The `MEERKATS` Cloud Security Architecture

Angelos D. Keromytis, Roxana Geambasu, Simha Sethumadhavan, Salvatore J. Stolfo, Junfeng Yang
Columbia University
{*angelos,roxana,simha,sal,junfeng*}@*cs.columbia.edu*

Azzedine Benameur, Marc Dacier, Matthew Elder, Darrell Kienzle
Symantec Research Labs
{*firstname_lastname*}@*symantec.com*

Angelos Stavrou
George Mason University
astavrou@gmu.edu

***Abstract*—MEERKATS is a novel architecture for cloud environments that elevates continuous system evolution and change as first-rate design principles. Our goal is to enable an environment for cloud services that constantly changes along several dimensions, toward creating an unpredictable target for an adversary. This unpredictability will both impede the adversary's ability to achieve an initial system compromise and, if a compromise occurs, to detect, disrupt, and/or otherwise impede his ability to exploit this success. Thus, we envision an environment where cloud services and data are constantly in flux, using adaptive (both proactive and reactive) protection mechanisms and distributed monitoring at various levels of abstraction. A key element of MEERKATS is the focus on both the software and the data in the cloud, not just protecting but leveraging both to improve mission resilience. MEERKATS seeks to effectively exploit "economies of scale" (in resources available) to provide higher flexibility and effectiveness in the deployment and use of protection mechanisms as and where needed, focusing on current and anticipated application and mission needs instead of an inefficient, "blanket" approach to protecting "everything the same way, all the time". We outline our vision for MEERKATS and describe our approach toward prototyping it.**

I. Introduction

As cloud computing gains more traction, whether in the form of public clouds (*e.g.,* Amazon EC2) or enterprise-private ones, an increasing number of critical applications are deployed and operated in such computational environments. Given the concentration of services and data, often from a large number of different entities, into a single logical location, cloud infrastructures represent a tempting and highly lucrative target for malicious entities. Therefore, the security expectations from cloud computing infrastructures are (or should be) arguably higher than those in more "diffuse" computing.

While there is probably some basis to the expectation that unified and concentrated administration and management will lead to better overall security, the current state of affairs regarding the (in)security of enterprise networks and systems does not inspire confidence in that respect. In other words, given existing security mechanisms and practices, it is likely that any gains from homogeneity and higher professional standards will not be sufficient to protect the cloud infrastructure and the applications running on it from even more motivated adversaries. We argue that we need to rethink cloud security, introducing new principles for cloud infrastructures and applications that leverage the strengths inherent in this form of computing to improve security.

We observe that a key characteristic of cloud computing is the overprovisioning of resources to comfortably accommodate the highest expected workload, with some margin of error. This leads to "dormant" resources, which are not used most of the time, to the tune of 20%–50% of typical workload levels. The argument has been made that such resources can be used, at least some of the time, to protect the cloud infrastructure and all applications on it. However, if spread across all applications, the quantitative improvement in security (*e.g.,* by being able to use a given security mechanism whose overhead is compensated by increased resource usage) will be incremental and small.

In `MEERKATS` we argue for a mission-oriented cloud computing security architecture that (a) focuses resources to improve the resiliency of components that are critical to the current mission/application, (b) learns and adapts to past, current, and anticipated threats, and (c) inherently presents an unpredictable target through continuous "motion" and mutation of services and data, and through the use of deception.

The two high-level challenges to our envisioned architecture are (1) the lack of the efficient and effective mechanisms for instantiating several of our architecture's core elements, and (2) the complexity of integrating and operating an architecture where "everything changes". To realize our vision we need to investigate, develop, and evaluate a number of individual components, and to integrate them in a coherent architecture.

The main components of `MEERKATS` are (a) DMCC, a distributed monitoring and cross-checking substrate that operates at multiple levels of abstraction (network, process, and function/instruction), whose goal is to learn models of normal behavior and to detect anomalous behavior; (b) CSSH, a set of mechanisms that allow for targeted and adaptive hardening and healing of programs, allowing for tunable protection schemes to be applied on select application instances at any given time, both leveraging and feeding the DMCC; (c) CISR, an artificial diversity mechanism based on Instruction-Set Randomization [1], [2], coordinated across program replicas to ensure non-alignment; (d) CSIFT, a fine-grained cross-host/cross-application information flow tracking system; (e) DIGIT, a deceptive information generation, injection, and tracking component aimed at detecting and confusing adversaries; (f) Evade, a system for splitting, moving and recombining information on demand across the cloud infrastructure using lightweight data migration; (g) N-Schedule, a mechanism for exploiting redundancy to detect attacks and failures due to concurrency errors and vulnerabilities; (h) MiSS, a system for fast and seamless process migration, data, and network reconstitution, effectively mitigating attacks that target individual application instances and "pinch-points" in cloud application instances and network communications; (i) DREME, an environment for enabling synchronized execution of software replicas with varying diversification strategies, using distributed correlation techniques for monitoring the health of those software replicas; and (j) hardware support for streamlining the execution of replicas and for fast checkpoint/migration.

The various components seek to address overlapping and complementary types of adversaries and attacks, and interact in several ways. CSSH and CISR seek to prevent altogether or detect a low-level software compromise shortly after the attack occurs. Both components can be targeted to specific applications (or even portions of an application), and can share information to coordinate cloud-wide protection. The DMCC provides anomaly detection capabilities (both for use with CSSH, and as a fallback detector); it also gathers information that can be used to improved the functionality of other mechanisms (especially CSSH and CSIFT), *e.g.,* identification of rescue points for use in self-healing [3]. MiSS and DREME provide code migration and containment capabilities, and Evade provides data migration capabilities—all these are meant to be used both proactively (as a "moving target" defense) and reactively (in response to a detected compromise). CSSH and MiSS will also provide recovery capabilities, and the same combination will also enable us to lower performance overheads through the use of replicated execution. DIGIT is a complementary mechanism that seeks to confuse and misdirect adversaries through the use of deceptive information and computation (coordinating with Evade for the former, depending on MiSS and DREME for the latter). CSIFT provides fine-grained data tracking and exfiltration-detection capabilities across the cloud, and interacts both with Evade (for tracking legitimate data) and DIGIT (for tracking deceptive information). N-Schedule addresses concurrency bugs by leveraging deterministic scheduling and replicated execution.

In the remainder of this paper, we provide detail on the specific components that we are developing for MEERKATS.

II. Research Direction

To set our discussion of the individual research thrusts in context, we provide an example scenario of how MEERKATS might be used in the context of a cloud infrastructure supporting multiple diverse applications that are composed into on-demand (or pre-planned) missions. At any given time, some missions (and therefore applications) will be more important than others; MEERKATS will concentrate its efforts in protecting those applications, while expending minimal (but sufficient) effort in preserving the remaining applications.

Once a mission is planned and the critical assets identified, MEERKATS will begin by using CSSH to harden these, and deploy several replicas using DREME, if that is not already the case. Some of these replicas may be diversified with CISR and N-Schedule. DMCC will be monitoring both the specific applications and the overall infrastructure, exploiting knowledge from previous executions and missions. Simultaneously, the critical data will be pre-positioned using Evade. CSIFT will be tracking the use of said data, while DIGIT will also leverage DREME, Evade and MiSS to create artificial targets. If an attack or compromise is identified, uncorrupted components will be migrated elsewhere with MiSS, and data will be moved by Evade. Any corrupted components might be recovered with CSSH, or they may simply be abandoned, or they may be replaced by deceptive computation (via DIGIT) to divert attacker attention and effort.

We now describe some of our components in more detail; due to space limitations, we cannot discuss them all equally.

A. DMCC: Distributed Monitoring and Crosschecking

The DMCC portion of the system is the major component that integrates and correlates information feeds from multiple MEERKATS components. The key capability we seek is to identify abnormal execution events that indicate attacks requiring immediately mitigation. This is largely accomplished via correlation of detected anomaly alerts.

The anomaly detection (AD) algorithm we will use is based on the Probabilistic Anomaly Detection (PAD) algorithm [4], [5], [6]. PAD has been applied to Windows Registry, process execution, file system access, and network packet header anomaly detection. Here, we will apply an improved and faster version of PAD to program execution state information to detect errant program behavior, as well as other novel audit sources as provided by other MEERKATS components such as CSIFT, N-Schedule, and CSSH.

One research thrust we are pursuing is how to extract from the system specific information, such as function names on the stack, the argument buffer name it may reference, and potentially other lower-level system and processor features, such as the Translation Lookahead Buffer access patterns, that may be used to identify abnormal system behavior previously tested by prior use of a common "test vector" that baselined the application. A focus problem in this area is to define the particular features extracted and that are efficiently processed to maximize detection performance (maximal true positive rates and minimized false positives). This information would enrich the potential alerts from other system components to reduce false positives and identify true attacks. Correlating this information is conducted by use of gold standard test vectors used to baseline the systems expected behavior. Careful creation of test vectors is necessary to learn the most effective correlation function that maximizes detection performance.

B. CSSH: Collaborative Self-healing and Service Hardening

A key component of MEERKATS is *CSSH*, a novel mechanism that allows for selective, tunable application of a number of software hardening and self-healing techniques to applications. CSSH itself is composed of a number of elements, primarily centered around PIN [7], an extensible dynamic binary rewriting framework that allows us to instrument arbitrary instructions and functions so that we can insert hardening and self-healing functionality with great precision. Furthermore, PIN allows us to instrument (and de-instrument) *already-running* programs. The combination of selectivity, fine granularity, and flexibility in injected instrumentation will allow us to deploy hardening functionality as and where it is needed, focusing computational resources to protecting current mission needs. We envision deployment under the guidance of DMCC, under manual control, and through an API exposed to other detection mechanisms.

The protection mechanisms that CSSH will enable include REASSURE [3], Write-Read Integrity Testing (WRIT), stack/heap buffer overflow protection, number handling vulnerability masking (divide-by-zero, integer overflow/underflow, *etc.*), NULL pointer protection, Control Flow Integrity (CFI) [8], and limited race condition detection/avoidance. CSSH will also be used to insert probes and detectors within an application, to provide fine-grained behavior information to the DMCC. REASSURE is a software self-healing mechanism that can be used to recover from unforeseen errors or vulnerabilities, using existing code locations that handle certain anticipated errors as "graceful" (if unintended) exception handlers, providing both integrity and availability guarantees. MEERKATS will extend REASSURE by enabling the sharing of alerts and fixes, and will leverage efficient replicated execution to provide low-overhead service and community resilience. WRIT is a novel mechanism that applies points-to analysis to determine the correct Read and Write Sets for each instruction (or group of instructions), and insert instrumentation to detect deviating memory accesses.

CSSH is also implementing Coordinated Instruction Set Randomization (CISR), an extension to our baseline ISR technique [1]. ISR obfuscates the "language" understood by a system to protect against code-injection attacks by presenting an ever-changing target. MEERKATS is extending ISR to coordinate randomization across replicas and independent application instances, to ensure attack-vector independence and thus rapid detection of attacks.

C. CSIFT: Cross-System Information Flow Tracking

A novel multi-use component that we are developing as part of MEERKATS is CSIFT, a fine-grained information flow tracking (IFT) system. CSIFT will allow us to *transparently* perform byte- or word-level IFT within and across individual processes and systems. To achieve this, we will use our high-performance data tracking library, *libdft*, which is built on top of PIN. CSIFT will allow for labels to be transparently propagated across CSIFT/CSSH-supervised processes by multiplexing label transmission with application data over the same channel (*e.g.,* pipe or network socket). CSIFT will operate on unmodified program binaries, and will offer flexible assignment of "taint" sources. CSIFT will track the provenance, consumption and attempted exfiltration of sensitive information across the distributed cloud application in several ways: combined with PPSE (Section II-B) and replicated execution (Section II-H), it will allow for low-overhead tracking (both by pipelining the tracking and through *post mortem* analysis via replicas); combined with DIGIT (Section II-D), it will allow for the detection and monitoring of adversaries that are accessing deceptive information; combined with Evade (Section II-E), it will provide both resilience against and detection of data exfiltration; combined with DMCC (Section II-A), it will learn each service's "normal" information flow pathways, and detect deviations.

D. DIGIT: Deceptive Information Generation, Injection, and Tracking

To deal with attackers that have managed to penetrate a system, and to divert attempted attacks, we propose to integrate the use of deceptive information. DIGIT is a novel approach to detecting, confusing, and misdirecting attackers by using deceptive information and "throw-away" computation. We propose to leverage our ability to create and operate a large number of application replicas, some of which (called the "deception set", or DS) are provided with fake inputs. Thus, an adversary controlling a malicious or compromised replica will be uncertain as to the validity of any captured data. Primary replicas will also periodically be placed in the DS simply by changing the source of inputs, and a new primary replica will be chosen. We will leverage the MEERKATS functionality for replicated execution, to modify some of the inputs on other replicas. The inputs will contain automatically generated *enticing* and *believable* deceptive information (DI) whose misuse by an adversary can be subsequently detected. Examples of such enticing bait information include documents with built-in "beacons" [9], URLs and username/passwords to honeypots or sites whose access can be directly or indirectly monitored [10], credit card and bank account numbers with triggers [11], *etc.* Other types of DI that we plan to use include deceptive documents in the filesystem and entries in database tables (or entire databases). Furthermore, CSIFT can provide DI tracking within the cloud infrastructure. The exact type of bait used depends on the application.

We are combining the use of DI with migration, containment and replication functionality (Sections II-G and II-H) to create execution environments (individual processes and complete VMs) that are partly or entirely deceptive. Thus, adversaries will either waste their efforts attempting to compromise these systems (and betray their presence) or, if already in the system, they will waste effort trying to exploit their initial success. Furthermore, by using VM and process migration, we will be able to move legitimate computation away from a targeted or compromised VM/host, and replace it with deceptive computation. CSSH instrumentation will also allow us to analyze adversary activities and take containment actions in VMs and hosts that are not yet compromised.

The main challenge in building DIGIT is the believability of generated DI. We propose to leverage the DMCC to learn from prior execution runs and construct realistic inputs that will cause the same execution path within a replica to be taken.

E. Evade: Avoiding Danger with Lightweight Data Migration

To protect data against attacks in a diversified cloud, we propose *Evade*, a security-oriented distributed storage system in which *data runs away from (or evades) danger*. In Evade, as soon as a machine is suspected to be compromised or vulnerable to future attacks, its data will immediately disappear from that location and emerge at a different location. Moreover, the data periodically "jumps" from one location to another randomly, impeding targeted attacks. Evade's data migration is fast, independent of the data's size, allowing us to create a fast-flux, unpredictable environment.

At a high level, Evade supports lightweight migration by combining proactive secret sharing [12] with distributed, replicated storage systems. Each data object is encrypted with its own symmetric key, and the ciphertext is replicated on N devices. The ciphertext replicas are of two types: *active* replicas (M replicas, $M < N$) and *inactive* replicas ($N - M$ replicas). Active replicas participate in a proactive secret sharing protocol to store the key in a distributed fashion, requiring any T of the M active replicas to be compromised in order for the data's privacy to be compromised. The active replicas also respond to authenticated clients' requests.

For swift migration, the system maintains a pool of inactive replicas that contain versions of the ciphertext, but do not participate in secret sharing or client requests. Capturing any number of inactive replicas provides no benefit to the attacker. Periodically, or whenever an active replica is in danger, the other active replicas choose at random an inactive replica and run a mobile proactive secret sharing protocol [13] to (1) include the new replica into the active set, and (2) exclude the old replica from the active set. Using pre-established ciphertext replicas, data migration in Evade involves merely transferring small key shares instead of the actual data.

F. N-Schedule

We are developing N-Schedule, a novel, unique component within MEERKATS to provide effective, transparent, and pervasive protection against *unknown* concurrency vulnerabilities. The key idea in N-Schedule is to diversify scheduling: making different replicas run different thread schedules. If one replica crashes or is exploited due to a concurrency vulnerability, the other replicas will be unaffected with high probability, because N-Schedule deliberately forces replicas to run diversified schedules. Thus, by diversifying schedules, N-Schedule will make a multithreaded program more robust against concurrency vulnerabilities. N-Schedule's explicit control over scheduling will also simplify forensic analysis of concurrency attacks. To deterministically replay a compromised execution, users run the same program on the same input, automatically reproducing the thread schedule that occurred in the attack.

G. MiSS: Migrating Software Swiftly

To prevent targeted attacks that originate both from insiders or from outside, we introduce a system for fast and seamless migration of the software and network state of application instances. The proposed system, called Migrating Software Swiftly (MiSS), leverages advances in lightweight process virtualization to move, re-instantiate, and re-route traffic between and for application instances between cloud nodes. Our aim is to provide a "moving target" defense for the hosted application instances. By moving application processes and re-routing their network connectivity, we can effectively mitigate attacks that target individual application instances and "pinch-points" in cloud application instances and network communications.

For our system, we are using open-source lightweight process virtualization systems that support Unix systems. They are effectively providing on top of the OS a thin virtualization layer called a container that provides a group of processes with a private namespace. The sandboxed process group always sees the same virtualized view of the system, which associates virtual identifiers with OS resources such as process identifiers and network addresses. This decouples sandboxed processes from dependencies on the host OS and from other processes in the system. This virtualization is integrated with a checkpoint-restart mechanism that enables the sandboxed processes to be migrated as a unit to another machine. These process groups are independent and self-contained, and can thus be migrated freely without leaving behind any residual state after migration, even when migrating network applications while preserving their network connections. We can therefore allow applications to execute after migration even if the machine on which they previously executed is no longer available.

H. DREME: Diversified Replica Execution and Monitoring Environment

To enable the execution of software replicas in support of MEERKATS' other components and capabilities, we propose DREME: an environment for (1) creating and executing multiple instances of a software application, possibly using varying diversification strategies (*e.g.,* CISR), and (2) monitoring the health of those software replicas by collecting a greater volume of data on possibly suspicious indications and then applying distributed correlation techniques to manage that data. DREME enables the synchronized execution of application replicas via the distribution of inputs, outputs, and key program state between replicas. The comparison of this data across replicas enables detection of attacks and compromise. Execution of mission applications using replicas within the cloud provides an opportunity to apply additional computational resources for more "aggressive" attack detection strategies, such as collection of more indications of "suspicious" behavior within replicas and their environments, and the use of correlation techniques across replicas to further vet those indications. Upon detection of replica compromise, DREME will provide capabilities for reconfiguring the remaining replicas, introducing additional possibly diverse replicas, and adapting the monitoring and detection strategies used.

DREME works in concert with the lightweight process virtualization technology of MiSS to provide this execution environment for application instances. DREME also takes ad-vantage of traditional "heavyweight" virtualization technology, using a hypervisor and virtual machine monitor to provide isolation of the monitoring environment from the application environment. This architecture enables additional diversification strategies to be employed with the replicas, while also enhancing the monitoring capabilities of the infrastructure.

From our vantage point monitoring the application from an external virtual machine, we can determine when changes occur that affect the integrity of either the running application instance or the operating system. We will also monitor for behaviors and characteristics that could be considered acceptable in certain situations but suspicious in other contexts, such as changes to a process's export table or kernel hooks. By collecting data on these acceptable but suspicious observables, we turn each replica into a "chatty" sensor that can be used to determine when a replica exhibits indications of an attack.

I. Hardware Support

Future cloud hardware chips will have several on-chip accelerators dedicated to application or domain specific tasks. These accelerators are the key to improving energy-efficiency and scalability in the face of diminishing gains from traditional Moore's Law scaling. Accelerators improve energy efficiency by implementing algorithms in silicon, reducing bookkeeping overheads. Instances of these special-purpose devices in commodity hardware today include GPUs (*e.g.,*, AMD Fusion, Sandybridge) or cryptographic units. For the fast migration support we envision in `MEERKATS` (through MiSS) these accelerators must be handled as first-class OS objects. The state of the art, however, is to use these accelerators as slave I/O devices that cannot be directly accessed by user-mode programs, thus requiring several user-kernel transitions, consequently reducing throughput for interactive applications. Further, as there is no common architectural specification or hardware support for checkpointing and migration in these accelerators, moving computation around may further degrade performance. To overcome these problems, we envision hardware support for tracking those pages that are touched by the accelerators, and a special purpose accelerator for managing migration itself. This will be used in concert with architectural and OS modifications that are required for treating these accelerators as processors instead of slave devices.

Naive execution of diversified replicas spawned by the DREME shim layer can significantly impact throughput of cloud oriented systems. To avoid slowdowns we propose hardware support in the form of memoization caches that selectively re-use values from previous computations in diversified replicas. User annotations, CSIFT and DIGIT analyses will all be used to determine computations that can be reused.

III. Summary

Current systems provide mostly "static" targets with limited ability to react, much less fundamentally change in security-relevant or security-significant ways. Even when such capabilities exist (*e.g.,* threshold cryptography, system migration capabilities), they operate in an isolated, uncoordinated fashion, without leveraging the distributed nature of the cloud. Such techniques are designed and used with no provision for adaptation; this leads to inefficient use of security resources on components and data that are peripheral to the current mission.

If our effort in developing `MEERKATS` is successful, we will create a "moving target" defense mechanism for the cloud that will leverage the inherent distributed nature of the cloud to improve service and data resilience to threats and attacks. The ability of `MEERKATS` to explicitly control the tradeoff between resilience/security and resource consumption is fundamental to the adoption of security mechanisms. We argue that the inherent availability of fungible resources in the cloud and the ability of MEERKATS to both strategically and tactically deploy them as the situation warrants (*e.g.,* in anticipation of or in response to an attack against a specific service or collection of data) will result in a *more secure environment* compared to current systems and services making, at best, small-scale tradeoffs between security and resource consumption.

Acknowledgements*:* This work was supported by DARPA through Contract FA8650-11-C-7190. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or DARPA.

References

[1] Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering Code-Injection Attacks With Instruction-Set Randomization. In: Proceedings of the 10th ACM CCS. (2003) 272–280

[2] Portokalidis, G., Keromytis, A.D.: Fast and Practical Instruction-Set Randomization for Commodity Systems. In: Proceedings of ACSAC. (2010)

[3] Portokalidis, G., Keromytis, A.D.: REASSURE: A Self-contained Mechanism for Healing Software Using Rescue Points. In: Proceedings of the 6th International Workshop on Security (IWSEC). (2011)

[4] Apap, F., Honig, A., Hershkop, S., Eskin, E., Stolfo, S.J.: Detecting malicious software by monitoring anomalous windows registry accesses. In: Proceedings of RAID. (2002)

[5] Wang, K., Parekh, J., Stolfo, S.J.: Anagram: A content anomaly detector resistant to mimicry attack. In: Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID). (2006)

[6] Frias-Martinez, V., Stolfo, S.J., Keromytos, A.D.: Behavior-profile clustering for false alert reduction in anomaly detection sensors. In: Proceedings of ACSAC. (2008)

[7] Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of PLDI. (2005) 190–200

[8] Abadi, M., Budiu, M., Erlingsson, U., Ligatti, J.: Control-flow integrity: principles, implementations, and applications. In: Proceedings of the ACM Conference on Computer and Communications Security. (2005)

[9] Bowen, B.M., Hershkop, S., Keromytis, A.D., Stolfo, S.J.: Baiting Inside Attackers Using Decoy Documents. In: Proceedings of SecureComm. (2009) 51–70

[10] Bowen, B.M., Kemerlis, V., Prahu, P., Keromytis, A.D., Stolfo, S.J.: Automating the Injection of Believable Decoys to Detect Snooping (Short Paper). In: Proceedings of WiSec. (2010) 81–86

[11] Bowen, B.M., Prabhu, P., Kemerlis, V.P., Sidiroglou, S., Keromytis, A.D., Stolfo, S.J.: BotSwindler: Tamper Resistant Injection of Believable Decoys in VM-Based Hosts for Crimeware Detection. In: Proceedings of RAID. (2010) 118–137

[12] Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: Proceedings of CRYPTO. (1995)

[13] Schultz, D., Liskov, B., Liskov, M.: Mobile proactive secret sharing. In: Proceedings of PODC. (2008) Brief Announcement.