2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

A Hierarchical Architecture for Distributed Security Control of Large Scale Systems

Yar Rouf, Mark Shtern, Marios Fokaefs and Marin Litoiu
Department of Electrical Engineering and Computer Science
York University
Toronto, ON, Canada
yarrouf@my.yorku.ca, mark@cse.yorku.ca, fokaefs@yorku.ca, mlitoiu@yorku.ca

***Abstract*—In the era of Big Data, software systems can be affected by its growing complexity, both with respect to functional and non-functional requirements. As more and more people use software applications over the web, the ability to recognize if some of this traffic is malicious or legitimate is a challenge. The traffic load of security controllers, as well as the complexity of security rules to detect attacks can grow to levels where current solutions may not suffice. In this work, we propose a hierarchical distributed architecture for security control in order to partition responsibility and workload among many security controllers. In addition, our architecture proposes a more simplified way of defining security rules to allow security to be enforced on an operational level, rather than a development level.**

***Keywords*-security control, cloud computing, ddos**

I. Introduction

As the Internet grows at a high rate and an increasing number of people and devices become interconnected over the web, large amounts of traffic are generated for software systems. From a security perspective, on one hand, it may not always be possible to filter all incoming requests and, on the other hand, there are more sophisticated cyber attacks. To mitigate the workload and rule complexity problems in security control, our proposal is for a hierarchical and distributed architecture. In the first layer, we partition incoming traffic among numerous intrusion detection system (IDS) monitors, which apply partial security control. In the second layer, there is a scalable analytics cluster, which uses business rules to aggregate the output of the IDS nodes and create a more complete picture for the system's security state. The rules and the cluster are organized in a hierarchy to progressively address more complicated intrusions. In this work, we present the general architecture of such a system and a set of preliminary experiments to expose the scalability problem of security control systems and the capacity of our proposal to solve it.

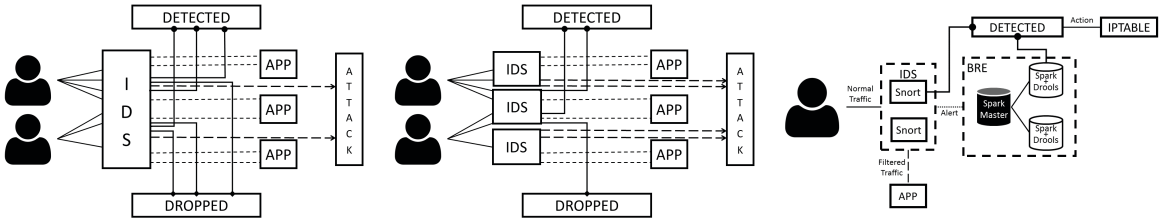
II. Architecture

Figure 1 shows the mental process of constructing the proposed architecture, as motivated by the problem at hand; security control may fail under heavy load in the face of complicated attacks. Figure 1a shows a standard topology for security control with a single IDS monitor in front of the applications. Under heavy load, the IDS starts failing in its cause; due to its limited capacity, it can no longer filter all incoming traffic and it starts to drop packets. The solution to this problem is a distributed set of multiple IDS so that the workload is spread among them (Figure 1b). This will decrease the drop ratio, but allow more sophisticated attacks through. To mitigate the increasing complexity of security threats, we can also partition the responsibility of IDS nodes so that they can analyze incoming traffic based on simpler sets of rules. However, this creates additional gaps in security due to lack of information. Therefore, on top of partitioned rule sets, we also need a hierarchical decision mechanism, which will be able to detect threats based on more complicated rule sets, but derived from partial information alerts coming from the IDS nodes.

The proposed architecture stems from the solution devised in Figure 1c, a hierarchical and distributed security control system. In this system, regular traffic enters the system through a cluster of IDS monitor nodes. The workload is distributed between IDS nodes, as are the security rules to relieve the analytics load on each node. Each IDS has the capacity to identify intrusions on each own based on the localized traffic data it receives, in which case it creates alert traces. The alerts are then forwarded to the Business Rule Engine (BRE) cluster for further analysis. The business rules are implemented as an analytics job to which there is a map function, i.e., analyze alert traces from IDS according to business rules, and a reduce function, i.e., aggregate map results to make more complete decisions for a larger part of the system. In a deep hierarchy with multiple intermediate BRE nodes, every node is reducing partial results from each children, while the root of the hierarchy reduces all results to create the complete state of the system and take actions on a system level. Depending on the completeness of information, even intermediate nodes can take action. Once a set of alerts is identified by the BRE as an actual attack, action is taken.

Figure 2 shows examples of the data that flows through the system. At the top we see an example of an alert as it is generated by the IDS. The next trace in the figure corresponds to the IDS rule that generated the previous alert. The bottom trace shows the business rule to process the corresponding IDS rule. In order to render the IDS alert consumable by the BRE, we develop a `translator` middleware. In practice, this is a simple parser/serializer that transforms the alert trace into an object, so that the BRE can directly access the alert's properties.

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.64



(a) *Standard security control with a single IDS node.* **(b)** *Distributed security control with multiple IDS nodes.* **(c)** *Proposed architecture for distributed and hierarchical security control.*

Fig. 1: *Various topologies for security control.*

Snort Alert:

```
[**] [1:40400007:1] (Machine 2: High Traffic Alert) [**]
[Priority: 2]
10/26-07:01:01.291092 142.104.17.133:35224 -> 10.2.7.17:80
TCP TTL:48 TOS:0x0 ID:37031 IpLen:20 DgmLen:303 DF
***AP*** Seq: 0xC14D5C20  Ack: 0x79CB07AD  Win: 0x409  TcpLen: 32
TCP Options (3) => NOP NOP TS: 647010 32192474
```

Snort Rule:

```
alert tcp 10.2.7.15  any -> $HTTP_SERVERS $HTTP_PORTS (msg:"(Machine
1: High Traffic Attempt)";  sid:40400067; threshold: type both, track
by_src, count 150, seconds 5; rev:1; priority:2; )
```

Drools Rule:

```
rule "Create Script"
    when
       a : Alert ( ( a.getScript() == "" && a.getIP() != "10.2.7.12"
&& a.getIPLength() >= 8 && a.getMsg() == "High Traffic Alert" &&
a.getMachine(0) == a.getMachine(1)) , myAlert : alert)

    then
          System.out.println("Potential DDoS Attack: ");
      a.setScript("iptables -A INPUT -s "+ a.getIP() +" -j DROP");
      update( a );
end
```

Fig. 2: *Examples of (a) Snort alert, (b) Snort rule, (c) Drools rule.*

III. Experiments

We designed four experiments to show (a) how a monolithic IDS fails under high traffic even without a DDoS attack (**Exp1**), (b) the same setting with a DDoS attack (**Exp2**), (c) how a distributed IDS architecture fails due to partial information (**Exp3**), and (d) how a distributed and hierarchical architecture, based on both IDS and BRE, can aleviate both situations (**Exp4**). To set up the experiments, we used iPerf [1] to emulate regular traffic and Skipfish [2] to inject a DDoS attack in the regular traffic. Snort [3] was used as the IDS monitor, Drools[4] as the BRE and Spark [5] as the analytics engine. WordPress [6] was used as our application. We generated high traffic in the amount of 300, 400 and 500Mbps. In the first two experiments, we used a single Snort node with 891 community rules[3]. For the third experiment, we used 2 Snort nodes with the workload split randomly in half and the rules also split in half (approximately 450 for each node). Finally, in the fourth experiment, we deployed the 2 Snort nodes as previously, but now feeding their alerts to a Spark cluster with a single worker and the Drools rule from Figure 2 as the Spark job. Our IDS focuses on generating two kinds of alerts: `HIGH TRAFFIC` (if more than 250 requests come from the same IP in 5 seconds) and `DDoS` (if more than 500 requests come from the same IP in 5 seconds). All experiments were executed 10 times and we report the averages.

TABLE I: *Experimental results.*

Traffic Intensity	Experiment	Packets Dropped	High Traffic	DDoS
300Mbps	**Exp1**	3.86	N/A	N/A
	Exp2	10.81	NO	NO
	Exp3	0—0.01	6—4.9	NO
	Exp4	0—0.04	6—3.9	YES
400Mbps	**Exp1**	21.46	N/A	N/A
	Exp2	27.32	NO	NO
	Exp3	0—0.78	5.9—3.6	NO
	Exp4	0—1.15	5.9—2.4	YES
500Mbps	**Exp1**	31.44	N/A	N/A
	Exp2	33.80	NO	NO
	Exp3	0—2.47	5.1—2.5	NO
	Exp4	0—2.159	5.8—2	YES

Experimental results are presented in Table I. Exp3 and Exp4 have two values reported, one for each Snort node. In Exp1 and Exp2, it can be seen that the amount of packets dropped (reported as percentage in the table) increases rapidly along with the traffic. As a result, Snort cannot generate any alerts, even when there is an attack. When we add a second Snort node, almost no packets are dropped and high traffic alerts start to get generated, although still with decreased frequency as the traffic increases, but there is no detection of DDoS, due to impartial information. When we add the BRE cluster, DDoS is correctly detected. Overall there were two exceptions in our experiments; in one iteration of Exp3, the system actually managed to detect a DDoS attack, while in a single iteration of Exp4, the system did not manage to detect the DDoS attack.

IV. Conclusion

In this work, we identify that current security control systems suffer from scalability problems. For this reason, we propose a distributed and hierarchical security control architecture to handle high workloads and more complex security rules. Preliminary results are promising about the capabilities of the proposed architecture. In the future, we plan to experiment with real applications and more realistic workloads, as well as with a greater variety of security vulnerabilities and attacks. Moreover, we plan to investigate the effect and challenges of deeper hierarchies in the BRE cluster.

REFERENCES

[1] "The TCP, UDP and SCTP network bandwidth measurement tool". [Online]. Available: https://iperf.fr/. Accessed: Feb. 13, 2017.

[2] "skipfish,". [Online]. Available: https://code.google.com/archive/p/skipfish/. Accessed: Feb. 13, 2017.

[3] "Network intrusion detection and prevention system," 2017. [Online]. Available: https://www.snort.org/. Accessed: Feb. 13, 2017.

[4] "Drools - business rules management system (Java, open source)," Drools, 2006. [Online]. Available: https://www.drools.org/. Accessed: Feb. 13, 2017.

[5] "Apache Spark - lightning-fast cluster computing,". [Online]. Available: http://spark.apache.org/. Accessed: Feb. 13, 2017.

[6] "WordPress.com: Create a website or blog,". [Online]. Available: https://wordpress.com/. Accessed: Feb. 13, 2017.