On Technical Security Issues in Cloud Computing

Meiko Jensen, Jörg Schwenk
Horst Görtz Institute for IT Security
Ruhr University Bochum, Germany
{meiko.jensen|joerg.schwenk}@rub.de

Nils Gruschka, Luigi Lo Iacono
NEC Laboratories Europe
NEC Europe Ltd.
{nils.gruschka@nw.neclab.eu, lo_iacono@it.neclab.eu}

Abstract

The Cloud Computing concept offers dynamically scalable resources provisioned as a service over the Internet. Economic benefits are the main driver for the Cloud, since it promises the reduction of capital expenditure (CapEx) and operational expenditure (OpEx). In order for this to become reality, however, there are still some challenges to be solved. Amongst these are security and trust issues, since the user's data has to be released to the Cloud and thus leaves the protection-sphere of the data owner. Most of the discussions on this topics are mainly driven by arguments related to organisational means. This paper focusses on technical security issues arising from the usage of Cloud services and especially by the underlying technologies used to build these cross-domain Internet-connected collaborations.

1. Introduction

The new concept of *Cloud Computing* offers dynamically scalable resources provisioned as a service over the Internet and therefore promises a lot of economic benefits to be distributed among its adopters. Depending on the type of resources provided by the Cloud, distinct layers can be defined (see Figure 1). The bottom-most layer provides basic infrastructure components such as CPUs, memory, and storage, and is henceforth often denoted as Infrastructure-as-a-Service (IaaS). Amazon's Elastic Compute Cloud (EC2) is a prominent example for an IaaS offer. On top of IaaS, more platform-oriented services allow the usage of hosting environments tailored to a specific need. Google App Engine is an example for a Web platform as as service (PaaS) which enables to deploy and dynamically scale Python and Java based Web applications. Finally, the top-most layer provides it users with ready to use applications also known as Software-as-a-Service (SaaS). To access these Cloud services, two main technologies can be currently identified. Web Services are commonly used to provide access to IaaS services and Web browsers are used to access SaaS applications. In PaaS environments both approaches can be found.

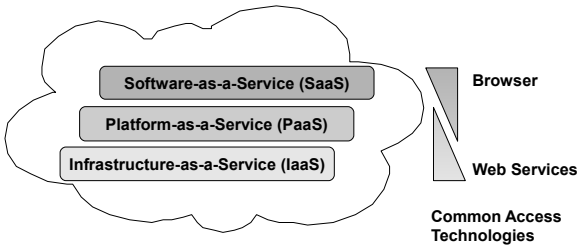

Figure 1. Cloud layers and access technologies

All of these layers come with the promise to reduce first of all capital expenditures (CapEx). This includes reduced hardware costs in the IaaS layer and reduced license costs in all layers. Especially in the IaaS layer it is not required anymore to engineer the own data center for peak performance cases, which occur in general very seldom and which usually result in a poor utilization of the available resources. Additionally, reductions of the operational expenditures (OpEx) in terms of reduced hardware, license and patch management are promised as well.

On the other hand, along with these benefits, Cloud Computing also raises severe concerns especially regarding the security level provided by such a concept. Completely relying the own data and execution tasks to an external company, eventually residing in another country with a different regulatory environment, may cause companies not to consider Cloud Computing but to stick to the conventional local data center approach.

Although there is a clear demand for in-depth discussion of security issues in Cloud Computing, the cur-

978-0-7695-3840-2/09 $26.00 © 2009 IEEE
DOI 10.1109/CLOUD.2009.60

IEEE computer society

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

rent surveys on Cloud security issues focus primarily on data confidentiality, data safety and data privacy and discuss mostly organizational means to overcome these issues [1]. In this paper, we provide an overview on technical security issues of Cloud Computing environments. Starting with real-world examples of attacks performed on Cloud computing systems (here the Amazon EC2 service), we give an overview of existing and upcoming threats to Cloud Computing security. Along with that, we also briefly discuss appropriate countermeasures to these threats, and further issues to be considered in future research on the way to a secure, trustworthy, reliable, and easily applicable Cloud Computing world.

The paper is organized as follows. In the next section, we outline the major technologies used in the context of Cloud Computing and security. Then, in Section 3, we provide a set of security-related issues that apply to different Cloud Computing scenarios. Each issue is briefly described and complemented with a short sketch on countermeasure approaches that are both sound and applicable in real-world scenarios. The paper then concludes in Section 4, also giving future research directions for Cloud Computing security.

2. Foundations

Various distinct technologies are used and combined to build Cloud Computing systems. Depending on the type of Cloud—either IaaS, PaaS or SaaS as defined above—the access technologies can vary from service-enabled fat clients to Web browser-based thin clients. This section provides some required foundations in order to set the fundament for the subsequently introduced and discussed security issues.

2.1. WS-Security

The most important specification addressing security for Web Services is WS-Security, defining how to provide integrity, confidentiality and authentication for SOAP messages. WS-Security defines a SOAP header (`Security`) that carries the WS-Security extensions. Additionally, it defines how existing XML security standards like *XML Signature* and *XML Encryption* are applied to SOAP messages.

XML Signature allows XML fragments to be digitally signed to ensure integrity or to proof authenticity. The XML Signature element has the following (slightly simplified) structure:

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod
                    Algorithm="..."/>
    <SignatureMethod Algorithm="..."/>
    <Reference URI="..." >
      <DigestMethod Algorithm="...">
      <DigestValue>...</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
</Signature>
```

The signing process works as follows: For every message part to be signed a `Reference` element is created and this message part is canonicalized and hashed. The resulting digest is added into the `DigestValue` element and a reference to the signed message part is entered into the `URI` attribute. Finally the `SignedInfo` element is canonicalized and signed. The result of the signing operation is placed in the `SignatureValue` element and the `Signature` element is added to the security header.

XML Encryption allows XML fragments to be encrypted to ensure data confidentiality. The encrypted fragment is replaced by an `EncryptedData` element containing the ciphertext of the encrypted fragment as content.

Further, XML Encryption defines an `EncryptedKey` element for key transportation purposes. The most common application for an encrypted key is a *hybrid encryption*: an XML fragment is encrypted with a randomly generated symmetric key, which itself is encrypted using the public key of the message recipient. In SOAP messages, the `EncryptedKey` element must appear inside the security header.

In addition to encryption and signatures, WS-Security defines security tokens suitable for transportation of digital identities, e.g. X.509 certificates.

2.2. TLS

Transport Layer Security [2] has been introduced, under its more common name “Secure Sockets Layer (SSL)”, by Netscape in 1996. It consists of two main parts: The *Record Layer* encrypts/decrypts TCP data streams using the algorithms and keys negotiated in the *TLS Handshake*, which is also used to authenticate the server and optionally the client. Today it is the most important cryptographic protocol worldwide, since it is implemented in every web browser.

TLS offers many different options for key agreement, encryption and authentication of network peers, but most frequently the following configuration is used:

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

- The Web server is configured with a X.509 certificate that includes its domain name. This certificate must be issued from a "trusted" certification authority (CA), where "trusted" means that the root certificate of this CA is included in nearly all Web browsers.
- During the TLS Handshake, the server sends this certificate to the browser. The browser checks that the certificate comes from a "trusted" CA, and that the domain name in the certificate matches the domain name contained in the requested URL. If both checks succeed, the browser continues loading the Web page. If there is a problem, the human user is asked for a (security) decision.
- The browser itself remains anonymous within this TLS configuration. To authenticate the user, most commonly a username/password pair is requested by the server through an HTML form.

This TLS configuration worked fine for all Web applications, until the first *Phishing* attacks surfaced in 2004[1]. In a Phishing attack, the attacker lures the victim to a fake Web page (either using spoofed emails or attacks on the DNS), where the victim enters username and password(s). This is possible even with TLS, since the human user fails to verify the authentication of the server via TLS (cf. [3]).

3. Cloud Computing Security Issues

In the following, we present a selection of security issues related to Cloud Computing. Each issue is explained briefly and accompanied with a short discussion on potential or real-world measured impacts.

3.1. XML Signature

A well known type of attacks on protocols using XML Signature for authentication or integrity protection is *XML Signature Element Wrapping* [4] (henceforth denoted shortly as wrapping attack). This of course applies to Web Services and therefore also for Cloud Computing.

Figures 2 and 3 show a simple example for a wrapping attack to illustrate the concept of this attack. The first figure presents a SOAP message sent by a legitimate client. The SOAP body contains a request for the file "me.jpg" and was signed by the sender. The signature is enclosed in the SOAP header and refers to the signed message fragment using an XPointer to

1. Interestingly, these attacks have been described as early as 1996, but these descriptions were ignored both by the bad and the good guys outside academia.

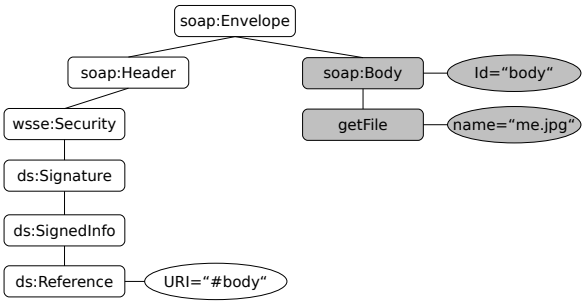

Figure 2. Example SOAP message with signed SOAP body

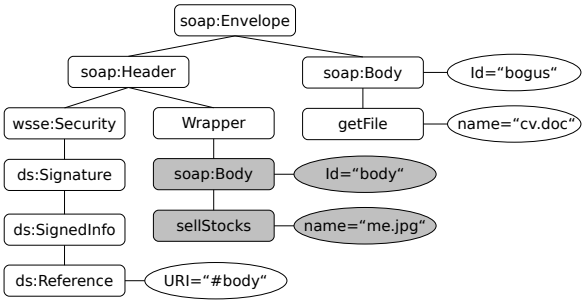

Figure 3. Example SOAP message after attack

the Id attribute with the value "body". If an attacker eavesdrops such a message, he can perform the following attack. The original body is moved to a newly inserted wrapping element (giving the attack its name) inside the SOAP header, and a new body is created. This body contains the operation the attacker wants to perform with the original sender's authorization, here the request for the file "cv.doc". The resulting message still contains a valid signature of a legitimate user, thus the service executes the modified request.

Since the discovery of wrapping attacks by McIntosh and Austel in 2005 a number of further variations, countermeasures and again attacks circumventing these countermeasures have be published. For example, in [5] a method – called *inline approach* – was introduced to protect some key properties of the SOAP message structure and thereby hinder wrapping attacks, but shortly later in [6] it was shown how to perform a wrapping attack anyhow.

However, mostly due to the rare usage of WS-Security in business applications these attacks remained theoretical and no real-life wrapping attack became public, until in 2008 it was discovered that Amazon's EC2 services were vulnerable to wrapping attacks [7]. Using a variation of the attack presented before an attacker was able to perform arbitrary EC2 operations on behalf of a legitimate user. In order to

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

exploit the SOAP message security validation vulnerability of EC2, a signed SOAP request of a legitimate, subscribed user needed to be intercepted. Since the vulnerability in the SOAP request validation allows to interfer any kind of operation and have it executed, it does not matter what kind of request the attacker has at its disposal. The instantiation of a multitude of virtual machine to send spam mails is just one example what an attacker can do—using the legitimated user's identity and charging his account.

3.2. Browser Security

In a Cloud, computation is done on remote servers. The client PC is used for I/O only, and for authentication and authorization of commands to the Cloud. It thus does not make sense to develop (platform-dependent) client software, but to use a universal, platform independent tool for I/O: a standard Web browser. This trend has been observed during the last years, and has been categorized under different names: Web applications, Web 2.0, or Software-as-a-Service (SaaS).

Modern Web browsers with their AJAX techniques (JavaScript, XMLHttpRequest, Plugins) are ideally suited for I/O. But what about security? A partial answer is given in [8], where different browser security policies (with the notable exception of TLS) are compared for the most important browser releases. With a focus on the *Same Origin Policy* (SOP), this document reveals many shortcomings of browser security. If we additionally take into account TLS, which is used for host authentication and data encryption, these shortcomings become even more obvious.

Web browsers can not directly make use of XML Signature or XML Encryption: data can only be encrypted through TLS, and signatures are only used within the TLS handshake. For all other cryptographic data sets within WS-Security, the browser only serves as a passive data store. Some simple workarounds have been proposed to use e.g. TLS encryption instead of XML Encryption, but major security problems with this approach have been described in the literature and working attacks were implemented as proofs-of-concept (cf. 3.2.2). Our goal is to propose provably secure solutions using TLS, but at the same time encourage the browser community to adapt XML based cryptography for inclusion in the browser core.

3.2.1. The Legacy Same Origin Policy. With the inclusion of scripting languages (typically JavaScript) into Web pages, it became important to define access rights for these scripts. A natural choice is to allow read/write operations on content from the same *origin*, and to disallow any access to content from a different origin. This is exactly what the legacy Same Origin Policy does, where origin is defined as "the same application", which can be defined in a Web context by the tuple (*domain name, protocol, port*). There are many special cases where problems with the SOP occur, but this could be solved if the basic definition of "origin" was sound. Unfortunately, for a distributed application like the WWW, this definition is not sound.

In 2008, Dan Kaminski showed that Domain Name System (DNS) caches can easily be "poisoned", i.e. filled with bogus data [9]. Since the DNS heavily relies on caching, domain names become unreliable. This attack could only be fixed outside the DNS protocol, by using UDP source port randomization, to achieve a moderate level of reliability. Other severe security problems with DNS have been described in the area of home routers [10], and in the end this attack vector renders all content loaded via an URL to be unreliable unless they are secured by other means.

For Web applications with high security requirements, TLS has been used for a long time to protect both data during transport, and to authenticate the servers domain name. Problems with this naive approach became apparent with the advent of Phishing attacks for online banking, and will be discussed in the next section.

3.2.2. Attacks on Browser-based Cloud Authentication. The realization of these security issues within browser-based protocols with Cloud Computing can best be explained using Federated Identity Management (FIM) protocols: Since the browser itself is unable to generate cryptographically valid XML tokens (e.g. SAML tokens) to authenticate against the Cloud, this is done with the help of a trusted third party.

The prototype for this class of protocols is Microsoft's Passport [11], which has been broken by Slemko [12]. If no direct login is possible at a server because the browser does not have the necessary credentials, an HTTP redirect is sent to the Passport login server, where the user can enter his credentials (e.g. username/password). The Passport server then translates this authentication into a Kerberos token, which is sent to the requesting server through another HTTP redirect. The main security problem with Passport is that these Kerberos tokens are not bound to the browser, and that they are only protected by the SOP. If an attacker can access these tokens, he can access all services of the victim.

Whereas Passport used a REST type of communication, its successors MS Cardspace and the SAML

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

family of protocols definitively belong to the world of Web Services. However, the same security problems persist: Groß [13] analyzed SAML browser profiles, and one of the authors of this paper described an attack on MS Cardspace [14], [15], which can also be applied to the SAML browser profiles (both token and artifact profiles).

To resume: Current browser-based authentication protocols for the Cloud are not secure, because (a) the browser is unable to issue XML based security tokens by itself, and (b) Federated Identity Management systems store security tokens within the browser, where they are only protected by the (insecure) SOP.

3.2.3. Secure Browser-based Authentication. However, the situation is not hopeless: If we integrate TLS and SOP in a better way, we can secure FIM protocols. In previous work, we identified four methods to protect (SAML) tokens with the help of TLS.

- **TLS Federation [16].** In this approach, the SAML token is sent inside an X.509 client certificate. The SAML token thus replaces other identification data like distinguished names. The certificate has the same validity period as the SAML token.
- **SAML 2.0 Holder-of-Key Assertion Profile [17].** Here again TLS with client authentication is used, but the client certificate does not transport any authorization information. Instead, the SAML token is bound to the public key contained in this certificate, by including this key in a Holder-of-Key assertion. For a more detailed analysis on the security improvements and requirements of this approach see [18].
- **Strong Locked Same Origin Policy [19].** Whereas the previous approaches relied on the server authenticating (in an anonymous fashion) the client, in this approach we strengthen the client to make reliable security decisions. This is done by using the server's public key as a basis for decisions of the Same Origin Policy, rather than the insecure Domain Name System.
- **TLS session binding.** By binding the token to a certain TLS session, the server may deduce that the data he sends in response to the SAML token will be protected by the same TLS channel, and will thus reach the same (anonymous) client who has previously sent the token.

3.2.4. Future Browser Enhancements. Even with the workarounds using TLS, the browser is still very limited in its capacities as an authentication center for Cloud Computing. Whereas many Web Service functionalities can be added within the browser by simply loading an appropriate JavaScript library during runtime (e.g. to enable the browser to send SOAP messages), this is not possible for XML Signature and Encryption, since the cryptographic keys and algorithms require much higher protection[2].

Therefore it would be desirable to add the following two enhancements to the browser security API:

- **XML Encryption:** Here standard APIs could easily be adapted, because only a byte stream has to be encrypted/decrypted, and no knowledge of XML is necessary. However, a naming scheme to access cryptographic keys "behind" the API must be agreed upon. DOM or (mostly) SAX based processing of XML data can be handled by a JavaScript library, since the decrypted data will be stored in the browser and is thus in any case accessible by a malicious (scripting) code.
- **XML Signature:** This extension is non-trivial, because the complete XML Signature data structure must be checked inside the API. This means that the complete `<ds:Signature>` element must be processed inside the browser core, including the transforms on the signed parts, and the two-step hashing. In addition, countermeasures against XML wrapping attacks should also be implemented.

In addition, the API should be powerful enough to support all standard key agreement methods specified in WS-Security family of standards natively, since the resulting keys must be stored directly in the browser. This could be done e.g. by enhancing known security APIs, e.g. PKCS#11.

3.3. Cloud Integrity and Binding Issues

A major responsibility of a Cloud Computing system consists in maintaining and coordinating instances of virtual machines (IaaS) or explicit service implementation modules (PaaS). On request of any user, the Cloud system is responsible for determining and eventually instantiating a free-to-use instance of the requested service implementation type. Then, the address for accessing that new instance is to be communicated back to the requesting user.

Generally, this task requires some metadata on the service implementation modules, at least for identification purposes. For the specific PaaS case of Web Services provided via the Cloud, this metadata may also cover all Web Service description documents

2. Obviously, keys should not be sent or processed in the clear within a JavaScript library.

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

related to the specific service implementation. For instance, the Web Service description document itself (the WSDL file) should not only be present within the service implementation instance, but also be provided by the Cloud system in order to deliver it to its users on demand.

Most of these metadata descriptions are usually required by any user prior to service invocation in order to determine the appropriateness of a service for a specific purpose. Additionally, these descriptions also represent some preliminary service identifiers, as assumably service implementations with identical WSDL descriptions provide the same functionality. Thus, these metadata should be stored outside of the Cloud system, resulting in a necessity to maintain the correct association of metadata and service implementation instances.

3.3.1. Cloud Malware Injection Attack. A first considerable attack attempt aims at injecting a malicious service implementation or virtual machine into the Cloud system. Such kind of *Cloud malware* could serve any particular purpose the adversary is interested in, ranging from eavesdropping via subtle data modifications to full functionality changes or blockings.

This attack requires the adversary to create its own malicious service implementation module (SaaS or PaaS) or virtual machine instance (IaaS), and add it to the Cloud system. Then, the adversary has to trick the Cloud system so that it treats the new service implementation instance as one of the valid instances for the particular service attacked by the adversary. If this succeeds, the Cloud system automatically redirects valid user requests to the malicious service implementation, and the adversary's code is executed.

A promising countermeasure approach to this threat consists in the Cloud system performing a service instance integrity check prior to using a service instance for incoming requests. This can e.g. be done by storing a hash value on the original service instance's image file and comparing this value with the hash values of all new service instance images. Thus, an attacker would be required to trick that hash value comparison in order to inject his malicious instances into the Cloud system.

3.3.2. Metadata Spoofing Attack. As described in [20], the *metadata spoofing attack* aims at maliciously reengineering a Web Services' metadata descriptions. For instance, an adversary may modify a service's WSDL so that a call to a `deleteUser` operation syntactically looks like a call to another operation, e.g. `setAdminRights`. Thus, once a user is given such a modified WSDL document, each of his `deleteUser` operation invocations will result in SOAP messages that at the server side look like—and thus are interpreted as—invocations of the `setAdminRights` operation. In the end, an adversary could manage to create a bunch of user logins that are thought to be deleted by the application's semantics, but in reality are still valid, and additionally are provided with administrator level access rights.

For static Web Service invocations, this attack obviously is not so promising for the adversary, as the task of deriving service invocation code from the WSDL description usually is done just once, at the time of client code generation. Thus, the attack here can only be successful if the adversary manages to interfere at the one single moment when the service client's developer leeches for the service's WSDL file. Additionally, the risk of the attack being discovered assumably is rather high, especially in the presence of sound testing methods.

These restrictions tend to fall away in the Cloud Computing scenario. As the Cloud system itself has some kind of WSDL repository functionality (comparable to a UDDI registry [21]), new users most assumably will gather for a service's WSDL file more dynamically. Thus, the potential spread of the malicious WSDL file—and thus the probability for a successful attack—rises by far.

Similar to the hash value calculation discussed for the Cloud malware injection attack, in this scenario a hash-based integrity verification of the metadata description files prior to usage is required. For instance, an XML digital signature performed on the WSDL by the original service implementor would ensure its integrity. If the WSDL is additionally extended with a hash value on the service instance's image file, this also ensures a cryptographically strong binding between the WSDL and the original service image.

3.4. Flooding Attacks

A major aspect of Cloud Computing consists in outsourcing basic operational tasks to a Cloud system provider. Among these basic tasks, one of the most important ones is server hardware maintenance. Thus, instead of operating an own, internal data center, the paradigm of Cloud Computing enables companies (users) to *rent* server hardware on demand (IaaS). This approach provides valuable economic benefits when it comes to dynamics in server load, as for instance day-and-night cycles can be attenuated by having the data traffic of different timezones operated by the same servers. Thus, instead of buying sufficient server hardware for the high workload times, Cloud

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

Computing enables a dynamic adaptation of hardware requirements to the actual workload occurring.

Technically, this achievement can be realized by using virtual machines deployed on arbitrary data center servers of the Cloud system. If a company's demand on computational power rises, it simply is provided with more instances of virtual machines for its services.

Under security considerations, this architecture has a serious drawback. Though the feature of providing more computational power on demand is appreciated in the case of valid users, it poses severe troubles in the presence of an attacker. The corresponding threat is that of *flooding attacks*, which basically consist in an attacker sending a huge amount of nonsense requests to a certain service. As each of these requests has to be processed by the service implementation in order to determine its invalidity, this causes a certain amount of workload per attack request, which—in the case of a flood of requests—usually would cause a Denial of Service to the server hardware (cf. [22], [23]).

In the specific case of Cloud Computing systems, the impact of such a flooding attack is expected to be amplified drastically. This is due to the different kinds of impact, which are discussed next.

3.4.1. Direct Denial of Service. When the Cloud Computing operating system notices the high workload on the flooded service, it will start to provide more computational power (more virtual machines, more service instances...) to cope with the additional workload. Thus, the server hardware boundaries for maximum workload to process do no longer hold. In that sense, the Cloud system is trying to work *against* the attacker (by providing more computational power), but actually—to some extent—even *supports* the attacker by enabling him to do most possible damage on a service's availability, starting from a single flooding attack entry point. Thus, the attacker does not have to flood all n servers that provide a certain service in target, but merely can flood a single, Cloud-based address in order to perform a full loss of availability on the intended service.

3.4.2. Indirect Denial of Service. Depending on the computational power in control of the attacker, a side-effect of the direct flooding attack on a Cloud service potentially consists in that other services provided on the same hardware servers may suffer from the workload caused by the flooding. Thus, if a service instance happens to run on the same server with another, flooded service instance, this may affect its own availability as well. Once the server's hardware resources are completely exhausted by processing the flooding attack requests, obviously also the other service instances on the same hardware machine are no longer able to perform their intended tasks. Thus, the Denial of Service of the targeted service instances are likely to cause a Denial of Service on all other services deployed to the same server hardware as well.

Depending on the level of sophistication of the Cloud system, this side-effect may worsen if the Cloud system notices the lack of availability, and tries to "evacuate" the affected service instances to other servers. This results in additional workload for those other servers, and thus the flooding attack "jumps over" to another service type, and spreads throughout the whole computing Cloud (see also [24]).

In the worst case, an adversary manages to utilize another (or the very same) Cloud Computing system for hosting his flooding attack application. In that case, the *race in power* (as defined in [22]) would play both Cloud systems off against each other; each Cloud would provide more and more computational resources for creating, respectively fending, the flood, until one of them eventually reaches full loss of availability.

3.4.3. Accounting and Accountability. As the major economic driver behind running a Cloud Computing service is charging the customers according to their actual usage (e.g. workload caused), another major effect of a flooding attack on a Cloud service consists in raising the bills for Cloud usage drastically. There are no "upper limits" to computational power usage[3], thus the user running the flooded service most likely has to foot the bill for the workload caused by the attacker—at least if the attacker is not determinable itself (cf. [24]).

4. Conclusion and Future Work

In this paper, we presented a selection of issues of Cloud Computing security. We investigated ongoing issues with application of XML Signature and the Web Services security frameworks (attacking the Cloud Computing system itself), discussed the importance and capabilities of browser security in the Cloud Computing context (SaaS), raised concerns about Cloud service integrity and binding issues (PaaS), and sketched the threat of flooding attacks on Cloud systems (IaaS).

As we showed, the threats to Cloud Computing security are numerous, and each of them requires an in-depth analysis on their potential impact and relevance to real-world Cloud Computing scenarios.

3. This holds only as long as there are no appropriate service-level agreements with upper boundaries between Cloud operators and users.

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.

As can be derived from our observations, a first good starting point for improving Cloud Computing security consists in strengthening the security capabilities of both Web browsers and Web Service frameworks, at best integrating the latter into the first. Thus, as part of our ongoing work, we will continue to harden the foundations of Cloud Computing security which are laid by the underlying tools, specifications, and protocols employed in the Cloud Computing scenario.

References

[1] J. Heiser and M. Nicolett, "Assessing the security risks of cloud computing," *Gartner Report*, 2009. [Online]. Available: http://www.gartner.com/DisplayDocument?id=685308

[2] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," IETF RFC 5246, 2008, http://www.ietf.org/rfc/rfc5246.txt.

[3] R. Dhamija, J. D. Tygar, and M. A. Hearst, "Why phishing works," in *Proceedings of the 2006 Conference on Human Factors in Computing Systems (CHI), Montréal, Québec, Canada*. ACM, 2006, pp. 581–590.

[4] M. McIntosh and P. Austel, "XML signature element wrapping attacks and countermeasures," in *SWS '05: Proceedings of the 2005 workshop on Secure web services*. ACM Press, 2005, pp. 20–27.

[5] M. A. Rahaman, A. Schaad, and M. Rits, "Towards secure SOAP message exchange in a SOA," in *SWS '06: Proceedings of the 3rd ACM workshop on Secure Web Services*. ACM Press, 2006, pp. 77–84.

[6] S. Gajek, L. Liao, and J. Schwenk, "Breaking and fixing the inline approach," in *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services*. New York, NY, USA: ACM, 2007, pp. 37–43.

[7] N. Gruschka and L. Lo Iacono, "Vulnerable Cloud: SOAP Message Security Validation Revisited," in *ICWS '09: Proceedings of the IEEE International Conference on Web Services*. Los Angeles, USA: IEEE, 2009.

[8] Google, "Browser security handbook," 2009. [Online]. Available: http://code.google.com/p/browsersec/

[9] D. Kaminski, "Dns server+client cache poisoning, issues with ssl, breaking *forgot my password* systems, attacking autoupdaters and unhardened parsers, rerouting internal traffic; http://www.doxpara.com/DMK_BO2K8.ppt," -, 2008.

[10] S. Stamm, Z. Ramzan, and M. Jakobsson, "Drive-by pharming," Indiana University Computer Science, Tech. Rep. 641, 2006.

[11] D. Kormann and A. Rubin, "Risks of the passport single signon protocol," *Computer Networks*, vol. 33, no. 1–6, pp. 51–58, 2000.

[12] M. Slemko, "Microsoft passport to trouble," 2001, http://alive.znep.com/∼marcs/passport/.

[13] T. Groß, "Security analysis of the SAML single sign-on browser/artifact profile," in *Proc. 19th Annual Computer Security Applications Conference*, 2003.

[14] S. Gajek, J. Schwenk, M. Steiner, and C. Xuan, "Risks of the cardspace protocol," in *ISC'09: Proceedings of the 12th Information Security Conference, LNCS*. Springer, 2009.

[15] X. Chen, S. Gajek, and J. Schwenk, "On the Insecurity of Microsoft's Identity Metasystem CardSpace," Horst Görtz Institute for IT-Security, Tech. Rep. 3, 2008.

[16] B. P. Bruegger, D. Hühnlein, and J. Schwenk, "TLS-Federation – A secure and Relying-Party-friendly approach for Federated Identity Management," in *Proceedings of* BIOSIG 2008: Biometrics and Electronic Signatures*, LNI 137*, 2008, pp. 93–104.

[17] T. Scavo, "SAML V2.0 Holder-of-Key Assertion Profile," Working Draft 09, 20.01.2009, 2009, http://www.oasis-open.org/apps/org/workgroup/security/download.php/30782/sstc-saml2-holder-of-key-draft-09.pdf.

[18] S. Gajek, T. Jager, M. Manulis, and J. Schwenk, "A Browser-based Kerberos Authentication Scheme," in *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, LNCS 5283*. Springer, 2008, pp. 115–129.

[19] J. Schwenk, L. Liao, and S. Gajek, "Stronger Bindings for SAML Assertions and SAML Artifacts," in *Proceedings of the 5th ACM CCS Workshop on Secure Web Services (SWS'08)*. ACM Press, 2008.

[20] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on web services," *Computer Science - Research and Development (CSRD), Springer Berlin/Heidelberg*, 2009.

[21] L. Clement, A. Hately, C. von Riegen, and T. Rogers, "UDDI Version 3.0.2," *OASIS UDDI Spec Technical Committee Draft*, 2004.

[22] M. Jensen, N. Gruschka, and N. Luttenberger, "The Impact of Flooding Attacks on Network-based Services," in *Proceedings of the IEEE International Conference on Availability, Reliability and Security (ARES)*, 2008.

[23] M. Jensen and N. Gruschka, "Flooding Attack Issues of Web Services and Service-Oriented Architectures," in *Proceedings of the Workshop on Security for Web Services and Service-Oriented Architectures (SWSOA, held at GI Jahrestagung 2008)*, 2008, pp. 117–122.

[24] M. Jensen and J. Schwenk, "The accountability problem of flooding attacks in service-oriented architectures," in *Proceedings of the IEEE International Conference on Availability, Reliability and Security (ARES)*, 2009.

Authorized licensed use limited to: Old Dominion University. Downloaded on January 8, 2010 at 12:52 from IEEE Xplore. Restrictions apply.