Securing Elastic Applications on Mobile Devices for Cloud Computing

Xinwen Zhang†, Joshua Schiffman‡, Simon Gibbs†,
Anugeetha Kunjithapatham†, and Sangoh Jeong†
†Computer Science Lab, Samsung Information Systems America
{ xinwen.z, s.gibbs, anugeetha.k, sangoh.j}@samsung.com
‡Computer Science and Engineering, Pennsylvania State University
jschiffm@cse.psu.edu

ABSTRACT

Cloud computing provides elastic computing infrastructure and resources which enable resource-on-demand and pay-as-you-go utility computing models. We believe that new applications can leverage these models to achieve new features that are not available for legacy applications. In our project we aim to build *elastic applications* which augment resource-constrained platforms, such as mobile phones, with elastic computing resources from clouds. An elastic application consists of one or more *weblets*, each of which can be launched on a device or cloud, and can be migrated between them according to dynamic changes of the computing environment or user preferences on the device. This paper overviews the general concept of this new application model, analyzes its unique security requirements, and presents our design considerations to build secure elastic applications. As first steps we propose a solution for authentication and secure session management between weblets running device side and those on the cloud. We then propose secure migration and how to authorize cloud weblets to access sensitive user data such as via external web services. We believe some principles in our solution can be applied in other cloud computing scenarios such as application integration between private and public clouds in an enterprise environment.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Security

Keywords

Elastic Application, Weblet, Cloud Computing, Security

1. INTRODUCTION

Cloud computing delivers new computing models for service providers and individual consumers including infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS), which enable novel IT business models such as resource-on-demand, pay-as-you-go, and utility-computing [6]. Although the benefits of cloud computing to enterprise consumers and service providers have been well explored, its effect on end users, applications, and application developers is still not clear. In particular, research is needed to explore the new design models and unique requirements of cloud aware applications in order to leverage their full potential.

In the scope of consumer electronic (CE) devices, applications traditionally are constrained by limited resources such as low CPU frequency, small memory, and a battery-powered computing environment. Cloud computing has the potential to radically alter how CE device software is developed and deployed while at the same time removing constraints on device functionality. In our project, we aim to design *elastic devices*, which are augmented CE devices with cloud-based functionality. A key part of this work is to provide a framework to develop, deploy, configure, and execute *elastic applications* that can run efficiently on resource constrained devices, by seamlessly and transparently making use of cloud resources whenever needed.

Figure 1 shows an overview of how an elastic mobile terminal consumes cloud resources. An elastic application can consist of one or more *weblets*, which function independently, but communicate with each other. When the application is launched, an *elasticity manager* running on the device monitors the resource requirements of the weblets of the application, and make decisions where they should be launched. Computation or communication intensive weblets such as image and video processing usually strain the processors of mobile devices, therefore they can be launched on one or more platforms in the cloud; while user interface components (UI) or those needing extensive access to local data may be launched on the device. If one weblet should be launched on the cloud, the elasticity manager talks to an *elasticity service* residing on the cloud, which arranges the execution resources of the weblet, e.g., on which cloud node it should be launched, and how much storage should be allocated. The service also returns some information after successfully launching the weblet, such as its endpoint URL. In some situations, even with heavy computational tasks, there may be times when running on the device is preferred. When, for example, the device is offline, or the media is small in size or number, or fast response is not a requirement, then it is certainly possible to consider running on the device. The mobile device and the elasticity service can work together to decide where and how the tasks can be executed. In very general scenarios, the elasticity manager can also make decisions about migrating running weblets from the device to cloud, or from cloud to device, according to changes in computing constraints on the device or changes in user preferences.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCSW'09, November 13, 2009, Chicago, Illinois, USA.
Copyright 2009 ACM 978-1-60558-784-4/09/11 ...$5.00.

During execution, the weblets of a single application can communicate with each other, e.g., to synchronize application state and exchange data as inputs/outputs, with an RPC mechanism or RESTful web services [15]. The elasticity service organizes cloud resources and delegates application requirements from mobile devices. As a service provider, the elasticity service may or may not be part of a cloud provider.

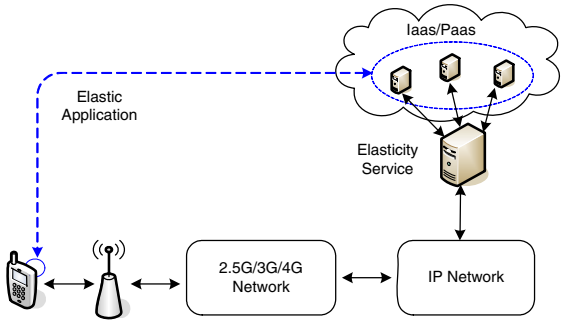

Figure 1: Overview of elastic application for mobile device. Components of an application can be executed on device and cloud platforms simultaneously.

Several challenges exist to enable this type of application. First, a new application model is needed in order to launch or migrate some parts of an application in the cloud and others on the device. The new application model should support applications partitioned into multiple components, each of which can run autonomously from the others. Besides the computation property of each weblet, communication and data dependency are also factors when partitioning an application. Second, an appropriate protocol is needed between weblets during runtime, e.g., to synchronize the state of the application, to respond to state change or user actions, and for other management issues such as resource usage accountability. Third, a set of cost objective functions are needed, which should be optimized when elastic scheduling decisions are made, such as when and where to migrate weblets. For example, although cloud-based resources can accelerate the computing task of a weblet, remote execution introduces network latency and communication overhead to the device, and further introduces monetary cost [1] to mobile users. Furthermore, security and privacy are important factors when considering some sensitive weblets and data migrating from device to cloud. One important design objective of elastic applications is to make the weblet and data migrations seamless and transparent to mobile users. Another objective is to build an infrastructure with enabling functions such as network protocols, secure communication, and resource management, such that the new elastic computing model introduces minimal extra considerations to application developers.

In this paper, based on an overview of how elastic applications work and some features of the elasticity framework, we focus on the security issues in designing elastic applications. We identify several challenges for a secure elastic runtime environment: (1) As naturally an elastic application runs in distributed manner where some weblets are in the cloud and some are on device, authentication between weblets should be provided, i.e., two weblets in one application launched by the same user/device should authenticate to each other during runtime. Typically, this authentication implies a secure communication channel between them; (2) When a cloud weblet needs to access sensitive user data, e.g., data either on the device or on another web service, authorization is needed to give the weblet access privileges, but only the minimum needed since the weblet may be running on a public and relatively untrusted environment in cloud; (3) How to build and verify a trusted runtime environment of a cloud node used for weblet execution is a fundamental problem for elastic applications and many other cloud-based applications. Essentially, we need to identify a trusted computing base (TCB) for the elastic application to deploy and run. We note that although we discuss security issues in the context of elastic applications, we believe some principles would be applicable to many other cloud users and service providers, such as, enterprise and SaaS providers. For example, we find that the integration of existing enterprise IT infrastructures and private clouds with public clouds has similar security requirements as our elastic devices.

The rest of this paper is organized as follows. In next section we overview the concept of elastic applications and the framework architecture. In Section 3 we identify general security threats and requirements for elastic applications. We then illustrate our security designs in Section 4 including weblet authentication and secure session management, secure migration, and permission authorization to cloud weblet to access external resources on behalf of mobile user. Section 5 presents some related work on security of cloud computing and computation offloading. Section 6 concludes this paper and highlights our future work.

2. ELASTIC APPLICATIONS FOR MOBILE PLATFORM

2.1 Concepts and Benefits

The objective of an elastic application is to dynamically leverage cloud computing for resource-constrained mobile devices. Rather than a simple and rigid solution where nearly all processing and storage are on the cloud, an elastic device should have the ability to migrate functionality between the device and cloud. This ability allows the device to adapt to different work loads, performance goals, and network latencies. For example, an application requiring compute resources may run locally if the work load is light, but as the work load increases, more and more of the computing is shifted off the device to the cloud.

There are several potential benefits that the elastic device concept offers to device users and application developers. For example, elastic applications need not be constrained by the current compute capabilities of mobile devices. If more compute (or storage) is needed then this can be obtained from the cloud. As devices become more powerful, compute and storage can shift back to the device. In addition, CE device compute and storage need not be designed to satisfy the most demanding applications. Device resources can be modest (and less power consuming) since the more demanding applications can acquire resources from the cloud. From a performance perspective, the ability to allocate resources in the cloud and migrate functionality gives the device great flexibility. For example, performance can be increased or optimized to fit various goals (such as responsiveness, monetary cost, or power consumption). Furthermore, application components that are partitioned for migration can also be replicated. The failure then of one instance of a replicated component need not compromise the application.

2.2 Elastic Framework Architecture

Figure 2 shows the main functional components needed to realize the elastic device concept. A typical elastic application includes a UI component and one or more weblets. On the device side, the key component is the device elasticity manager (DEM) which is responsible for configuring applications at launch time and making

configuration changes during run time. The configuration of an application includes: where the application's components (weblets) are located, whether or not components are replicated or shadowed (e.g., for reliability purpose), and the selection of paths used for communication with weblets (e.g., WiFi or 3G if such a choice exists). The elasticity manager maintains a cost model which accounts for such factors as power costs and the monetary costs resulting from network and cloud usage. The elasticity manager runs an optimizer which is responsible for determining the best application configuration given costs and user goals (such as running in a lower power mode or a high performance mode). The router passes requests from UI components to weblets. It insulates the UI logic from weblet location. When a weblet is migrated, the router will be aware of the new location and will continue passing requests from the UI to the weblet (and passing replies back to the UI). Each device also provides sensing data on the device such as processor utilization or battery state. This data is made available to the elasticity manager and is used by the cost model.

The cloud elasticity service (CES) consists of cloud manager, application manager, and sensing information collection. The cloud manager oversees resources on the cloud that are used for elastic applications. It is responsible for allocating resources from, and releasing to, the underlying cloud platform. It maintains usage information, including compute, bandwidth and storage, for the various parts of applications running on the cloud. The application manager provides functions to install and maintain applications on behalf of elastic devices, and helps to launch weblets in different cloud nodes. Sensing information refers to the collection of operational data on the cloud platform. These data are made available to the cloud manager to assist it in tracking usage. In addition to application performance data, sensing may collect information on cloud infrastructure, failures of various forms, and resource availability. As a service provider, CES provides a web service, referred to as the cloud fabric interface (CFI) to elastic devices and applications.

A node manager on each node oversees resources associated with a particular node (server) within the cloud. It communicates directly with the cloud manager and application manager. Each node runs one or more weblet containers which are the weblet runtime environments hosted on the cloud platform. Examples could include Java VM, .Net CLR, or Python VM.

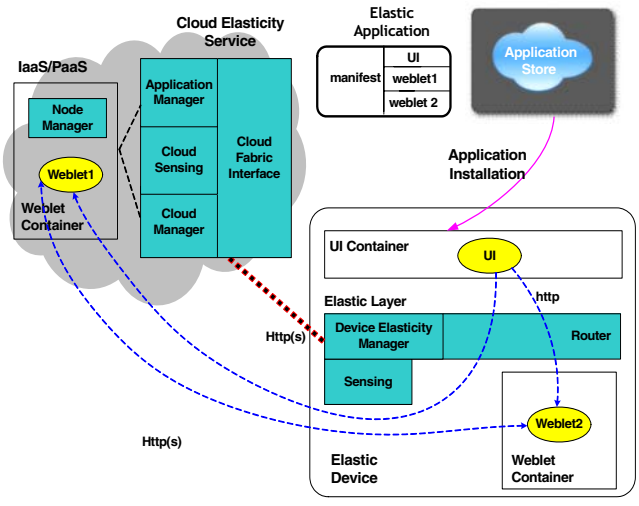

Figure 2: Elastic framework architecture.

We have developed a set of example applications based on this architecture. We build a CES web service which allocates Amazon EC2 instances [1] for our applications. Each instance is installed with a node manager service which handles weblet launching according to request from a root weblet running on a client device such as a mobile phone. The node manager also monitors the resource usage of the instance, mainly the CPU and network loads, which are used to decide whether more weblets can be launched on the same instance. In one application, we leverage cloud resources for image processing by assigning image processing tasks to weblets located in EC2 instances while a UI on the client displays the processed results on the device. Based on user configuration, several weblets can be launched on multiple EC2 instances. Another example is augmented reality on a Samsung mobile device. Here we plan to track identified features from photos taken with the phone's camera by sending feature vectors to a set of weblets running on EC2 instances. The cloud weblets then query, in parallel, online image collections and attempt to match the features from these photos.

2.3 Elastic Application Model

Partitions of an elastic application To achieve seamless and transparent migration and offloading, each application should be partitioned into components called weblets. Previous work has proposed many mechanisms for splitting an application into modular components for remote execution or *cyber foraging* purposes, such as [7, 14, 11]. For elastic devices we assume application developers have determined how to organize weblets based on their different behaviors such as computation demand, data dependency, and communication need. One unique requirement for elastic applications is that a weblet's functionality should not be affected by the location or environment where it is running. Essentially, the location of individual weblets should be transparent to users.

Data dependency of weblets An elastic application should allow reasonable data dependency between weblets of the same application or different applications, such as: (1) Different weblets of an application or different applications may require the same native (device) data for their input, (2) Different components of an application or different applications may update the same native (device) data during run-time. However, different components of an application may not update the same application-specific data at run-time.

Communication protocols between weblets As an elastic application is naturally distributed between the cloud platform and a mobile device, weblets should communicate with each other and an appropriate protocol is desirable. Also, infrastructure components such as the CFI and elasticity manager need to communicate with weblets for management purposes such as launch and shutdown. As one design goal for elastic applications is to maintain good robustness and failure recovery, lightweight web services protocols such as REST are used in our framework. The CFI is a web service provided by the CES. It is primarily used by the elasticity manager to request, configure, and release cloud resources. It can also be used by tracking software to monitor cloud usage. Each weblet implements HTTP REST interfaces, which consist primarily of application specific requests; however a small number of generic management requests (e.g., ping, die, prepare-to-migrate) are also part of the weblets.

3. SECURITY ASSUMPTIONS, THREATS, AND REQUIREMENTS

As an elastic application can be distributed among mobile devices and clouds, security should be an essential part of the appli-

cation framework. With the open and multi-lateral nature of cloud computing environment, the problem becomes more complex than existing web services environments.

3.1 Security Assumptions

In designing security mechanisms for elastic applications, we place trust in the CES including cloud manager, application manager, cloud node manager, and CFI. Note that this assumption does not mean we completely trust the cloud infrastructure (IaaS) or platform (PaaS) providers. In general, the CES and CFI can be provided by an independent service provider, e.g., as a SaaS to elastic application developers and mobile users. Also, as part of the elastic framework we trust the elasticity manager on each device. We require that each user should first pair their DEM with a CES (e.g., upon installing the first elastic application on the device).

3.2 Threat Model

Threats to Mobile Devices Malware targeting mobile devices such as cellphones and more recently smartphones,including Symbian and iPhone platforms [12, 2], have become prevalent. In addition to regular attacks on integrity and confidentiality of application code and user data on a device, malware can compromise the DEM. One possible attack is to drain power on the device by changing the configuration of an application such that heavy computation weblet is launched on local device. Similarly, compromise of the device's sensing components could cause the DEM to manage weblets based on incorrect data. For instance, a malware can change the battery status of the device thus DEM does not make decision of offloading execution when an application is launched. Furthermore, malware could bypass the elasticity manager and launch weblets on cloud platforms on behalf of the user, which would be billed to the user's account.

Threats to Cloud Platform and Application Container As a weblet has the potential to run on commercially available cloud platforms, security of the hosting environment must be considered. Many cloud providers offer some level of protection to their clients, but misconfigurations of critical cloud components could lead to weblet compromise. Weak authentication and access control settings on cloud platforms, software vulnerabilities, and attacks on the weblet VM, code, and data from within the cloud side are possibilities. More particular to elastic applications, malicious entities can change network and cost settings, or even cloud sensing information to confuse the CES into making decisions such as using overly expensive network connections to devices. Other malicious activities can consume resources of cloud platform such as CPU cycles, storage, and network traffic. This could lead to degradation of performance and network bandwidth of the cloud platform and generate hidden bills to the application user.

Threats to Communication Channels Weblets must be able to communicate with the device and may need to communicate with other web services on behalf of the user. Experience with Internet-based services have shown that attacks from worms and viruses such as Code Red, and SQL Slammer are a common threat to network oriented applications. Threats also exist from active network entities such as packet injection and Man-in-the-Middle (MITM) attacks if care is not taken to properly authenticate and secure weblet communication with elastic devices. Passive attackers can also eavesdrop on traffic and violate user confidentiality and privacy requirements. DDoS is another network threat. For elastic applications, an attacker can sit in the middle of the network and generate tremendous network traffic to both sides. DDoS attacks can not only exhaust bandwidth resources, but also result in excessive charges to user accounts or disable them if they exceed their quotas.

3.3 Security Objectives

With respect to these threats, we identify the following security objectives that should be achieved during elastic application installation and runtime.

- **Trustworthy weblet containers (or VMs) on both device and cloud:** Weblets must be installed and execute in trusted runtime environments in all locations. The elasticity manager on both the cloud and device should have some assurance that the weblet's execution environment can adequately protect its expected functionalities. How trust is established with the container should not only rely on social and legal agreements (e.g., those for cloud providers), but also via technical mechanisms such as integrity measurement and attestation [4, 16]. Trusted elastic devices and cloud platforms are the foundation necessary to achieve high assurance of other security objectives.

- **Authentication and secure session management:** The elastic framework should provide a mechanism to authenticate weblets belonging to the same application and user to each other. This is especially important when they are running on different platforms. Authentication is the prerequisite to building secure communication between weblets. Also, as we extensively leverage http-based web service mechanisms between CES, DEM, and weblets (even on the same platform), secure session management is needed especially when multiple instances of the same applications can be launched concurrently.

- **Authorization and access control:** A weblet on the cloud should adhere to the property of least privileges. Which permissions a weblet might have may depend on its execution location. Implicit access to device resources may require additional scrutiny when the weblet is no longer running local to the device.

- **Logging and auditing:** Behaviors of weblets should be logged and audited routinely to prevent malicious activities, such as consumption of too many resources on a cloud platform thus affecting its QoS or cost.

As a first step toward achieving the above security objectives, we focus on authentication and session management in the next section. We note that other security objectives are heavily related to authentication and secure communication, especially the trusted weblet runtime containers, which will be our future work.

4. AUTHENTICATION AND SECURE SESSION MANAGEMENT

4.1 Secure Installation of Elastic Applications

For our elastic applications, we use an Java-like application package in a single bundle, which includes the binaries of weblets, UI components, and necessary data. Each application bundle also has a set of meta-data which encoded into a manifest, including the description of the application, and most importantly, the developer signed SHA1 hash values of the individual weblets. Unlike a typical Java package, the application developer can also specify where individual weblets can be installed and executed, (e.g., migratable, cloud side only, or device side only).

When a user downloads and installs an application, the integrity of all weblets are verified by the installer of the elastic device by

re-computing and comparing their hashes and with those in the bundle. After successful integrity verification, the installer registers the application with the DEM. Specifically, the DEM maintains a table of installed applications on the device which need elasticity manager support, each with detailed information of weblets including signed hash values and migration settings.

As an installation option, parts of the elastic application can be installed by the application manager into the CES, which maintains installed applications for users. To do this, the user has to register (for the first time to use the CES service) and authenticate with the CES during installation, (e.g., creating an account and login with username and password). To save communication overhead during installation, the cloud-based application manager also can download the same application from an application store instead of uploading from the device.

4.2 Building Authentication between Weblets

Our goal is to enable a weblet to authenticate another weblet of the same application, which could be launched in different locations, and can be migrated during execution. Here, by an "application" we mean it is installed from a signed bundle and authenticated by the user. Two weblets installed on different devices are not in the same application, although they may downloaded and installed from the same bundle by the same user. We leverage the elasticity manager on device side and the elasticity service on cloud side to establish a shared secret between weblets.

The DEM, CES, and the node manager on each cloud node are implemented as web services. As previously stated, both user applications and DEM are registered with CES, which manages multiple cloud nodes in one IaaS or PaaS. In our implementation, we use Amazon EC2 for cloud nodes. However similar mechanisms can be deployed for PaaS like Google AppEngine. Figure 3 shows the basic work flow, which is briefly explained as follows. The interfaces of web methods for each entity are shown in Figure 4 and 5, respectively.

```
  //wid is the first weblet of the application
  If (no wsk & wss for the application) {
    generate random number wsk and wss;
    //to identify a single session
    update session table with (app,wsk,wss);
  }
 }
DEM::LaunchWeblet(location, wid, wsk, wss) {
  If location==localhost {
    executeWeblet(wid,wsk,wss);
    get wid_url;
    update weblet table with (wid,wid_url,wsk)
  }
  If location==cfi
    call LaunchWeblet(cfi, wid, wsk, wss) via https
    if get_WebletOK(wid_url,wid, wsk, sig)
      update weblet table with (wid_url,wid,wsk)
 }
DEM::GetWeblet(wsk) {
  return the list of running weblets
  in a single session identified by wsk
 }
CFI::LaunchWeblet(nodeid, wid, wsk, wss) {
  select cloud nodeid to lauch wid;
  call LaunchWeblet(nodeid,wid,wsk,wss) via https
 }
Node::LaunchWeblet(wid, wsk, wss){
  executeWeblet(wid,wsk,wss);
  get wid_url;
  return WebletOK(wid_url,wid,wsk,sig);
 }
```

Figure 4: Web methods to establish shared secret and session key when launching weblets in different locations.

```
Weblet::weblet(wid, wsk, wss) {
  ...
  public id=wid;
  private sk=wsk;
  private ss=wss;
 }
Weblet::GetSession () {
  return this.sk;
 }
Weblet::call(p1,..., pn, wid) {
  //call a method of another wid
  generate nonce N;
  sig=HMAC(p1|...|pn|this.sk|N|this.id|wid|this.ss);
  this.send(p1,...,pn,this.sk,N,this.id,wid,sig);
 }
Weblet::get(p1,...,pm, wsk,N,wid1,wid2,sig) {
  if wid2 != this.id
    return with exception;
  sig1 = HMAC(p1|...|pm|wsk|N|wid1|wid2|this.ss);
  if sig1 != sig // signature verification fail
    return with exception;
  this.get(p1,...,pm);
 }
```

Figure 5: Weblet methods to obtain shared secret and session key when being launched and use them for authentication and message integrity during communication.

1. Whenever an elastic application wants to launch a weblet, e.g., via the `main()` or any UI component invoked by the user, it first connects to the DEM, which decides where to launch the weblet.

2. DEM generates a pair of weblet session keys (`wsk`) and a secret (`wss`) for the application if this is the first weblet to be launched. These are shared by all weblets during a single session.

3. When DEM decides to launch a weblet in local device, it executes the installed weblet binary with `LaunchWeblet ( localhost,wid,wss,wsk)`. Upon invoking, the weblet construction method records `wid`, `wsk`, and `wss` into its member variables. Also, the weblet returns a valid URL endpoint which is used to communication with other weblets with http(s). DEM then updates a weblet table which records the active weblet's URL, `wid`, and `wsk`.

4. If DEM decides to launch a weblet in a cloud, it calls the CFI's web method `LaunchWeblet(cfi,wid,wsk,wss)`. Note that this method has to be done with https as it transfers a session secret `wss`.

5. Based on its service logic, the CFI queries its cloud manger and decides on which cloud node the weblet will be loaded. Note that during the installation phase, the corresponding weblet binary is either installed in the application manager of CES, or download from the URL provided by DEM (not shown in Figure 3). Once this is decided, CFI call the target node manager's `LaunchWeblet(nodeid, wid, wsk, wss)`, again with https as it goes via public Internet.

6. The node manager executes weblet binaries provided by the application manager of the CES, similar to launching a weblet by the DEM locally. A code transportation mechanism is needed between the node manager and CES, which is not shown in Figure 3. The successfully launched weblet returns a valid URL endpoint to the node manager, which in turn is passed back to CFI and DEM. DEM updates the weblet table with returned result. Before updating, DEM verifies if the

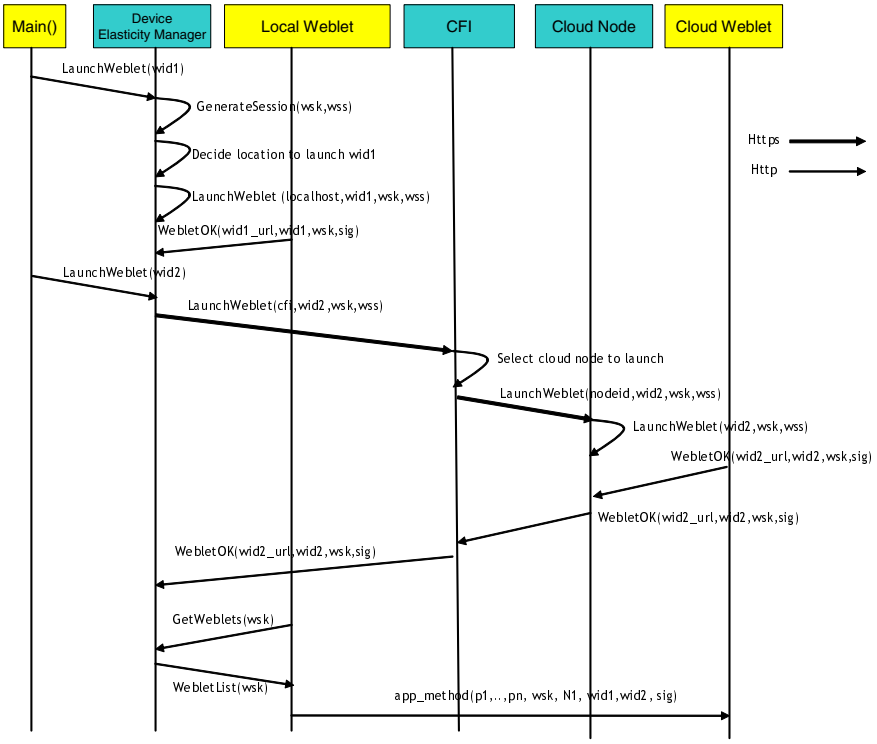

Figure 3: Establish shared secret and session key when launching weblets in different locations. Upon launching weblets of an application, the DEM generates a pair of weblet session key (wsk) and weblet session secret (wss), which are transferred to launched weblets and used to establish authenticated and secure channel between two weblets.

`WebletOK` message is generated by the launched weblet, by checking the HMAC value with `wss`.

7. A local weblet can query DEM to obtain the list of all active weblets in the same session by call `DEM::GetWeblet(wsk)`. DEM returns the URLs of all weblets by querying the table.

8. The local weblet can broadcast the URLs to any other weblet that needs to communicate. Figure 5 lists example interfaces of a weblet to invoke another weblet's method or receive a call from another weblet. Specifically, when calling, the calling weblet generates a nonce, and creates a HMAC value by calculating all parameters with the nonce, its own `wid`, the target `wid`, and its own `wss`. When responding to a call, the weblet first verifies the HMAC with its `wss`, and processes the request if successes; otherwise, it denies the calling.

Security Analysis Through the above protocol, we note that during runtime, only weblets and the DEM know about `wss`, while node managers, CFI, and cloud manager do not. This reduces the risk of a compromised CES and node managers using the secret. Furthermore, communication between weblets are authenticated by the shared secret, and identified by the session token. Thus, an entity which is not launched by the DEM for the same application cannot hijack the session and arbitrarily send requests to cloud weblets on behalf of the user. We assume there is authentication mechanism between the application user and the DEM, i.e., DEM only launches weblets for authenticated applications. Similarly, we trust that each user and/or device is authenticated with the CFI before using the elastic service. For simplicity we omit the details of authentication between user, device, DEM, and CFI here.

Another issue is the trust of the weblet environment. In IaaS and Paas clouds, the DEM should be assured, either via the CES or directly from the cloud, that the hosting system is trustworthy. By leveraging trusted computing techniques like integrity measurement [16] and trusted distributed computing [13, 18, 9], the hosting cloud can generate attestations of the infrastructures integrity (code, data, proper configurations). Only after validating the correctness of the attestation would the elasticity manager load the weblet on the cloud. During the weblet's execution, continued inspection of the platform's integrity may be necessary.

4.3 Secure Migration

According to the elasticity principle, weblets of a running application may be migrated between device and cloud, and even between different cloud nodes. Figure 6 shows the work flow of migrating a weblet from device to cloud. Other scenarios can be done in a similar way. Usually a migration request is triggered by the DEM. For instance, due to the low battery energy level of a device, the DEM decides that a weblet (wid_s) involving heavy computing should be migrated to the cloud (step 1 in Figure 6). Upon this request, the weblet enters a migration state and saves its current running state including its session secret, and returns to the DEM (step 2-3). The DEM then sends the migration request to the CFI. CFI then decides where the weblet should be migrated, by either picking one cloud node in an available node pool or creating a new node (step 4-6). Then migrating the weblet to this node is similar to launching a new weblet, except that the saved weblet state from device is used in the launching, and then the new weblet (wid_d) obtains the shared session secret with other running weblets of the application. To make this migrated weblet visible to other weblets, the node manager updates its new URL to the routing table on the de-

vice after successful migration, and the migrated weblet broadcasts its new URL to other weblets by querying the DEM. Corresponding interfaces and methods should be defined and implemented by the weblet, details of which we omit in this paper.

When the DEM initiates migration of a weblet, it starts queuing any requests to the weblet and releases (relays) the requests only after migration has completed. Consequently, the weblet will not reply to other running weblets during migration. We note that what we describe here is a simplified view of code and data migration but still illustrates the problem and our principle.

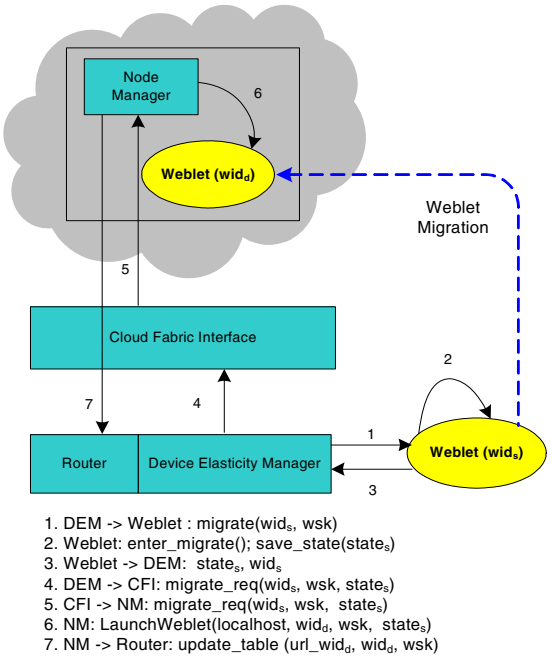

Figure 6: Secure weblet migration from device to cloud. Upon migration request, the client weblet enters migration state and saves its state information. The CFI identities the new destination node on cloud, where the node manger launches new weblet with the state of migrating weblet from device.

Another type of migration occurs when a cloud provider migrates a VM from one physical cloud node to another, e.g., for load balancing. In this situation, a cloud-based weblet may migrate along with other applications running on the VM. Usually, for IaaS and PaaS cloud computing, VM migration by cloud infrastructure is transparent to end users and applications including SaaS providers, in which case no action need be taken by the DEM. If however a weblet's endpoint URL is changed after VM migration, the CFI should capture the migration event from the cloud provider and update the router table on the DEM. We assume that if the VM's IP address changes, the cloud provider would provide migration events and status to other service providers (e.g., the CFI).

4.4 Authorization of Weblets

Until now, we have considered authentication between weblets of a single elastic application. Each weblet is treated equally; each has a pair of session token and secret that identifies the authenticated single application instance. We consider a typical use case where the weblet should be considered with different permissions. Typically, a weblet may access external web services on behalf of the user or any weblet or UI running on an elastic device. Figure 7 shows one instance of this, where an elastic application accesses a Facebook service, (e.g., to retrieve some pictures of a user, and do some image processing) which is performed by a weblet running on cloud. Besides the authentication and session information between weblets, the weblet in cloud needs authorization to access the external web service before it can obtain user data. As the UI part of an application is usually running on the device side, the user authentication step should be performed by the user with a local weblet. Without loss of generality we also assume that upon authentication the application obtains a pair of application session key (`ask`) and secret (`ass`) from the web server. There are different approaches for a cloud weblet to obtain authentication and authorization to access external services. We briefly discuss them and tradeoffs below.

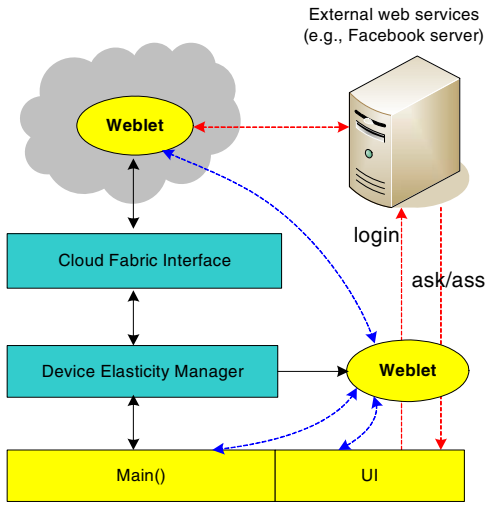

Figure 7: Authenticating and authorizing cloud weblet to access external web services, which is separated from the authentication mechanism between weblets of an elastic application.

Shared user credentials Each weblet has user credentials such as username and password or digital certificate of the web service. A method can be implemented by weblets to retrieve this via the UI component. This is the simplest solution, but implies that each weblet can represent the user and introduces risks, especially for those on cloud. A hostile environment on cloud node can save the user credentials and impersonate the user later.

Shared session information After a device weblet authenticates with the web server, it shares `ask` and `ass` with other weblets. A method can be implemented by the weblet class to share or broadcast secret, e.g., with https. This is a safer solution than the first one as it only shares the session secret, which usually is only valid for a short time period after authentication.

Use session information only on device weblet Whenever a cloud weblet needs access to user data on external web services, it forwards the requests to the authenticated device weblet, which has `ask` and `ass`. For example, if any request to the web server needs signature-based authentication, the cloud weblet sends the request to the device weblet, which signs the request with the session secret, and returns back to the calling weblet, which in turn accesses the server. This enhances the security as session information is only available on the device. However, multiple re-directions are needed in this approach, thereby introducing communication overhead to the device.

OAuth-like [3] authentication When the cloud weblet accesses the web service, it generates an authentication challenge on behalf of

the user and redirects any responded authentication URL to the UI or device weblet. When the web server authenticates the user successfully, the UI or device weblet re-directs the resulting session information to the original requesting cloud weblet. What is different from the shared session approach is that a user can select which weblet on cloud can access which external web services, such as assign different permissions to them based on demand, thus fine-grained authorization can be supported. The cost of this approach is that extra authorization management should be considered in application logic, thus more burden for application developers.

5. RELATED WORK

Computation offloading, remote execution, and cyber foraging have been extensively discussed in previous literature[7, 14, 11]. Among the differences between earlier work and our approach, one salient feature of the elastic framework is that it enables running applications between resource-constrained devices and Internet-based clouds. Most existing approaches focus on offloading computing tasks to nearby PCs and servers such as those in office and home environments. Another major difference is that elasticity is not a feature in most earlier work. Furthermore, although security is often mentioned, few actually have built-in security mechanisms. CloneCloud [8] is the closest to implementing an elastic application. Similar to traditional computation offloading work, CloneCloud focuses on execution augmentation with less consideration on elasticity based on user preference or device status. There is no concrete security mechanism in CloneCloud so far.

Santos et al. [17] propose a trusted cloud computing platform (TCCP) for IaaS. Using TCG/TPM [5], TCCP verifies the platform integrity of a computing node in IaaS before it offers services to cloud users. Similar to Shamon [13], this can be leveraged in our elastic framework to enhance the trustworthiness of weblet containers running on the cloud.

Goayl and Carter propose a secure cyber foraging mechanism for resource-constrained devices [10]. However, this mechanism is based on public-key infrastructure, where each VM, device and user should have a public/private key pair. In our approach, we use a simple secret-key based authentication mechanism, which is more efficient and easier to manage. Also, we offer separate security mechanisms at the application level (between weblets) and infrastructure level (between application, DEM, and CES), which makes our approach more scalable.

6. CONCLUSIONS AND FUTURE WORK

Emerging cloud computing models, IaaS, PaaS and SaaS, support flexible and efficient ways to augment computing, storage, and communication capabilities of applications for resource-constrained devices. To achieve this in our project we are developing an elastic application framework with new application model and elasticity infrastructure. This paper analyzes security threats to elastic applications and identifies security objectives that should be provided by the infrastructure. We then propose authentication and secure communication mechanisms for elastic applications, which consist of weblets running on device and cloud nodes concurrently. Our mechanism also supports secure migration of weblets between device and cloud. We leverage the elasticity infrastructure to build shared session key and secret between weblets of a single application instance. We further propose different approaches to authorize weblets running on the cloud to access user sensitive data such as provided by other web services, and analyze their tradeoffs. One of our design goals is to give minimum security considerations to users and application developers.

As discussed in Section 4, security of elastic applications relies heavily on the trusted behavior of weblets running on remote cloud platforms, which in turn depends on the integrity of the cloud platform and software stack. We plan to explore the open and general question of building a large-scale cloud-based trusted environment. Also, we are developing a cost service for mobile users running elastic applications. The cost service will incorporate factors such as performance, latency, and monetary cost, and guide the dynamic configuration of elastic applications.

7. REFERENCES

[1] Amazon EC2, http://aws.amazon.com/ec2/.

[2] Mcafee mobile security report 2009, http://www.mcafee.com/us/local_content/reports/mobile_security_report_2009.pdf.

[3] Oauth, http://oauth.net.

[4] Tcg mobile reference architecture specification, https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-reference-architecture-1.0.pdf.

[5] *TCG Specification Architecture Overview*. https://www.trustedcomputinggroup.org.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, 2009.

[7] R. K. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proc. of MobiSys*, 2003.

[8] B.-G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *USENIX HotOS XII*, 2009.

[9] L. S. Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and sustaining system integrity via root of trust installation. In *Proc. of ACSAC*, 2007.

[10] S. Goyal and J. Carter. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications*.

[11] G. C. Hunt, M. L. Scott, G. C. Hunt, and M. L. Scott. The coign automatic distributed partitioning system. In *Proc. of OSDI*, 1999.

[12] M. Hypponen. State of cell phone malware in 2007, http://www.usenix.org/events/sec07/tech/hypponen.pdf.

[13] J. McCune, S. Berger, R. Caceres, T. Jaeger, and R. Sailer. Shamon: A system for distributed mandatory access control. In *Proc. of ACSAC*, 2006.

[14] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. Giuli, and X. Gu. Towards a distributed platform for resource-constrained devices. Technical Report HPL-2002-26, HP Laboratories, 2002.

[15] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big web services: Making the right architectural decision. In *Proc. of WWW*, 2008.

[16] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proc. of ACM CCS*, 2004.

[17] N. Santos, K. P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *USENIX HotCloud*, 2009.

[18] E. Shi, A. Perrig, and L. V. Doorn. Bind: a fine-grained attestation service for secure distributed systems. In *Proc. of IEEE Symposium on Security and Privacy*, 2005.