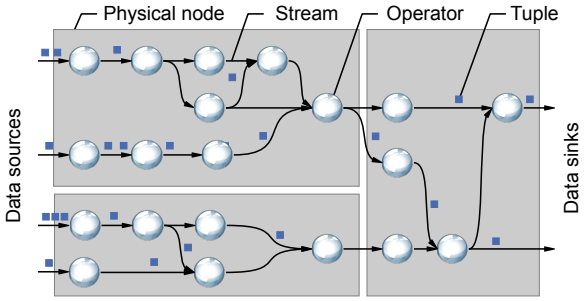

Figure 3: Processing data in a stream-based fashion.

Extensibility captures the ease of designing and adding new algorithms as well as the ease of connecting them to the appropriate data channels, without jeopardizing the normal operation of the system.

2.1 Illustrative Example

In the following example, we illustrate the utility of multiple information sources to detect botnet activity. We use this example later in this paper to demonstrate the ease of designing and implementing detectors on top of our platform.

In Figure 2, we schematically depict the typical activities after a botnet infection of an end user machine, e.g., a laptop. In step 1, the machine is infected by botnet malware which propagates, e.g., through e-mail, a malicious website, or a worm. Once the malware is activated on the victim's machine, it announces its availability to receive orders from an elaborate command and control (C&C) infrastructure. This infrastructure typically hides behind a complex domain name update mechanism, which frequently changes the IP addresses with fluxing domain names [23]. This mechanism allows attackers to conceal their activities behind a resilient infrastructure. After resolving the IP address of the C&C site in the domain name system (DNS) in step 2, the malware contacts the C&C site and obtains commands or additional malware code to be installed (steps 3 and 4). From this point onwards, the victim's machine is remotely controlled (a *bot*) and may be employed to conduct various activities such as transmission of Spam, execution of a scan, or participation in a distributed denial of service attack (step 5).

2.2 Stream-based Event Processing

Our platform builds on the concepts of stream computing. The concept is similar to a typical database management system in that both implement middleware for application development. However, stream processing and database management focus on different parts of the problem, or at least are solving the problem in different ways. While a database management system is a platform for developing applications to store and process data, stream processing platforms are used to develop on-the-fly, incremental processing of data as it arrives.

In stream computing, events (called *tuples*) are processed in real time, filtered, combined, or otherwise transformed to obtain new event streams that contain information of interest. As depicted schematically in Figure 3, data processing is organized and programmed in terms of data flows that connect data *sources*, *sinks*, and *operators*. Operators may be distributed over multiple *physical nodes*, and produce streams based

on the consumption of other streams, where a stream is a sequence of records or events. Operators may make use of external input to enrich a data stream (e.g., annotate or classify data) or receive feedback streams from other operators. The stream processing infrastructure takes care of the routing and allocation of the streams across the operators and physical nodes. It comes with a standard set of these operators along with the ability to create new operators.

Stream computing gives us the power to build cybersecurity analytics that reason about behaviors composed of many events across time and across data sources, in a highly scalable way and with a flexible programming model. Moving to a new computing model for cybersecurity analytics is, however, non-trivial. Streaming concepts such as event windows, aggregators, processing elements, stream joining and stream splitting, do not map directly to programming models known in the security domain. As we will present in Sections 4 and 5, we have developed an architecture and implemented a prototype that we currently use on a real network to better understand the benefits of stream computing for cybersecurity analytics.

3 Related Work

In this section we compare our own distributed architecture with existing IDS approaches, and we compare our own results with those from existing approaches for the detection of botnets and malicious domain names.

3.1 Distributed Intrusion Detection

Distributed IDSes were proposed to combine the alert information from multiple detection sensors placed around the network and on its hosts. In 1991, DIDS was designed to perform distributed monitoring of a heterogeneous network of computers and to aggregate user-login audit information [27]. Subsequent work expanded on this concept by using hierarchical aggregation to improve scalability, as done in GrIDS [29] and AAFID [5], by using a peer-to-peer system to propagate alerts, as done in the CSM framework [31], in Crosbie and Spafford's IDS [7], or by using a combination of hierarchical and P2P alert aggregation, as done in CARDS [32], in DOMINO [33], and in Aussibal and Gallon's IDS [3]. In the commercial space, software such as ArcSight [16] and NetWitness [26] implement a similar IDS architecture based on a two-level hierarchy (or, more simply, a hub-and-spoke scheme).

Common to all of these architectures are two conceptual tiers, one of data collection and one of alert correlation. First, host and network activity data is collected from local sensors in network segments of interest, then possibly aggregated and filtered in the sensor, and then analyzed locally to determine whether the activity is suspicious. Second, in the case of a suspicious activity, an alert and maybe its supporting data are sent to a central system that decides whether this alert is relevant, given the data reported from many other sensors. This two-tier architecture forces a security analyst into an unnatural separation of the detection task of two distinct components, one for local analysis and one for global analysis.

Our cybersecurity analytics platform avoids this issue by providing a unified one-network abstraction for designing the detection analytics, as if the complete network is as readily accessible as a local area network. At deployment time, our analytics are distributed across the network to achieve the desired performance and scalability. In our experience, this provides a significant simplification of the detection task since it matches the global view of the attacker (whose goal is to subvert the network in any way to achieve his goal) to the view of the defender (whose goal is to defend each part of the network).

A second improvement in our cybersecurity analytics platform is the fact that we do not impose any particular structure on how the analytics actually communicate with each other. Thus the security analyst can decide whether a strict hierarchy or a loose P2P community or any hybrid in between is preferable.

3.2 Botnet Detection Analytics

Early botnet detection systems relied primarily on static models. For example, Goebel and Holz [10] recognize IRC-based bots by looking for suspicious IRC nicknames in plaintext traffic. On a larger scale, Karasaridis *et al.* [20] combined network IDS alarm events with network flow information to identify botnets with centralized

command and control (C&C) in a tier-1 ISP network. BotHunter [12] uses customized IDS rules and event correlation to recognize specific stages in bot life cycle models from initial infection to attack propagation. Without reproducing these systems in their entirety, our experimental evaluation in Section 5 shows that even a basic streams platform can easily handle static models and event correlation at comparable levels of complexity and traffic volume.

More recent botnet detection approaches take advantage of machine learning to detect botnet members by the similarities in their behavior. For example, BotSniffer [13] uses an IDS to detect both attack/spam traffic and potential command/control traffic, and uses spatial-temporal correlation of network activity to recognize coordinated bot activity. BotMiner [11] cross-correlates clusters of IDS events with clusters of traffic in netflow logs, and TAMD [34] detects bots by aggregating network traffic with similar communication patterns. Our streams platform currently implements correlation over a time window that is limited by available (non-persistent) memory. For long-term correlations, we are implementing a data-at-rest component. This will form the basis for dynamic models based on machine learning, and will allow us to perform analyses of communication graphs similar to BotGrep [22] and Graption [19].

3.3 DNS Analytics

Currently, botnets deploy multiple DNS-based techniques to increase their availability: especially popular are domain flux, where the domain name is generated with a pseudo-random algorithm, and fast flux, where DNS servers reply with a frequently-changing list of IP addresses. Early fast-flux detection approaches, such as [15, 23], relied on a limited number DNS response properties such as TTL, multiplicity of IP addresses and their ASN numbers, and third-party black- and whitelists. Our experiences in Section 5 show that the streams platform easily supports quick experiments with similar static models.

Later implementations of DNS reputation systems use a more general approach that is based on machine learning. EXPOSURE [6] and Notos [2] are examples of DNS reputation systems that periodically train a classifier on labeled DNS traffic, using lists of known-benign and known-malicious domains. We are currently implementing the data-at-rest component that will enable analyses over long-term data, including dynamic modeling approaches based on machine learning.

4 Architecture

The architecture of our cybersecurity analytics platform revolves around the requirements identified earlier in Section 2. We structure the discussion in this section along two dimensions, *analytics programming* and *analytics runtime*. A security analyst is tasked with developing new detectors, and requires a platform that provides multi-channel data, correlation capabilities, and both real-time and historical data. A security administrator is tasked with deploying the cybersecurity platform, and requires scalability, timeliness, and distributed-processing capabilities. We note that extensibility is a desirable feature for both analytics development and analytics execution.

4.1 Analytics Programming

A security analyst develops detectors by dividing the detection task into logical steps, implementing each step as a component that receives input data and creates output data (alerts or aggregated data for later processing stages), and then combining the components by connecting output and input interfaces. The division of the implementation into components allows reuse of these components, such that detectors that depend on common input data can share the components that produce that data. This natural approach to implementing analytics that detect network intrusion behavior patterns informed our view of the programming model of such analytics. Fundamentally, a detector running on our cybersecurity analytics platform consists of data sources that produce time-ordered series of events and of analytic components that process events to produce other events.

A *data source* is simply a (possibly infinite) series of events that are ordered in time. Data sources include, for example, a router generating Netflow/IPFIX records, a Unix server generating syslog entries, and a packet-capture tool generating raw network packets. Data sources have unique identifiers and are homogenous in the type of events they generate. An *event* is a data item with multiple attributes, a timestamp, and a data-source identifier. For example, an event of the network flow type may have ten attributes: a source IP address, a source port number, a destination IP address, a destination port number, a protocol identifier, a count of incoming octets, a count of outgoing octets, an application protocol identifier, and, of course, a timestamp and a data-source identifier. The data-source identifier allows the analyst to discriminate events of the same type from different data sources.

An *analytic component* processes events of a particular type and generates other events. Analytic components can act as simple filters that discard irrelevant events from their input, they can remember past events and use them in processing the current event, or they can aggregate events into time-based summaries. Because events are both the input and the output of an analytic component, components can be chained together to form long processing pipelines. Furthermore, the analyst has the option to merge events from multiple sources or multiple analytics and to create processing loops where an analytic component receives some events from another analytic component that is further downstream.

Data sources and analytic components form a processing graph, whose root nodes are data sources and interior and leaf nodes are analytic components. Connecting data sources and analytic components to each other is based on the event types ingested by analytics and produced by sources and analytics. For example, one can write an analytic component that inspects flow events without explicitly specifying the data sources for such events. The platform merges events from multiple and flow data sources dynamically as such sources become available and automatically routes them to the analytic component.

4.2 Analytics Runtime

The runtime environment has to allow both single-event processing (in the style of Snort [25] and related IDSes) and the processing of large sets of events (e.g., "compute statistical models from 3 years of DNS traffic"). To achieve this level of flexibility we introduce two runtime systems that individually each address one style of analytics and together form a functionally complete infrastructure. The first system is a stream-processing engine, which performs real-time computation over data streams, and the second system is a Map-Reduce engine [8], which performs batch computation over stored *historical* data (i.e., data-at-rest). Additionally a visualization system retrieves the results from the stream-processing and the Map-Reduce engines and presents them graphically to an analyst. We illustrate in Figure 4 the main components of the cybersecurity analytics platform architecture. Not depicted is an event routing layer, which ensures that the appropriate events are connected to the correct analytic components.

4.2.1 Stream Computing for Real-Time Analytics

A stream-processing engine is a system designed to feed continuous, high-speed data streams through complex data-processing pipelines. Its design makes it particularly suitable for real-time analysis of network traffic and host events. A data stream corresponds naturally to events produced by data sources in our platform. The pipelines in a stream-processing engine are formed by connecting stream operators, which are grouped into processing elements when running on the same machine. Processing elements serve as the functional containers both for our data sources and for our analytic components and are distributed across many machines. Finally, the property-based routing of a stream-processing engine allows the association of properties with streams and the automatic connection of streams to processing elements based on requested properties.

In the cybersecurity analytics platform, the interaction between data streams, processing elements, and property-based routing is straightforward. Initially, event streams are instantiated for raw network traffic, e.g., each monitoring point produces one event stream. Processing elements filter, transform, or otherwise modify these raw event streams to derive event streams with higher information content. Each of these

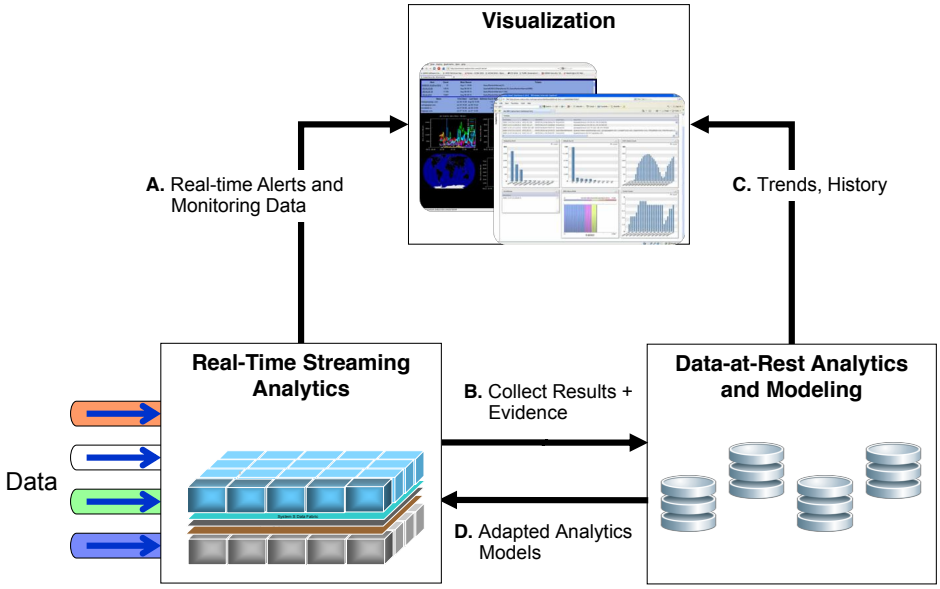

Figure 4: The high-level architecture of the cybersecurity analytics platform.

derived streams can be tagged with one or more properties; processing elements can then use these tags to locate event streams of interest.

4.2.2 Map-Reduce for Data-At-Rest Analytics

The data-at-rest analytics component is work in progress and will complete the overall architecture. Real-time, low-latency analysis of network traffic often requires the complementary strength of long-term or historical analysis. Due to hardware resource limits it is not feasible, for example, to cache in memory and analyze one year's worth of HTTP traffic. The offline cybersecurity-analysis component is being designed and implemented to process large amounts of stored network data and derive network behavior models that can then be operated in real time by the stream-processing engine of the platform. The stored data is accumulated output from streaming analytic components of the cybersecurity analytics platform, thus forming a feedback loop between the streaming analytics that analyze instantaneous data and summarize it for offline use, and the offline analytics that extract trends from historical data and that produce behavioral models to inform the streaming analytics.

4.3 Discussion

We now revisit the requirements set out in Section 2 and discuss how our cybersecurity analytics architecture fulfills them.

Multi-channel data and context are supported by the capability to connect various data sources dynamically and to merge related event streams before they are input into an analytic component. Thus changes to the physical network topology are addressed in real time in the cybersecurity engine.

The **event correlation and history** requirements are realized by the property-based routing layer and by the data-at-rest environment underlying the Map-Reduce system. The property-based stream routing layer acts as a "glue" between analytics, which as a result can easily combine and correlate data to derive higher-level information.

Scalability and timeliness are respectively achieved by distributing the analytics and by employing the event-driven stream-processing engine. Analytics are distributed both in the stream-processing system and in the Map-Reduce system, thus ensuring scalability across the real-time volume of network data sources

and the massive volume of data-at-rest. The facts that the stream-processing engine is event-driven, and that it can handle high volumes of streaming data, both guarantee timeliness of identifying suspicious activities and generating alerts.

Extensibility arises from the ease with which an analyst can connect fine-grained analytics together and can add analytics that reuse output event streams. The reuse of analytics reduces the overhead at deployment time and increases operational flexibility (i.e., adding, removing, and updating of analytics are done at the level of individual analytic components/processing elements), all while the cybersecurity analytics platform is running.

5 Implementation

We chose IBM InfoSphere Streams [18] as the underlying stream-processing platform. InfoSphere Streams is a high-performance software platform for developing applications for real-time analysis and processing of structured and unstructured data. The strength of InfoSphere Streams stems both from its ability to create distributed and scalable applications and from its well-suited domain-specific Streams Processing Language (SPL) [14].

As a general platform, InfoSphere Streams has been applied across a wide variety of industries including government, telecommunications, financial markets, health care, energy and utilities, and manufacturing. Measurements on quad-core 3.0 GHz machines with 256-byte messages and an application with simple analytics have shown that InfoSphere Streams achieves throughput rates of over 400 000 messages per second on a single machine, and over 1.3 million messages per second on six machines with an average latency lower than 120μs on Ethernet networks [17].

5.1 Platform Extensions

We extended the InfoSphere Streams platform to make it fit our needs, using the APIs provided by the platform. First, we created new source operators to process raw network traffic and convert it directly into Streams tuples (cf. Section 5.2). This avoids the cost of an extra data conversion step. Second, we created new operators to expose dynamic property-based streams bindings, which were not exposed in the streams programming language. By exposing these bindings we are able to allow operators to configure their input streams based on their location, so that they can consume data streams local to them. Third, we enhanced the operator placement functionality to provide more flexible run-time control. This extension allows us to ensure that operators are placed close to the data sources and the data they are processing, and reduces communication overhead between physical nodes.

5.2 Data-Collection Plane

A common issue with analyzing data from different sources is the need to bring the data into a normalized format. In some cases, we use custom operators, implemented in C++, which extend existing InfoSphere Streams operators. The custom operators decode the raw input events and convert them into tuples. In other cases, we leverage Streams built-in input methods (such as CSV input) and provide external adapters for further processing units.

In the following, we describe the data-channel operators we implemented in our system:

Name Service (DNS) records present a valuable source of information about network activity. DNS messages are ingested into the system using the port mirroring feature of network switches. All network traffic to and from DNS servers is mirrored to a network tap system. The packets are captured on the tap using the packet capturing (PCAP) facility. The PCAP-encapsulated DNS messages are fed into the streams platform, where IP address and domain name mappings are extracted and converted into streams tuples.

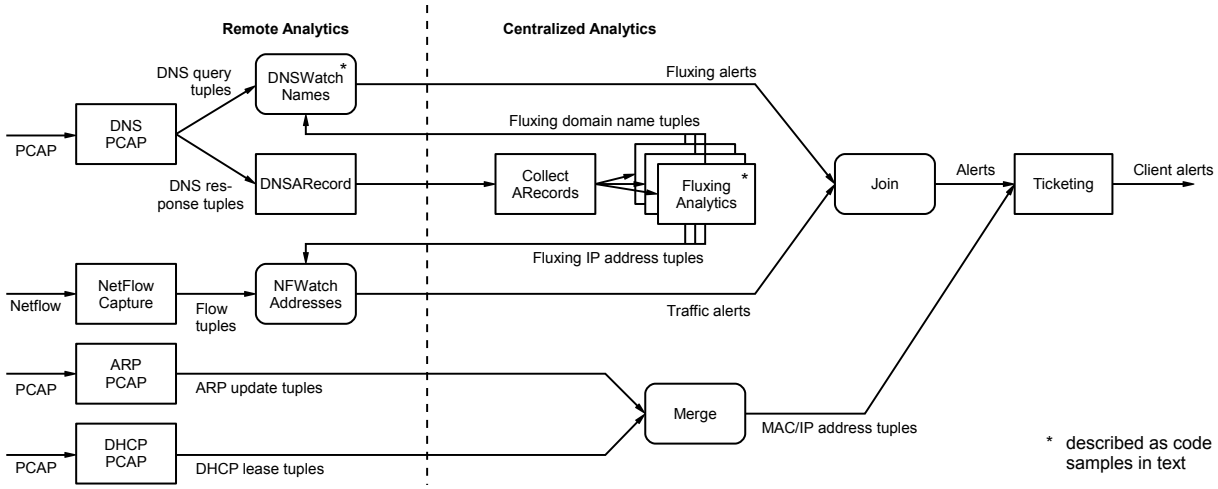

Figure 5: Stream-processing operators in DNS analytics and alerting.

Address Resolution Protocol (ARP) messages provide a mapping between MAC addresses and IP addresses. ARP messages are captured using PCAP and are fed directly into a custom stream operator for conversion into tuples. We use ARP queries instead of responses; queries are broadcast to everyone and contain the MAC to IP mapping of the requesting system.

Dynamic Host Configuration Protocol (DHCP) messages, similar to ARP messages, provide a mapping between MAC addresses and IP addresses. We use DHCP messages to complement coverage of the ARP-based mappings of hosts with dynamically assigned IP addresses. The ingest of DHCP is handled in a manner similar as DNS, using port mirroring and PCAP at the DHCP servers.

Netflow/IPFIX records provide important summary properties of network traffic flows and are consumed by multiple methods in our implementation. A modified NFdump collector [1] is used for traditional Netflow records. Instead of archiving the records, the records are passed into the streams system, where they are converted to tuples. Syslog-based collection of Netflow records is also supported. In addition, a custom flow collector is used for Netflow and IPFIX messages to provide full flexibility. The flow records from this collector are passed into streams, where they are again converted into streams tuples.

Hypertext Transfer Protocol (HTTP) data is collected by mirroring the traffic from the desired network link and reassembling the TCP streams. For these streams, a custom stream operator filters and extracts the relevant HTTP header fields of all web requests and responses, including the URL, content types, user agent and server version strings, and cookies.

In addition to the channel operators described above, we are experimenting with operators ingesting non-network centric channels such as syslog and user authentication logs to our platform.

5.3 Analytics Plane

One of the design goals for the platform is to avoid sending large amounts of data to the central location. This is realized by filtering and aggregating near the data source taps, and by co-locating analytics which do not require a global view near those data source taps. An example of such reduction is for detecting fluxing domain names. One of the core inputs to any such analytic is the set of name to IP address mappings. At the DNS data input, the vast majority of these inputs are duplicates due to names that are looked up frequently

(for example, www.google.com). The reduction step at the DNS sensor removes these duplicates within a time window, which significantly reduces the amount of data that is sent to the central fluxing analytic.

Another design goal is ease of adding new functionality. This is achieved by making heavy use of run-time import and export of data streams. Our operators are small, typically processing one or two imported streams and exporting one or more result streams. This allows us to replace or augment functionality in the running system by replacing or adding operators incrementally.

The import and export of data streams also allows us to have the same operator instantiated multiple times. Through an architected schema for stream properties, we can create processing clusters which import and export certain streams from the other operators in the same cluster. For example, we use this to create a cluster of operators on a DNS tap to perform all local processing on that system. All operators in that DNS cluster can be submitted on to a different DNS tap and will cluster together on that tap. Any globally-exported streams from these clusters are available to the central analytics.

5.4 Example: Fluxing Domain Analytics

The purpose of our fluxing domain analytics is to detect queries for domain names that appear to be mechanically-generated, or that appear to have a large and continuously-evolving IP address footprint. Both techniques are popular in the deployment of botnet C&C infrastructure.

Figure 5 shows a graphical view of the stream-processing graph. The vertical dashed line separates the analytics on the left, which are running remotely near the data sources, from the centralized analytics on the right.

DNS messages flow into the `DNS PCAP` operator, which separates the queries and responses into different output streams. The response messages are streamed to the `DNSARecord` operator. This operator is responsible for eliminating duplicate name-to-address mappings within a given time window (we use a 24-hour window). The name-to-address mappings flow to the central `CollectARecords` operator which is responsible for collecting name-to-address mappings from all network taps. Note that we only show one DNS tap here, but typically there would be multiple. The collected mappings are then annotated with additional features (such as ASN, country code, etc.) and are then sent to the various fluxing analytics. When an analytic flags a name as fluxing, it writes the name to one stream and the set of associated IP addresses to another.

The names flow into the `DNSWatchNames` operator which is receiving the DNS query messages. This operator runs at the data source. When a client queries a name that has been flagged, the information is sent to the central ticketing system. Likewise, the IP addresses flow into a `NFWatchAddresses` operator which matches NetFlow records against the flagged addresses. Flows matching flagged addresses result in a message to the ticketing system. The DHCP and ARP operators provide the ticketing system with IP address-to-MAC address mappings which identify the physical host and its owner. The ticketing system uses these to coalesce tickets based on the MAC address.

The following example illustrates the `DNSWatchNames` operator (cf. Figure 5) which imports two streams from other operators. Note that the code samples leave out the boilerplate code and are simplified for brevity's sake. The first is a stream of DNS names which the system is interested in, i.e., names which have been flagged as fluxing. The second imported stream contains all the queries. Note that the first import is global, while the second is local to only the queries on the local tap. Localization is not a feature of streams, but is implemented through designed usage of imports and runtime operator placement.

```
# DNSWatchNames operator
import stream MonitoredNamesIn(schemaFor(MonitoredNames))
    tapping ‘‘stream[class=’MonitoredNames’]’’
import stream QueriedNamesIn(schemaFor(QueriedNames))
    tapping ‘‘stream[class=’QueriedName’ & tap=’thisTapID’]’’

export properties [class:‘‘QueryMatchedNames’’]
stream CheckNames(schemaFor(QueryMatchedNames)) :=
    Match(QueriedNamesIn; MonitoredNamesIn) [ JoinKey1: RRName; JoinKey2: domainName ] {}
```

The output of this operator can be imported by another operator simply by tapping `stream[class='QueryMatchedNames']`. Connections between operators are managed by the Streams platform, and the order of operator submissions does not matter. To simplify the coding, as well as provide some compile-time checking, we leverage the M4 macro utility [21]. The example above simplifies to the code shown next.

```
# DNSWatchNames operator, simplified
Import_MonitoredNames_As(MonitoredNamesIn)
Configure_Import_QueriedNames_As(QueriedNamesIn)

Export_QueryMatchedNames_From(CheckNames) :=
     Match(QueriedNamesIn; MonitoredNamesIn) [ JoinKey1: RRName; JoinKey2: domainName ] {}
```

As a more concrete example, the following shows a simple ASN-based detector for fluxing names (cf. Figure 5). Any domain with IP addresses in more than five ASNs will be emitted to an exported stream of monitored names, which in turn is imported in the code sample above.

```
# Fluxing analytics operator
Import_DNSARecords_As(ARecordsIn)

stream ARecordAddASN(domainName: String, address: String, ASN: Integer) :=
    Functor(ARecordsIn)[] { ASN := getASN(address) }

stream ARecordCounts(domainName: String, addresses: StringList,
                     uniqueASNs: IntegerList, ASNcount: Integer) :=
    Collect(ARecordAddASN)
    [GroupBy: domainName;Emit:60;Expire:604800]
    { domainName := ~OneOf(domainName),
      addresses := ~Unique(address),
      uniqueASNs := ~Unique(ASN),
      ASNcount := ~UniqCount(ASN)}

Export_MonitoredNames_From(FluxingName)
Export_MonitoredAddresses_From(FluxAddr) :=
     Functor(ARecordCounts) [ ASNcount > 5 ] {}
```

We can easily expand the system by creating new operators that tap into existing streams. Note that the previous code sample is actually exporting two data streams from the final operator (i.e., `ARecordAddASN` and `ARecordCounts`). One stream has the names, while the other has the addresses. We can create a new analytic that reports names that have matching addresses other than the reported name:

```
# Sample new operator
Import_MonitoredAddresses_As(MonitoredAddrIn)
Import_DNSARecords_As(ARecordsIn)

stream CandidateMatches(domainName: String, address: String, flaggedName: String) :=
     Match(ARecordsIn; MonitoredAddrIn)
     [ JoinKey1: domainName; JoinKey2: domainName ]
     { domainName := $1.domainName, flaggedName := $2.domainName }

Export_OverlapAddress_From(CandidateMatches) :=
      Functor(CandidateMatches) [ domainName != flaggedName ] {}
```

When we submit this operator into the system, the streams infrastructure will automatically connect it to the appropriate streams of data, and begin processing.

5.5 User Interface

We implemented a web-based user interface that runs on top of the analytics plane, and that enables analysts and administrators to interact with the cybersecurity platform. The front-end user interface, implemented in JavaScript, connects to a back-end asynchronous communication infrastructure that passes requests to the centralized analytics and the remote analytics.

In the current version of the platform, we implemented a number of interactive modules for event monitoring, context-sensitive drill-down on DNS and network flow data, geographic mapping of communication peers, and ticket handling. Moreover, an additional set of modules enable administrators to monitor the platform performance in detail, including the performance (e.g., throughput, tuple rates) of individual data streams as well as the performance of the physical machines in the distributed setup.

5.6 Discussion

Our platform is able to scale to a large number of data sources, while correlating over multiple different types of data. By performing discovery on live streams of data, our platform minimizes the delay from the time that an issue is discovered (such as a malicious fluxing domain name) to the time that additional monitoring is turned on for clients affected by that issue.

As the examples illustrate, we prefer to implement the analytics with a large number of small operators, instead of using one large operator to implement everything related to, say, DNS. We find that small, single-function, operators take better advantage of the dynamic and extensible aspects of the system. New functionality can be added incrementally to the running system simply by submitting operators that leverage output streams of existing operators. Improved analytics can be leveraged by starting the operator with the enhancements and stopping the old one, without impacting the rest of the system.

6 Performance Evaluation

In this section, we evaluate the performance of the cybersecurity engine described in Section 5. We describe the data sets and testbed first, then the results of a performance evaluation.

6.1 Data Set and Evaluation Testbed

The evaluation testbed includes a medium-sized corporate production network comprising about twelve thousand nodes, including server and end-user work stations, laptops, and mobile devices. In our current deployment the central analytics are hosted on two blade servers (each with two 2.5 GHz dual-core CPUs and 40 GB main memory) that are connected over an Ethernet backplane. The remote analytics run on server systems (each with two 2 GHz quad-core CPUs and 24 GB main memory) and are deployed in proximity to the data sources.

In Table 1 we list the input rates that the taps deployed in our testbed observe on average per day. We distinguish the byte rate and the event rate of the incoming data stream. In the DNS streams we observe an average of 340 000 unique fully qualified names every day with an average of about 110 000 unique primary-level domains. Our current deployment captures network flows originating from the border routers of the network, observing all traffic that enters and leaves the network. The testbed has been continuously expanded over the last three years to incorporate new data sources.

6.2 Performance Results

We evaluated the performance of the implementation in two ways in our testbed. To understand the performance limits of the analytics, we injected events from an archive collected over the last years and measured the throughput of the streams along the operators. Separate from this stress test, we constantly measured the performance of the system under the real load observed in our testbed. In this section, we present the results of both evaluations.

Tap	Ingress Rate (in MB)	Ingress Event Rate (in thousands)
DNS tap 1	19,300 (σ = 4,755)	122,648 (σ = 30,221)
DNS tap 2	349 (σ = 123)	1,860 (σ = 652)
Flow tap 1	1,468 (σ = 657)	12,123 (σ = 5,424)
Flow tap 2	1,788 (σ = 927)	14,764 (σ = 7,649)
DHCP tap 1	31 (σ = 22)	93 (σ = 65)
DHCP tap 2	54 (σ = 38)	163 (σ = 114)
HTTP tap 1	3,649 (σ = 1,896)	6,473 (σ = 3,363)
ARP tap 1	5 (σ = 1)	87 (σ = 18)
ARP tap 2	31 (σ = 12)	557 (σ = 219)

Table 1: Input data rates (average and standard deviation per day) for the remote analytics in our testbed. Tap 1 captures Intranet traffic, tap 2 captures DMZ traffic.

Aggregated Data Stream	Egress Rate (in MB)	Egress Event Rate (in thousands)
DNS (tap 1+tap 2)	84 (σ = 19)	1,020 (σ = 220)
Flow (tap 1+tap 2)	367 (σ = 193)	3,369 (σ = 1,969)
DHCP+ARP (tap 1+tap 2)	150 (σ = 41)	1,684 (σ = 465)

Table 2: Aggregated data rates (per day) between remote analytics and centralized analytics. "Egress" refers to event streams from taps to centralized analytics.

DNS analytics stress test. A stream of raw DNS messages (request and response) were sent over the network to a single blade. The tap read the messages from the wire and injected them into the platform. The operators deployed in the evaluation included all analytical steps described in Section 5.4. We measured peak throughput rates as high as 50 000 to 70 000 DNS messages a second on a single blade. To process more DNS traffic, additional blades may be operated in parallel as performance scales linearly due to the inherent parallelism of the stream-computing programming model. In the production network, we did not observe message rates exceeding 6 550 DNS messages per second.

Filtering and data reduction. By filtering data near the source, our remote analytics significantly reduced the amount of traffic which was routed into the central analytics (cf. Section 5.3). In Table 2, we report the event and byte rates originating from the remote analytics measured in our testbed. Monitoring of the network links at the central analytics during peak hours indicates an inbound data rate of 96 kB/s and an outbound data rate (back to remote analytics) of 23 kB/s. These rates include the actual tuple data, administrative messages for the middleware, as well as any encapsulation overhead. During the same time period, the remote analytics were processing an aggregate 34 MB/s of incoming data.

While the data reduction is significant, one could envision that a large number of data sources would still result in a significant amount of traffic destined for the central analytics. In such a scenario, multiple "localized central" analytics could be deployed that each handle a subset of the data source outputs, reducing that data before forwarding it to the central analytics.

CPU and memory utilization. We monitored CPU and memory utilization at the central analytics site. With the data rates as shown in Table 2, the aggregate memory used by the central analytics amounted to 1.1 GB. About 50 % of this memory was used by the DNS A-record aggregator `CollectARecords`, while the rest was evenly distributed across other jobs. The CPU used by the central analytics averaged 1 520 seconds of CPU per day. Again, the DNS A-record aggregator was responsible for half of the total whereas the rest was spread across the other jobs.

7 Lessons Learned

In this section, we summarize the most relevant lessons we learned while designing, implementing, and deploying our cybersecurity analytics platform on top of the stream-processing engine.

1. **Stream processing suits cybersecurity analytics.**
   The natural flow of events from network and system activity is modeled as streams of tuples that travel through the analytics graph. Stream processing supports easy incremental deployment of new analytics that tap into existing streams. This frees the analyst from having to worry about how to get data and allows him to focus on the detection logic.

   The native concurrency in stream processing leads to a highly scalable analytics solution. Furthermore, the distribution of analytics makes up for the often long distances between data sources (taps) and centralized global analytics, allowing us to filter and aggregate data on or close to the taps. The self-managing middleware abstracts from physical distribution of data sources and processing nodes, further simplifying the task of the analyst.

2. **Window size matters in real-time analytics.**
   The amount of history kept across the processing of events (often expressed as the size of the "time window") has significant performance implications. High-speed streaming analytics require a large amount of state to achieve their optimal detection rate. This state information must be kept in memory to maintain adequate performance levels. A large amount of memory may be required when multiple analytics share, due to data locality, the same physical system.

3. **Real-time analysis is necessary, yet not sufficient.**
   Irrespective of how much data real-time analytics can keep with their state, an advanced attacker might be able to avoid detection by being patient enough to deploy their attack over many months or even years. It is thus necessary to complement real-time analytics based on stream processing with offline analytics based on massively scalable batch processing (e.g., Map-Reduce). Offline analytics can then correlate data from many different time frames and build models of expected, benign behavior and of anomalous, malicious behavior. These models in turn can be deployed as real-time analytics, thus completing the feedback loop between real-time and offline.

4. **Distributed processing needs security hardening.**
   Distributed processing for cybersecurity applications, whether based on stream processing or on Map-Reduce, requires management beyond what is normally provided by the middleware. Monitoring and management of CPU, memory, and disk utilization, restarting down systems, etc., are needed in order to keep the system functioning as a whole. Furthermore, as the middleware is now part of the trusted computing base of the cybersecurity analytics platform, it has to be enhanced to withstand and recover from direct (denial-of-service) and algorithmic attacks.

8 Future Directions

We are currently extending the described streaming analytics with massive-scale data-at-rest analytics for three reasons. First, we wish to overcome the limitations of real-time processing in detecting attacks that stretch their behavior over extensive time windows. Second, data-at-rest will enable the application of automatic forensic capabilities to attacks long after they occur, and the estimation of the impact, progression, and motives of attackers. Third, we aim to extract dynamically evolving malicious-behavior models based on data mining and machine learning, models which can be automatically translated into streaming analytics to improve coverage of future detection. Another focus is on integrating privacy-aware algorithms that preserve the privacy of users with minimal impact on detection accuracy.

9 Conclusions

We have presented an extensible, high-performance analytics platform based on a general-purpose real-time streams processing engine. The first benefit of our enhanced engine is a separation of analytics development from deployment. This frees analysts from concerns about physical network topologies, and allows administrators to deploy analytics without concerns about stream-processing connection topologies. Second, our unified network view is a major improvement over conventional distributed IDSes that separate local alert generation from global event correlation. Other major benefits of general-purpose stream processing are incremental analytics deployment and reusability.

We have shown that the platform is scalable and that it can correlate vast amounts of information across multiple channels in real time. We evaluated our approach in the context of botnet detection, and showed that a basic streams processing system can easily implement the complexity and performance levels of existing systems that are based on static analytics. We are confident that the data-at-rest component will provide the power that is needed for the next step, adaptive analytics that capture the ever-changing nature of cyber threats.

10 Acknowledgments

We wish to thank Vugranam C. Sreedhar, Mitchell A. Cohen, Tracy Kimbrel, and the Exploratory Streams Processing Team at IBM Research for their insights, guidance, and support during the development of the present work.

References

[1] NFdump. http://nfdump.sourceforge.net/. Accessed: June 8, 2011.

[2] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for DNS. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[3] J. Aussibal and L. Gallon. A new distributed IDS based on CVSS framework. In *Proceedings of the 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems (SITIS'08)*, pages 701–707, 2008.

[4] W. Baker, M. Goudie, A. Hutton, C. D. Hylender, and J. Niemantsverdriet. 2010 data breach investigations report – a study conducted by the Verizon Business RISK team in cooperation with the US Secret Service. http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf, 2010.

[5] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. H. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th Annual Computer Security Applications Conference*, 1998.

[6] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proceedings of the 18th Network and Distributed System Security Symposium (NDSS '11)*, 2011.

[7] M. Crosbie and G. Spafford. Defending a computer system using autonomous agents. Technical report, Purdue University, 1995.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Comm. ACM*, 51:107–113, Jan. 2008.

[9] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet dossier (Symantec Security Response). http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, Nov. 2010. Accessed: June 3, 2011.

[10] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *Proceedings of the 1st Wksp. on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[11] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, pages 139–154, 2008.

[12] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[13] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS'08)*, 2008.

[14] M. Hirzel, H. Andrade, B. Gedik, V. Kumar, G. Losa, M. Mendell, H. Nasgaard, R. Soulé, and K.-L. Wu. SPL stream processing language specification. Technical report, IBM Research, Nov. 2009.

[15] T. Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS'08)*, 2008.

[16] HP Corp. ArcSight. Product webpage. http://www.arcsight.com/. Accessed: June 7, 2011.

[17] IBM Corp. IBM InfoSphere Streams V1.2.1 offers new Resource Value Unit Pricing and offers a Developer Edition. http://www-01.ibm.com/common/ssi/rep_ca/8/897/ENUS210-428/ENUS210-428.PDF. Accessed: June 13, 2011.

[18] IBM Corp. InfoSphere Streams. Product webpage. http://ibm.com/software/data/infosphere/streams/. Accessed: May 20, 2011.

[19] M. Iliofotou, H.-C. Kim, M. Faloutsos, M. Mitzenmacher, P. Pappu, and G. Varghese. Graph-based P2P traffic classification at the internet backbone. In *Proceedings of the 28th IEEE international conference on Computer Communications (INFOCOM'09)*, pages 37–42, 2009.

[20] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the 1st Wksp. on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[21] B. W. Kernighan and D. M. Ritchie. The M4 macro processor. Technical report, Bell Laboratories, 1977.

[22] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. BotGrep: Finding P2P bots with structured graph analysis. In *Proceedings of the 19th USENIX Security Symposium*, pages 95–110, 2010.

[23] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (Malware'08)*, 2008.

[24] V. Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, 1999.

[25] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration (LISA '99)*, pages 229–238, 1999.

[26] RSA Corp. NetWitness. Product webpage. http://www.netwitness.com/. Accessed: June 7, 2011.

[27] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C. lin Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, 1991.

[28] E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, 2000.

[29] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, pages 361–370, 1996.

[30] G. Vigna and R. A. Kemmerer. NetSTAT: A network-based intrusion detection approach. In *Proceedings of the 14th Annual Computer Security Applications Conference*, 1998.

[31] G. White, E. Fisch, and U. Pooch. Cooperating security managers: a peer-based intrusion detection system. *Network, IEEE*, 10(1):20 –23, jan/feb 1996.

[32] J. Yang, P. Ning, X. S. Wang, and S. Jajodia. CARDS: A distributed system for detecting coordinated attacks. In *Proceedings of the IFIP TC11 15th Working Conference on Information Security for Global Information Infrastructures*, 2000.

[33] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the DOMINO overlay system. In *Proceedings of the 11th Network and Distributed System Security Symposium*, 2004.

[34] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA'08)*, 2008.