An abstraction layer for cybersecurity context

R. Bolla
DITEN – University of Genoa
Genoa, Italy Email: raffaele.bolla@unige.it

A. Carrega, M. Repetto
S3ITI Lab, CNIT
Genoa, Italy
Email: {alessandro.carrega,matteo.repetto}@cnit.it

***Abstract*—The growing complexity and diversification of cyber-attacks are largely reflected in the increasing sophistication of security appliances, which are often too cumbersome to be run in virtual services and IoT devices. Hence, the design of cyber-security frameworks is today looking at more cooperative models, which collect security-related data from a large set of heterogeneous sources for centralized analysis and correlation.**

In this paper, we outline a flexible abstraction layer for access to security context. It is conceived to program and gather data from lightweight inspection and enforcement hooks deployed in cloud applications and IoT devices. We also provide a preliminary description of its implementation, by reviewing the main software components and their role.

I. Introduction

Virtualization and the cloud paradigm usually bring agility and cost-effectiveness in building and operating ICT services, but they pose a number of additional security concerns, when compared to current legacy deployments [1], [2].

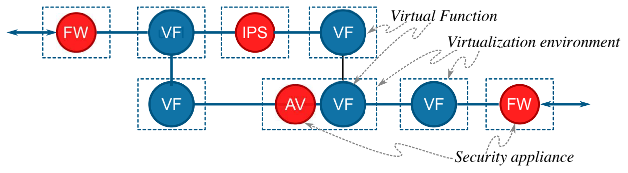
Motivated by the substantial limitations of security mechanisms in the virtualization infrastructure (i.e., distributed firewalling, micro-segmentation, and security groups [3]–[5]), the difficulty to coordinate them in cross-cloud deployments, and the typical diffidence in trusting security services provided by third parties, there is an increasing trend to insert legacy security appliances in the topology of virtual services (see Fig. 1a). However, this approach has several drawbacks. First, each appliance has its own inspection hooks, and this may result in unnecessary duplication of operation (i.e., the same packet may be processed by different appliances for retrieving very similar information). Second, given the ever-growing number and complexity of protocols and applications, detection is a cumbersome task and requires large amount of computing resources, which may significantly increase the cost of running the graph. Third, it is difficult to balance the need for pervasive protection (i.e., many cybersecurity appliances) with performance of the whole graph (e.g., in case of NFV packet processing should happen at wire speed). Last, but not least, complex security appliances are not immune to bugs and vulnerabilities, which eventually increase the overall attack surface of the deployed service.

Building situational awareness for virtual services requires new architectural paradigms, able to overcome the above limitations by combining fine-grained and precise information with efficient processing, elasticity with robustness, autonomy with interactivity [2]. In this respect, the transition from standalone security appliances to more cooperative models has already proven to improve the detection rate while decreasing the overhead on each terminal [6]. For cooperative model, we mean a centralized architecture where security information, data, and events are collected from multiple sources within a given domain for common analysis and correlation. This is a common trend today for all major vendors of cyber-security applications, which are increasingly developing Security Events and Information Management and Security Analytics software for the enterprise, leveraging machine learning and other artificial intelligence techniques for data correlation and identification of attacks. They are usually designed as integration tools of existing security applications and require to run heavyweight processes on each host; hence, they are not suitable for virtual services.

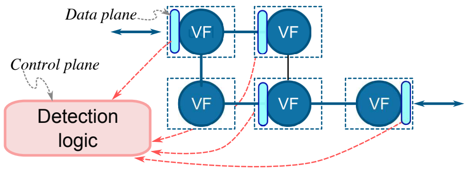
Recently, we have outlined the general architecture of a novel framework for AddreSsing ThReats for virtualIzeD services (ASTRID) [7]. The underlying concept is the decoupling of inspection tasks (to be integrated into the different forms of virtualization boxes, as Virtual Machines or containers) from a (logically) centralized ans shared detection logic (to be kept outside the graph), as schematically shown in Fig 1b. In this paper, we describe our on-going work about the definition of an abstraction layer to provide the detection logic with uniform and "bi-directional" access to heterogeneous security context of virtualized services. The novelty of our work consists in abstracting lightweight *programmable* hooks in the kernel or system libraries, without the need to deploy complex and cumbersome security appliances inside VMs or as separate components in the overall service graph. The ability to program both the collection of security context and the configuration of enforcement rules (which is the mean for "bi-directional" access) is just a major improvement over the number of log collection tools already available as commercial or open-source implementations.

The rest of the paper is organized as follows. We describe the overall ASTRID architecture in Section II. We then elaborate on the concept of abstraction layer and its architectural design in Section III, while we discuss relevant technologies and the current implementation stage in Section IV. We briefly report related work in Section V. Finally, we give our conclusion in Section VI.

The final publication is available at IEEE Xplore via https://doi.org/10.1109/ICCNC.2019.8685665.



(a) Security appliances in service graph



(b) Decoupling inspection and detection

Fig. 1: The difference between the current approach (left side) and the proposed paradigm (right side).

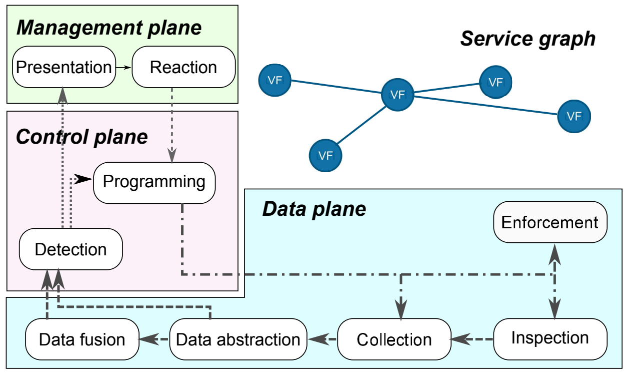


Fig. 2: Multi-layer security architecture

II. The ASTRID architecture

The ASTRID multi-layer architecture is organized in three complementary planes, as shown in Fig. 2. Due to functional similarities (especially with architectures for software-defined networking), we used a typical networking terminology, though our architecture is not directly tied to network operations. ASTRID is a multi-tier architecture, where a common, programmable, and pervasive *data plane* feeds a powerful set of multi-vendor detection and analysis algorithms (*business logic*). On the one hand, the challenge is to assemble a wide knowledge over multiple sites by real-time collection of massive events from a multiplicity of capillary sources, while maintaining essential properties such as forwarding speed, scalability, autonomy, usability, fault tolerance, resistance to compromises, and responsiveness. On the other hand, the ambition is to support better and more reliable situational awareness by inter- and intra-domain data correlation in both space and time, in order to timely detect and respond even the more sophisticated multi-vector and interdisciplinary cyber-attacks.

The *data plane* is the only part of the architecture that is deployed in the virtualization environment. It collects the security *context*, i.e., a knowledge base including events (failed login attempts, denied access, system calls), logs (service requests, operations, anomalies, client identity, execution traces, memory dumps), measures (network metrics, usage profiles) that can be useful for detection of known attacks or identification of new threats.

One of the main advantages of a common control plane is the availability of data from different subsystems (disk, network, memory, I/O), instead of relying on a single source of information as is the common practice nowadays. Since the collection of data from multiple sources may easily result in excessive network overhead, it is important to shape the inspection, monitoring, and collection processes to the actual need. The data plane must therefore support re-configuration of individual components and programming of their virtualization environments, to change the reporting behavior, including parameters that are characteristics of each app (logs, events), network traffic, system calls (e.g., disk read/write, memory allocation/deallocation), RPC toward remote applications (e.g., remote DB). Programming also include the capability to offload lightweight aggregation and processing tasks to each virtual environment, hence reducing bandwidth requirements and latency.

The data plane is responsible for enforcing security policies, including packet filtering, access control, and re-configuration of the execution environment. A fundamental property for the data plane is programmability, that is the capability to shape the deep of inspection according to the current need, in both spacial and temporal dimensions, so to effectively balance granularity of information with overhead.

The *control plane* is a (logically) centralized collections of algorithms for detection of attacks and identification of new threats. Every algorithm retrieves the data it needs (e.g., number of packets intended to given port on specific host, number of login failures, username used for failed authentication, etc.) from the common data plane. This represents one the main innovation behind the proposed framework: indeed, every algorithm has complete visibility on the overall system, removing the need to have local agents deployed in each virtual function, which often perform the same or very similar inspection operations. The control plane should also include programming capabilities, i.e., a framework to configure and offload local processing tasks to the data plane, so to effectively balance the depth of inspection with the generated overhead.

Beyond the mere (re-)implementation of legacy appliances for performance and efficiency matters, the ASTRID approach is specifically conceived to pave the road for a new generation of detection intelligence, arguably by combining detection methodologies (rules-based, machine learning) with big data techniques; the purpose is to locate vulnerabilities in the graph and its components, to identify possible threats, and to timely detect on-going attacks. In this respect, the application of machine learning and artificial intelligence would be useful to inspect and correlate the large amount of data, events, and measures that have to be analyzed for reliable detection and identification of even complex multi-vector attacks.

The *management plane* is conceived to keep humans in the loop. It notifies detected attacks and anomalies, allowing access to the full context in case the human expertise is needed to complement artificial intelligence in the inspection process. The management plane supports quick and effective remediation actions, by the definition of high-level policies that are then translated in specific data plane configurations from the control plane. The management plane also seamlessly integrates with orchestration tools, which are expected to be widely used for automating deployment and life-cycle operations of virtual services.

The main tasks at the management plane are the representation and usage of situational awareness built by underlying security applications. Specific challenges include data visualization (e.g., to pinpoint the actual position of attacks and threats in the network topology, to point out the possible correlation between events in different domains), and decision support (e.g., to suggest remediation and countermeasures, to define automatic response to well-known attacks). Also, the presentation layer should provide seamless integration with CERT networks to share information about new threats and attacks among different administrative domains (e.g., with STIX), in order to facilitate continuous update of the attack data base and the elaboration of common reaction and mitigation strategies [8]. Integration with existing risk assessment and management tools is also possible, so to automate most procedures that are still carried out manually.

III. An abstraction layer for the data plane

The main purpose for an abstraction layer is to provide uniform access to the underlying data plane capabilities. According to the general description in Section II, the data plane is made of heterogeneous inspection, measurements, and enforcement hooks, which are implemented in the virtualization environment.

These hooks include logging and event reporting developed by programmers into their software, as well as monitoring and inspection capabilities built in the kernel and system libraries that inspect network traffic and system calls. They are 'programmable' because they can be configured at runtime, hence shaping the system behavior according to the evolving context. This means that packet filters, types and frequency of event reporting, and verbosity of logging are selectively and locally adjusted to retrieve the exact amount of knowledge, without overwhelming the whole system with unnecessary information. The purpose is to get more details for critical or vulnerable components when anomalies are detected that may indicate an attack, or when a warning is issued by cyber-security teams about new threats and vulnerabilities just discovered. This approach allows lightweight operation with low overhead when the risk is low, even with parallel discovery and mitigation, while switching to deeper inspection and larger event correlation in case of anomalies and suspicious activities. This allows to scale with the system complexity, even for the largest services (e.g., carriers large scale virtual networks, and worldwide mass applications as social nets).

There are two main aspects to be covered by the abstraction layer:

- hiding the technological heterogeneity of the monitoring hooks;
- abstracting the whole service graph and the capabilities of each node.

Fig. 3 shows a schematic view of the envisioned abstraction. Locally, within each virtualization box, a Local Security Agent (LSA) provides a common interface to different hooks (e.g., different logging interfaces, different packet filtering/inspection tools). Then, the whole graph topology is abstracted as a hub-and-spokes graph. In this model, each node represents a virtual function and each link a communication path. Satellites of nodes are security properties; they include both monitoring/inspection capabilities (what can be collected, measured, and retrieved) and relative data (metrics, events, logs). Similarly, links have properties too (though not explicitly shown in the picture), related to the usage of encryption mechanisms and utilization metrics. This abstraction, effectively decouples the detection logic from the distributed data plane: a common language can be used to query security-related attributes and to re-program inspection and enforcement tasks, without the need to use different interfaces and heterogeneous semantics.

To provide composite metrics, data fusion is also envisioned as part of the overall abstraction framework. Pre-processing and aggregation of elementary data can be accomplished by the same query, hence optimizing look ups in the abstraction model. The abstraction layer also includes storage capabilities, so to provide both real-time and historical information for both on-line and off-line analyses.

IV. Implementation

We started the implementation of our concept by an in-depth analysis of design requirements and the selection of suitable technologies to implement the whole data plane.

Remote collection of logs is already a well established practice, with many frameworks available for this purpose: Scribe[1], Flume[2], Heka[3], Logstash[4], Chukwa[5], fluentd[6], nsq[7] and Kafka[8]. There are two ways to collect logs from applications: either forcing applications to directly write to these sources through specific APIs (as happens for Scribe, nsq and Kafka) or parsing their own log files (this option is available for Logstash, Heka, fluentd and Flume). The advantage of the first method is to reduce latency and improve reliability. A quite standard method for logging under Linux is syslogin this

[1]https://github.com/facebookarchive/scribe.

[2]https://flume.apache.org.

[3]http://hekad.readthedocs.io/en/v0.10.0.

[4]https://www.elastic.co/products/logstash.

[5]http://chukwa.apache.org.

[6]https://www.fluentd.org/.

[7]https://nsq.io.

[8]http://kafka.apache.org.

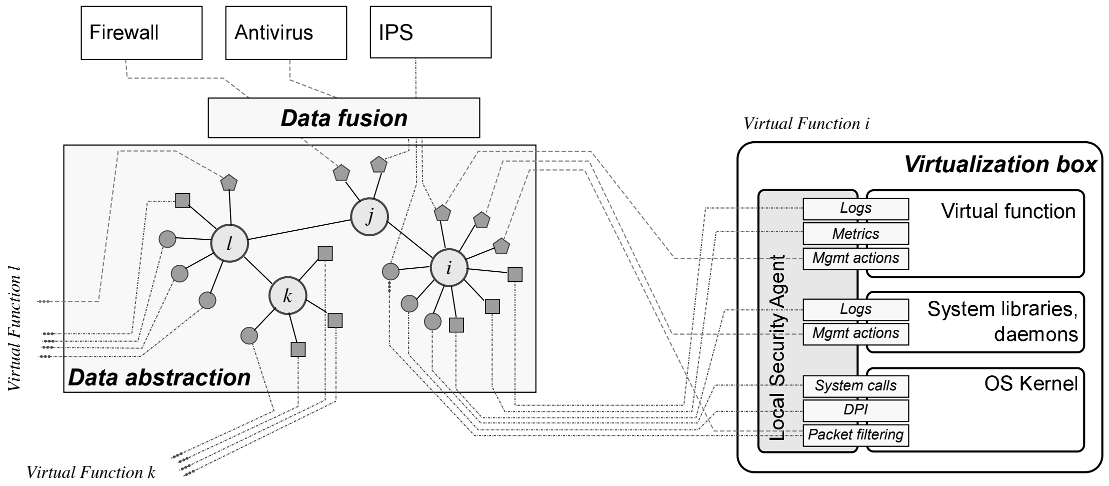


Fig. 3: The data plane collects and abstract the security context for the whole virtual graph.

case, remote collection would be possible through rsyslog[9] and Syslog-ng[10], but not all applications use syslog.

Network statistics are usually collected on flow-level basis through protocols as SNMP, NetFlow, sFlow, IPFIX, and, more recently, OpenFlow [9]. These are usually static and pre-defined kinds of data, so the interest in recent years has shifted towards stateful inspection. In this respect, we chose the extended Berkeley Packet Filter (eBPF) as the main packet inspection tool. eBPF enables arbitrary code to be dynamically injected and executed in the Linux kernel while at the same time providing hard safety guarantees in order to preserve the integrity of the system. While originally conceived to filter network packets only, it has now evolved to catch a broader set of kernel events; in general, any kernel event can be potentially intercepted (Kprobes, Uprobes, syscalls, tracepoints), making eBPF capable of analyzing message (socket-layer) received, data written to disk, page fault in memory, files in /etc folder being modified.

Beyond collection of data, the data plane must also tackle storage and querying. We considered the following technical issues in the selection of the most suitable storage technology: (1) the nature and composition of data (2) validity and volume of data, (3) query language and access interface.

The basic principle behind the abstraction layer is uniform access to heterogeneous security context, including data from network measurements, application logs, system calls, and other security events. This drives towards non relational (NoSQL) databases. The validity and volume of data affect the size of the database and the need for scalability. Local installations are suitable when the data are kept for days or months, but cloud storage services may be necessary for longer persistence or large systems.

On the other hand, cloud storage is not suitable for real-time or even batch analysis. Distributed storage systems such as HDFS[11], Cassandra[12], MongoDB[13], or ElasticSearch[14] allow to work with the raw data more effectively and can scale-out horizontally if data volume become large. In addition, they often offer inborn support for parallel processing and big data analytics (e.g., HDFS with Apache Hadoop[15] or Apache Spark[16]).

Based on the above considerations, graph databases as Neo4j[17], OrientDB[18], and ArangoDB[19] looks the best solution for building our abstraction layer. Indeed, unlike tabular databases like Cassandra, they support fast traversal and improve look up performance and data fusion capabilities. Our current implementation is based on Neo4j, which has a really great interface for searching, exploring and analyzing graph data in an intuitive way without requiring special domain knowledge.

The other fundamental component is the ability to perform quick look-ups and queries, also including some forms of data fusion. We adopted the data and manipulation language APIs called GraphQL[20]. It is an open source project, and a runtime for fulfilling queries with existing data. It provides a more efficient, powerful and flexible alternative to REST and ad-hoc web service architectures. It allows clients to define the structure of the data required, and exactly the same structure of the data is returned from the server, therefore preventing excessively large amounts of data from being returned.

V. Related work

The increasing complexity of cyber-attacks is urging the evolution of legacy security applications towards new central-

[9]https://www.rsyslog.com.
[10]https://www.syslog-ng.com/products/open-source-log-management.
[11]https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[12]http://cassandra.apache.org.
[13]https://www.mongodb.com.
[14]https://www.elastic.co.
[15]https://hadoop.apache.org/.
[16]https://spark.apache.org.
[17]https://neo4j.com/.
[18]https://orientdb.com.
[19]https://arangodb.com.
[20]https://graphql.org.

ized models, which enable deeper correlation and analytics than today [10]. As a matter of fact, the stages of advanced persistent threats and multi-vector attacks are quite easily mistaken to be independent events that occur in unrelated slots of time or network segments. Similar, zero-day attacks and outdated definitions are the main causes for the propagation and infection of malware. Typical recent architectures are made of four components:

1) monitoring probes (that may include log collectors, packet inspection, tracing of system calls),
2) local analysis,
3) data collection, and
4) centralized processing.

Most commercial tools for Security Information and Event Management (SIEM) and many research papers collect logs directly from local applications and legacy security appliances, focusing on centralized correlation to detect advanced persistent threats and other multi-vector attacks [11], [12]. N-version protection and big data techniques have been proposed to improve detection capabilities, though sometimes this may increase the rate of false positive [6], [13]. Evolving algorithms for anomaly detection are progressively introducing artificial intelligence (neural networks, fuzzy logic, support, vector machines, genetic algorithms, machine learning), but they are still largely tied to the concept of independent frameworks and appliances for different threats (e.g., firewalling, host-based intrusion detection, network-based intrusion detection) [14], [15]; indeed, the definition of distributed intrusion detection systems is mostly seen as a grid of independent appliances.

To improve efficiency, distributed firewalling and hypervisor-based security appliances are already available, but all these solutions introduce dependency and trust on the infrastructure provider (which is often an external entity). Further, the impact of hardware virtualization on the attack surface has already been largely recognized [1], [16].

To reduce the overhead on small devices and lightweight cloud services, all the detection logic may be logically centralized (even with redundancy for scalability and availability purposes), limiting local agents to inspection and collection tasks [6]. Indeed, cyber-security appliances are not immune to bugs and vulnerabilities, as witnessed by the continuous threat reporting to the NIST National Vulnerabilities Database. This means that beyond protection, the usage of cyber-security appliances increases the attack surface of virtual services.

Finally, we argue that, though in some cases the verbosity of the collected information can be dynamically tuned, there is currently a lack of more advanced programmability, which can offload specific inspection and detection tasks to local systems.

VI. Conclusion

In this paper, we have outlined the main features and the preliminary design of an abstraction layer that provide bi-directional access to an heterogeneous set of information and sources. This approach makes large data sets available for application of machine learning and other artificial intelligence mechanisms, which are currently the main research frontier for a new generation of threat detection algorithms. Differently from existing approaches, our target is to expose programmable features of the execution environment, which can be used to program local inspection and monitoring tasks.

Our next steps will be functional validation and extensive performance evaluation of a proof-of-concept implementation, including integration with local monitoring/enforcement agents (eBPF) and detection logic (based on OSSEC). Relevant aspects under investigation will include scalability, latency, and overhead, to demonstrate its applicability for both on-line and batch analysis and detection.

Acknowledgment

This work was supported in part by the European Commission, under Grant Agreement no. 786922.

References

[1] G. Pék, L. Buttyán, and B. Bencsáth, "A survey of security issues in hardware virtualization," *ACM Computing Surveys*, vol. 45, no. 3, pp. 40:2–40:34, June 2013.

[2] R. Rapuzzi and M. Repetto, "Building situational awareness for network threats in fog/edge computing: Emerging paradigms beyond the security perimeter model," *Future Generation Computer Systems*, vol. 85, pp. 235–249, August 2018.

[3] "Security groups – openstack networking guide," OpenStack documentation, August 2017. [Online]. Available: http://docs.ocselected.org/openstack-manuals/kilo/networking-guide/content/section_securitygroups.html

[4] "Security groups for your vpc," AWS documentation, August 2017. [Online]. Available: http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html

[5] "Vmware vcloud air," web site. [Online]. Available: http://vcloud.vmware.com

[6] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-version antivirus in the network cloud," in *Proceedings of the 17th conference on Security symposium (SS'08)*, San Jose, CA – USA, Jul. 28th–Aug. 1st, 2008, pp. 91–106.

[7] S. Covaci, R. Rapuzzi, M. Repetto, and F. Risso, "A new paradigm to address threats for virtualized services," in *In IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, Sep. 23rd–27th, 2018, pp. 689–694.

[8] F. Skopik, G. Settanni, and R. Fiedler, "A problem shared is a problem halved: A survey on the dimensions of collective cyber defense through security information sharing," *Elsevier Computers & Security Journal*, 2016.

[9] T. A. Limoncelli, "Openflow: A radical new idea in networking," *Commun. ACM*, vol. 55, no. 8, pp. 42–47, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2240236.2240254

[10] J. de Vries, H. Hoogstraaten, J. van den Berg, and S. Daskapan, "Systems for detecting advanced persistent threats: A development roadmap using intelligent data analysis," in *In International Conference on Cyber Security*, Washington, DC – USA, Dec.14th–16th, 2012, pp. 54–61.

[11] P. Bhatt, E. T. Yano, and P. Gustavsson, "Towards a framework to detect multi-stage advanced persistent threats attacks," in *In IEEE 8th International Symposium on Service Oriented System Engineering*, Oxford, UK, Apr.7th–11th, 2014, pp. 390–395.

[12] I. Friedberg, F. Skopik, and R. Fiedler, "Cyber situational awareness through network anomaly detection: state of the art and new approaches," *Elektrotechnik und Informationstechnik*, vol. 132, no. 2, pp. 101–105, March 2015.

[13] J. Therdphapiyanak and K. Piromsopa, "Applying hadoop for log analysis toward distributed ids," in *In Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC '13)*, Kota Kinabalu, Malaysia, Jan.17th–19th, 2013.

[14] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, January 2013.

[15] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, January 2013.

[16] R. Roman, J. Lopez, and M. Mambo, “Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges,” *Future Generation Computer Systems*, vol. 78, no. 2, pp. 680–698, January 2018.