About an Immune System Understanding for Cloud-native Applications

Biology Inspired Thoughts to Immunize the Cloud Forensic Trail

Nane Kratzke
Lübeck University of Applied Sciences, Germany
Center of Excellence for Communication, Systems and Applications (CoSA)
Email: nane.kratzke@fh-luebeck.de

***Abstract*—There is no such thing as an impenetrable system, although the penetration of systems does get harder from year to year. The median days that intruders remained undetected on victim systems dropped from 416 days in 2010 down to 99 in 2016. Perhaps because of that, a new trend in security breaches is to compromise the forensic trail to allow the intruder to remain undetected for longer in victim systems and to retain valuable footholds for as long as possible. This paper proposes an immune system inspired solution, which uses a more frequent regeneration of cloud application nodes to ensure that undetected compromised nodes can be purged. This makes it much harder for intruders to maintain a presence on victim systems. Basically, the biological concept of cell-regeneration is combined with the information systems concept of append-only logs. Evaluation experiments performed on popular cloud service infrastructures (Amazon Web Services, Google Compute Engine, Azure and OpenStack) have shown that between 6 and 40 nodes of elastic container platforms can be regenerated per hour. Even a large cluster of 400 nodes could be regenerated in somewhere between 9 and 66 hours. So, regeneration shows the potential to reduce the foothold of undetected intruders from months to just hours.**

***Keywords*–cloud computing; node regeneration; container platform; append-only log; forensic trail;**

I. Introduction

Cloud computing has become a great enabler for a variety of different IT-enabled business and service models. The ability to deploy new systems rapidly without concern for forward planning, accessing corporate budgets and the ability to scale up (or down) on demand has proved particularly attractive for a continuously rising number of companies and organizations. Many research studies and programs have been actively involved in trying to develop systems in a responsible way to ensure the security and privacy of users. But compliance with standards, audits and checklists, does not automatically equals security [1]. Furthermore, there is a fundamental issue remaining, which presents a serious challenge, and is of great concern. Once an attacker successfully breaches a cloud system and becomes an intruder, usually escalating privileges the longer they are in the system, there is nothing then to prevent them from deleting or modifying the forensic trail. Preventing this from happening presents a serious challenge, and in the light of forthcoming regulation from various countries, and of particular interest the forthcoming EU General Data Protection Regulation (GDPR), which has a regime of penalties which can rise up to the greater of €20 million or 4% of global turnover. The other challenging aspect of this legislation is that any security breach must be reported within 72 hours. While the original idea was to do this "within 72 hours of the occurrence of a breach", it has been somewhat watered down to read "within 72 hours of discovery of a breach". We believe this is a backward step, since there will be less incentive for firms to deal with the real problem, and instead will perhaps encourage some to delay "discovery" to suit their own agendas. For cloud users who are breached, particularly where the intruder deletes or modifies the forensic trail, the longer the intruder remains in the system, the more difficult it becomes to be able to properly report the full impact of the breach, which is also likely to lead to higher levels of fines.

In our recent research [2], we exploited successfully elastic container platforms and their "designed for failure" capabilities to realize transferability of cloud-native applications at runtime. By transferability, we mean that a cloud-native application can be moved from one (public or private) Infrastructure as a Service (IaaS) provider infrastructure to another without any downtime and therefore without being noticed by its users. These platforms are more and more used as distributed and elastic runtime environments for cloud-native applicatons [3].

Table I lists some elastic container platforms that gained a lot of interest in recent years. These platforms can be understood as a kind of cloud infrastructure unifying middleware for cloud-native applications [4]. These platforms can be even used to transfer cloud applications between different cloud service providers at runtime. We think that it should be possible to make use of the same features to immunize the forensic trail simply by moving an application within the same provider infrastructure. To move anything from A to A makes no sense at first glance. However, let us be paranoid and aware that with some probability and at a given time, an attacker will be successful and compromise at least one virtual machine. In these cases, a transfer from A to A would be an efficient counter measure – because the intruder immediately loses any hijacked machine that is moved. To understand that, the reader must know that our approach does not effectively move a machine, it regenerates it [2]. So, to move a machine means to launch a compensating machine unknown to the intruder and to terminate the former (hi-jacked) machine. So, whenever an application is moved (basically the hosting container platform is moved and not the application itself) all of its virtual machines are regenerated. And this would effectively eliminate undetected hi-jacked machines, as well as those which have not been compromised. The biological analogy of this strategy is called *"cell-regeneration"* and the attack on ill cells is coordinated by an immune system. This paper describes the first ideas for such a kind of immune system for cloud applications.

The paper will explain these basic and unconventional thoughts following this **outline**. To provide some context for the reader, Section II will explain the general lifecycle of a cyber attack and will show that two aspects have to be addressed to protect the forensic trail. First of all, it is assumed that every system can be penetrated due to unknown exploits [5]. Nevertheless, systems can be built which are capable of being able to regenerate penetrated nodes. That makes it harder for intruders to maintain a presence on penetrated systems over a longer time. Section III will summarize some of our recent research to explain how such systems can be built. However, even in a short amount of time, the forensic trail can be deleted or compromised. So, the task is to find a solution to store the forensic trail in such a way as to make it un-erasable and un-compromise-able. Section IV will propose a double logging architecture which exploits messaging systems like Kafka [6] to realize regenerating append-only logging systems which are recovering from penetrations. Section V shows some evaluation results measured from transferability experiments. These numbers are used to estimate possible regeneration intervals for systems of different sizes and to compare it with median dwell times reported by security companies over the last seven years (see Table II). The advantages and limitations of this proposal are related to other work in Section VI. We conclude our considerations in Section VII.

II. The Lifecycle of Cyber Attacks

Figure 1 shows the cyber attack life cycle model, which is used by the M-Trends reports [7] to report developments in cyber attacks over the years. According to this model an attacker passes through different stages to complete their mission. It starts with initial reconnaissance and compromising of access means (often using mobile phones or the desktop PCs of employees, who are generally not security experts). These steps are very often supported by social engineering methodologies [8] and phishing attacks [9]. The goal is to establish a foothold near the system of interest. All these steps are not covered by this paper, because the proposed technical solution is not able to harden the weakest point in security - the human being. We refer to corresponding research like [8] [9].

The following steps of this model are more interesting for this paper. According to the cyber attack life cycle model the attacker's goal is to escalate privileges to get access to the target system. Because this leaves trails on the system which could reveal a security breach, the attacker is motivated to compromise the forensic trail. According to the M-Trends 2017 report attackers make more and more use of counter-forensic measures to hide their presence and impair investigations. The report refers to batch scripts used by financial intruders to delete pre-fetch entries, clear Microsoft Windows event logs and securely delete arbitrary files. These batch scripts are run to hide the execution of malware that was scraping payment card information from memory. The technique is simple, but the intruders' knowledge of forensic artifacts demonstrate increased sophistication, as well as their intent to persist in the environment.

With a barely detectable foothold, the internal reconnaissance of the victim's network is carried out to allow the lateral movement to the target system. This is a complex and lengthy process and may even take weeks. So, infiltrated machines have worth for attackers and tend to be used for as long as possible, even after mission completion. Although the numbers are decreasing, Table II shows how astonishingly long a period on average an intruder has access to a victim system at the time of writing this paper. According to this reference model for cyber attacks two conclusions can be drawn.

(1) An undetected attacker should lose access to compromised nodes of the system as fast as possible. The Sections III and V will propose a solution on how the undetected days on a system can be reduced from months down to days, or even hours. Even if undetected days on systems can be reduced dramatically, it **(2) must be still impossible to compromise the forensic trail**. Otherwise intrusions might be short, but remain undetectable. Section IV will propose a solution on how to log the forensic trail using append-only logging systems (which could itself be compromised).

III. From Transferable to Regenerate-able Cloud Applications

Our recent research dealt with the question of how to design cloud-native applications that are transferable between different cloud service providers to reduce vendor lock-in situations. One aspect that can be learned from this is that there is no common understanding of what a cloud-native application really is. A kind of software that is *"intentionally designed for the cloud"* is an often heard but vacuous phrase. However, noteworthy similarities exist between various view points on *cloud-native applications* (CNA) [3]. A common approach is to define maturity levels in order to categorize different kinds of cloud applications. Table III shows a maturity model proposed by the *Open Data Center Alliance*. And common motivations for CNA architectures are fault isolation, fault tolerance, and automatic recovery to improve **safety**, and to enable horizontal (instead of vertical) application **scalability** [10]. Fehling et al. [11] proposed the IDEAL model for CNAs. A CNA should strive for an **isolated state**, is **distributed**, provides **elasticity** in a horizontal scaling way, and should be operated on **automated deployment machinery**. Finally, its components should be **loosely coupled**.

Balalaie et al. [13] stress that these properties are addressed by cloud-specific architecture and infrastructure approaches like **Microservices** [14], **API-based collaboration**, adaption of **cloud-focused patterns** [11], and **self-service elastic platforms** that are used to deploy and operate these microservices via self-contained deployment units (containers). These platforms provide additional operational capabilities on top of IaaS infrastructures like automated and on-demand scaling of

TABLE I. SOME POPULAR OPEN SOURCE ELASTIC PLATFORMS

Platform	Contributors	URL
Kubernetes	Cloud Native Found.	http://kubernetes.io (initiated by Google)
Swarm	Docker	https://docker.io
Mesos	Apache	http://mesos.apache.org/
Nomad	Hashicorp	https://nomadproject.io/

TABLE II. UNDETECTED DAYS ON VICTIM SYSTEMS [7]

Year	External notification	Internal discovery	Median
2010	-	-	416
2011	-	-	?
2012	-	-	243
2013	-	-	229
2014	-	-	205
2015	320	56	146
2016	107	80	99

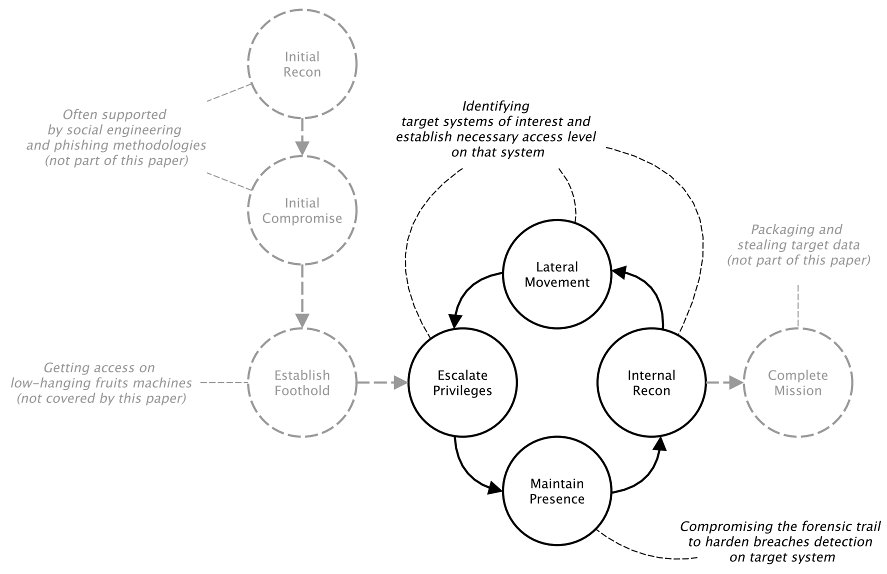
Initial Recon
Often supported by social engineering and phishing methodologies (not part of this paper)
Initial Compromise
Identifying target systems of interest and establish necessary access level on that system
Lateral Movement
Packaging and stealing target data (not part of this paper)
Getting access on low-hanging fruits machines (not covered by this paper)
Establish Foothold
Escalate Privileges
Internal Recon
Complete Mission
Maintain Presence
Compromising the forensic trail to harden breaches detection on target system

Figure 1. The cyber attack life cycle model. Adapted from the cyber attack lifecycle used by the M-Trends reports, see Table II

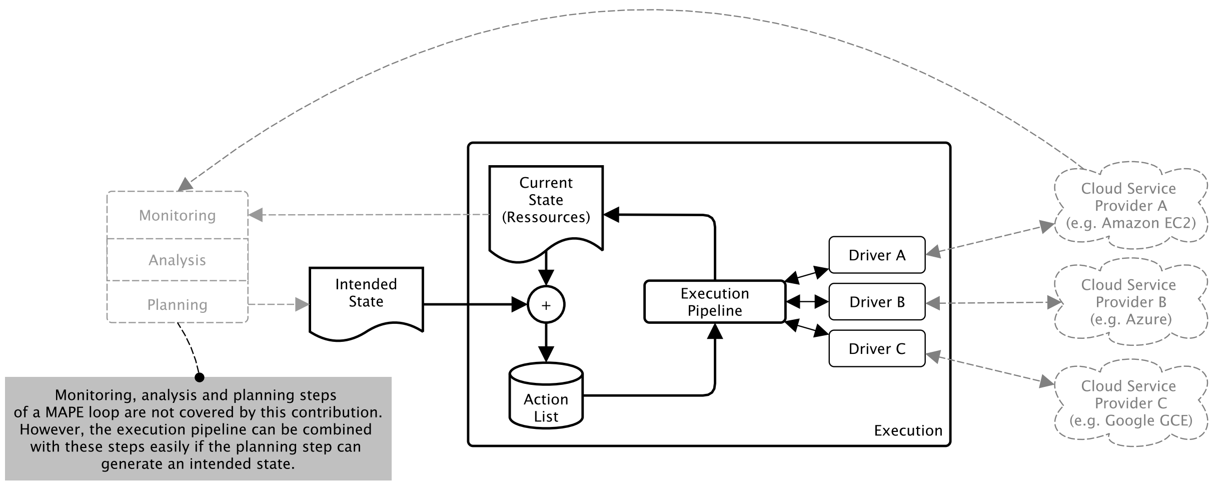
Monitoring
Analysis
Planning
Monitoring, analysis and planning steps of a MAPE loop are not covered by this contribution. However, the execution pipeline can be combined with these steps easily if the planning step can generate an intended state.
Intended State
Current State (Ressources)
+
Action List
Execution Pipeline
Driver A
Driver B
Driver C
Execution
Cloud Service Provider A (e.g. Amazon EC2)
Cloud Service Provider B (e.g. Azure)
Cloud Service Provider C (e.g. Google GCE)

Figure 2. The execution loop and synchronized security group concept

application instances, application health management, dynamic routing and load balancing as well as aggregation of logs and metrics [3]. Some open source examples of such kinds of elastic platforms are listed in Table I.

If the reader understands the commonality that cloud-native applications are operated (more and more often) on elastic – often container-based – platforms, it is an obvious idea to delegate the responsibility for the forensic trail to these platforms. Furthermore, our recent research shows that the operation of these elastic container platforms and the design of applications running on-top should be handled as two completely different engineering problems. This often solves several issues in modern cloud-native application engineering [15]. And that is not just true for the transferability problem but might be an option for the forensic trail as well. These kinds of platforms are an essential part of the immune system of modern cloud-native applications.

Furthermore, **self-service elastic platforms** (see Table I) are really ”bulletproofed” [10]. *Apache Mesos* [16] has been successfully operated for years by companies like Twitter or Netflix to consolidate hundreds of thousands of compute nodes. Peinl and Holzschuher [17] provide an excellent overview for interested readers. Elastic container platforms are **designed for failure** and provide self-healing capabilities via auto-placement, auto-restart, auto-replication and auto-scaling features. They will identify lost containers (for whatever reasons, e.g., process failure or node unavailability) and will restart containers and place them on remaining nodes. These features are absolutely necessary to operate large-scale distributed systems in a resilient way. However, these exact same features can be used intentionally to **realize transferability requirements** or to **purge “compromised nodes”**.

In recent research [2] [15] we demonstrated a software prototype that provides the control process shown in Figure 2. This process relies on an *intended state* ρ and a *current state* σ of a container cluster (attached nodes, existing security groups. If no multi-cloud settings are considered, security groups can be ignored almost completely. This is done by this paper for simplicity reasons. If the intended state differs from the current state ($\rho \neq \sigma$), necessary adaption actions are deduced (creation and attachment/detachment of nodes, creation and termination of security groups) and processed by an execution pipeline fully automatically (see Figure 3) to reach the *intended state* ρ, so that $\rho = \sigma'$ where σ' is the current state in time after σ.

With this kind of control process, a cluster can be simply resized by changing the intended state of the cluster (adding $\sigma = N \mapsto \sigma' = N + i$ or decreasing $\sigma = N \mapsto \sigma' = N - i$ the intended amount of nodes). If the cluster is shrinking and nodes have to be terminated, affected containers will be rescheduled to other available nodes. Migrations between different cloud service providers become possible as well. Let us assume a cluster with N nodes provided by *Amazon Web Services* (*AWS*) that shall be transferred to *Google Compute Engine* (*GCE*). So, the starting current state would be $\sigma = (N, 0)$. If we want to move from *AWS* to *GCE* we can transfer an elastic container platform in the following way. In a first step, we simply add the amount of nodes provisioned by *AWS* to the clusters intended state $\rho = (N, N)$ – but on *GCE*'s side, so we get $\sigma' = (N, N)$ after a cycle of the control process. In a second step, we shut down all nodes provided by *AWS* by removing them from the intended state $\rho = (0, N)$ and get $\sigma'' = (0, N)$ after another cycle. The cluster will observe node failures and trigger its self-healing features to reschedule lost containers accordingly. From an inner point of view of the platform, rescheduling operations are tasked due to node failures. From an outside point of view, it looks like (and in fact **is**) a migration from one provider to another provider at run-time. This should make the general idea clear – and the reader is referred to [2] [15] for more details.

It is essential to understand that a node is migrated by adding a new node to the cluster and delete the former node. Therefore, the intruder looses access to every migrated node, because this instance is terminated and replaced by a complete new node instantiated from a virtual machine image. This is mainly done to keep things simple for the above mentioned control process. A migration is no "live-migration" and keeps no user-related state on the affected machine during or after migration. The container platform will detect container unavailabilities due to node failures and will reschedule lost containers on other nodes. This is all handled by the container platform. For an intruder, the only way to keep a foothold in the system would be to inject malicious code into a virtual machine or container image that is used to launch nodes for the container platform or container on the platform. However, that is a completely different kind of attack, which is not covered by this paper.

The downside of this approach is,that this will only work for level 2 (cloud resilient) or level 3 (cloud native) applications (see Table III) which by design, can tolerate dependent service failures (due to node failures and container rescheduling) which may occur for a limited amount of time. However, for that kind of level 2 or level 3 application, we can use the same control process to regenerate nodes of the container cluster. The reader shall consider a cluster with $\sigma = N$ nodes. If we want to regenerate one node, we change the intended state to $\rho = N + 1$ nodes which will add one new node to the cluster ($\sigma' = N + 1$). And in a second step, we will decrease the intended size of the cluster to $\rho' = N$ again, which has the effect that one node of the cluster is terminated ($\sigma'' = N$). So, we regenerated one node simply by adding one node and deleting one node. We could even regenerate the complete cluster by changing the cluster size in the following way: $\sigma = N \mapsto \sigma' = 2N \mapsto \sigma'' = N$. But, this would consume more resources because the cluster would double its size for a limited amount of time. A more resource efficient way would be to regenerate the cluster in N steps: $\sigma = N \mapsto \sigma' = N + 1 \mapsto \sigma'' = N \mapsto ... \mapsto \sigma^{2N-1} = N + 1 \mapsto \sigma^{2N} = N$.

Whenever such a regeneration is triggered, all (even undetected) hijacked machines would be terminated and replaced by other machines, but the applications running on-top of this platform would be unaffected. For an attacker, this means losing their foothold in the system completely. Imagine if this were to be done once a day or even more frequently? The question is whether it is possible to do it with these frequencies and this paper will return to this question in Section V.

TABLE III. CLOUD APPLICATION MATURITY MODEL [12]

Level	Maturity	Criteria
3	Cloud native	- A CNA can migrate across infrastructure providers at runtime and without interruption of service. - A CNA can automatically scale out/in based on stimuli.
2	Cloud resilient	- The application state is isolated in a minimum of services. - The application is unaffected by dependent service failures. - The application is infrastructure agnostic.
1	Cloud friendly	- The application is composed of loosely coupled services. - Application services are discoverable by name (not by IP). - Application components are designed using cloud patterns. - Application compute and storage are separated.
0	Cloud ready	- The application runs on virtualized infrastructure. - The application can be instantiated from image or script.

IV. PROPOSAL OF AN IMMUNE SYSTEM ARCHITECTURE

Section III showed that it is possible for level 2 or level 3 cloud-native applications to be operated on a permanently regenerating platform that makes it hard for an intruder to maintain a foothold in the system. Yet the forensic trail can be deleted or compromised in only a short amount of time. So, there is the need to store the forensic trail in a way to be undeleteable and uncompromiseable. One obvious solution is to store the logs not on the same system but to consolidate them in an external logging system or an external logging service. Such logging services are becoming more and more widespread and the term Logging-as-a -Service (LaaS) has been established. Furthermore, research has been carried out to enable secure LaaS, even in untrusted environments [18]. So, from a regulatory point of view it would be sufficient to log into an external logging service and to make this service provider responsible to fulfill the criteria through service level agreements.

There might be regulatory constraints (e.g., data privacy) that prohibit to make use of external logging services. In these cases, we have to consolidate our logs by ourselves. The canonical way to do this in modern cloud engineering is to make

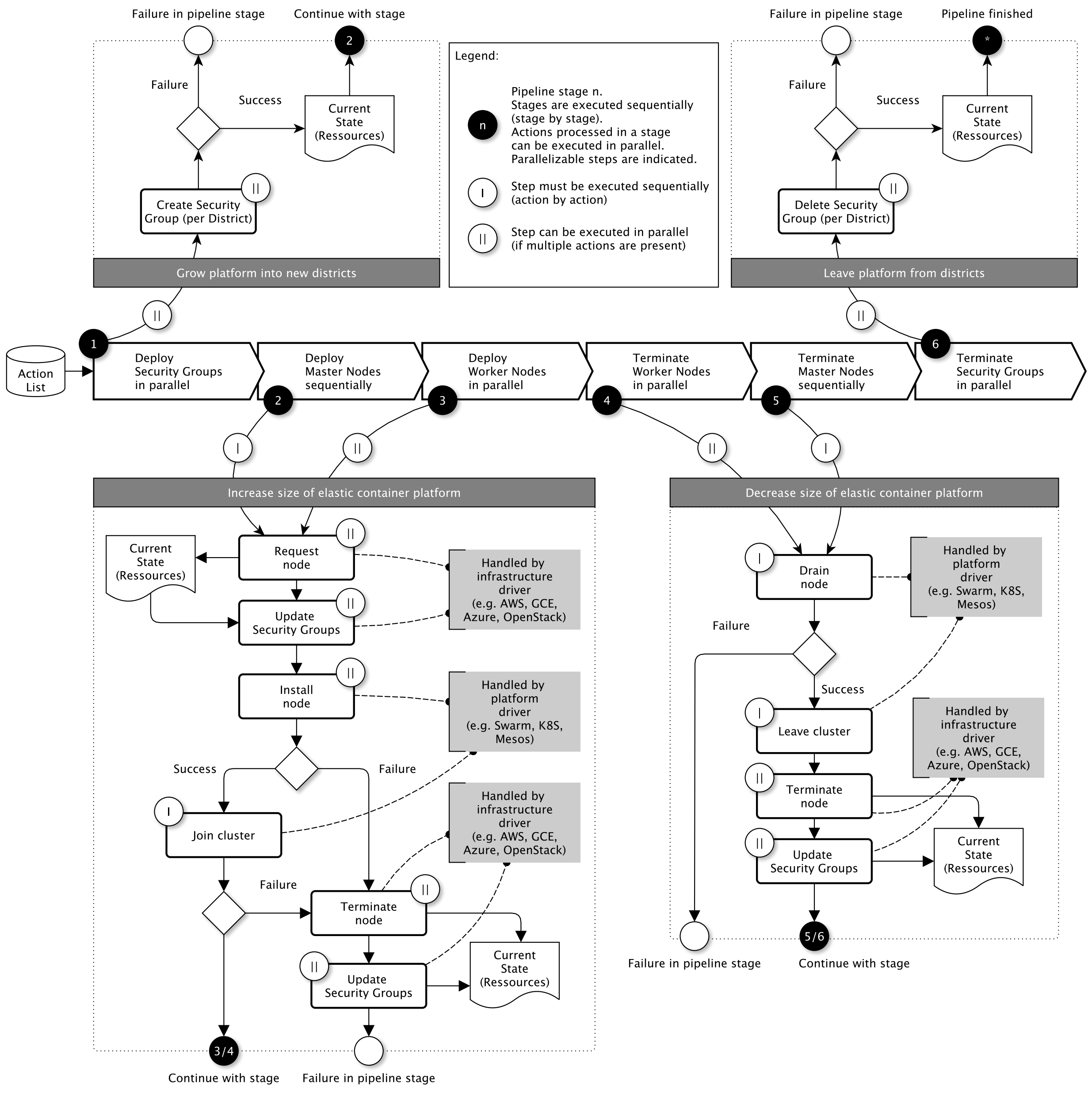
Failure in pipeline stage
Continue with stage
Failure
Success
Current State (Ressources)
Create Security Group (per District)
Grow platform into new districts
Legend:
Pipeline stage n. Stages are executed sequentially (stage by stage). Actions processed in a stage can be executed in parallel. Parallelizable steps are indicated.
Step must be executed sequentially (action by action)
Step can be executed in parallel (if multiple actions are present)
Pipeline finished
Delete Security Group (per District)
Leave platform from districts
Action List
Deploy Security Groups in parallel
Deploy Master Nodes sequentially
Deploy Worker Nodes in parallel
Terminate Worker Nodes in parallel
Terminate Master Nodes sequentially
Terminate Security Groups in parallel
Increase size of elastic container platform
Request node
Update Security Groups
Install node
Join cluster
Terminate node
Handled by infrastructure driver (e.g. AWS, GCE, Azure, OpenStack)
Handled by platform driver (e.g. Swarm, K8S, Mesos)
Decrease size of elastic container platform
Drain node
Leave cluster
3/4
5/6

Figure 3. Execution pipeline (explained in details here: [15])

use of logging stacks like the ELK stack (https//elastic.io) or Heapster (https://github.com/kubernetes/heapster) or to use distributed streaming platforms like Kafka [6]. These stacks are composed of three basic tiers. Nodes are instrumented with *metric* and *log shippers* on tier 1 (the nodes of the payload system according to Figure 4). These shippers send their data to distributed and horizontal scalable *timeseries databases* or *streaming platforms* (that can be operated on elastic platforms according to Figure 4). And a visualization and analytical component on tier 3 is responsible to visualize timeseries or streams and perform analytics on log data. These analytical and visualizing components can be run on the same hardware like the timeseries database/streaming platform or somewhere else.

This paper proposes to do the same. However, is has to be considered that the logging system might be compromised as well. And that is why this paper proposes to log and analyze the logging system by a second logging system and vice versa. In fact there are three systems. The *payload system*, a *logging system A* which logs the *payload system* and a second *logging system B* which logs the *logging system A*. The *logging system A* logs and supervises *logging system B* as well, to avoid an unsupervised system. All three systems are operated on regenerating elastic container platforms as shown in Section III that make it hard for an intruder to maintain a foothold in any of these three systems. *Logging systems A* and *B* have not just the responsibility to log but also to detect anomalies in the supervised systems. If they detect anomalies they trigger a node regeneration. Such anomalies

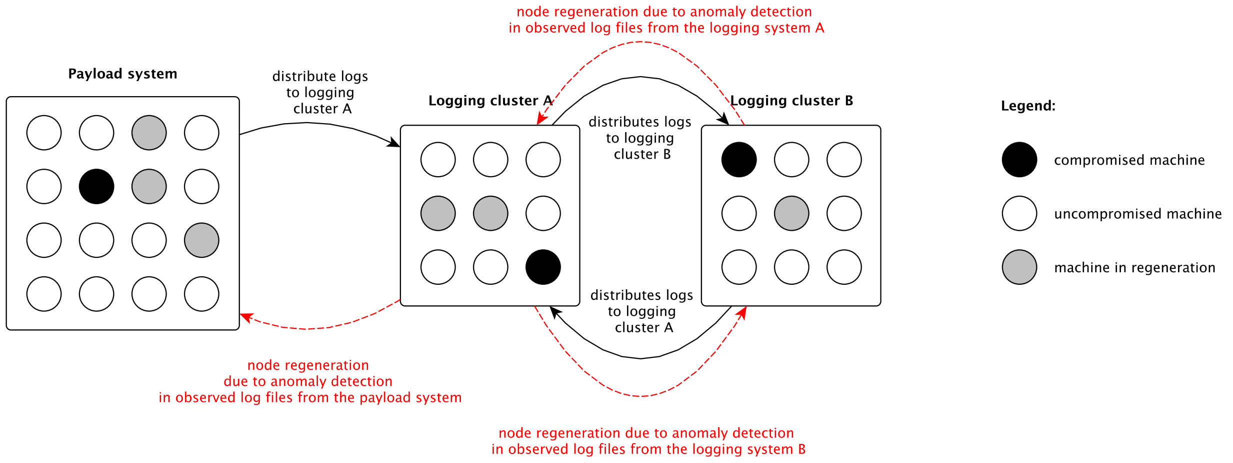
node regeneration due to anomaly detection in observed log files from the logging system A
Payload system
distribute logs to logging cluster A
Logging cluster A
distributes logs to logging cluster B
Logging cluster B
distributes logs to logging cluster A
node regeneration due to anomaly detection in observed log files from the payload system
node regeneration due to anomaly detection in observed log files from the logging system B
Legend:
compromised machine
uncompromised machine
machine in regeneration

Figure 4. An Immune System Architecture based on a Double Logging System

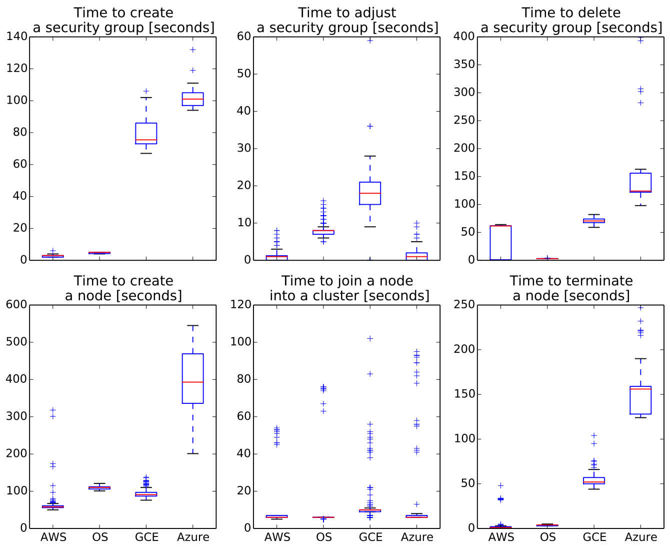

Figure 5. Differences in infrastructure specific regeneration durations [15]

could be detected automatically by approaches described by [19]. And it might be interesting for the reader that there are even approaches making use of bioinformatics tools to detect outliers in log data [20] which is aligned to some degree with our biology analogy throughout this paper. This anomaly detection must not be perfect, it would be sufficient to rate a node just vaguely as suspect (and not as precisely hostile). A false positive would result only in an unnecessary regeneration of one node. However, if there are too many suspects identified, then too many nodes would be attacked unnecessarily by the immune system to be regenerated, thus the system would become ”hot” due to a lot of regenerations. The reader might know this health state as ”fever” from their own experiences. A medical scientist might even want to talk about an autoimmune disease.

V. Evaluation Results

The execution pipeline presented in Figure 3 was evaluated by operating and transferring two elastic platforms (*Swarm Mode of Docker 17.06* and *Kubernetes 1.7*) across four public and private cloud infrastructures. All experiments were repeated 10 times. The platforms operated a reference ”sock-shop” application being one of the most complete reference applications for microservices architecture research [21]. Table IV lists the machine types that show a high similarity across different providers. These machine types have been selected according to [22]. The OpenStack *m1.large* and *m1.medium* are research institute specific machine types that have been intentionally defined to show maximum similarities with the other mentioned machine types.

Although the evaluation of [2] [15] was not done to investigate the current use case, it is possible to use some of the data to estimate reasonable regeneration cycles of elastic container platforms. First of all, the evaluation demonstrated that most time is spent on the IaaS level (creation and termination of nodes and security groups) and not on the elastic platform level (joining, draining nodes). The measured differences on infrastructures provided by different providers is shown in Figure 5. For the current use case the reader can ignore the times to create and delete for a security group (because that is a one time action). However, there will be many node creations and node terminations. According to our execution pipeline shown in Figure 3 a node creation ($\sigma = N \mapsto \sigma' = N + 1$)

TABLE IV. USED MACHINE TYPES AND REGIONS

Provider	Region	Master type	Worker type
AWS	eu-west-1	m4.xlarge	m4.large
GCE	europe-west1	n1-standard-4	n1-standard-2
Azure	europewest	Standard_A3	Standard_A2
OpenStack	*own datacenter*	m1.large	m1.medium

TABLE V. DURATIONS TO REGENERATE A NODE (median values)

Provider	Creation	Adj. Secgroup	Joining	Term.	Total
AWS	70 s	1 s	7 s	2 s	**81 s**
GCE	100 s	8 s	9 s	50 s	**175 s**
Azure	380 s	17 s	7 s	180 s	**600 s**
OpenStack	110 s	2 s	7 s	5 s	**126 s**

involves the durations to **create a node** (request of the virtual machine including all installation and configuration steps for the elastic container platform), to **adjust of security group** the cluster is operated in and to **join the new node** into the cluster. The shutdown of a node ($\sigma = N \mapsto \sigma' = N - 1$) involves the **termination of the node** (this includes the platform draining and deregistering of the node and the request to terminate the virtual machine) and the necessary **adjustment of the security group**. So, for a complete regeneration of a node ($\sigma = N \mapsto \sigma' = N + 1 \mapsto \sigma'' = N$) we have to add the times to create, to join, to terminate the node and to add two times the adjustment of the security group the cluster is operated in. Table V lists these values per infrastructure.

So, even on the "slowest" infrastructure Azure, a node can be regenerated in about 10 minutes. That means one can regenerate six nodes every hour or up to 144 nodes a day or a cluster of 432 nodes every 72h (which is the reporting time requested by the EU GDPR). If the reader compares a 72h regeneration time of a more than 400 node cluster (most systems are not so large) with the median value of 99 days that attackers were present on a victim system in 2016 (see Table II) the benefit of the proposed approach should become obvious.

Obviously there are some open questions and drawbacks regarding this proposal that should be tackled by ongoing research. For example, what would be the penalty of additional load on the cloud infrastructure? What would be the degradation in application performance caused by frequent VMs re-generation? An educated guess would be to expect $1/n$ behavior. At one point in time only one node of a n node cluster is affected. So, the performance degradation would be much more severe for small clusters and hardly observable for larger clusters. However, this has been not investigated so far.

VI. Related Work

To the best of the author's knowledge, there are currently no approach making intentional use of virtual machine regeneration for security purposes [23]. A literature search using Google Scholar and the Semantic Web did not turn up any noteworthy papers in this field. However, the proposed approach stands on the shoulders of giants and is derived from multi-cloud scenarios and their increased requirements on security. And there are several promising approaches dealing with multi-cloud scenarios. So, all of them should show comparable opportunities but come along with a lot of inner complexity. A container based approach seems to handle this kind of complexity better. There are some good survey papers on this [24] [25] [26] [27].

To make the execution pipeline work seamlessly, an efficient and pragmatic deployment description format is needed. The current format is based on JSON and shows similarities with other kind of deployment description languages like **TOSCA** or **CloudML** [28]. Nonetheless, the proposed approach is focused on a more container-centric approach and separates the platform and application level which enables a high-frequency regeneration of hosting virtual machines. This is hardly realizable with TOSCA and comparable approaches without accepting downtimes in regeneration cycles.

Duncan and Whittington emphasize the requirement to beef up the need to use the humble audit trail on all cloud systems to improve the ability to retain some level of forensic trail. They propose to use an immutable database for this purpose, which they suggested to be kept in a remote location from the main cloud system [29] [30]. Further research deals with append-only data structures on untrusted servers [31]. Other approaches propose building a secure and reliable file synchronization service using multiple cloud synchronization services as untrusted storage providers [32]. Further approaches focus on the integrity of logs and ensure their integrity by hash-chain schemes and proofs of past logs published periodically by the cloud providers [18]. The question remains, whether these approaches are scalable enough to provide robust logging means for the forensic trail of up to thousands of nodes. Messaging solutions like Kafka [6] or logging stacks like the ELK-Stack are bullet-proofed technologies for consolidating logs but assume to be operated in a trusted environment which ends in a kind of double logging architecture. So, the above mentioned research approaches show the potential to simplify the double logging architecture and should be considered in ongoing research. The same is true for anomaly detection approaches in log data [19] [20].

Taken all together, the proposed approach leverages more up-to-date container technologies with the intent to be more "pragmatic". On the downside, it might be only applicable for container-based applications being on the level 2 or 3 of the maturity model shown in Table III. But this architecture style gets more and more common in cloud application engineering [3].

VII. Conclusion

Once attackers successfully breach a cloud system there is little to prevent them from modifying the forensic trail. This involves a serious challenge - from a security but also from an economic point of view. The forthcoming EU General Data Protection Regulation (GDPR) has a regime of penalties which can rise up to 4% of global turnover in those cases where failure by the company to protect systems in a sufficiently robust manner will be seen as complicit in the loss of customer data.

Although the presented approach evolved mainly from transferability research questions for cloud-native applications, it can be adopted as a foundation for an approach one could call 'immune system" inspired. This paper proposed to regenerate virtual machines (the cells of an IT-system) with a much higher frequency than usual to purge even undetected intruders. Our evaluations on infrastructures provided by AWS, GCE, Azure and OpenStack showed that a virtual machine can be regenerated in somewhere between two minutes (AWS) and 10 minutes (Azure). The reader should compare these times with recent cyber security reports. E.g., the M-Trends report from 2016 reported that an attacker was undetected on a victim system for about 100 days. For an attacker this means that their undetected time on a victim systems drops from months down to minutes, thus minimising the potential for damage.

Nevertheless, even in a very short amount of time an attacker could delete (parts) of the cloud forensic trail. To use external and trusted append-only logging systems seems somehow obvious. However, existing solutions rely on trusted environments. But if that environment can not be assured, the need for complex and "ugly" double logging architectures arises, as Section IV, has shown. Our further research will investigate how "regenerating" platforms and append-only logging systems could be integrated more straightforward.

Acknowledgment

This research is partly funded by the Cloud TRANSIT project (13FH021PX4, German Federal Ministry of Education and Research). I would like to thank Bob Duncan for his efforts making this paper much more readable.

References

[1] B. Duncan and M. Whittington, "Compliance with standards, assurance and audit: does this equal security?" in Proc. 7th Int. Conf. Secur. Inf. Networks - SIN '14. Glasgow: ACM, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2659651.2659711

[2] N. Kratzke, "Smuggling Multi-Cloud Support into Cloud-native Applications using Elastic Container Platforms," in Proc. of the 7th Int. Conf. on Cloud Computing and Services Science (CLOSER 2017), 2017.

[3] N. Kratzke and P.-C. Quint, "Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study," Journal of Systems and Software, vol. 126, no. April, 2017.

[4] N. Kratzke and R. Peinl, "ClouNS - a Cloud-Native Application Reference Model for Enterprise Architects," in 2016 IEEE 20th Int. Enterprise Distributed Object Computing Workshop (EDOCW), Sep. 2016.

[5] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in ACM Conference on Computer and Communications Security, 2012.

[6] Wang et al., "Building a Replicated Logging System with Apache Kafka," in Proc. of the VLDB Endowment, vol. 8, no. 12, 2015.

[7] FireEye Inc., "Security Predications 2018," 2017, retrieved: 12, 2017. [Online]. Available: https://www.fireeye.com/current-threats/annual-threat-report.html

[8] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Advanced social engineering attacks," Journal of Information Security and Applications, vol. 22, 2015.

[9] S. Gupta, A. Singhal, and A. Kapoor, "A literature survey on social engineering attacks: Phishing attack," 2016 International Conference on Computing, Communication and Automation (ICCCA), 2016.

[10] M. Stine, Migrating to Cloud-Native Application Architectures. O'Reilly, 2015.

[11] Fehling et al., Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer Publishing Company, Incorporated, 2014.

[12] Ashtikar et al., "OPEN DATA CENTER ALLIANCE Best Practices: Architecting Cloud-Aware Applications Rev. 1.0," 2014, retrieved: 12, 2017. [Online]. Available: https://www.opendatacenteralliance.org/docs/architecting_cloud_aware_applications.pdf

[13] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to Cloud-Native Architectures Using Microservices: An Experience Report," in 1st Int. Workshop on Cloud Adoption and Migration (CloudWay), Taormina, Italy, 2015.

[14] S. Newman, Building Microservices. O'Reilly Media, Incorporated, 2015.

[15] N. Kratzke, "About the complexity to transfer cloud applications at runtime and how container platforms can contribute?" in Cloud Computing and Service Sciences: 7th International Conference, CLOSER 2017, Revised Selected Papers, Communications in Computer and Information Science (CCIS). Springer International Publishing, 2018, to be published.

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in 8th USENIX Conf. on Networked systems design and implementation (NSDI'11), vol. 11, 2011.

[17] R. Peinl and F. Holzschuher, "Docker cluster management state of the art and own solution," Journal of Grid Computing, vol. 14, 2016.

[18] S. Zawoad, A. K. Dutta, and R. Hasan, "Towards building forensics enabled cloud through secure logging-as-a-service," IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 2, 2016.

[19] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," in 2009 Ninth IEEE Int. Conf. on Data Mining, 2009.

[20] M. Wurzenberger, F. Skopik, R. Fiedler, and W. Kastner, "Applying High-Performance Bioinformatics Tools for Outlier Detection in Log Data," in CYBCONF, 2017.

[21] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," in Proc. of the 1st Int. Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ser. ECASE '17. Piscataway, NJ, USA: IEEE Press, 2017.

[22] N. Kratzke and P.-C. Quint, "About Automatic Benchmarking of IaaS Cloud Service Providers for a World of Container Clusters," Journal of Cloud Computing Research, vol. 1, no. 1, 2015.

[23] D. A. B. Fernandes, L. F. B. Soares, J. V. P. Gomes, M. M. Freire, and P. R. M. Inácio, "Security issues in cloud environments: a survey," Int. Journal of Information Security, 2014.

[24] A. Barker, B. Varghese, and L. Thai, "Cloud Services Brokerage: A Survey and Research Roadmap," in 2015 IEEE 8th International Conference on Cloud Computing. IEEE, jun 2015.

[25] D. Petcu and A. V. Vasilakos, "Portability in clouds: approaches and research opportunities," Scalable Computing: Practice and Experience, vol. 15, no. 3, oct 2014.

[26] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected Cloud Computing Environments," ACM Computing Surveys, vol. 47, no. 1, may 2014.

[27] N. Grozev and R. Buyya, "Inter-Cloud architectures and application brokering: taxonomy and survey," Software: Practice and Experience, vol. 44, no. 3, mar 2014.

[28] M. Lushpenko, N. Ferry, H. Song, F. Chauvel, and A. Solberg, "Using Adaptation Plans to Control the Behavior of Models@Runtime," in MRT 2015: 10th Int. Workshop on Models@run.time, co-located with MODELS 2015: 18th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems, ser. CEUR Workshop Proceedings, N. Bencomo, S. Götz, and H. Song, Eds., vol. 1474. CEUR, 2015.

[29] B. Duncan and M. Whittington, "Creating an Immutable Database for Secure Cloud Audit Trail and System Logging," in Cloud Comput. 2017 8th Int. Conf. Cloud Comput. GRIDs, Virtualization. Athens, Greece: IARIA, ISBN: 978-1-61208-529-6, 2016.

[30] ——, "Cloud cyber-security: Empowering the audit trail," Int. J. Adv. Secur., vol. 9, no. 3 & 4, 2016.

[31] T. Pulls and R. Peeters, Balloon: A Forward-Secure Append-Only Persistent Authenticated Data Structure. Cham: Springer International Publishing, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-24177-7_31

[32] Han et al., "MetaSync: File Synchronization Across Multiple Untrusted Storage Services," in USENIX Annual Technical Conference, 2015.