On the Application of Anomaly Detection in Reliable Server Pooling Systems for Improved Robustness against Denial of Service Attacks

Pascal Schöttle, Thomas Dreibholz, Erwin P. Rathgeb
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany

***Abstract*— The Reliable Server Pooling (RSerPool) architecture is the IETF's upcoming standard of a lightweight server redundancy and session failover framework for availability-critical applications. RSerPool combines the ideas from different research areas into a single, resource-efficient and unified architecture. Although there have already been a number of research papers on the pool management, load distribution and failover handling performance of RSerPool, the robustness against intentional attacks has not been intensively addressed yet.**

Therefore, the first goal of this paper is to provide a robustness analysis in order to outline the attack bandwidth necessary for a significant impact on RSerPool-based services. After that, we present our anomaly detection approach that has been designed to protect RSerPool systems against attacks. We also show the effectiveness of this approach by simulations.[1]

Keywords: Reliable Server Pooling, Anomaly Detection, Attacks, Robustness, Denial of Service

I. Introduction and Scope

Reliable Server Pooling (RSerPool) denotes the IETF's generic, application-independent framework for server pool [1] and session management [2]. While there have already been a number of publications on the performance of RSerPool for load balancing [3]–[5] and server failure handling [6], there has been very little research on security and robustness. Until now, only some simple thresholds to avoid flooding the pool management with misinformation have been analysed in [7]. The underlying transport protocol SCTP[2] already provides protection against blind flooding attacks [9] and the Internet Draft [10] of RSerPool mandatorily requires applying mechanisms like TLS [11] or IPSEC [12] in order to ensure authenticity, integrity and confidentiality. Nevertheless, these techniques are still insufficient: in a distributed system, there is always a chance that an attacker compromises a legitimate component (e.g. by exploiting a software bug) and obtains the private key. It is therefore important to analyse the behaviour of the RSerPool protocols under attack situations.

The goal of this paper is to first analyse the robustness of RSerPool systems against a denial of service (DoS) attack by compromised components and to show the impact of different attack scenarios on the application performance. Using these analyses as a baseline performance level, we will present a counter-measure approach based on anomaly detection techniques to efficiently reduce the impact of such attacks.

[1]Parts of this work have been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

[2]Stream Control Transmission Protocol, see [8].

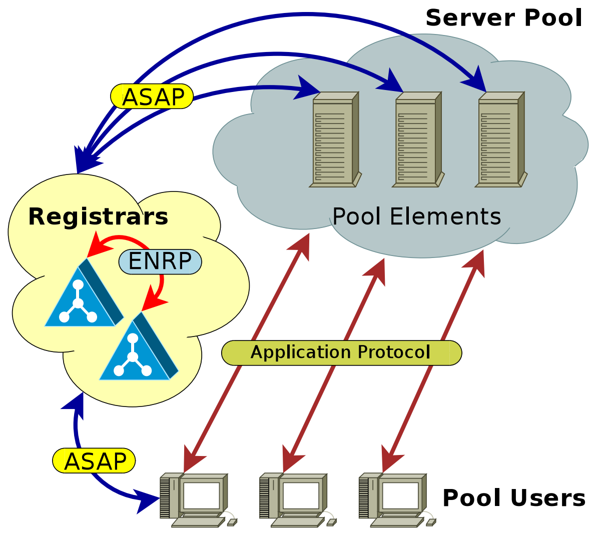

Fig. 1. The RSerPool Architecture

II. The RSerPool Architecture

Figure 1 illustrates the RSerPool architecture [13], [14] which consists of three types of components: servers of a pool are called *pool elements* (PE), a client is denoted as *pool user* (PU). The *handlespace* – which is the set of all pools – is managed by redundant *pool registrars* (PR). Within the handlespace, each pool is identified by a unique *pool handle* (PH).

A. Components and Protocols of RSerPool

PRs of an *operation scope* synchronize their view of the handlespace by using the Endpoint haNdlespace Redundancy Protocol (ENRP) [15], transported via SCTP [16]–[19] and secured e.g. by TLS [11] or IPSEC [12]. In contrast to GRID computing [20], an operation scope is restricted to a single administrative domain. That is, all of its components are under the control of the same authority (e.g. a company or an organization). This property results in a small management overhead [1], [21], which also allows for RSerPool usage on devices providing only limited memory and CPU resources (e.g. embedded systems like telecommunications equipment or routers). Nevertheless, PEs may be distributed globally for their service to survive localized disasters [22].

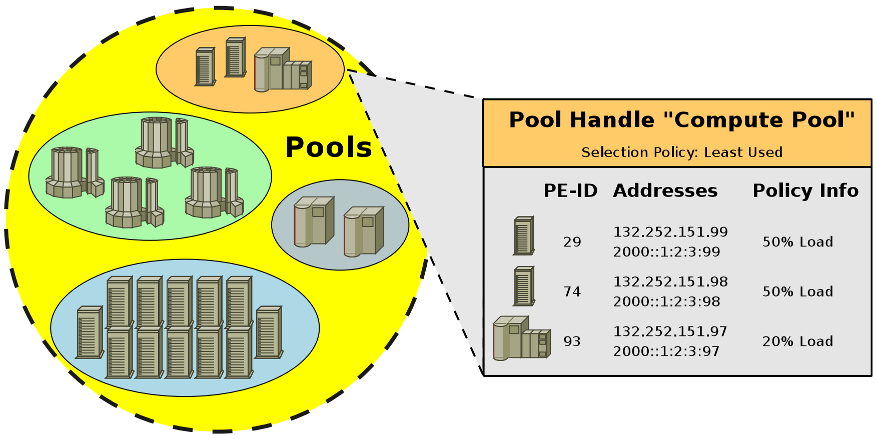
Pools
Pool Handle "Compute Pool"
Selection Policy: Least Used
PE-ID Addresses Policy Info
29 132.252.151.99 2000::1:2:3:99 50% Load
74 132.252.151.98 2000::1:2:3:98 50% Load
93 132.252.151.97 2000::1:2:3:97 20% Load

Fig. 2. A Handlespace Example

PEs choose an arbitrary PR of the operation scope to register into a pool by using the Aggregate Server Access Protocol (ASAP) [23], again transported via SCTP and using TLS or IPSEC. Within its pool, a PE is characterized by its PE ID, which is a randomly chosen 32-bit number. Upon registration at a PR, the chosen PR becomes the Home-PR (PR-H) of the newly registered PE. A PR-H is responsible for monitoring its PEs' availability by keep-alive messages (to be acknowledged by the PE within a given timeout) and propagates the information about its PEs to the other PRs of the operation scope via ENRP updates. PEs re-register regularly (in an interval denoted as *registration lifetime*) and for information updates.

In order to access the service of a pool given by its PH, a PU requests a PE selection from an arbitrary PR of the operation scope, again using ASAP. The PR selects the requested list of PE identities by applying a pool-specific selection rule, called *pool policy*. RSerPool supports two classes of load distribution policies: non-adaptive and adaptive algorithms [3]. While adaptive strategies base their assignment decisions on the current status of the processing elements (which of course requires up-to-date states), non-adaptive algorithms do not need such data. A basic set of adaptive and non-adaptive pool policies is defined in [24]. Relevant for this paper are the non-adaptive policies Round Robin (RR) and Random (RAND) as well as the adaptive policy Least Used (LU).

PUs may report unreachable PEs to a PR by using an ASAP Endpoint Unreachable message. A PR locally counts these reports for each PE. If the threshold MaxBadPEReports [6] is reached, the PR may decide to remove the PE from the handlespace. The counter of a PE is reset upon its re-registration.

An example handlespace consisting of four pools is illustrated in figure 2. The pool using the PH "Compute Pool" consists of 3 dual-homed PEs (IPv4 and IPv6). Since the pool policy is LU, the handlespace also stores the latest known load state of each PE in this pool.

B. Application Scenarios for RSerPool

While the initial motivation of RSerPool has been the availability of SS7 (Signalling System No. 7 [25]) services over IP networks, it has been designed for application independence. Current research on applicability and performance of RSerPool includes application scenarios like VoIP with SIP [26], SCTP-based mobility [27], web server pools [14], e-commerce systems [2], video on demand [28], battlefield networks [29], IP Flow Information Export (IPFIX) [30] and workload distribution [3], [14], [31].

A generic application model for RSerPool systems has been introduced by [3], including performance metrics for the provider side (pool utilization) and user side (request handling speed). Based on this model, the load balancing quality of different pool policies has been evaluated [3]–[5], [14], [22], [32].

III. Anomaly Detection

The term *anomaly detection* [33] denotes the continuous monitoring of network traffic or user behaviour and the checking for deviation from "normality". It bases on the assumption that attackers behave different from regular users and therefore can be identified as such. Unlike traditional anti-threat applications like Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), anomaly detection is able to identify attack scenarios that have never occurred before – because malicious systems do not behave like normal network elements.

Anomaly detection consists of two phases: in the *monitoring phase*, network traffic or user behaviour is monitored and learned as a baseline "normal" system behaviour. The second phase – denoted as *detection phase* – is the comparison of the actual network traffic or user behaviour with the baseline. In case of significant differences from the normal state, an attack is assumed and an alarm or counter-measure procedure is triggered.

For a good determination whether a monitored behaviour is normal or anomalous, thresholds have to be defined after the monitoring-phase. There are two ways of specifying these thresholds. First, there are the *static thresholds*, which do not change once they are defined. The second kind of thresholds are *dynamic thresholds*. They adapt themselves to usual changes which may occur in the system (e.g. added or removed components or new applications).

IV. Quantifying an RSerPool System

For our quantitative performance analysis, we use the application model from [3], [14]: the service provider side of an RSerPool system consists of a pool of PEs. Each PE has a request handling *capacity*, which we define in the abstract unit of calculations per second[3]. Each request consumes a certain number of calculations; we call this number *request size*. A PE can handle multiple requests simultaneously – in a processor sharing mode as provided by multitasking operating systems.

On the service user side, there is a set of PUs. The number of PUs can be given by the ratio between PUs and PEs (*PU:PE ratio*), which defines the parallelism of the request handling. Each PU generates a new request in an interval denoted as *request interval*. The requests are queued and sequentially assigned to PEs.

The total delay for handling a request d_{Handling} is defined as the sum of queuing delay d_{Queuing}, startup delay d_{Startup} (dequeuing until reception of acceptance acknowledgement) and processing time $d_{\text{Processing}}$ (acceptance until finish):

$$d_{\text{Handling}} = d_{\text{Queuing}} + d_{\text{Startup}} + d_{\text{Processing}}. \qquad (1)$$

That is, d_{Handling} not only incorporates the time required for processing the request, but also the latencies of queuing, server selection and message transport. The *handling speed* is defined as: $\text{handlingSpeed} = \frac{\text{requestSize}}{d_{\text{handling}}}$. For convenience reasons, the handling speed (in calculations/s) is represented in % of the average PE capacity. Clearly, the user-side performance metric is the handling speed – which should be as fast as possible.

Using the definitions above, it is possible to delineate the average system utilization (for a pool of NumPEs servers and a total pool capacity of PoolCapacity) as:

$$\text{systemUtil} = \text{NumPEs} * \text{puToPERatio} * \frac{\frac{\text{reqSize}}{\text{reqInterval}}}{\text{PoolCapacity}}. \qquad (2)$$

Obviously, the provider-side performance metric is the system utilization, since only utilized servers gain revenue. In practise, a well-designed client/server system is dimensioned for a certain *target system utilization* of e.g. 80%. That is, by setting any two of the parameters (PU:PE ratio, request interval and request size), the value of the third one can be calculated using equation 2 (see [3], [14] for details).

V. The Simulation Setup

For our performance analysis, the RSerPool simulation model RSPSIM [3], [14], [31] has been used. This model is based on the OMNeT++ [34] simulation environment and contains the protocols ASAP [23] and ENRP [15], a PR module, an attacker module and PE as well as PU modules for the request handling scenario defined in section IV. Network latency is introduced by link delays only. Therefore, only the network delay is significant. The latency of the pool management by PRs is negligible [1].

[3]An application-specific view of capacity may be mapped to this definition, e.g. CPU cycles or memory usage.

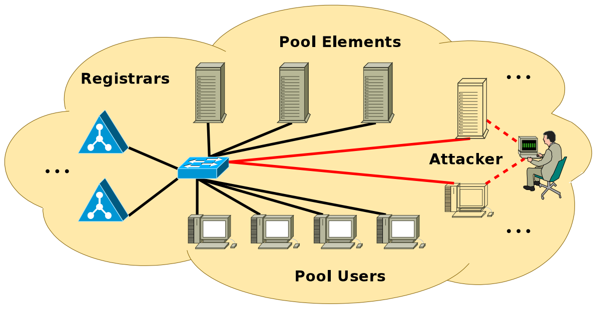

Fig. 3. The Simulation Setup

Unless otherwise specified, the basic simulation setup – which is also presented in figure 3 – uses the following parameter settings:

- The target system utilization is 80%. Request size and request interval are randomized using a negative exponential distribution (in order to provide a generic and application-independent analysis [3], [14]). There are 10 PEs; each one provides a capacity of 10^6 calculations/s.
- A PU:PE ratio of 10 is used (i.e. a non-critical setting as shown in [3]).
- We use request size:PE capacity settings between 1 and 100; i.e. being processed exclusively, the processing takes between 1s and 100s respectively – see also [3].
- There is a single PR only, since we do not examine PR failure scenarios here (see [3] for the impact of multiple PRs). PEs re-register every 30s (registration lifetime) and on every load change of the adaptive LU policy.
- MaxBadPEReports is set to 3 (default value defined in [15]). A PU sends an Endpoint Unreachable if a contacted PE fails to respond within 10s (see also [6]).
- The system is attacked by a single attacker node.
- The simulated real-time is 90min; each simulation run is repeated at least 15 times with a different seed in order to achieve statistical accuracy.

GNU R is used for the statistical post-processing of the results. Each resulting plot shows the average values and their 95% confidence intervals.

VI. Possible Attacks and Effects

The attack targets of RSerPool are the PRs, PEs and PUs. But due to the restriction of RSerPool to a single administrative domain, a protection of the small number of PRs is assumed to be feasible [7]. Instead, the most likely attack targets are the PEs and PUs. These components are significantly more numerous [14] and may be distributed over multiple, less controllable locations [22].

In the following, we will demonstrate that – without protection mechanisms – even a *single* compromised PE or PU is already able to cause a complete DoS.

A. A Compromised Pool Element

The most likely scenario of an attacker masquerading as a PE is to perform as many fake PE registrations

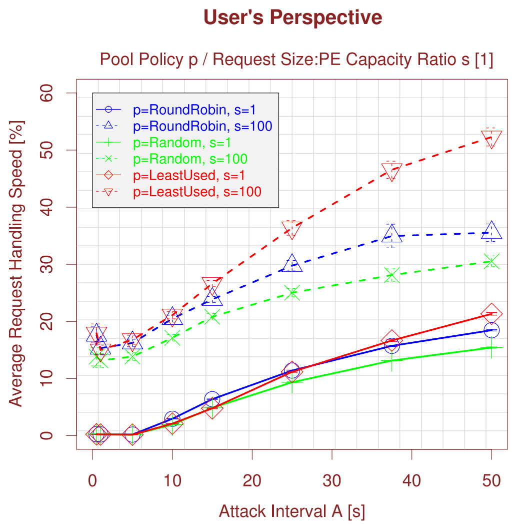

Fig. 4. The Impact of a Compromised Pool Element

as possible [7], [35]: each registration request sent to a PR simply has to contain another PE ID (randomly chosen), which makes it appear as registration of a new PE. Obviously, using an adaptive pool policy, the attacker will also set the policy information for the fake PE to be selected as often as possible (e.g. claiming a load of 0% for LU). For a non-adaptive policy, the attacker simply has to register as many fake PEs as possible, and the more registrations he can perform, the more likely is the choice of a fake PE.

In both cases, the impact of the attack depends on the attack interval A – which is the delay between two fake registrations. Figure 4 shows the handling speed for a single attacker performing registrations in the interval A (see section V for the other parameters). Obviously, the smaller the attack interval, the more fake entries go into the handlespace. This leads to an increased probability for a PU to select a non-existing PE and therefore to a decreased overall request handling speed. It takes only a single registration every second (i.e. A=1 for a request size:PE capacity ratio of s=1) to cause a handling speed of nearly 0% – which means a complete DoS. Clearly, the results are worse with multiple attackers.

B. A Compromised Pool User

An attacker masquerading as a PU has two possibilities to degrade a pool's service [7], [35]: First, it can flood PRs with handle resolution requests. Since the handlespace management – and therefore the server selection procedure – can be realized very efficiently (as shown in [1], [21], [36]), such an attack requires a large attack bandwidth to cause problems. The second attack possibility is therefore much more likely: an attacker can report real PEs as being unreachable. Depending on the setting of $\mathrm{MaxBadPEReports}$ [6] (default is 3, see [15]), the PR will eventually remove the reported PE out of the handlespace. Using a sufficiently small attack interval, an attacker can clear the whole handlespace from PEs and cause a complete DoS.

Figure 5 presents the impact of an attacker masquerading as a PU: handle resolutions are performed in the attack interval A, and an unreachability report is sent for the selected PE at a probability u. The left-hand side shows the variation of A, the right-hand side the variation of u for A=0.1. Even without sending unreachability reports (i.e. u=0%), there is an impact on the performance of RR. The reason for this effect is the "stateful" [3] operation of RR: the round robin pointer is advanced by the selection procedure itself (i.e. without actually using a PE), causing the selection of less appropriate PEs. On the other hand, LU and RAND are "stateless" policies and not affected by such an attack.

However, the impact of also reporting all PEs as being unreachable is disastrous: PEs are kicked out of the handlespace, and the handling speed quickly sinks. This effect is even emphasised by a larger PE lifetime L (see the right-hand plot): the later PEs re-register (and therefore re-appear in the handlespace), the worse the pool's performance. Because an attacker's goal is to cause the highest performance degradation, unreachability reports are clearly the most probable threat scenario for PU-based attacks.

VII. Our Approach for Anomaly Detection

As mentioned in section VI, we assume the PRs to be relatively safe against attacks and therefore design our anomaly detection for protection against PE and PU-based attacks as part of the PRs. First, we have to distinguish between PE attacks and PU attacks.

A. Concept

Therefore, we introduce three new lists for the handlespace management: the first list stores the source addresses (not the PE IDs!) of all PEs registering at the PR. Each entry contains a counter which increments on each registration or re-registration from the same source address. The second list records all PUs with their addresses and number of handle resolution requests. In the third list, PU source addresses and their corresponding number of unreachability reports are stored. It is important to note that address spoofing is already avoided by the SCTP protocol [9], [17], [18], [37]. So, we can assume that an attacker has not more than a few usable source addresses – since RSerPool is deployed into a single administrative domain. As already shown in section VI, the attack interval is the most important factor of an attack. Therefore, comparing the lists contents of an attacker to a normal component, we can assume a significant difference.

For interpretation of the list contents, it is necessary to define a time window interval l for the anomaly detection, after which the lists are periodically analysed and reset. The time window l has to be long enough for the list contents to be meaningful, but also not too long for an attack to remain undetected for some time. Resetting the list each period allows components to use the pool again after cleaning from an infestation.

We now differntiate the attack scenarios into two categories:

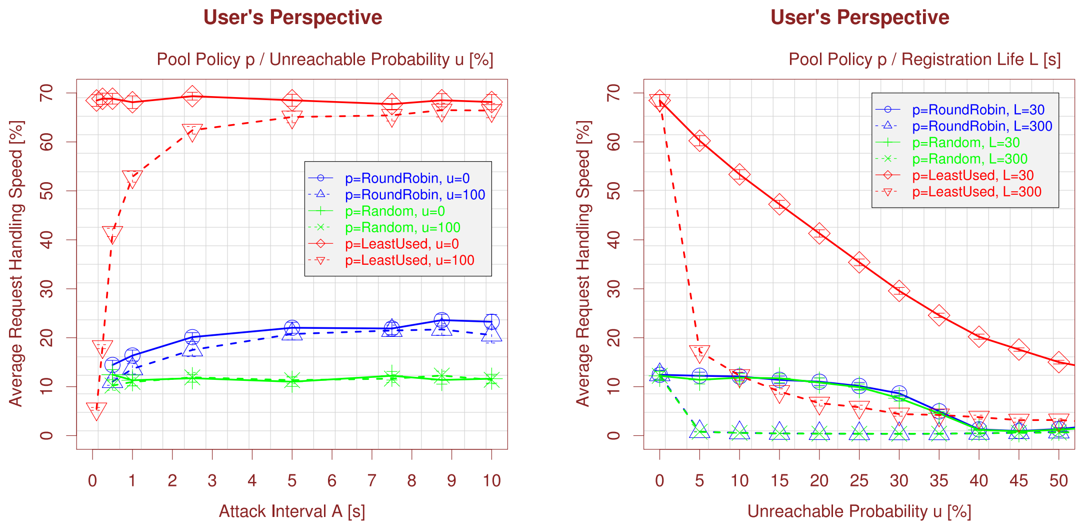

Fig. 5. The Impact of a Compromised Pool User

1) PE attacks with non-adaptive policy and
2) PE attacks with adaptive policy or PU attacks.

As described in section III, there are static and dynamic thresholds. While static thresholds are useful for the first category, dynamic ones are useful for the second. In the following subsections, we will introduce our anomaly detection approach for the two categories.

B. Pool Element with Non-Adaptive Policy

Using a non-adaptive policy, a PE registers in a fixed re-registration interval r at its PR-H. Except for the case of a PR-H failure (which is seldom), there is no need to re-register more frequently. Therefore, for non-adaptive policies, we can simply calculate an upper threshold S:

$$\text{upper threshold } S = \frac{\text{time window } l}{\text{re-registration interval } r}.$$

By slightly increasing the actual setting of S, PR-H failures will also be covered.

Choosing a different PE ID for each registration, an observation of the per-ID behaviour would not be suspicious. But since we look for the source address (which cannot change arbitrarily in a controlled environment), the attacker's behaviour is definitely different from the behaviour of the real PEs – unless his attack interval is equal to the re-registration interval r (which would be clearly ineffective, though).

C. Pool Element with Adaptive Policy or Pool User

For a PE with an adaptive policy or a PU, it is not possible to specify an universal upper threshold for re-registrations or handle resolutions. Both operations strongly depend on the current application requirements. Therefore, we apply a statistical analysis of the saved values for our counters. In different tests, we have observed that attackers in a pool with adaptive policy do not need to re-register more frequently than regular PEs. Instead, using the right policy value (e.g. a load of 0% for LU), an attack is already effective with significantly less re-registrations than the interval r. Therefore, we span a confidence interval K with an upper and a lower threshold. PEs are considered to be attackers when their counter is outside of this range. Analogously, we handle PUs and their handle resolutions.

We have considered two statistical measures for locating the middle of the upper and lower thresholds: the *mean* and the *median*. The mean $\bar{X}_n$ of a sample $X_n = \{x_1, x_2, \ldots, x_n\}$ is defined as:

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \ldots + x_n}{n}.$$

However, the mean is not appropriate for anomaly detection: as described in [38], it has a *breakdown point* of $\frac{1}{n}$. The breakdown point was introduced to measure the resilience of a statistical measure with respect to outliers. A breakdown point of $\frac{1}{n}$ means that if only one value in the sample runs towards infinity (in our case: the counter of an attacker node), the mean runs towards infinity as well. Therefore, a single attacker is already able to displace the mean value – and *regular* nodes fall outside of the confidence interval.

The median has a breakdown point of $\frac{1}{2}$, i.e. more than half of the sample's values have to be outliers (in our case: attackers) to affect the value of the median. This is indeed the best possible value for the breakdown point of a statistical measurement, as shown by [39].

The median $\mathrm{med}(X_n)$ for a given sample of values $X_n = \{x_1, x_2, \ldots, x_n\}$ is defined as:

$$\mathrm{med}(X_n) = \begin{cases} x_{\frac{n+1}{2}} & \text{if n is odd} \\ \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}) & \text{if n is even} \end{cases}$$

Using the median as an appropriate value for what is approximately "normal", we have to span the confidence interval. In statistics besides the measures of location, there are measures of dispersion of a sample of values. Directly linked to the median is the MAD (Median Absolute Deviation). The MAD $\mathrm{MAD}(X_n)$ of a sample of values $X_n = \{x_1, x_2, \ldots, x_n\}$ is defined as:

$$\mathrm{MAD}(X_n) = \mathrm{med}(|x_1 - \mathrm{med}(X_n)|, \ldots, |x_n - \mathrm{med}(X_n)|).$$

We have chosen the median as measurement for the location of the saved counters and the MAD as measurement of their dispersion. Therefore, these two values span the confidence interval K with lower value K_- and upper value K_+:

$$\begin{aligned} K_- &= \mathrm{med}(x_1, \ldots, x_n) - p_- * \mathrm{MAD}(x_1, \ldots, x_n) \\ K_+ &= \mathrm{med}(x_1, \ldots, x_n) + p_+ * \mathrm{MAD}(x_1, \ldots, x_n) \end{aligned}$$

The two parameters p_- and p_+ define how wide the confidence interval is spanned up and down.

VIII. Dealing with Anomalies

Using the threshold settings defined in subsection VII-B and subsection VII-C, PEs and PUs falling below or exceeding their counter threshold are assumed to be attackers. In order to handle such components, we introduce two more lists: the *suspect PE list* and the *suspect PU list*. Requests (registration or handle resolution) from components on these lists are simply ignored by the PR during the next period of the time window.

A. Dealing with a Suspect Pool Element

Upon start of the next time window period, a PR de-registers all PEs on the suspect PE list. Furthermore, upon each registration of a PE, the PR checks whether the PE address of the incoming request is on the suspect PE list. If a PE is suspect, its registration requests are ignored, but "success" is reported back to the PE. There are two reasons for reporting a successful registration to the PE:

1) The attacker should not get an information of being blacklisted and
2) A real PE, which is mistakenly on the list as a false positive, would – without successful registration – assume a PR problem and try again at another PR.

When a real PE gets blacklisted, it will remain on the list only for one period: as long as it is on the list, it is not selected for PUs. Therefore, the re-registration interval should normalize again and the PE will be removed from the suspect PE list for the next period. However, it remains blacklisted if the anomaly persists.

B. Dealing with a Suspect Pool User

The handlespace itself does not maintain a list of PUs, i.e. there is no need to de-register suspect PUs. Instead, a suspect PU only gets an empty list of PEs upon its handle resolution request. A real PU, misidentified as attacker, will then assume that the pool is currently empty and retry after some delay h (see also [14], [32]). Due to its reduced request rate, it will be removed from the suspect PU list in the next period. However, an attacker would not care for the reply and keep sending handle resolution requests. Therefore, we introduce a new parameter $\mathrm{minPURate}$, which is defined as:

$$\mathrm{minPURate} = \frac{\text{time window } l}{\text{handle resolution retry timer } h}.$$

At the end of each anomaly detection time window, the counters on the suspect PU list are checked against $\mathrm{minPURate}$. If the counter is higher, the PU remains on the list. Otherwise, it is removed.

The second target of PU-based attackers are the unreachability reports. A real PU will not send such reports if it gets an empty list of PEs upon handle resolution requests: if there are no PEs, how should one be unreachable? Therefore, a PU on the suspect PU list sending unreachability reports (for a list of previously recorded PEs!) is definitely an attacker. Utilizing this fact, we can make sure that a real PU will be removed from the list in the next period – while keeping attackers blacklisted.

Another behaviour which indicates an attacker is the composition of the pool. If there are a lot of PUs (e.g. some thousands) and only a few PEs (e.g. a dozen), an attack can be assumed when only a small subset of the PUs reports unreachable PEs. Therefore, such PUs are also put on the list of suspect PUs as a precaution.

IX. Performance Evaluation

In order to evaluate the performance of our anomaly detection approach, we have set up scenarios similar to the attack simulations in section VI, but with anomaly detection activated. The anomaly detection window has been set to l=5min, which has turned out to be a reasonable value. We use these attack scenarios as baseline and show how the system performance is improved. Furthermore, we also increase the number of attackers.

A. Avoiding Pool Element Attacks

The left-hand side of figure 6 presents the handling speed results for varying the number of attackers from 0 to 5 at an attack interval A=2 for request size:PE capacity ratios s=1 and s=100. Unlike the results in subsection VI-A – where even a single attacker was able to cause a complete denial of service – the performance difference between no attack and 5 attackers is minimal. Using 3 attackers and varying the attack interval A (shown on the right-hand side of figure 6), also only very small performance degradations can be observed at very short attack intervals. In particular, the handling speed does not even come close to a DoS. That is, our evaluation shows that our PE anomaly detection approach – for adaptive policies (here: LU) and non-adaptive policies (here: RR, RAND) – is of significant benefit in attack situations.

B. Avoiding Pool User Attacks

Handling speed results for PU attacks are presented in figure 7. The left-hand plot shows the results for increasing the number of attackers from 0 to 5 for never sending unreachability reports (u=0%) and always sending them (u=100%) for a PE lifetime of L=90s. On a pure handle resolution attack (i.e. u=0%), no service degradation can be observed. Unreachability reports (here: u=100%) significantly affect LU only. To explain this effect, the right-hand side of figure 7 presents the handling speed for varying the the probability to report a PE unreachability u for 2 attackers and an attack interval of A=1s. While the impact of the attack is smaller for RR

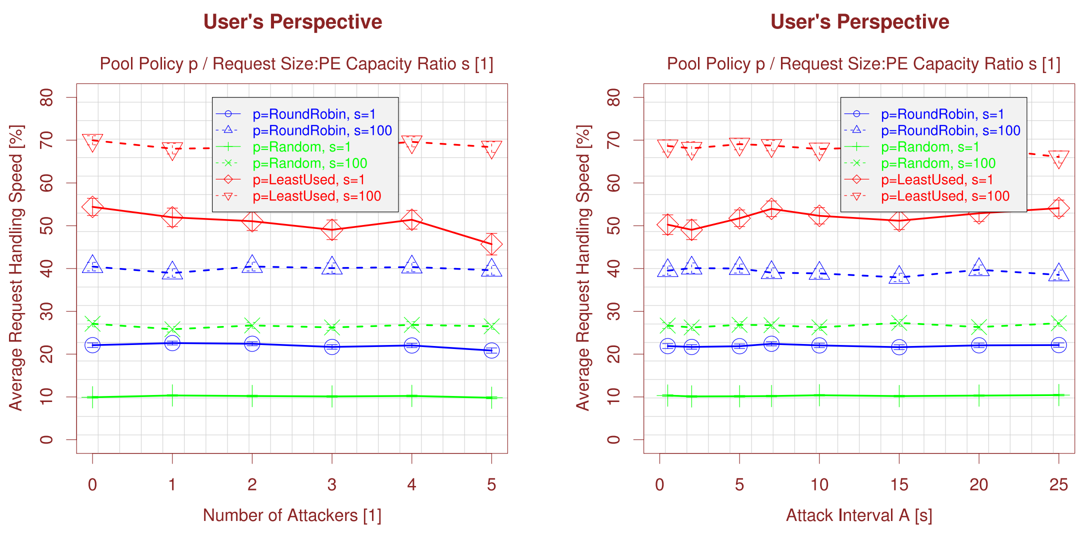

Fig. 6. Effects of the Anomaly Detection against the Pool Element Attack

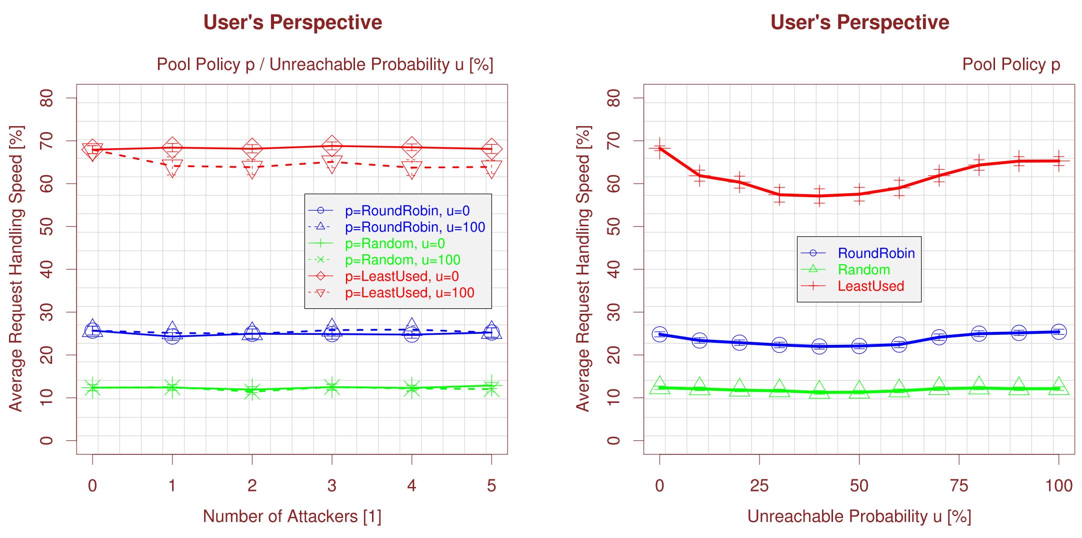

Fig. 7. Effects of the Anomaly Detection against the Pool User Attack

and RAND, unreachability reports for LU target the least-loaded PE, which is obviously the best choice for a new request. Therefore, the performance decreases for an increasing report probability (since the least-loaded PE gets kicked out of the pool) – until there is a sufficient report rate that triggers the anomaly detection which blacklists the attacker. Clearly, this occurs earlier for higher u, resulting in an again increasing handling speed. However, in comparison to the complete DoS for only a single attacker without anomaly detection (see subsection VI-B), even the worst performance degradation with anomaly detection is quite small. That is, our anomaly detection approach also achieves a significant benefit in PU-based attack scenarios.

X. Conclusions

The goal of this paper has been the application of anomaly detection to protect RSerPool systems against two important attack scenarios:

- PE-based attacks (registration) and
- PU-based attacks (handle resolution/failure report).

Without any protection, we have shown that even a single attacker is able to cause a complete DoS. Therefore, we have designed and described our anomaly detection approach, using statistical methods to define lower and upper thresholds for certain pool management operations. In simulations using our RSerPool simulation model RSPSIM, we have shown that our approach is effectively protecting RSerPool setups against the PE/PU-based attacks.

As part of our future work, we consider a similar anomaly detection approach for the protection against PR-based attacks. Furthermore, we are also going to evaluate our approach in the reality, using our RSerPool prototype implementation RSPLIB [14], [40] in large-scale Internet setups – based on the PlanetLab [22].

References

[1] T. Dreibholz and E. P. Rathgeb, "An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems," *SERSC International Journal on Hybrid Information Technology (IJHIT)*, vol. 1, no. 2, pp. 17–32, Apr. 2008.

[2] T. Dreibholz, "An Efficient Approach for State Sharing in Server Pools," in *Proceedings of the 27th IEEE Local Computer Networks Conference (LCN)*, Tampa, Florida/U.S.A., Oct. 2002, pp. 348–352, ISBN 0-7695-1591-6.

[3] T. Dreibholz and E. P. Rathgeb, "On the Performance of Reliable Server Pooling Systems," in *Proceedings of the IEEE Conference on Local Computer Networks (LCN) 30th Anniversary*, Sydney/Australia, Nov. 2005, pp. 200–208, ISBN 0-7695-2421-4.

[4] T. Dreibholz, X. Zhou, and E. P. Rathgeb, "A Performance Evaluation of RSerPool Server Selection Policies in Varying Heterogeneous Capacity Scenarios," in *Proceedings of the 33rd IEEE EuroMirco Conference on Software Engineering and Advanced Applications*, Lübeck/Germany, Aug. 2007, pp. 157–164, ISBN 0-7695-2977-1.

[5] X. Zhou, T. Dreibholz, and E. P. Rathgeb, "Improving the Load Balancing Performance of Reliable Server Pooling in Heterogeneous Capacity Environments," in *Proceedings of the 3rd Asian Internet Engineering Conference (AINTEC)*, ser. Lecture Notes in Computer Science, vol. 4866. Springer, Nov. 2007, pp. 125–140, ISBN 978-3-540-76808-1.

[6] T. Dreibholz and E. P. Rathgeb, "Reliable Server Pooling – A Novel IETF Architecture for Availability-Sensitive Services," in *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)*, Sainte Luce/Martinique, Feb. 2008, pp. 150–156, ISBN 978-0-7695-3087-1.

[7] T. Dreibholz, E. P. Rathgeb, and X. Zhou, "On Robustness and Countermeasures of Reliable Server Pooling Systems against Denial of Service Attacks," in *Proceedings of the IFIP Networking*, Singapore, May 2008, pp. 586–598, ISBN 978-3-540-79548-3.

[8] R. Stewart, "Stream Control Transmission Protocol," IETF, Standards Track RFC 4960, Sept. 2007.

[9] E. Unurkhaan, "Secure End-to-End Transport - A new security extension for SCTP," Ph.D. dissertation, University of Duisburg-Essen, Institute for Experimental Mathematics, July 2005.

[10] M. Stillman, R. Gopal, E. Guttman, M. Holdrege, and S. Sengodan, "Threats Introduced by RSerPool and Requirements for Security," IETF, RSerPool Working Group, Internet-Draft Version 15, July 2008, draft-ietf-rserpool-threats-15.txt.

[11] A. Jungmaier, E. Rescorla, and M. Tüxen, "Transport Layer Security over Stream Control Transmission Protocol," IETF, Standards Track RFC 3436, Dec. 2002.

[12] S. Bellovin, J. Ioannidi, A. Keromytis, and R. Stewart, "On the Use of Stream Control Transmission Protocol (SCTP) with IPsec," IETF, Standards Track RFC 3554, July 2003.

[13] P. Lei, L. Ong, M. Tüxen, and T. Dreibholz, "An Overview of Reliable Server Pooling Protocols," IETF, RSerPool Working Group, Internet-Draft Version 06, May 2008, draft-ietf-rserpool-overview-06.txt, work in progress.

[14] T. Dreibholz, "Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture," Ph.D. dissertation, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, Mar. 2007.

[15] Q. Xie, R. Stewart, M. Stillman, M. Tüxen, and A. Silverton, "Endpoint Handlespace Redundancy Protocol (ENRP)," IETF, RSerPool Working Group, Internet-Draft Version 21, July 2008, draft-ietf-rserpool-enrp-21.txt, work in progress.

[16] A. Jungmaier, E. P. Rathgeb, and M. Tüxen, "On the Use of SCTP in Failover-Scenarios," in *Proceedings of the State Coverage Initiatives, Mobile/Wireless Computing and Communication Systems II*, vol. X, Orlando, Florida/U.S.A., July 2002, ISBN 980-07-8150-1.

[17] C. Hohendorf, E. P. Rathgeb, E. Unurkhaan, and M. Tüxen, "Secure End-to-End Transport Over SCTP," *Journal of Computers*, vol. 2, no. 4, pp. 31–40, June 2007.

[18] ——, "Secure End-to-End Transport Over SCTP," in *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, ser. Lecture Notes in Computer Science, vol. 3995. Springer, 2006, pp. 381–395.

[19] A. Jungmaier, "Das Transportprotokoll SCTP," Ph.D. dissertation, Universität Duisburg-Essen, Institut für Experimentelle Mathematik, Aug. 2005.

[20] I. Foster, "What is the Grid? A Three Point Checklist," *GRID Today*, July 2002.

[21] T. Dreibholz and E. P. Rathgeb, "An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems," in *Proceedings of the IEEE International Conference on Future Generation Communication and Networking (FGCN)*, vol. 1, Jeju Island/South Korea, Dec. 2007, pp. 136–143, ISBN 0-7695-3048-6.

[22] ——, "On Improving the Performance of Reliable Server Pooling Systems for Distance-Sensitive Distributed Applications," in *Proceedings of the 15. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, Bern/Switzerland, Feb. 2007, pp. 39–50, ISBN 978-3-540-69962-0.

[23] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen, "Aggregate Server Access Protcol (ASAP)," IETF, RSerPool Working Group, Internet-Draft Version 21, July 2008, draft-ietf-rserpool-asap-21.txt, work in progress.

[24] T. Dreibholz and M. Tüxen, "Reliable Server Pooling Policies," IETF, RSerPool Working Group, Internet-Draft Version 10, July 2008, draft-ietf-rserpool-policies-10.txt, work in progress.

[25] ITU-T, "Introduction to CCITT Signalling System No. 7," International Telecommunication Union, Tech. Rep. Recommendation Q.700, Mar. 1993.

[26] P. Conrad, A. Jungmaier, C. Ross, W.-C. Sim, and M. Tüxen, "Reliable IP Telephony Applications with SIP using RSerPool," in *Proceedings of the State Coverage Initiatives, Mobile/Wireless Computing and Communication Systems II*, vol. X, Orlando, Florida/U.S.A., July 2002, ISBN 980-07-8150-1.

[27] T. Dreibholz, A. Jungmaier, and M. Tüxen, "A new Scheme for IP-based Internet Mobility," in *Proceedings of the 28th IEEE Local Computer Networks Conference (LCN)*, Königswinter/Germany, Nov. 2003, pp. 99–108, ISBN 0-7695-2037-5.

[28] A. Maharana and G. N. Rathna, "Fault-tolerant Video on Demand in RSerPool Architecture," in *Proceedings of the International Conference on Advanced Computing and Communications (ADCOM)*, Bangalore/India, Dec. 2006, pp. 534–539, ISBN 1-4244-0716-8.

[29] Ü. Uyar, J. Zheng, M. A. Fecko, S. Samtani, and P. Conrad, "Evaluation of Architectures for Reliable Server Pooling in Wired and Wireless Environments," *IEEE JSAC Special Issue on Recent Advances in Service Overlay Networks*, vol. 22, no. 1, pp. 164–175, 2004.

[30] T. Dreibholz, L. Coene, and P. Conrad, "Reliable Server Pooling Applicability for IP Flow Information Exchange," IETF, Individual Submission, Internet-Draft Version 06, July 2008, draft-coene-rserpool-applic-ipfix-06.txt, work in progress.

[31] T. Dreibholz and E. P. Rathgeb, "A Powerful Tool-Chain for Setup, Distributed Processing, Analysis and Debugging of OMNeT++ Simulations," in *Proceedings of the 1st ACM/ICST OMNeT++ Workshop*, Marseille/France, Mar. 2008, ISBN 978-963-9799-20-2.

[32] X. Zhou, T. Dreibholz, and E. P. Rathgeb, "A New Approach of Performance Improvement for Server Selection in Reliable Server Pooling Systems," in *Proceedings of the 15th IEEE International Conference on Advanced Computing and Communication (ADCOM)*, Guwahati/India, Dec. 2007, pp. 117–121, ISBN 0-7695-3059-1.

[33] A. Sundaram, "An Introduction to Intrusion Detection," *Crossroads*, vol. 2, no. 4, pp. 3–7, 1996.

[34] A. Varga, *OMNeT++ Discrete Event Simulation System User Manual - Version 3.2*, Technical University of Budapest/Hungary, Mar. 2005.

[35] P. Schöttle, "Einsatz von Anomalieerkennung in Reliable-Server-Pooling-Systemen zur Verbesserung der Robustheit gegen Denial-of-Service-Angriffe," Universität Duisburg-Essen, Institut für Experimentelle Mathematik," Bachelor's thesis, Mar. 2008.

[36] C. S. Chandrashekaran, W. L. Johnson, and A. Lele, "Method using Modified Chord Algorithm to Balance Pool Element Ownership among Registrars in a Reliable Server Pooling Architecture," in *Proceedings of the 2nd International Conference on Communication Systems Software and Middleware (COMSWARE)*, Bangalore/India, Jan. 2007, pp. 1–7, ISBN 1-4244-0614-5.

[37] E. Unurkhaan, E. P. Rathgeb, and A. Jungmaier, "Secure SCTP - A Versatile Secure Transport Protocol," *Telecommunication Systems*, vol. 27, no. 2-4, pp. 273–296, 2004.

[38] D. L. Donoho and P. J. Huber, "The notion of breakdown point," in *A Festschrift for Erich L. Lehmann*, P. J. Bickel, K. A. Doksum, and J. J. L. Hodges, Eds. Belmont, California: Wadswort, Feb. 1983, pp. 157–184.

[39] P. L. Davies and U. Gather, "Breakdown and Groups," *The Annals of Statistics*, vol. 33, no. 3, pp. 977–1035, 2005.

[40] T. Dreibholz, "An Introduction to Reliable Server Pooling and the RSPLIB Implementation," in *Invited talk*, Haikou, Hainan/People's Republic of China, Dec. 2007.