A Distributed Active Response Architecture for Preventing SSH Dictionary Attacks

J. Lane Thames
Georgia Institute of Technology
lane.thames@gatech.edu

Randal Abler
Georgia Institute of Technology
randal.abler@gatech.edu

David Keeling
Georgia Institute of Technology
dkeeling@gatech.edu

Abstract

Dictionary attacks against Internet servers that provide the Secure Shell (SSH) service for secure, remote login is very common. The dictionary attack is an attempt to gain unauthorized access to a server by continuously guessing username and password pairs using sophisticated brute force techniques. Solutions exist that can detect and prevent this attack for a local host. However, a technique that distributes the detection and prevention information to a server's trusted neighbors can provide a gain in security by way of preemptive protection. This paper describes a distributed active response architecture that provides proactive, preemptive protection against the SSH dictionary attack.

1. Introduction

All computer systems connected to the Internet are at risk of being attacked. There is an attack class known as the *dictionary attack* that is in wide spread use by attackers. A dictionary attack is an attempt to exploit a user authentication system by systematically guessing authentication tokens. An authentication system has a primary goal of proving an entity's *identity*. Current authentication systems use three types of credentials [2], also referred to as *factors*, to authenticate entities: *knowledge, ownership,* and *self-characteristics*. Knowledge based authentication is based on something one knows such as a username and password pair. Ownership authentication is based on something one owns such as a smart-card or hardware token. Self-characteristic authentication is based on biological traits of humans such as fingerprints, voice patterns, or retinal patterns. An authentication system can be 1-factor, 2-factor, or 3-factor. A 1-factor system relies on the entity using only one of the three types of authentication factors, 2-factor employs two types, and 3-factor requires three types. Of course, the security provided by the authentication system increases as the number of factors increases.

Single factor authentication is by far the most commonly used method for allowing access to computing systems and services during a ***login*** process whereby a user must know a username and password pair for access to be granted. If configured correctly, a single factor authentication system can provide a reliable authentication service. However, "correct configuration" is a subject of debate. The global problem that arises with the common login process is that many systems exist that allow users to choose extremely weak authentication tokens, namely login passwords. It is commonplace for users to create passwords that are based on easily guessable patterns. For example, these patterns could be simple permutations of the user's personal information such as Lthames210 or permutations of common phrases such as Linuxabc, Password1a2, or HelloWorld123. A dictionary attack systematically guesses, using a "brute force" technique, username and password pairs based on a *dictionary* of common phrases. The brute force aspect of the attack refers to the rules that the attack code uses to continuously permute the patterns of the tokens in the dictionary database while attempting to authenticate to some login process. Dictionary attacks take advantage of the password-pattern phenomenon, which increases the probability that a successful intrusion can be made. Essentially, the dictionary attack exploits a human behavioral weakness in the way passwords are chosen.

The secure shell program is a remote terminal login service that allows users to access a computing system from across the network. SSH is a replacement of the legacy telnet program and uses encryption techniques to offer secure connections between the user's SSH client and the SSH server. SSH can be configured to use various forms of authentication for remote login, but the most common configuration is based on single factor authentication using a username and password pair. Many Internet servers provide SSH services so that system administrators can access the machine for administrative purposes. Attackers will scan the Internet IP address space looking for machines that have SSH services available. When these machines are located, the attackers will execute dictionary attacks against these SSH servers trying to gain unauthorized access. Figure 1 shows dictionary attack results for an SSH server connected to the Internet. The data in Figure 1 represents the total number of failed login attempts per day for 44 consecutive days against the SSH service of a server that the authors administer under the EDU hierarchy of the DNS namespace. Figure 2 displays the total number of unique IP addresses associated with these failed login attempts. From this data, there was an average of 324.3 failed login attempts per day, an average of 3.2 unique IP addresses issuing the attacks per day, and an average of 81.8 failed login attempts per IP address per day. This data included a maximum of 3533 failed login attempts on day number 18. Another study [5] showed SSH dictionary attack data comparable to the values shown in Figures 1 and 2.

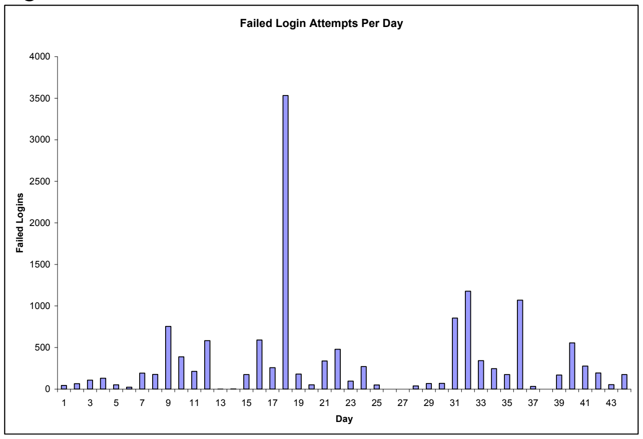

Figure 1: SSH dictionary attacks against an Internet server.

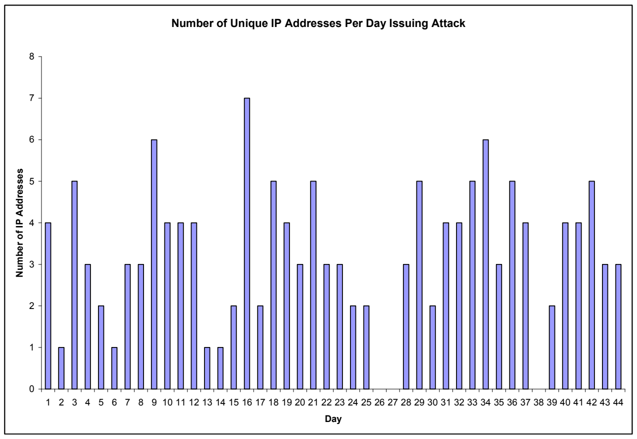

Figure 2: Number of Unique IP addresses attacking the server.

2. Preventing SSH Dictionary Attacks

A first line of defense in preventing a computer attack is the firewall. Firewalls, either network based or host based, are configured to allow or deny connections into or out of its perimeter based on the protected entity's security policy. The policy states what types of connections are allowed into or out of the entity. However, when network service such as SSH must be offered, the firewall is configured to allow connections to the service. The solution needed is an application that can detect a dictionary attack and dynamically apply firewall blocking rules against the source of the dictionary attack. This solution already exists. The following is a short list of available applications that offer protection against the SSH dictionary attack:

- Snort Intrusion Detection System
- DenyHosts application program
- OpenSSH timelox software patch
- SSHD_Sentry application program

These applications detect SSH dictionary attacks against a local host and can be configured and/or modified to automatically apply firewall blocking rules to the local host from the source of the attack.

Classically, firewalls and access control mechanisms are implemented as static protection mechanisms. The rules are configured based on the security policy for the host or network that these devices protect, and the rules remain constant unless there is a security policy change. Intrusion detection is technology designed to monitor hosts and/or networks. These systems monitor a host or network based on configurable rules and specifications. Once abnormal activity is observed, the system will send an alert to the system administrator. It is the responsibility of the system administrator to act on the alerts they receive from the detection system. Unfortunately, the response time of human administrators is too slow compared to the speed of modern day attacks. As a result, research in the field of intrusion detection has begun to focus on the concept of ***Active Response*** [4]. Active response is the act of detection systems dynamically responding to real time attacks without the need of human advisory. Indeed, the detection techniques listed in this section are simple mechanisms that offer active response by dynamically blocking access to servers from the sources of SSH dictionary attacks.

The solutions listed above work well for protecting the local host from a dictionary attack. However, we posit that a distributed solution that offers dissemination of the detection information and security policy to participating neighbors can offer greater security by way of preemptive and proactive protection.

3. Distributed Active Response

3.1. Motivation

Attackers employ multiple tools and techniques when attempting to invade a computer or networking system. Considering this, if one applies a "deny-all" blocking policy against an offending source IP address when an SSH dictionary attack is detected, then there is also the gain of preemptively protecting the server from other types of attacks from the same malicious source. The deny-all policy achieves preemptive protection because it prohibits the attacker from accessing any services on the local host, not just the SSH service. This can be viewed as an active response preemptive protection mechanism for the local host. However, with a distributed active response system, the local host and its trusted neighbors can achieve preemptive protection. The fundamental design principle of the proposed distributed active response architecture is the following: *Once a source IP address has been classified with an anomaly detection mechanism as being untrustworthy, deny access from the source to all services running on any servers in the trusted domain of administration.* In this paper, we define the anomaly detection mechanism as any application that can detect SSH dictionary attacks. Being classified as untrustworthy implies that the source of the dictionary attack has malicious intentions and can issue other harmful exploits against the servers in the Trusted Domain of Administration (TDA). The TDA is defined as the set ∑ of servers under the control of a single administrator.

3.2. High Level System Description

A simple description of this distributed active response architecture follows. Consider a TDA composed of $\sum = \{S_1, S_2, S_3, S_4, S_5\}$, where each S_i is a server offering network services including SSH to the Internet. Each S_i has a process that is monitoring for dictionary attacks against its SSH service, i.e. one of the solutions listed in section 2. Once an attack is detected, the offending IP source address is added to a local database and a filtering rule is created within its local host-based firewall that denies all access from the offending IP address to the server. Then, the server sends this information to each of its neighbors within the TDA so that they can also add a deny rule for the offending address. This process is an act of distributed active response that implements preemptive protection. Consider the case where a malicious node issues a dictionary attack against S_1. Once S_1 has detected the attack, a blocking rule will be created for the offending IP address, and after distribution the other neighbors will have a blocking rule too. At some time in the near future, the same malicious node issues a sequence of attacks against S_5. But, because this server has previously added a blocking rule based on the distributed security policy received from S_1, it will not be affected by the attacks. Hence, S_5 has been preemptively protected.

Trust is a critical aspect of this architecture. If trust is neglected, there is a possibility that false policies can be injected into the system such that denial of service could be issued on random hosts. Therefore, a well defined TDA is required. Using the definition of the TDA as proposed in section 3.1, trust can be achieved. Since a server must "trust" its administrator by default, and each server in the TDA trusts the same administrator, then each server in the TDA can trust one another as a consequence of transitive trust.

3.3. Detailed Architectural Description

Figure 3 illustrates the framework for the distributed active response architecture. The architecture is composed of 4 primary management modules: *Firewall Management, Policy Management, Distribution Management,* and *Anomaly Detection System (ADS) Management*. Further, the architecture contains 2 specification interfaces: *Firewall Rule Specification Interface (FRSI)* and *Policy Description Specification Interface (PDSI)*. This system is designed to be completely distributed with no centralized control.

ADS Management Module: The ADS management module is simply the collection of independent ADS components (anomaly detection mechanisms) that are active on the local system and the collection of rules that define how the ADS must interact with the policy and distribution management modules. Currently, the architecture defines one rule: the ADS must adhere to the contract offered by the PDSI.

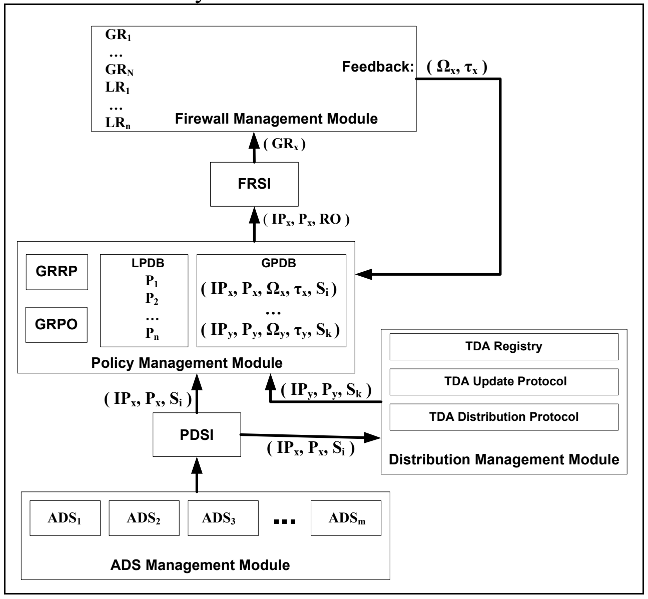

Figure 3: Distributed active response architecture.

Policy Description Specification Interface: The PDSI is an interface that offers a globally recognized Policy Description Specification (PDS) contract that all members of the TDA interpret with the same meaning, i.e. it is a formal syntax describing a policy. The following is the PDSI contract: **(IP_x, P_x, S_i)**. The contract is a 3-tuple containing the IP address of the attacking host (IP_x), the policy action to take against the attacking host (P_x), and the server identifier (S_i). The server identifier is some value that uniquely identifies the TDA server that generated the PDS. P_x is a policy action construct. The architecture currently defines only one action, P_x = DENY. Future extensions can be made whereby one might offer rate-limiting definitions. However, P_x shall never be extended to offer an operation that elevates a service privilege, i.e. P_x = ALLOW. The ADS issues the PDS to the policy management and the distribution management modules via the PDSI.

Policy Management Module: The Policy Management Module (PMM) contains 4 components: *Local Policy Database (LPDB), Global Policy Database (GPDB), Global Rule Removal Protocol (GRRP),* and *Global Rule Placement Optimizer (GRPO)*. The LPDB is the collection of security policies that have been defined by the system administrator for the local host. For example, if the local host is a web server, the list of local policies could be stated like the following: {allow from any to localhost service http; allow from any to localhost service ssh; deny from any to localhost service any}. The policy states that the local host can receive inbound HTTP or SSH connections and all others are to be denied. The GPDB is the collection of 5-tuples having the following form: **(IP_x, P_x, Ω_x, τ_x, S_i)**. This list of variables represents the IP address of the anomalous host, the policy action to apply against the anomalous host, the firewall hit-count (Ω_x), the firewall hit-time (τ_x), and the server identifier. The firewall hit-count and hit-time are feedback variables that are received from the local host's firewall module and are used by the GRRP and GRPO. The architecture defines that the hit-count is initialized to a value of 1 and the hit-time is set to the local host's current system time upon insertion into the GPDB. These values are recalculated over time by the firewall management module. The GRRP and GRPO are responsible for removal of global firewall rules and optimization of their placement within the firewall rule base, respectively. These functions will be described in more detail in section 3.4. The IP address, policy action, and server identity within the GPDB are received either by the local PDSI or the distribution management module. The ones received via the distribution management module represent the policies received from one of the neighboring servers within the TDA. Once new policies are received by the PMM, they are inserted into the GPDB via the PMM and into the firewall via the FRSI.

Firewall Rule Specification Interface: The FRSI offers a contract between the Firewall Management Module (FMM) and the PMM. The FRSI's contract has the following form: **(IP_x, P_x, RO)**. The 3-tuple contract contains the IP address of the anomalous host, the policy action to apply against the anomalous host, and the Rule Operation (RO). Three types of rule operations have been defined: {RO = Insert, RO = Delete, RO = Modify}. These define the firewall operations of inserting a new rule, deleting an existing rule, or modifying an existing rule. The FRSI is responsible for dynamically mapping the data contained within a received contract into a firewall rule based on the firewall's native syntax. Using an interface such as this offers extensibility because the architecture will not be constrained to one type of firewall product. The FRSI can be modified by the system implementer to map the data to the syntax of the desired firewall to be used.

Firewall Management Module: The FMM is responsible for dynamically configuring the firewall rule base when rule structures are received from the FRSI. Further, the FMM collects the hit-count and hit-time feedback variables and sends these variables to the PMM. The hit-count represents the number of times that a rule within the set of global rules has been matched (hit) by a particular incoming IP address and the hit-time represents the local system time of the last hit.

Distribution Management Module: The DMM receives the 3-tuple **(IP_x, P_x, S_i)** either from the local host or from one of the local host's neighbors within the TDA. If the tuple is received from the local host, it is immediately distributed to the other members of the TDA. And, if it is received from one of its neighbors, it is immediately sent to the PMM for processing.

The DMM contains three components: *TDA Registry, TDA Update Protocol,* and *TDA Distribution Protocol*. The TDA registry stores server identifier information. At minimum, this is a list of all the members of the TDA. The identifier values are not strictly defined by the architecture. However, the suggested values are S_i = { IP_i, $Server\_Name_i$, $Public\_Key_i$ }, which is a set containing a member's IP address, its DNS name, and its public encryption key. Of course, if encryption keys are used, an agreed upon encryption standard must be designated. The TDA update protocol defines policy update procedures. The architecture currently defines one procedure, which is to update immediately upon receiving new policies. Finally, the TDA distribution protocol defines how the TDA members will distribute information between each other. This is not strictly defined by the architecture, but encrypted communication streams are suggested. The authors use encrypted data transfers by employing the SSH protocol.

3.4. Solutions for Limitations of the Architecture

The architecture inherently contains some limitations that require solutions. These limitations include the possibility of a large number of firewall rules, and non-constant IP addresses assigned to hosts within networks that employ Dynamic Host Configuration Protocol (DHCP) and Network Address Translation (NAT).

The GRPO and GRRP components of the PMM were designed to solve these limitations. The GRPO component is designed to optimize the placement of the global rules within the firewall rule base. This optimization is used to help alleviate the issue of large numbers of global rules. The global rules are of the following form: {deny from IP_x to localhost service any}. All of the global rules must be processed in the firewall first, and then the local policy rules must be processed. This is a consequence of the logical ordering of rules needed in an access control list. An example will be given to illustrate. Consider the following firewall rule list:

```
{
        deny from 1.2.3.4 to localhost service any;
        allow from any to localhost service http;
        allow from any to localhost service ssh;
        deny from any to localhost service any;
}
```

This sequence of rules states that IP address 1.2.3.4 should not be allowed to access any services on the localhost. But, all other IP addresses are allowed access to HTTP and SSH. The last rule is the default rule that is used to deny access to any other service on the localhost. If the first rule is placed after the second rule, then 1.2.3.4 would be able to access HTTP. However, that defeats the goals of this architecture, which is to deny access to any service on the localhost once a host has been classified as anomalous. Hence, all of the global rules generated with the ADS or received from another TDA member must be processed before the local policy rules. As seen from the FMM in Figure 3, there are a total of n + N firewall rules configured within a local host's firewall. The n LPDB policies map to n local rules (LR), and the N GPDB policies map to the N global rules (GR). As the number of global rules increases, the amount of time to process a connection request through the firewall increases. While a connection is being processed by the firewall, other incoming connections must wait in the server's connection queue. The firewall service time and queue waiting time are inherently probabilistic. However, there is an opportunity for some queue optimization under certain conditions. During a period where the server is not under an attack, the queue will contain connection requests from non-hostile clients. Each non-hostile connection will wait in the queue while the connections ahead of it are being processed through the firewall with the n + N rules. However, during times when the server is under attack from one or more hostile clients, the queue will contain connection requests from hosts who have been classified as anomalous and have a blocking rule in the global rule list. If these connections can be processed quicker, then the waiting time of the non-hostile connections can be reduced. The GRPO component is designed to provide this small optimization. The idea is that the anomalous IP addresses are monitored by the FMM, and the "heavy-hitters" (the anomalous IP addresses) that are currently attacking the server have their global rules placed at or near the top of the global rule list. For this to be achieved, the hit-count and hit-time feedback variables are used to make intelligent placement decisions. The FMM sends the hit-count and hit-time to the GPDB once a match is made in the firewall rule base for the anomalous IP addresses. The GRPO component uses the following algorithm to optimally order the rules in the global rule list.

Algorithm GRPO:

```
{
        X := {GPDB.($IP_i$, $\Omega_i$, $\tau_i$)};
        $\Delta\tau_i := \tau_c - \tau_i$;
        Z := {X: $\Delta\tau_i < T_0$};
        Y := X – Z;
        maxsort(Z, $\Omega_i$), minsort(Y, $\Delta\tau_i$);
        X ← Z || Y;
        UpdateRules(X. $IP_i$);
}
```

The set X is extracted from the GPDB and all of the members that satisfy $\Delta\tau < T_0$ are extracted from X and stored in the set Z. The temporal optimization parameter T_0 defines a threshold value for the time range calculated in $\Delta\tau$. $\Delta\tau$ is defined as the time difference between the current system time, τ_c, and the last time, τ_i, that the anomalous host IP_i has attempted a connection. This implies that Z contains the 3-tuples for some set of anomalous IP addresses that have attempted to connect within the last T_0 seconds. Y contains the remainder of 3-tuples for the anomalous IP addresses that have not attempted a connection recently, where recently is defined by T_0. The maxsort() function takes the list of 3-tuples and sorts them using the hit-count variable as the sorting key. This list is sorted from maximum hit count to minimum hit count. The minsort() function takes the list of 3-tuples and sorts them using the hit-time variable as the sorting key. This list is sorted from minimum $\Delta\tau$ to maximum $\Delta\tau$. Then, X is replaced with the concatenation of Z and Y. Finally, the UpdateRules() function updates the logical arrangement of global rules in the firewall based on the updated ordering of IP addresses stored in X. The overall result is that the most recent hitters are sorted in such a way that the ones that are *currently* hitting the most are near the top of the list. The remaining hitters are sorted according to the time since their last hit.

The Global Rule Removal Protocol (GRRP) is designed to remove global rules that have become stale. DHCP and NAT offer dynamic mapping of IP addresses to hosts. Generally, these mappings are not constant over time, which implies that hosts can obtain different IP addresses over time. Since the firewall will filter connections based on IP addresses, old rules will need to be removed. Consider the case where the ADS detects an anomalous host that is within a network that uses DHCP. Once the host's lease time for its IP address has expired, it will need to renew its IP address and the renewed IP address can be different. If the new IP address is different, the current rule in the firewall and GPDB will be stale as it no longer references the true attacking host. Further, if the IP address is issued to a different host, the system will be causing denial of service to the current owner of the IP address that was marked as anomalous. Another issue is that the recurrence of attack from hostile hosts has a heavy tailed distribution. This means that a small number of hosts will issue many attacks against a particular server over time. The majority of hostile hosts will attack a server only a small number of times. Considering these issues, the architecture needs to remove rules for hosts that have not attacked the server *recently*. Removing stale rules will alleviate the issues with large numbers of rules, DHCP, and NAT. GRRP is designed to remove global rules based on the 2 variables, Ω_x and $\Delta\tau_x = \tau_c - \tau_x$, associated with an IP address, IP_x. If it has been a long time since IP_x has attacked the server, then $\Delta\tau_x >> 0$ and we want to consider removing the global rule for IP_x. However, we want to also consider how often IP_x was attacking the server, and this information is contained in Ω_x. We define the rule removal optimization parameter ω_x as the following:

$$\omega_x = \frac{\Delta\tau_x}{\Omega_x} \qquad (1)$$

Equation 1 allows ω_x to be normalized relative to the hit count for a particular attacking host IP_x. By definition, $\Omega_x \geq 1$. Hence, ω_x will be bounded by $\omega_x = \Delta\tau_x$ and $\omega_x = 0$, for all $\Delta\tau_x \geq 0$. We define the rule removal timing threshold T_R as a time interval such that if $\omega_x > T_R$ then the rule should be removed. The following is a formal statement of the algorithm.

Algorithm GRRP:

```
{
        $\forall x \in X$
        If ($\omega_x > T_R$) Then Remove(GPDB[x], FR[x]);
}
```

In this algorithm, X is the set of all IP addresses associated with the GPDB and the firewall rule (FR) base. The instance x can be used to index into both the GPDB and the FR base. Observing the dynamics of Equation 1, one can see that if host x issues many attacks (implying that $\Omega_x >> 1$) then the rule will remain in the GPDB and FR base for larger amounts of time because $\Delta\tau_x$ will need to grow larger in order to satisfy $\omega_x > T_R$. Hence, the policies and rules will remain active longer for hosts that are more active with their attacks.

4. Simulation Results

To obtain empirical data describing the behavior of this architecture, one will need to have machines connected to the Internet that have various publicly available services in operation. Further, the data should be collected over a reasonable amount of time so that valid conclusions can be drawn. The authors are currently collecting data that describes this architecture's behavior, but there is not enough data available now to publish in this paper. However, a simulation of the distributed active response behavior of this system was created, and the results are shown in Table 1.

Table 1: Simulation results.

	S1	S2	S3	S4	S5
Attack Method					
Normal	0	0	10	0	0
Random	7	8	8	10	4
Weighted	10	1	4	2	0
Round Robin	9	10	9	9	9
Non-distributed, Any Method	10	10	10	10	10

The simulation was based on a TDA of 5 members, S1-S5. There was a single host attacking the servers using 4 types of SSH dictionary attack methods: *Normal, Random, Weighted,* and *Round Robin.* The normal method is based on the common type of SSH dictionary attack—the attacker attempts many login attempts against a single server before targeting another server. The random method models an attack where the attacker issues single login attempts against the servers in a random order. The weighted method models an attack where the attacker focuses on a primary server but occasionally issues an attack on another server randomly during the process. Finally, the round robin method assumes a model whereby the attacker issues one login attempt per server in a round robin fashion. The first four rows of data reveal the number of failed login attempts on each server before being blocked with the distributed active response system. A threshold value of 10 was used in the SSH ADS. The threshold value is the number of failed login attempts from a single source IP address that is needed before a blocking rule is applied, i.e. we assume that 10 consecutive failed login attempts is due to an SSH dictionary attack. The data shows that the architecture performs well for the normal dictionary attack method. The random and weighted methods perform marginally well. But, the round robin method is only slightly better than the non-distributed active response method. In the non-distributed active response case, each server will have 10 login attempts before creating its local blocking rule against the attacking host. The simulation results show that the architecture can achieve very good preemptive protection during the event of a normal dictionary attack.

5. Conclusion

This paper described a distributed active response architecture that provides preemptive protection against the common SSH dictionary attack. Simulation results show that high levels of preemptive protection can be achieved for the normal SSH dictionary attack method. This paper focused on the dictionary attack. However, due to the extensibility of the ADS module, preemptive protection can also be achieved for other classes of attacks. Our future work will focus on developing trust mechanisms such that inter-TDA policy sharing and collaboration can be established.

6. References

[1] S.M. Bellovin, "Distributed Firewalls", *;login:*, pp. 37-47, 1999.

[2] W.R. Cheswick, S.M. Bellovin, and A.D. Rubin, *Firewalls and Internet Security, 2nd Edition*, Addison-Wesly, Boston, MA, 2003.

[3] D. Ramsbrock, R. Berthier, and M. Cukier, "Profiling Attacker Behavior Following SSH Compromises", *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007.

[4] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for Intrusion Detection and Response", *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, 2000.

[5] C. Seifert, "Analyzing Malicious SSH Login Attempts", Technical Report, http://www.securityfocus.com/infocus/1876, 2006.