

Preserving Data Query Privacy in Mobile Mashups through Mobile Cloud Computing

Rodney Owens
Department of SIS
UNC Charlotte
Charlotte, NC 28223
Email: rvowens@uncc.edu

Weichao Wang
Department of SIS
UNC Charlotte
Charlotte, NC 28223
Email: weichaowang@uncc.edu

Abstract—Mobile mashups promise great data aggregation and processing capabilities for all end users. During the data collection procedures, some data providers fail to protect confidentiality and privacy of user queries and transmit information in plain text. This enables attackers to eavesdrop on networks and compromise user information. Since mobile mashups can adopt server-side, client-side, or hybrid architectures, no one-size-fits-all solutions can be designed to solve this problem.

In this paper, we propose to design two mechanisms using mobile clouds to preserve data query privacy in mobile mashups. For server-side mashups, we propose to use dynamically created virtual machines as proxies to process data collection and aggregation in order to prevent information leakage through eavesdropping. For client-side mashups, we propose to use live migration of the application level virtual machines into mobile cloud to hide the data collection and aggregation procedures from attackers. We will evaluate the proposed approaches through both analysis and experiments on real platforms.

I. INTRODUCTION

With the proliferation of smart phones, numerous new widgets based on the Web 2.0 standard are developed. Among these applications, many are built upon the mashup technique. A mashup application is a system which combines the local contents with the information from other web providers, such as Google, Ebay, and Craigslist into an integrated information presentation. The published web service interfaces such as Google Maps and Yahoo! Flickr greatly simplify the creation of the mashups by hiding their internal complexity.

While the mashup technique enables the development of new applications, it also raises security concerns. Previous research has been focusing on the prevention of information leakage among data providers in the same mashup pages [1], [2]. In this paper, we investigate the security problem from a new perspective. As an example, a health care company develops a mobile mashup: a patient needs to input only an address and a disease. The mashup will search in the company's database to locate all doctors specialized in this disease within 10 miles of the input address. It will then retrieve information from two other providers to label these doctors on a map and show their patient reviews. The company knows that the privacy of health information is critical. Therefore, it encrypts the query results from the database. Unfortunately, the map and doctor review companies transmit information in plain text. Now if an attacker monitors the returned data of this

mashup, he can easily derive the disease that the user might have. This kind of information leakage is beyond the control of the developer of this mashup since she/he cannot determine the interface design of the data providers.

In parallel to the development of mobile mashups is the deployment of mobile cloud computing. Mobile cloud provides transparent services to portable devices in order to resolve the discrepancy between the limited resources of such devices and the demands of innovative applications. While the security of mobile cloud has attracted interests from researchers [3], [4], using it to provide security services for other applications deserves more efforts. At least two reasons make us believe that mobile cloud is an appropriate solution to privacy preservation in mashups. First, mobile cloud provides transparent services to end users so that we can hide the sources of aggregated information. Second, mobile cloud allows components of the mashup software to move freely between end users and cloud infrastructure. In this way, it becomes more difficult for attackers to track the information flow.

Recent research [5] shows that for mobile devices client-side, server-side, and even hybrid mashups are all popular. Therefore, in this paper we propose to design two mechanisms for privacy preservation in data acquirement for client-side and server-side mashups respectively. For server-side mashups, the server can dynamically create virtual machines in the mobile cloud to work as proxies to handle information collection and integration. The nondeterministic location of a virtual machine makes it extremely difficult for attackers to eavesdrop on the communication contents. For client-side mashups, we propose to use the techniques described in [6], [7], [8] to decompose the application into multiple components and move the data collection and aggregation part into the cloud. The overhead and safety of both approaches will also be studied.

Contributions of the paper can be summarized as follows. First, we explore using mobile cloud to improve information privacy in mobile mashups. Second, we have designed different mechanisms for client-side and server-side approaches so that they can be integrated with various widgets. Last but not least, we evaluate the proposed approaches through both analysis and experiments on real platforms.

The remainder of the paper is organized as follows. In Section II we will discuss the related work. In Section III we will

present the details of the proposed approaches. Section IV will investigate the performance and security of the approaches. Finally, Section V will conclude the paper.

II. RELATED WORK

Previous research on mashup security focuses on preventing information leakage among multiple sub-frames belonging to different data providers. The approaches can be classified into three groups. In the first group, researchers define the “object-capability” languages in which access control, message passing, and object reference are all tightly managed. For example, Caja [9] implements a subset of javascript to protect third party contents in web applications. The second group use the concept of sandboxing [2], [10]. They encapsulate information from each source with a subspace. Specially designed communication channels are then implemented among these components. In the third group, a complete set of communication protocols are implemented so that all sub-frames must communicate with each other through the protocols. Examples of the approaches include OMOS [11] and SMash [1].

Using proxy-based approaches to preserve privacy in networks has been investigated in different environments. For example, both Crowds [12] and Hordes [13] use proxies to defend against traffic analysis attacks in Internet. Similar technique has been adopted to achieve anonymity in HTTP requests [14] and P2P networks [15]. The techniques of cloud and virtualization make the dynamic establishment and maintenance of a proxy very easy.

Decomposing a web application into multiple components and moving some of them into cloud can serve different purposes of mobile computing. The early models such as Dryad [16] and IBM SPL [17] all enable an application to be composed by connecting modules. CloneCloud [6], MAUI [7], and COMET [8] focus on reducing energy consumption and computation overhead by offloading some components of an application into the cloud.

III. THE PROPOSED APPROACHES

In this section, we present the details of the proposed approaches. We will first elaborate on the system assumptions and the attacker model. We will then discuss the privacy preservation procedures for server-side and client-side mashups respectively.

A. System Assumptions and Attacker Model

Figure 1 illustrates the application scenarios that we need to protect. In this environment, end users depend on mashups through mobile cloud to get the needed information. The figure shows both server-side and client-side mashups. We assume that end users have enough computation power to support secure encryption algorithms. However, neither the end users nor the servers have any control over the communication standard with the data providers. A data provider can choose to return data in plain text and any eavesdropper will be able to understand the contents.

We assume that an attacker will be able to eavesdrop on the incoming and outgoing traffic of the mobile end users. If a mashup server has a fixed position and static IP address in the infrastructure, the attacker will be able to eavesdrop on the node as well. The attacker does not have the computation power to compromise the secure encryption algorithms adopted by the mobile devices. In theory, for a dynamically created virtual machine (VM) in the mobile cloud, the attacker could trace and locate the node, launch a co-residence attack [18], and steal information from the VM through side-channel attacks [19]. However, the analysis in subsequent sections will show that the difficulty level and overhead of such attacks are very high. Therefore, we assume that the attackers cannot eavesdrop on a dynamically created VM in the mobile cloud.

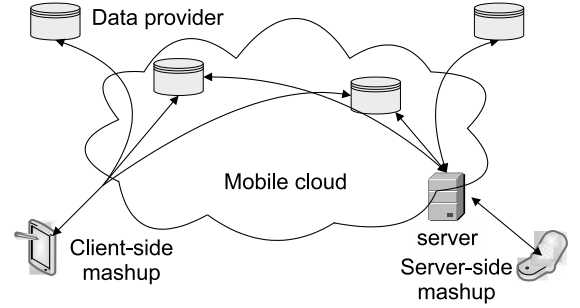


Fig. 1. Mashup scenarios in mobile cloud.

B. Privacy Preservation for Server-side Mashups

An attacker can eavesdrop on the network traffic of the mashup server if it has a fixed position and static IP address. However, it becomes much more difficult to locate and monitor a dynamically created VM in the mobile cloud. In this part, we will design a mechanism based on this observation.

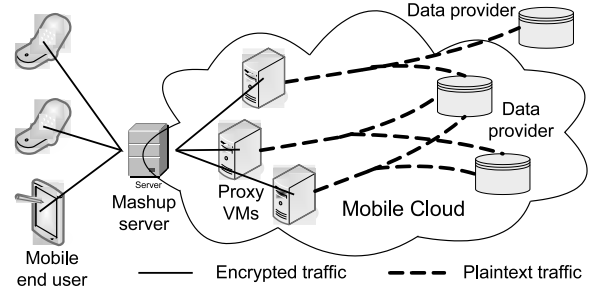


Fig. 2. Privacy preservation for server-side mashups.

Figure 2 illustrates the proposed mechanism to protect data privacy in server-side mashups. Instead of letting the mashup server directly connect to data providers, we request it to initiate multiple virtual machines in the cloud to serve as proxies. Since the VMs are created by the server, it can determine the communication methods with the VMs. Therefore, secure encryption algorithms can be used to protect network traffic on this segment. Note that traffic between the proxies and data providers could still be transmitted as plain text.

Through monitoring the outgoing traffic from the server, attackers can figure out the IP addresses of the proxies. However,

eavesdropping on the proxies will be a very difficult job that demands a lot of resources from the attackers. Experiments in [18] show that the attackers need to launch 40 to 80 virtual machines to achieve co-residence with one of the proxies if the mobile cloud owner does not adopt any VM placement policies to prevent cartography. This number could be even larger if the cloud owner randomly places VMs in the infrastructure.

Even after putting a malicious VM onto the same physical box as a proxy, the attacker still faces the challenge to derive out the network traffic of the proxy through side channel attacks. Please note that this is different from the cross VM private key extraction attack [20] in the following aspects. First and most importantly, in [20] the attacker must be able to remotely and repeatedly activate the encryption algorithm in the target VM and put the secret key into cache. In our scenario, the mashup procedure is initiated by the end user as a one-time execution so the attacker has no control over its happening. Second, in [20] the attacker knows exactly where the key will reside in the cache through code analysis. In our scenario, the returned information for the mashup could be put at any place in the cache. Last but not least, in [20] the attacker must be able to regain control of the CPU sufficiently frequently to conduct side-channel observations through the trigger of Inter-Processor Interrupts (IPI). Such functionality may not be enabled in the mobile cloud environment.

The mashup server can create multiple VM proxies to further increase the difficulty for the attackers to trace and intercept network traffic. Here we propose two dispatching mechanisms for the server to place requests from end users to different proxies. In the “vertical” dispatching scheme, every request from a mobile user will be treated as a separate transaction and assigned to a single proxy. Round-robin or least workload based placement can be used to maintain balance among the proxies. The advantage of this approach is that when the server determines to terminate a proxy, we can stop assigning new requests to it. Therefore, the switch to a new proxy can be accomplished smoothly.

In the “horizontal” dispatching scheme, each proxy will be in charge of handling all interactions with one or a few data providers. In this way, every user request needs the collaboration of multiple proxies and only the server itself can integrate the returned data. The advantage of this approach is that when an attacker intercepts the network traffic to a proxy, it can get access to only a part of the mashup result. Since there could be many returned messages from the same provider, the attacker cannot identify the request that it is interested in. The disadvantage, however, is that a malfunctioned proxy may impact the processing of a large number of user requests. The designer of a mashup application could choose the dispatching mechanism that fits her/his needs the best.

C. Privacy Preservation for Client-side Mashups

In theory, we can adopt a similar technique to protect the privacy of information in client-side mashups. However, products such as Mobile Virtualization Platform by VMware cannot yet provide full control of a virtual environment through

an app on a thin client such as a smart phone. Therefore, we need to design a new mechanism to protect client-side mashups. Previous research [6], [7], [8] shows that partitioning the execution of a mobile application and offloading some operations into the cloud has become a practical solution. In this paper, we propose to build an approach upon COMET [8]. The basic idea is shown in Figure 3.

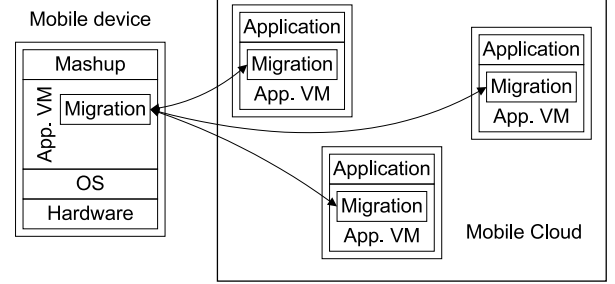


Fig. 3. Live migration based privacy preservation.

In this approach, the mobile device will host an application level virtual machine such as JVM, Microsoft .NET CLR, or the DalvikVM in Android to execute the mashup application. The VM will contain a partitioning and migration management component. When the mashup application is ready to pull information from providers, the partitioner will suspend the execution on the mobile device, and migrate the thread to a VM clone that is hosted in the cloud. The migrated thread will execute on the clone to acquire and integrate information from different providers. Finally, the thread will return to the mobile device and merge the remotely created states into the original process.

This approach does not depend on a static server since the VM clone can be hosted by any physical box in the cloud. The mobile device can execute the mashup in an application level VM until the information acquirement phase. It will then package the states of the VM and port it to its clone. Since a mobile user can randomly choose the placement of its clone, it is very difficult for an attacker to intercept the mashup traffic unless it can eavesdrop on the whole cloud simultaneously. Since the thread migration traffic is encrypted, the attacker will not learn anything by monitoring the inbound and outbound traffic of the mobile user.

Most functionality needed for this approach can be found in the implementation of COMET. We can choose the start position of thread migration in the mashup application so that all communication with the data providers is conducted by the clone VM. The functionalities of suspension, porting, resume, and merge have been included in the DalvikVM of Android. Since the mobile end users can randomly choose machines in the cloud to host their clones, we do not expect unbalanced workload to occur frequently.

IV. EXPERIMENT RESULTS AND EVALUATION

In this section, we will present the evaluation results. We will focus on the changes in computation and communication overhead in both mobile devices and servers. We will also study power consumption of the proposed approaches.

A. Server-side Mashups

The proxy based approach has no impact on the computation or communication overhead at the mobile devices since all operations are conducted by the server. The end users, however, could expect longer mashup response time that is introduced by the extra communication segment. The extra overhead at the server comes from the following aspects. First, the server needs to initiate, manage, and terminate the proxy VMs in the cloud. Second, to prevent attackers from eavesdropping on the server, it has to encrypt the communication traffic with the proxies. Last but not least, depending on the adopted dispatching algorithm for proxies, the server may need to integrate returned data from different providers to generate the mashup results.

To assess the overhead in real environments, we setup an evaluation platform. The mashup server is a Ubuntu virtual machine with 1GB of RAM and an allotment of two processors from a Dell Optiplex 980. We choose the open source mashup WSO2 as the application so that we can easily change its configuration. The mashup server's Internet connection was 1.47Mbps down and 490Kbps up. The proxies were running on remote servers with VMWare ESXi as the hypervisor. Each proxy was assigned one processor with 512 MB of RAM, using an Internet connection with 9.59Mbps down and 1.34Mbps up. All connections for the mashup server to data providers on the Internet were routed through the remote proxies either encrypted with Blowfish (128 bit key size) or in plain text, both by point-to-point links in OpenVPN. Our evaluation focuses on the increase in response time that is introduced by the communication segment between the mashup server and the proxies.

TABLE I
DELAY OF PROXY-BASED SERVER-SIDE MASHUP.

	Measured delay (ms)		
	maximum	minimum	average
baseline case	820	274	293
proxy w/o encryption	1890	519	613
proxy with encryption	1954	523	615

Table I shows the transmission and processing delay that is measured at the end user. It represents the time duration between the sending of a request and the return of corresponding mashup results. The decryption delay at the end user is not included since it is not impacted by our approach. The average amount of returned traffic from data providers is approximately 700KB. To better differentiate the delays caused by communication and computation overhead, we conduct three groups of experiments. In the first group, the mashup server will directly connect to the data providers and retrieve information. This is the baseline case. In the second group, the mashup server will retrieve data with the help of proxies. However, the data traffic between the proxies and mashup server is not encrypted. In the last group, the proxies will encrypt the returned data and then deliver it to the mashup server. 250 experiments for each group are conducted at different time in a weekday.

From the table we find that the proposed approach roughly doubles the waiting time of the end users. This is mainly caused by the long path between the mashup server and proxies. Comparing the second and third groups of experiments, we find that the communication delay between the mashup server and proxies dominates the increase in response time. The encryption/decryption of the returned data does not impact the delay to a large extent thanks to the computation resources available to the server.

B. Client-side Mashups

For client-side mashups, most overhead introduced by the proposed approach will be put upon mobile devices. Therefore, we need to carefully evaluate its impacts on these thin clients before this approach can be widely deployed. In the following discussion, we will focus on the increases in communication overhead and delay, computation overhead during the migration of the application level VM, and power consumption.

TABLE II
OVERHEAD COMPARISON BEFORE AND AFTER OFFLOADING.

	Overhead on the mobile phone	
	client-side mashup	offload to mobile cloud
communication overhead	(1) directly get contents from data providers;	(1) VM state migration and merge; (2) reception of the aggregation results;
computation overhead	(1) aggregate the fetched data;	(1) decryption of the aggregation results; (2) state separation and merge for VM migration;

To assess the proposed approach on real mobile platforms, we build our prototype upon the COMET code offload system [8]. Here a Samsung Captivate smart phone with Cyanogen-Mod (CM10) OS is used as the mobile device. The mobile cloud service (app-level VM migration and execution offload) is provided by a PC with 4GB RAM and 2.4GHz CPU. The smart phone uses WLAN to connect to the Internet and the code offload server. The measured phone download bandwidth is 371KB/s while the upload bandwidth is 522 KB/s (download is slower since it is flash based). We choose two mashup applications to evaluate our approach. The "Landmark Manager" app aggregates information from Facebook, Twitter, Google Places, Yelp, YouTube, Flickr, Groupon, and many other sites. It triggers a large amount of network traffic and relatively heavy computation load. The second app we choose is EarthAlbum, which is a lightweight Google Map and Flickr mashup. Table II compares the communication and computation overhead on the smart phone when it runs the mashup directly or offloads the task to mobile cloud.

The majority of the communication overhead comes from the reception of the aggregation results, and the migration and return of the application level VM states. In [6], [7] researchers have tested about 10 different applications on the Android system and found that on average the state migration will cause 10KB to 25KB traffic. The aggregation results of mashup applications usually have the size of several hundred kilo-byte to several mega-byte. For example, our experiments

with Landmark Manager show that if the smart phone directly gets data from various providers, the download traffic size is about 700KB. If the aggregation is offloaded to mobile cloud, the returned results to the smart phone have the size of 4.8MB. On the contrary, for the EarthAlbum mashup, both the raw data and the aggregation result have the size of 300KB.

To prevent information leakage through eavesdropping, data traffic between the mobile device and the offload server must be encrypted. We conduct measurement on the smart phone with RC4 with 64bit key size and find out that a mobile device with 1GHz CPU running Android system can encrypt 5MB data with AES or RC4 within 2 seconds.

Our experiments show that the proposed approach has a response time comparable to the scheme when the mobile phone directly executes the mashup. For example, for the EarthAlbum app, the proposed approach has the average response time of 8.52 seconds while the direct execution on the smart phone has an average response time of 7.89 seconds. For the Landmark Manager app, the proposed approach shortens the response time by about 35% since the server in the mobile cloud can aggregate the data more efficiently.

TABLE III
COMPARISON OF THE POWER CONSUMPTION RESULTS.

	Power consumption on the mobile phone per execution (% of the battery capacity)	
	client-side mashup	offload to mobile cloud
EarthAlbum	0.147%	0.127%
Landmark Manager	0.55%	0.327%

Since the most valuable resource for mobile devices is energy, we must carefully assess the power consumption of the proposed approach. Our measurements show that for the offloading approach, the most energy consuming operation is the rendering of the final results. On the contrary, for the direct execution scheme the most energy consuming operation is the aggregation procedure. From the results shown in Table III, we find that the proposed approach reduces power consumption on the mobile device by offloading the mashup procedures to the cloud environment. Our experiments show that the proposed approach brings improvements in both user privacy preservation and power usage efficiency.

V. CONCLUSION

In this paper, we study the problem of privacy preservation for mobile mashups. Since end users do not have control over the communication protocols with the data providers, new mechanisms must be designed to defend against eavesdropping attacks. For server-side mashups in mobile clouds, we propose to use a proxy based approach to protect confidentiality of the communication between the mashup server and data providers. Experiments show that except for the lengthened response time, other aspects of the application performance are not impacted. For client-side mashups, we propose to use live migration of application level virtual machines to hide the data acquirement and aggregation procedures from eavesdroppers. We investigate the computation, communication, and power consumption overhead of the approaches.

Immediate extensions to our approaches consist of the following aspects. First, we plan to test our approach for client-side mashups and collect feedbacks on user experiences. This effort will allow us to assess the potential for the deployment of our approaches in real mobile cloud environments. Second, we plan to explore the integration of the two approaches so that application developers and end users do not have to explicitly distinguish server-side mashups from client-side applications.

REFERENCES

- [1] F. De Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama, "Smash: secure component model for cross-domain mashups on unmodified browsers," in *WWW*, 2008, pp. 535–544.
- [2] S. Van Acker, P. De Ryck, L. Desmet, F. Piessens, and W. Joosen, "Webjail: least-privilege integration of third-party components in web mashups," in *Annual Computer Security Applications Conference (AC-SAC)*, 2011, pp. 307–316.
- [3] D. Huang, X. Zhang, M. Kang, and J. Luo, "Mobicloud: Building secure cloud framework for mobile computing and communication," in *IEEE International Symposium on Service Oriented System Engineering (SOSE)*, 2010, pp. 27–34.
- [4] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *ACM CCSW*, 2009, pp. 127–134.
- [5] V. Agarwal, S. Goyal, S. Mittal, and S. Mukherjee, "A middleware framework for mashup device and telecom features with the web," IBM Research, RI 10009, Tech. Rep., 2010.
- [6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.
- [7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.
- [8] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: Code offload by migrating execution transparently," in *Proceedings of OSDI*, 2012.
- [9] "Caja: Safe active content in sanitized javascript," a Google Research project, 2008.
- [10] C. Jackson and H. J. Wang, "Subspace: secure cross-domain communication for web mashups," in *WWW*, 2007, pp. 611–620.
- [11] S. Zarandoon, D. Yao, and V. Ganapathy, "Omos: A framework for secure communication in mashup applications," in *Annual Computer Security Applications Conference (ACSAC)*, 2008, pp. 355–364.
- [12] M. K. Reiter and A. D. Rubin, "Crowds: anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, 1998.
- [13] C. Shields and B. N. Levine, "A protocol for anonymous communication over the internet," in *ACM CCS*, 2000, pp. 33–42.
- [14] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer, "Consistent, yet anonymous, web access with lpwa," *Commun. ACM*, vol. 42, no. 2, pp. 42–47, 1999.
- [15] V. Scarlata, B. N. Levine, and C. Shields, "Responder anonymity and anonymous peer-to-peer file sharing," in *International Conference on Network Protocols (ICNP)*, 2001, pp. 272–280.
- [16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM European Conference on Computer Systems*, 2007, pp. 59–72.
- [17] M. Hirzel, H. Andrade, B. Gedik, V. Kumar, G. Losa, M. H. Nasgaard, R. Soule, and K.-L. Wu, "Spl stream processing language specification," Tech. Rep. RC24897 (W0907-066), IBM Research, Tech. Rep., 2009.
- [18] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *ACM CCS*, 2009, pp. 199–212.
- [19] R. Owens and W. Wang, "Non-interactive os fingerprinting through memory de-duplication technique in virtual machines," in *IEEE International Performance Computing and Communications Conference*, 2011.
- [20] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *ACM CCS*, 2012, pp. 305–316.