Tamper-Resistant Autonomous Agents-Based Mobile-Cloud Computing

Pelin Angin, Bharat Bhargava
Department of Computer Science
Purdue University
West Lafayette, IN, USA
Email: {pangin, bb}@cs.purdue.edu

Rohit Ranchal
Watson Health Cloud
IBM
Boston, MA, USA
Email: ranchal@us.ibm.com

***Abstract*—The rise of the mobile-cloud computing paradigm has enabled mobile devices with limited processing power and battery life to achieve complex tasks in real-time. While mobile-cloud computing is promising to overcome limitations of mobile devices for real-time computing needs, the reliance of existing models on strong assumptions such as the availability of a full clone of the application code and non-standard system environments in the cloud makes it harder to manage the performance of mobile-cloud computing based applications. Furthermore, offloading mobile computation to the cloud entails security risks associated with sending data and code to an untrusted platform and perfect security is hard to achieve due to the extra computational overhead introduced by complex mechanisms. In this paper, we present a dynamic computation-offloading model for mobile-cloud computing, based on autonomous agent-based application partitions. We propose a dynamic tamper-resistance approach for managing the security of offloaded computation, by augmenting agents with self-protection capability using a low-overhead introspection and integrity-preserving communication mechanism. Experiments with a real-world mobile application demonstrates the effectiveness of the approach for high-performance, tamper-resistant mobile-cloud computing.**

I. Introduction

Mobile computing devices have replaced desktops and mainframes for daily computing needs during the past decade. Despite the everyday advances in mobile computing technology, size restrictions impose limitations on the processing power and battery life of these devices, which limits their capabilities for real-time, computing-intensive applications such as image processing. Cloud computing offers the ability to fill the gap between the resource needs of mobile devices and availability of those resources, through the concept of *mobile-cloud computing* (MCC), which partitions mobile applications between mobile and cloud platforms for execution, by dynamically offloading computation to cloud hosts.

Achieving high performance with mobile-cloud computing requires optimal partitioning of the mobile application components between the mobile and cloud platforms based on dynamic runtime conditions. Recent work on this problem has resulted in frameworks with various partitioning and optimization techniques. However, most of these frameworks impose strict requirements on the cloud side, such as a full clone of the application code or special application management software, hindering wide applicability in public clouds. The other major obstacle for wider adoption of MCC is the security risks associated with sending sensitive data and code to an untrusted platform. In order to provide complete security, the application should ensure all communication with/execution on the cloud platforms are trusted. The performance requirements of real-time MCC call for a generalized computation offloading model that requires minimal involvement of the mobile platform for monitoring of offloaded computation, where all decision-making and integrity checks use lightweight components.

In this paper we present a mobile-cloud computation model based on autonomous agent-based application modules, which are augmented with integrity verification units for dynamic detection and reporting of code tampering. Through experiments with a real-world application, we show that self-protecting mobile agents are promising tools for performance and security management in mobile-cloud computing.

II. Related Work

Early work in dynamic mobile-cloud computing models includes CloneCloud [1] and MAUI [2], both of which partition applications using a framework that combines static program analysis with dynamic program profiling, and optimizes execution time and energy consumption on the mobile device. The disadvantage of these approaches is that they require a copy of the whole application code/virtual machine at the remote execution site, which is a strict requirement for public clouds and makes the application code vulnerable to analysis by malicious parties on the same platform. Lin et al. [3] present a model for energy-aware task scheduling on mobile devices, where tasks are assigned to cores on the device or a cloud resource based on their precedence requirements. The ThinkAir [4] framework provides better scalability and parallelism features than its predecessors, however it still requires the existence of the complete application code on the cloud server, exhibiting the aforementioned disadvantages. All these approaches lack a mechanism for integrity protection.

The two main approaches to provide security in MCC are (a) ensuring the security of the cloud platform on which the mobile code will execute for all users of that platform and (b) ensuring the security of the mobile code and data sent to the cloud platform for execution, without relying on the trustworthiness of the cloud platform. One approach taken by previous research [5], [6] in mobile code security and secret program execution is homomorphic encryption, which operates with encrypted functions obtained by homomorphic transformations. However this approach was only shown to be usable for polynomial and rational functions, limiting its use.

Zhang et al. [7] propose an authentication and secure communication framework for elastic applications, which consist of weblets running on a mobile device and cloud nodes concurrently. Their proposed method leverages elasticity managers on the cloud and the mobile device to establish a shared secret between weblets, to provide authentication between the weblets. While this is one of the few models for secure MCC, the authors do not mention implementation details.

Most of the existing security solutions for mobile-cloud computing focus on the confidentiality of mobile data offloaded to the cloud. Frameworks for protecting the runtime integrity of code offloaded to the cloud are still at a mostly immature state. Although not tailored for MCC, one state-of-the-art approach for protecting the integrity of computation is the *software guards* algorithm proposed by Chang and Atallah [8]. This algorithm is based on the partitioning of program code into regions, where each region is either user code, a checker guard that checks the integrity of a code region/other guard code or a responder guard that replaces a tampered code region with the original code for that region. The two main issues with this approach are that integration of an increased number of guards into the program code results in increased code size and the integration of guard code into the program could result in significantly lower performance, especially if the guard code dominates the program, making it unfit for real-time MCC.

III. Autonomous Agents-Based Mobile-Cloud Computing (AAMCC)

In our proposed computation-offloading model, an autonomous application module is a chunk of application code packed in an autonomous (mobile) agent that is executable on a cloud virtual machine instance (VMI). Figure 1 shows a high level view of the agent-based MCC architecture. Each module of a mobile application in AAMCC is either an agent-based application module (JADE [9] agents in the prototype) that is offloadable to the cloud, or a native application component. When a mobile application is launched, the *execution manager* contacts the *cloud directory service* to get a list of possible *cloud hosts* (EC2 [10] machine instances in the prototype) for offloadable application modules to run on. After this step, an execution plan containing offloading decisions for the agent-based modules is created by the execution manager. If the execution plan requires offloading a particular application module, a bridge is formed between the caller of that module and the *cloud host* selected by the execution manager, through which the offloaded module migrates to the container in the host, carrying along its input parameters. Upon migration, the module starts executing and communicates its output data to the caller through the same bridge. The details of the execution manager's optimization model for migration decisions can be found at [11] and are omitted here due to space limitations.

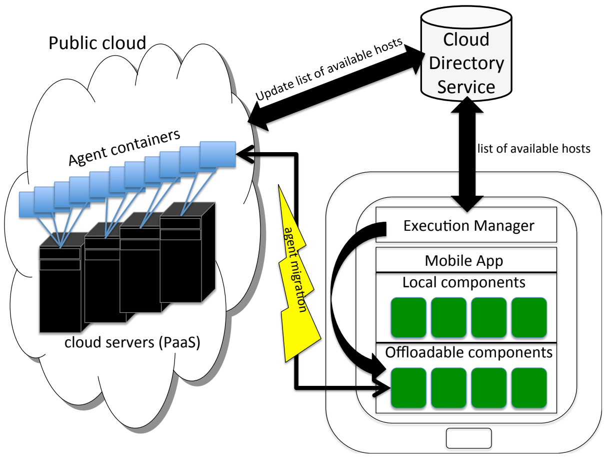

Fig. 1: High level view of AAMCC architecture.

IV. Self-Protecting Agents for Tamper Resistance

The approach we propose for tamper-resistant execution of offloaded code in the cloud is based on augmenting the mobile agents sent to a cloud platform with self-protection capability, using aspect-oriented programming (AOP) [12] based integrity checkpoints distributed throughout the agent code (placed around every method call using [13]) to ensure timely detection of tamper. Upon tamper detection, the agent stops execution, moves to a different platform and resumes execution from the last integrity-verified checkpoint. The self-protection method relies on the following two main ideas:

1) Tamper-checking guards: As the agent code is executing in the cloud, the integrity of the code is checked using *introspection* by *software guards* placed at various *integrity checkpoints*, which report to the mobile platform. The method used for introspection is the augmentation of the program with code that computes a hash value over a code region and compares it to the expected value for that region [14].
2) Guard tamper tracking code: Each time a software guard checks for tamper, its own hash value is saved in agent data. The agent accumulates the guard hash values in a variable, which forms the key for *authenticated encryption* [15], [16] (with the Galois/Counter (GCM) mode of operation [17]) of the computation result sent to the mobile platform. The mobile platform is capable of computing the key value independently, therefore it can decrypt the message received to extract the result from the autonomous application module.

Using an encryption key formed by the dynamic hash values of the software guard code in the authenticated encryption – which provides both integrity and authenticity of the result– has the following implication: For the correct encryption key to be formed, the guard hash value at each integrity checkpoint needs to be integrated into the key and the integrated value needs to match the true (original) hash value for that guard. If the result received from the mobile agent cannot be decrypted by the mobile platform, either the message containing the result was modified in transit or it was not encrypted with the correct key on the cloud platform. In either case, the result cannot be integrated into the program on the mobile platform. In the proposed security approach, accumulation of guard hash values during agent code execution is a stealthy tamper-resistance mechanism in the sense that it does not involve any checks (conditional expressions) that are prone to detection by pattern matching methods used by attackers. This incremental formation of the key serves two purposes:

1) The encryption key is not revealed until the end of program execution, which makes it harder for an attacker to dynamically swap the value of the corresponding variable with the original key value to hide tampering.
2) The mobile platform can ensure that all integrity verification blocks in the offloaded module were executed in the case of a correct key value, which allows for checking the authenticity of the results received.

Let H be a collision-resistant hash function used to calculate the guard hash values. The algorithm to compute the encryption key is provided in Algorithm 1.

Algorithm 1: Encryption key generation

```
input  : $guardCode$: guard bytecode array
         $H$: secure hash function
         $F$: guard hash accumulation function
output: encryption key for autonomous application
        module result

$numBlocks \leftarrow guardCode.size$;
$key \leftarrow 0$;
$mostRecentGuard \leftarrow 0$;
while $mostRecentGuard < numBlocks$ do
    $hashOfGuard \leftarrow$
      H($guardCode[mostRecentGuard]$);
    $key \leftarrow$ F($key, hashOfGuard$);
    $mostRecentGuard \leftarrow mostRecentGuard + 1$;
end
```

Figure 2 shows the UML activity diagram of the execution lifecycle of an autonomous agent in the proposed model. The main steps taken by the agent during execution are as follows:

1) The running *guard hash* and *agent state* values for mobile agent *MA* are reset.
2) MA is sent to a selected VMI for execution.
3) The integrity of the code and data of the MA is checked before starting execution on the cloud platform.
4) If the integrity of MA was preserved during the transfer, MA starts executing the first code block in the application module and goes onto Step 5. Otherwise, a new VMI *V* is selected, a new MA is sent to *V*, and execution restarts from Step 1.
5) The software guard for the most recently executed code block checks the runtime integrity of the block. If no tampering is detected, the running guard hash value is XORed with the hash of the guard code performing the most recent check and the agent state is updated to the last integrity checkpoint. Step 5 is repeated until there are no more code blocks to execute. If tampering is detected by the guard, a new VMI *V* is selected, MA sends a message to the mobile platform including the tamper report and its new VMI address, moves to V and resumes execution from the last tamper-free checkpoint.
6) The computation result is encrypted with the guard hash value and sent to the mobile platform.
7) If the mobile platform is able to successfully decrypt the result message received from MA, it integrates the result into the program. If the decryption of the result message fails on the mobile platform, the whole process of MA's execution is repeated starting with Step 1.

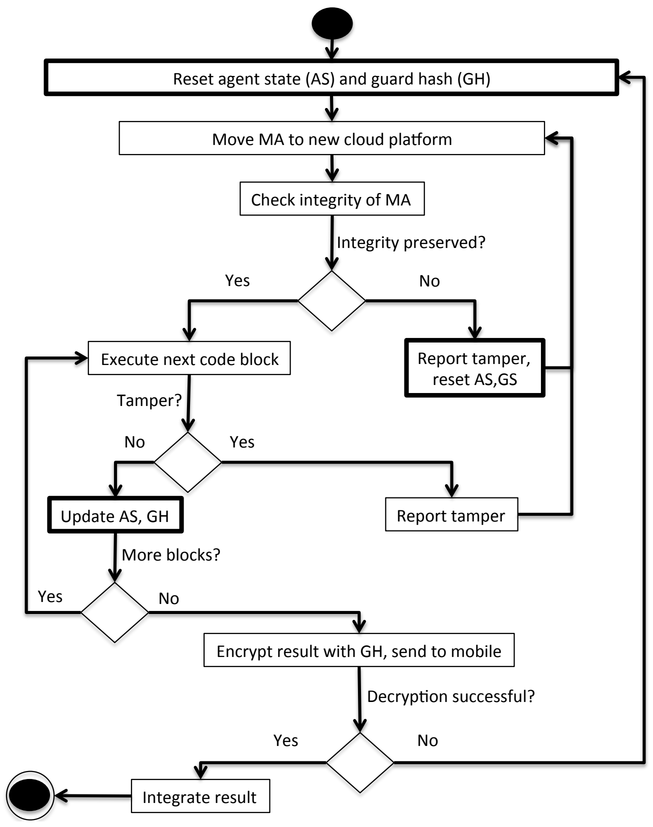

Fig. 2: Activity diagram for tamper-resistance mechanism.

V. Security Discussion

A. Resilience against Application Code Tampering

In the proposed model, the guards detect tampering with agent code as soon as the code section that was tampered with is executed, which means the time of detection is dependent on the location (order) of the tampering in the program. In a more complex attack scenario, an adversary can tamper with the guard code protecting a specific code segment in addition to the code segment itself, in a way that will cause the application code tampering to go undetected. The resilience of the approach against kind of attack is discussed below.

B. *Resilience against Guard Code Tampering*

For this attack, we consider an adversary who modifies guard code in the agent to prevent the detection of tampering with the application code. If the adversary is able to successfully suppress the tamper-reporting mechanism, the agent will continue execution on the platform until program completion. Modification in the guard code during or before runtime will however cause an incorrect hash value for the corresponding guard, which will result in a ciphertext that cannot be decrypted by the mobile platform, making the approach resilient against these types of attacks.

C. *Resilience against Communication Tampering*

According to the authenticated encryption scheme used in the model, an attacker would need to have knowledge of the secret shared key of the mobile platform and the autonomous agent in order to be able to impersonate the agent in a message sent to the mobile platform. Therefore, the model provides protection against any tampering with the communication content between the mobile platform and the agents.

VI. Performance Evaluation

A. *Performance of Agent-Based Computation Offloading*

To evaluate the performance of the proposed computation model, we performed experiments with *NQueens puzzle*, which is the problem of placing n chess queens on an n x n chessboard such that no two queens can attack each other. In these experiments, we used an NQueens solver, which finds all possible solutions to the puzzle for different number of queens using a recursive backtracking algorithm and returns the number of solutions. The experiments were performed using a Motorola Atrix 4G [18], where the cloud host was a medium VMI in EC2. Figure 3 provides a comparison of the device-only vs. offloaded execution times. While the execution times are close to each other for up to 12 queens, offloaded execution achieves significantly better performance than on-device execution for more than 13 queens. Actually, it outperforms on-device execution by about 15 times when the number of queens is 15.

B. *Tamper Resistance Overhead*

We also performed experiments to evaluate the performance overhead of the proposed tamper-resistance mechanism for the makespan of the NQueens solver. Figure 4 shows the comparison of the makespan of the NQueens application with code offloading in the case of no tamper-resistance mechanism present and the makespan of the same application with the proposed tamper-resistance mechanism integrated, for different number of queens. As seen in the figure, the difference between the average makespans for the two cases is always less than 2.2 sec, which is less than 3.6% of the total execution time for the case with 15 queens. The average makespan for on-device execution for the same case is 872 sec, therefore

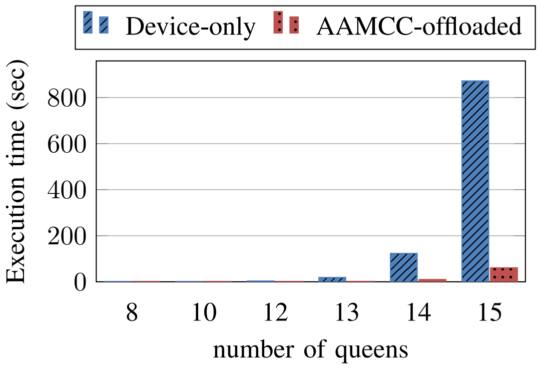

Fig. 3: NQueens makespan with device vs. AAMCC.

AAMCC with tamper detection still outperforms device-only execution, and the tamper-resistance overhead is negligible.

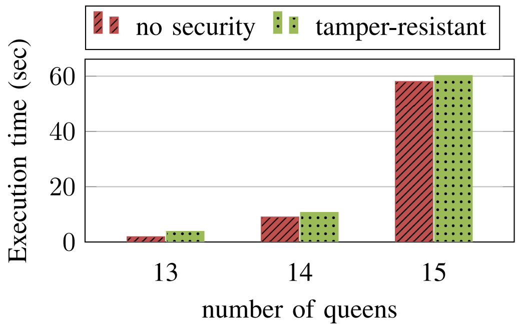

Fig. 4: Comparison of execution times for the NQueens solver application with vs. without tamper resistance.

VII. Conclusion

In this paper we presented a mobile-cloud computation model based on autonomous agent-based application modules using various integrity checkpoints of code introspection to detect and report tampering. We showed through experiments with an Android application that autonomous agent-based computation offloading is an effective tool for MCC, providing application execution times significantly lower than on-device execution. The proposed tamper-resistance model was also shown to incur very low runtime overhead and successfully detect load-time and runtime code tampering attacks. The mobile-cloud computation model presented in this paper is promising to meet high-performance computing and integrity assurance needs in real-time mobile-cloud computing. The principles laid out in this work regarding self-protecting mobile agents also provide a basis for simplified performance and security management of applications and services in cloud computing, which can utilize an agent-based structure for migration between platforms and self-performance monitoring.

Acknowledgment

The authors would like to thank Dr. Lotfi ben Othmane for his valuable comments and discussions.

References

[1] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the 6th ACM European Conference on Computer Systems (EuroSys'11)*, 2011, pp. 301–314.

[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, 2010, pp. 49–62.

[3] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Energy and performance-aware task scheduling in a mobile cloud computing environment," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD'14)*. IEEE, 2014, pp. 192 – 199.

[4] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM'12)*, 2012, pp. 945–953.

[5] M. Brenner, J. Wiebelitz, G. von Voigt, and M. Smith, "Secret program execution in the cloud applying homomorphic encryption," in *Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies (DEST'11)*, 2011, pp. 114–119.

[6] T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security*. Springer, 1998, pp. 44–60.

[7] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *Proceedings of the ACM Workshop on Cloud Computing Security*, 2009, pp. 127–134.

[8] H. Chang and M. J. Atallah, "Protecting software code by guards," in *Security and Privacy in Digital Rights Management*. Springer, 2002, pp. 160–175.

[9] Telecom Italia Lab. Java agent development framework. http://jade.tilab.com/. Accessed: 2016-02-10.

[10] Amazon Web Services Inc. Amazon elastic compute cloud. http://aws.amazon.com/ec2. Accessed: 2016-02-10.

[11] P. Angin and B. Bhargava, "An Agent-based Optimization Framework for Mobile-Cloud Computing," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 2, pp. 1–17, 2013.

[12] N. Påhlsson. Aspect-oriented programming. http://oberon2005.oberoncore.ru/paper/np2002.pdf. Accessed: 2016-02-10.

[13] AspectJ. http://eclipse.org/aspectj/. Accessed: 2016-02-10.

[14] C. Collberg and J. Nagra, *Surreptitious software: Obfuscation, watermarking, and tamperproofing for software protection*. Addison-Wesley Professional, 2009.

[15] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *Journal of Cryptology*, vol. 21, no. 4, pp. 469–491, 2008.

[16] J. Katz and M. Yung, "Unforgeable encryption and chosen ciphertext secure modes of operation," in *Fast Software Encryption*. Springer, 2001, pp. 284–299.

[17] D. McGrew and J. Viega, "The Galois/Counter mode of operation (GCM)," *NIST Special Publication 800-38D*, 2004.

[18] Motorola Mobility LLC. Motorola Atrix 4G. http://www.motorola.com/us/consumers/Motorola-ATRIX-4G/72112,en_US,pd.html. Accessed: 2016-02-10.