DISCUS: A massively distributed IDS architecture using a DSL-based configuration

Damien Riquet, Gilles Grimaud, Michaël Hauspie

DISCUS: A massively distributed IDS architecture using a DSL-based configuration

Damien Riquet, Gilles Grimaud, Michaël Hauspie
Laboratoire d'Informatique Fondamentale de Lille
Université Lille 1
59650 Villeneuve d'Ascq
Email: firstname.lastname@lifl.fr

***Abstract*—Nowadays, cloud computing becomes quite popular and a lot of research is done on services it provides. Most of security challenges induced by this new architecture are not yet tackled. In this work, we propose a new security architecture, based on a massively distributed network of security solutions, to address these challenges. Current solutions, like IDS or firewalls, were not formerly designed to detect attacks that draw profit from the cloud structure. Our solution DISCUS is based on a distributed architecture using both physical and virtual probes, along with former security solutions (IDS and firewalls). This paper describes DISCUS SCRIPT, a dedicated language that provides an easy way to configure the components of our solution.**

I. INTRODUCTION

Cloud computing is a popular model that provides large resources. A commonly admitted definition by the NIST [15] describes cloud computing as "a model enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources". With the increasing popularity of cloud computing, its security importance is rising and could delay cloud computing adoption for hesitant companies.

In 2010, a study [24] reported that 62 % of small and medium businesses had no plan to move to cloud computing due to security concerns. Academia researchers [2], industry actors [22], and government organizations [11] share the same perspective. That is why cloud providers need to propose a turnkey solution that deals with security concerns like intrusion detection or large-scale coordinated attacks.

Even though the "cloud computing" designation is quite recent, a lot of security concerns have already been studied. In [7], Chen et al. list what are the new challenges of cloud computing. One of these challenges is facing black hats who maintain the appearance of regular users but in fact perpetrate cybercrime or cyber attacks [5, 6]. Also, they mention that shared resource environment could introduce unexpected side or covert channels. Eventually, they point out that cloud providers need to isolate their structure at different granularities (virtual or physical machines, LANs, clouds or datacenters).

Isolation of the cloud components prevents some attacks that draw profit from its structure such as internal attacks (attacks led from the inside of the cloud targeting internal hosts).

Figure 1 pictures a cloud structure with several entry points, several levels of network devices and physical hosts. This basic structure could be either targeted from outside (horizontally dashed) or used to target external hosts (vertically dashed). Our work focuses on detecting these attacks as well as internal attacks (plain), even inside the same cluster or physical machine. The later attacks are yet seldom studied in the literature and, to the best of our knowledge, no solution has been proposed to deal with them.

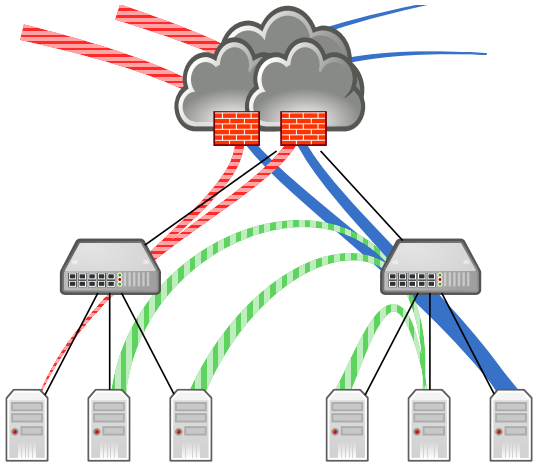

Figure 1: Attacks on the cloud

Current network security solutions in the cloud are Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS) and firewalls. For example, Amazon Web Services (AWS) cloud [1] uses instance isolation and firewalls. This type of device is designed to detect intrusion or attacks from outside to inside of the structure. This is why traditional security solutions are not able to detect internal intrusions, because they are located at the border of the network or simply not aware of the global security state.

In [19], we showed that distributed attacks could easily evade these security solutions, especially in structure with several entry points, like cloud computing.

Though cloud computing relies on datacenter structures, a new challenge arising is to protect the cloud network as well as the end users. Cloud computing security must be adaptable to dynamic architecture (end users may change during an attack). Moreover, cloud users may be malicious or an attacker could have the control of one or several internal hosts. Cloud security must ensure applications availability and data integrity. Finally, it must prevent distributed denial of service (DDoS), worms spreading and such large-scale coordinated attacks (even inside the cloud infrastructure).

Our proposal introduces a new architecture designed for large scale infrastructure, located at multiple places and subject to attacks (either internals or not). This solution, named DISCUS, is based on a massively distributed structure using heterogeneous probes. Physical and virtual probes are scattered

across the network and collaboratively detect intrusions and attacks. Physical probes could be a cheap and reconfigurable device (typically a FPGA) placed behind an host or directly integrated into the switches. Virtual probes are integrated into physical hosts and could be based on existing security solutions like Snort [20] or Bro [17].

The main feature of our solution is to provide a common framework for distributed security based on heterogeneous security devices. Any security solution could be integrated into DISCUS, providing that a back-end exists for this device. One of the supported back-end is Snort and we intend to develop back-ends for firewalls (for example pfSense) or operating systems (through a kernel module).

One of the challenge induced by this architecture is the administration of all the probes. The administrator uses DISCUS SCRIPT to write global security rules once, then the compiler is responsible for instantiating every device with relevant rules.

Our solution tackles new challenges arising with cloud computing. Also, in such structures, customers change constantly and a security solution must be aware of it. Goals of DISCUS are to provide a reconfigurable security solution that is fitted to large-scale infrastructures and also identify or isolate precisely an host among thousands.

The paper is structured as follows. First, Section II introduces related work. Section III describes the architecture of DISCUS, its components and its deployment process. Section IV presents the dedicated language used to configure devices of our solution and Section V proposes an analysis of several security rules written with this language. Finally, Section VI concludes this paper.

II. RELATED WORK

There are three main areas of prior work that are related to this research: (a) security devices used in large scale networks, (b) techniques to distribute these devices and (c) higher-level domain-specific language for security rules description.

Rimal et al. present cloud computing in [18] as the concept that addresses the next evolutionary step of distributed computing. They add that there is a growing concern about its security. Users store confidential information in these architectures, and in wrong hands, it could create civil liability.

Main components of network security structures are firewalls and intrusion detection systems. Bellovin et al. describe firewalls in [3] as components placed between two networks. All the traffic between these networks must pass through the firewall. Also, firewalls filter authorized traffic, defined by local security policies. Debar et al. outline in [9] that the main task of an Intrusion Detection System (IDS) is to monitor the usage of systems and to detect insecure states. IDS detect attempts and active misuses by legitimate users or external parties to abuse their privileges or exploit security vulnerabilities. Popular IDS, like Bro [17] or Snort [20], are based on signatures rules that can match undesired behavior or network traffic.

Early motivation to distribute security systems was to tackle modern network topologies. In [23], Snapp et al. introduce DIDS, a distributed IDS monitoring an heterogeneous network of computers. In the late 90s, Bellovin et al. also proposed in [4] to distribute firewalls because conventional firewalls rely on the notions of restricted topology and controlled entry points. While first solutions were centralized (several probes on the network and a central correlation unit), recent solutions are distributed (each component detects attacks and tries to collaborate with the other components) [26]. The latter are based on agents that analyze network traffic in order to collaboratively detect intrusions using alert correlation.

What we propose through DISCUS is not an agent-based infrastructure that correlates alerts coming from every probes. We want these agents to be able to share data so that all components can be aware at anytime of the state of the network. Our solution uses a content-based Peer-to-Peer system where each agent collaborates and disseminates local analysis. Previous works, like [13, 14, 25], use Distributed Hash Tables to distribute IDS or to detect large-scale attacks as DDoS, worm spreading or distributed portscan.

Dedicated languages, also known as Domain-Specific Languages, are tailored for a specific application domain. A network security DSL can describe an exploited vulnerability, how an attack is performed, what are the consequences of such attack, or how to react.

Exploit languages [21, 10] are used to describe how an attack is performed and the stages of the attack. Knowledge languages [16, 8] have a global expertise of described attacks and know the implications of a single event. They use pre and post conditions to trigger security rules. Detection languages [17, 20] describe the detection of an attack according to a number of occurrences of events. Response languages [21, 12] describe how to react when an attack is detected.

Our solution consists in describing security rules as events, structures and functions. Following languages use the same concepts. One of the first network security language proposed was the Bro language [17] that describes security rules using variables, structures, functions and statements. STATL [10] is a state-based language that use security rule to describe transition from an automaton.

III. DISCUS

The main goal of DISCUS is to detect and stop attacks targeting or using large network architecture, like cloud computing. Also, we want this solution to be easily (re)configurable because new exploits are discovered every day. People who are responsible for the security of a network should be able to load the new security rule easily, that is why we provide a simple and expressive dedicated language.

This section introduces DISCUS. First, it describes the proposed solution, then the security devices integrated into the solution, and eventually how to configure the whole structure.

A. Architecture overview

Commonly used network security solutions are firewalls and Intrusion Detection Systems. These solutions usually consider that attacks are led from an external network targeting the protected network (or vice versa). That is not true anymore for large networks like in cloud computing. Because these devices are not adequate for this type of network, we propose another security structure based on common security solutions, combined with a massively distributed structure composed of security probes.

Figure 2 represents the proposed structure. In this basic structure, firewalls remain the entry points from the outside of the network and filter undesired traffic. New elements are security probes, either physical or virtual. We want these security probes to be as close to the hosts as possible. Thus, we can finely analyze network traffic that goes through these

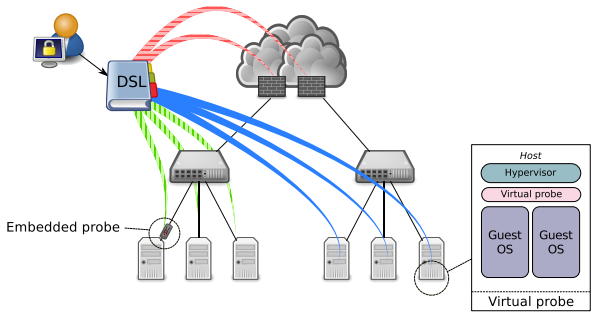

Figure 2: DISCUS architecture

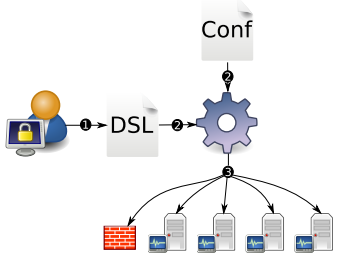

Figure 3: DISCUS deployment

security probes and stop attempts of intrusions and attacks, directly at their source. This can also let security administrator isolate one or several hosts which have been detected as malicious. Physical probes are placed behind physical hosts or plugged to the switch or directly integrated into it. Virtual probes are necessary when virtual machines are running on a physical machine (typical scenario in cloud computing). In this case, we need to analyze traffic that comes from every virtual machine, in order to prevent attacks inside a physical host. These probes are configured by an administrator responsible for the security of the structure. He describes security rules thanks to DISCUS SCRIPT, a dedicated language, and uses the provided compiler to generate a unique binary for every security probe. After being configured, security components cooperate to detect intrusion occurring on the network. If a decision has to be made (for example, how to react when an intrusion has been detected), decision can be delegated to powerful components of the structure like firewalls.

B. Security probes

In our solution proposal, we introduce new security probes. The goal of these security probes is to be placed in cut-through configuration between hosts and network elements. Physical probes are physically plugged directly on the host or the closest switch. Virtual probes are integrated into physical hosts running virtual machines. The latter could be existing security solutions such as Snort or Bro, customized in order to be able to collaborate into DISCUS. They process network data from these virtual hosts in order to prevent attacks occuring inside a same physical host. Also, they can isolate a virtual machine which has been detected as being a malicious host.

C. Configuration of the structure

Configuration of security devices of our solution is done in few steps. Figure 3 depicts these steps. At first, (1) the security administrator write security rules using the dedicated language. Then, thanks to a configuration file describing the network structure, (2) he compiles these rules using DISCUS SCRIPT's compiler. Eventually, (3) the compiler generates a binary program for every probe and components of the security structure. Each binary is produced by selecting and adapting a subset of the rules matching the context of the target. For example, a rule specifying that network traffic from a specific internal host is forbidden will only be consistent for the security probes monitoring this specific host.

IV. DISCUS SCRIPT

In this section, we introduce the dedicated language proposed along with our solution DISCUS. In our case, we wanted to configure easily probes and security elements of the structure. We use a static typing and declarative language. Besides, security rules expressed in this language are described as events, because it is natural to describe a sequence of network packets as a sequence of events. Following subsections introduce main syntax elements of the language. We will not discuss in this paper some syntax elements (like structures or enumerations) because they are very similar to C instructions, as well as types (because of the length of the paper). The only types used in this paper are enums, netaddr and *intX*, where *X* is the number of bits of the integer.

A. Table declaration

A security device usually keeps contextual data in order to run analysis. It includes hosts on the network, state of the connections, number of connections, and so on. Tables are aggregate types that store contextual data. For example, it can be used to store the number of failed TCP connections in order to detect portscan. Such tables are local or shared between components of the security solution.

Listing 1: Example of a local table declaration

```
table attack_attempts {
    int32 attacker;
    int attempt_counter;
    time last_attempt;
};
```

Listing 1 declares a table storing the number of attack attempts for a specific IPv4 source. When this number reaches a fixed threshold, the security device reacts or raises an alert.

One of the goal of DISCUS is to provide a distributed system able to detect large-scale attacks. This type of detection is possible through tables which can be global or local. A local table could be assimilated to a database stored directly on the security device (like common security solutions). Shared tables are spread over the network and each device is able to get or update entries of an table. When the compiler is processing the security rules, it has the full knowledge of which security devices can store data (thanks to the topology configuration), which security devices insert or update entries in which table and so on. It is the role of the compiler whether or not to distribute a table and where to distribute it.

Such distributed architecture could be based on existing systems like Distributed Hash Table. A network overlay such as Peer-to-Peer systems deals with joining and leaving device and provides fault-tolerance and scalability.

B. Purge and deletion of tables

Our solution is based on a multitude of security probes storing tables. Unfortunately these devices have limited resources and cannot store unlimited data. That is why table entries need to be deleted. There are two scenarios for deletion of entries: no more resources on the device (need to purge the system) or obsolete entries (deletion of obsolete entries).

Listing 2: Example of a deletion statement

```
remove attack_attempts
    when (now - last_transmission) > 3600;
```

In Listing 2, we specify that we do not want to keep entries that are too old (in this case, last attempt has been made at least one hour ago). Purge statements are declared in a similar manner, that is why we won't discuss it here.

C. Event-based security rules

Major statements of this dedicated language are security rules. We chose to adopt an event-based approach because network attacks could be naturally separated into events spread over the time. Moreover, event based systems can be easily and very efficiently implemented on FPGA. Listing 3 presents the syntax used to define a new event. A rule is triggered when an event (**on** `A`) occurs and conditions are satisfied. An event is defined using a name and a list of parameters. When a rule is triggered, it can execute several statements and/or raise one or several events (**raise** `D`), immediately or after a delay. There are several statements: create or delete a table entry, update a field in a table or run a command.

Listing 3: Security rule syntax

```
on A(args)
    where  ... /* Conditions      */
    insert ... /* Tables */
    update ... /* Tables fields  */
    run ... /* Script or action from the library */
    raise D [in ...];
```

Conditions are optional and a rule must have at least a statement. More examples of security rules are given in Section V.

D. Characteristics of the language

In this subsection, we point out several characteristics of the dedicated language. First, we want the compiler to be minimalist, in other words, we want that almost all elements are defined using the language (for example, ip or tcp structure will be defined using the language and not hard-coded in it). Also, the compiler is able to detect inconsistent or conflicting rules and alert the security administrator so that he could correct corresponding rules. Finally, the compiler decide where the data is stored across the network of security devices.

Our language has several properties as well. It is a statically typed language, mainly because we want to avoid programming mistakes or semantic misinterpretations. Also, the language detects cycles, thus it can guarantee the detection process will terminate. Finally, the compiler deletes inconsistent or unused rules : orphan event, sequences of events that lead to a dead leaf (an terminal rule that does nothing for example).

V. Security rules examples

In this section, we describe several security rules using Discus Script. In the following example, we want to prevent a SYN flood attack, a simple but effective denial of service. A basic idea to detect this type of attack is to pay attention to the number of opened connections between two hosts that are not fully established. When this number exceeds a fixed threshold, the security solution can react (for example, blacklist the establishing host, close all connections and so on).

We consider that the table `attack_attempts` from Listing 1 is defined for the remainder of this example. We also consider that an event `tcp_packet`, triggered when a TCP packet is received, is implemented into the library of the solution.

At first, we need to create a contextual structure that stores the state of TCP connection.

```
enum tcp_state {...};
table tcp_table {
    netaddr src, dst;
    int16 p_src, p_dst;
    enum tcp_state state;
    time last_trans;
};
```

When a new TCP connection is being established, we create an entry and store it locally. Because of the length of the paper, we will omit parameters not used in this example.

```
on tcp_packet(..., netaddr src, netaddr dst,
    int16 p_src, int16 p_dst, int9 flags, ...)
    where flags == SYN
    insert into tcp_table {
        src = src; dst = dst;
        p_src = p_src; p_dst = p_dst;
        tcp_state = TCP_HANDSHAKE_SYN;
        last_transmission = now;
    };
```

When such entry table is created and the remote host answers back a TCP SYN-ACK, we try to detect SYN flood. This attack consists in partially opening a lot of TCP connections in order to exhaust the target's resources. So, a basic detection method is to look for TCP connections not fully opened after a short delay (here 250 ms). We consider for the following listings that the arguments are the same.

```
on tcp_packet(...)
    where flags == SYN | ACK
    and exists t in tcp_table
        with t.src == src, t.dst == dst,
             t.p_src == p_src, t.p_dst == p_dst,
             t.state == TCP_HANDSHAKE_SYN
    raise syn_flood_attempt(src, dst, p_src, p_dst)
        in 250 ms;
```

If the TCP connection is still not fully established after that short delay, we use a local table entry that store the number of attempted attacks. When the structure exists, we only need to update `attempt_counter` and the time of the last attempt.

```
on syn_flood_attempt(...)
    where exists t in tcp_table
        with t.src == src, t.dst == dst,
             t.p_src == p_src, t.p_dst == p_dst,
             t.state == TCP_HANDSHAKE_SYN
    and not exists a in attack_attempt
        with a.attacker == dst
    insert into attack_attempt {
```

```
        attacker = dst;
        attempt_counter = 1;
    };

on syn_flood_attempt(...)
    where exists t in tcp_table
        with t.src == src, t.dst == dst,
             t.p_src == p_src, t.p_dst == p_dst,
             t.state == TCP_HANDSHAKE_SYN
    and exists a in attack_attempt
        with a.attacker == dst
    update a.attempt_counter += 1
    update a.last_attempt = now
    raise syn_flood_check(src)
```

When the number of attempted attacks reaches a fixed threshold, the security solution reacts. In this case, we decide to blacklist the attacker for one hour.

```
on syn_flood_check(netaddr src)
    where exists a in attack_attempts
        with a.src == src,
             a.attempt_counter >= THRESHOLD
    run blacklist(src, 3600);
```

VI. Conclusion

Cloud computing is a powerful architecture. It can provide several layers of services according to the needs of end users. Currently, only security concerns delay its massive adoption.

In this paper, we propose Discus, a new architecture that could solve security issues arising with cloud computing. Our proposal is based on a massively distributed structure that relies on commonly used security solutions but also probes scattered across the cloud structure. These probes could be physical or virtual and are located close to rented hosts. They collaboratively detect intrusions and attacks that target or use the cloud structure. To configure all these probes, we propose in this article Discus Script, a dedicated language, that describes security rules.

At the moment, the compiler is implemented, as well as the Snort backend. We are currently experimenting this backend and working on the configuration and deployment of security rules. Future work includes distribution of data across the probes and probes reconfiguration on the fly.

References

[1] Amazon. Amazon web services: Overview of security processes. Technical report, 2011. URL http://aws.amazon.com/security.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, 2009.

[3] S. Bellovin and W. Cheswick. Network firewalls. *Communications Magazine, IEEE*, sept. 1994.

[4] S. M. Bellovin. Distributed firewalls. *Journal of Login*, 24(5):37–39, 1999.

[5] Bitweasil. Cryptohaze cloud cracking. *Defcon 20*, 2012.

[6] D. Bryan and M. Anderson. Cloud computing : A weapon of mass destruction? *Defcon 18*, 2010.

[7] Y. Chen, V. Paxson, and R. H. Katz. What's new about cloud computing security? Technical report, University of California, Berkeley, Jan 2010.

[8] F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *Recent advances in intrusion detection*. Springer, 2000.

[9] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 1999.

[10] S. Eckmann, G. Vigna, and R. Kemmerer. Statl: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

[11] European Network and Information Security Agency. Cloud computing risk assessment. Technical report, 2009.

[12] W. Kanoun, S. Dubus, S. Papillon, N. Cuppens-Boulahia, and F. Cuppens. Towards dynamic risk management: Success likelihood of ongoing attacks. *Bell Labs Technical Journal*, 2012.

[13] Z. Li, Y. Chen, and A. Beach. Towards scalable and robust distributed intrusion alert fusion with good load balancing. In *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, LSAD '06. ACM, 2006.

[14] M. Marchetti, M. Messori, and M. Colajanni. Peer-to-peer architecture for collaborative intrusion and malware detection on a large scale. In *Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 475–490. Springer Berlin Heidelberg, 2009.

[15] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical report, July 2009.

[16] C. Michel and L. Mé. Adele: an attack description language for knowledge-based intrusion detection. In *Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001)*, 2001.

[17] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 1999.

[18] B. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *NCM '09*, aug. 2009.

[19] D. Riquet, G. Grimaud, and M. Hauspie. Large-scale coordinated attacks : Impact on the cloud security. *The Second International Workshop on Mobile Commerce, Cloud Computing, Network and Communication Security 2012*, page 558, July 2012.

[20] M. Roesch et al. Snort-lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, pages 229–238. Seattle, Washington, 1999.

[21] P. Salgueiro, D. Diaz, I. Brito, and S. Abreu. Using constraints for intrusion detection: the NeMODe system. *Practical Aspects of Declarative Languages*, 2011.

[22] S. Shankland. Hp's hurd dings cloud computing, ibm, 2009. URL http://news.cnet.com/8301-30685_3-10378781-264.html.

[23] S. R. Snapp, , J. Brentano, and G. V. Dias. Dids (distributed intrusion detection system) - motivation, architecture, and an early prototype. *in proceedings of the 14th National Computer Security Conference*, 1991.

[24] Spiceworks. New study sees rise in cloud services adoption among small and medium businesses in first half of 2010, 2010. URL http://www.spiceworks.com/news/press-release/2010/07-28.php.

[25] V. Yegneswaran and P. Barford. Global intrusion detection in the domino overlay system. In *Proceedings of Network and Distributed System Security Symposium*, 2004.

[26] C. V. Zhou, C. Leckie, and S. Karunasekera. A survey of coordinated attacks and collaborative intrusion detection. *Computers and Security*, 2010.