Rethinking security in a cloudy world

Federico Maggi Stefano Zanero

{fmaggi,zanero}@elet.polimi.it

Technical Report no. 2010-11
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy

The world of information and communication technology is experiencing changes that, regardless of some skepticism, are bringing to life the concept of "utility computing". The nostalgics observed a parallel between the emerging paradigm of cloud computing and the traditional time-sharing era, depicting clouds as the modern reincarnation of mainframes available on a pay-per-use basis, and equipped with virtual, elastic, paid disks-as-a-service that replace the old physical disks with quotas. This comparison is fascinating, but more importantly, in our opinion, it prepares the ground for constructive critiques regarding the security of such computing paradigm. In this paper we explore similar analogies to discuss our position about the current countermeasures (e.g., intrusion detection systems, anti-viruses), developed to mitigate well-known security threats. By reasoning on said affinities, we focus on the simple case of anomaly-based approaches, which are employed in many modern protection tools, not just in intrusion detectors. We illustrate our position by the means of a simple running example and show that attacks against injection vulnerabilities, a current menace that is easily recognizable with ordinary anomaly-based checks, can be difficult to detect if web services are assumed to be regular web applications. Along this line, we concentrate on a few, critical hypotheses that demand particular attention. We conclude that, although only a minority of threats qualify as *novel*, they are well camouflaged and can be difficult to recognize behind the confusion caused by the cloud computing excitement.

1 Introduction

The emerging concept of cloud computing is both widespread and puzzling [41]. Indeed, it has been talked, blogged, written and discussed about as an unprecedented paradigmatic change in the information and communication technology world. As it often happens, this also brought forth confusion in terms and buzzwords such as *"public cloud"*, *"private cloud"*, *"*-as-a-service"*, etc. According to the definition given in a recent, comprehensive and sound analysis [1], cloud computing enables organizations to run web applications (often referred to as services) on a pay-as-you-go basis on top of reliable, highly-available, scalable software and hardware infrastructures referred to as *clouds*. This is also, in general, the widespread perception shared by users and developers. In some sense, a cloud can be seen as a modern, large mainframe [28] with virtually infinite resources, and the term cloud *computing* refers to the use of such resources to deliver web services. However, as noted by other researchers, the lack of a widely-accepted definition may distract users and experts [8].

Cloud security is also, unsurprisingly, an ill-defined and vague field. Community-driven initiatives [3] and organizations such as the Cloud Security Alliance [30] are pursuing the common objective of gathering knowledge and joining efforts to devise security measures appropriate for cloud computing. The rapid growth of Cloud Security Alliance [17] suggests that the community is indeed concerned about security. While cloud computing certainly poses new challenges, however, it may be difficult to distinguish between issues *specifically* caused by the emerging computing paradigm, and issues that, by coincidence, occur on a system deployed on a cloud [8]. On one hand, the community of cloud users is worried about the fragility of cloud computing [6, 29, 5, 20, 36, 12, 35, 21] and concerns have arisen recently regarding the significant outages of the major cloud providers. On the other hand, it is quite easy to observe that security issues are mainly caused, as usual, by programming errors. In other words, although a cloud unquestionably offers a sophisticated and flexible platform, it still runs pieces of software, which can be just as insecure as any piece of software running on traditional environments. For instance, Amazon's S3 experienced two outages in the 2008 due to an overload of the authentication service [37] and an error in a single bit [38]. Also, Google AppEngine suffered a "blackout" because of a programming error [44]. Clearly, such vulnerabilities have nothing to do with cloud computing itself.

In environments characterized by distraction and lack of solid understanding, it might be difficult to reason about security threats[1]. At a first glance, it is almost superfluous to recall that the Internet is not a safe place, with more than 97,500 *known* web application vulnerabilities disclosed in 2009 [18], and more than 350 million sensitive records involved in security breaches in the United States since 2005 [9]. Moreover, less the one year ago, the number of entries in the Google Safe Browsing Malware List has doubled between June 2008 and August 2009. The black hats are somehow "weaponizing" malware [7] and, in addition [39], provide full-fledged attacking infrastructures (e.g., botnets) available (to inexperienced users) on a pay-per-use basis. It may be even argued that such threats are, unavoidably, part of the Internet. In some way, they have contributed to its natural evolution and therefore, they are also unavoidably part of the cloud computing realm.

[1]We refer to the most general definition of *security threat* as any opportunity (for a malicious entity) to compromise confidentiality, integrity or availability of a system. In this context, the term *system* may refer to different objects, such as a bare-metal computer, a virtual or physical network, an application, a service, a file-system, or a database.

Nevertheless, as discussed in Section 2, we believe it is important to critically re-examine the known security issues in the light of the changes induced by the adoption of cloud computing. This is the motivation of this paper, which analyzes the aforesaid changes by leveraging the parallel between the new and the old computing approaches. The consequences of such changes are the key point of our position, and are illustrated by means of a running example that considers automatic protection techniques for web applications. We chose this case primarily because web applications are widespread and their security is a major concern. In addition, this example is simple to analyze, it explains the essence of several modern detection approaches and, as summarized in Section 3, it has been generalized beyond tools to detect attacks against web application.

Although throughout our discussion we borrow examples and concepts from real-world applications of cloud computing (e.g., Amazon EC2, web services), our position focuses on cloud computing as a paradigmatic shift (discussed in Section 4), and the extent to which it may affect known security issues and countermeasures. Other aspects related cloud computing, such as new business models, economics, or mobility, are far from the scope of this paper and thus are not considered, as they do not impact directly the security concerns (while they are, of course, the fundamental reason for the adoption of cloud computing).

2 Security challenges

As discussed thoroughly in [1], the cloud computing paradigm presents some novel challenges to computer scientists. In our opinion, security-related challenges can be divided into two groups. On one hand, there are challenges that are already evident to cloud users (i.e., service providers), and as such demand for practical solutions (Section 2.1). On the other hand, there are challenges that will significantly influence the security of cloud computing in the long run. As explained in Section 2.2, this second group of challenges is relevant, yet less obvious.

2.1 Challenges with immediate impact

The challenges described in this section are already visibly impacting cloud users and providers. In particular, given the amount of data shared across these infrastructures, data confidentiality, trust relationships and shared reputation are concerning issues.

2.1.1 Data confidentiality

The obvious and, in general, effective measure to protect data confidentiality is encryption. However, encryption is not always a feasible solution, especially for data-intensive applications that require high I/O throughput. Although a scheme to compute arbitrary circuits over encrypted data (so to avoid encryption/decryption) has been proposed recently [16], in its current stage this solution requires significant efforts to be adopted in the real world. In addition, encryption is not straightforward when data is distributed; also, this solution may have a low acceptance rate and, more importantly, raises the issue of data property. As most of the users refrain from encrypting their laptop hard-drive because of the technical and computational overhead, in a similar vein, would users bother encrypting their virtual, remote storage? Moreover, if a remote storage is transparently encrypted (i.e., by the provider), whom the data belongs to? The user, the provider? And, is this fact provable? How?

2.1.2 Sharing shared resources

The security issues typical of shared hosting environments are magnified in the case of clouds, because the additional, *unperceived*, complexity due to dynamic resource slicing, allocation, replication and optimization, gives each user the illusion of being unique. In reality, each user (e.g., an actual system user or an application) operates in a *shared* environment. Therefore, users may behave maliciously, or compromise virtualization software (vulnerabilities in VMware have grown 35 times between 1999 and 2007 [25]), affecting other users and their reputation. A recent incident [22] that affected the reputation of a whole, shared Amazon EC2 cloud is discussed in [8] as a noteworthy example of this specific issue.

Observation To what extent users sharing the same cloud are isolated? Is it feasible to employ simple fail-over mechanisms to transparently "move" a mis-behaving user or process onto another cloud; and would this offer an adequate degree of protection?

2.2 Challenges with delayed impact

Debugging and auditing in large-scale, distributed systems unavoidably affect the foundations of secure software development. Although their impact may be delayed, and no incidents can be attributed directly to them as of now, we believe that these obstacles will influence significantly the security of the software developed for, or deployed onto, modern computing infrastructures.

2.2.1 Debugging in large distributed systems

Programmers know how to pinpoint and solve software flaws using debuggers, which allow to track the execution of even complex, multi-threaded processes and inspect the memory content. This routine task turns out to be a challenging research problem in the case of distributed applications [11, 15, 34]. Besides the intrinsic difficulties that programmers have to face, i.e., understanding what is "the memory", or the "process state", debugging tools devised for large-scale distributed systems are quite obtrusive (e.g., they require code annotation). In general, the existing tools are designed for critical systems and suitable for C-like languages, rather than web-oriented frameworks. In addition, bugs are difficult to reproduce in local, smaller configurations because testing and development environments might differ significantly from deployment conditions.

A less obvious complication that affects debugging is the "invasion" of web development frameworks. Rapid development frameworks are indeed very popular[2] and can speed up significantly the work of a programmer, because they hide many low-level details, exposing powerful abstract primitives. In some notable cases, such frameworks *are* the cloud service, thus are tightly coupled with the cloud provider, e.g., Google AppEngine. Because debugging "cloud-oriented" applications is an inherently difficult task, software flaws may become more prevalent. And, since such flaws are the main cause of security vulnerabilities, these aspects are likely to result in new venues for intrusions, and thus need to be considered thoroughly.

Observation A natural question regarding debugging may arise. What is the actual "programming language" a modern web application is written in? Is it the high-level, abstract framework or the language the framework is built upon? A flaw may hide deep down in this programming stack and, thus, affect *more* than one deployment (just like bugs in shared libraries in older systems were an enormous security issues in the mid-nineties).

2.2.2 Auditability

When disasters occur, reconstructing a "picture" of the system's status is vital. From a purely forensic point of view, monitoring and keeping track of a system's activity is as important as debugging. Unfortunately, this might in turn be very difficult in large-scale systems, since data and processes are distributed rather than contained within well-defined boundaries. Even simple tasks such as collecting logs are inherently more challenging when applications are distributed. In case of successful exploitation, a likely event in immature systems, the risk is that the compromised applications might leave insufficient or unreachable tamper evidence. For instance, is it always feasible to access the logs of all those hosted web services leveraged by an application that *we* developed? Despite the level of abstraction and the transparency offered by cloud services, developers should be aware that software is *not* running on bare-metal hardware.

3 Available countermeasures

Given the threat scenario briefly summarized in Section 1 and the obstacles mentioned in Section 2 (in particular, those described in Section 2.2), it is useful to discuss the potential consequences of combining a growing underground economy, armed with fast-spreading automated attacks, with the inherent fragility of a new, sophisticated and exciting infrastructure. In order to do so, we need to look at current security countermeasures: but of course, this is by no means a complete or exhaustive survey; it is limited to those concepts that are needed to explain our position.

Until now, research on security of web services and service oriented architectures (which are a key component of cloud computing) has focused on designing secure protocols for

[2]According to the most comprehensive ranking we were able to find, there are 98 web application development frameworks (as of April 2010) `http://hotframeworks.com/rankings`

exchanging messages that meet the confidentiality, integrity and availability requirements [2, 13, 4, 32]. Other approaches concentrated on secure patterns for modeling and developing services, mostly from a software engineering perspective [33, 19]. The goals of such measures are, basically, to prevent programmers — who are not necessarily security experts — from writing vulnerable code, rather than protecting services directly.

Protection for existing systems often revolves around the use of detection and prevention mechanisms, which nowadays are quite sophisticated, but still based on two complementary approaches that have been part of intrusion detection from its first inception [14]: recognizing known patterns of malicious signs (misuse detection), *versus* recognizing deviations from known normal activity (anomaly detection). For instance, the *Intrusion Detection System* (IDS) described in [23] models the normal characteristics of benign interactions between clients and the server-side applications at the HTTP layer. This system can effectively detect, for instance, code-injection attacks, which are visible into HTTP parameters. A similar system, described in [10], further develops client-side protection measures and, in addition, can detect attacks against the database tier by profiling benign queries to recognize deviating ones. These systems are said to be application-aware, because the knowledge they leverage is specific to the application layer protocol (e.g., HTTP).

To explain our position, we make use of the following example.

Running Example Let us assume that an HTTP-aware, anomaly-based IDS employs two simple models that capture, respectively, the length and the alphabet of the string parameters submitted through HTTP requests, which have the following structure:

```
POST /authenticate HTTP/1.1
Host: www.example.com
Content-Type: text/xml
Content-Length: ...

&name=administrator
```

In this case, the `/authenticate` handler accepts the `name` parameter via POST. If we exclude the data encoded in the header, the payload of the HTTP request is `&name=administrator`.

Let us suppose that the protected web application is designed to accept a limited set of values for the `name` parameter, i.e., `administrator` (length 13, alphabet `[a-z]`), `logs` (length 4, alphabet `[a-z]`), or `activities` (length 10, alphabet `[a-z]`). After having examined a sufficient amount of benign requests, the IDS learns that the normal length is, for example, 9, the variance is 21, and the alphabet is `[a-z]` (i.e., lowercase letters).

Let us now assume that the procedure invoked by the `/authenticate` handler is vulnerable to cross-site scripting injections, because the `name` parameter is not sanitized. Thus, an attacker may attempt to deploy a drive-by download kit, by submitting `name=<script src=//j.mp/xss>` (length 23, alphabet `[.<>/=a-z]`). The simple models used in this example (yet employed in advanced prototypes such as [23, 10]) are able to recognize the injection because `[.<>/=a-z]` $\not\subseteq$ `[a-z]`. Note that, in the case of particularly short injection vectors, the length model cannot recognize the attack. To alleviate this type of problems, modern systems actually adopt multiple models as well as probabilistic approaches, to assess the degree of anomaly in a more sound manner. However, the goal of this example is to show the essential concepts of anomaly detection to recognize evidence of malicious activity.

The aforesaid techniques are also suitable to protect operating systems. For example, a very effective technique to detect misbehaving processes consists in modeling the data passed to system calls and extracting some representative characteristics of the *Control Flow Graph* (CFG). The tools described in [31, 26] adopt this approach to detect, for example, whenever a program is forced to invoke an out-of-sequence system call, or a system call with an unexpected, i.e., too long, string argument that is likely to be evidence of a buffer overflow. In simple words, these techniques encode a process' behavior in terms of some features, which are then leveraged to assess the goodness of the system call sequences generated at runtime.

4 Signs of a paradigmatic shift

Even if cloud computing has been touted as a revolution in the information technology world, it is an ongoing change, and thus it might be difficult to assess its impact on security. Indeed, as noticed in Section 1, and thoroughly detailed in [8], the majority of current threats actually exploit vulnerabilities of the software, rather than of the cloud computing paradigm itself. Thus, at a first glance, one may think of protecting applications using traditional countermeasures. On the other hand, applying available countermeasures as they are, under the assumption that applications running in cloud computing environments are *just* regular applications, may reveal pitfalls in some cases.

In order to reason about security in the new scenario, we deem it useful to recall some analogies between the centralized approach of mainframes and the apparently centralized (but actually distributed) approach of cloud computing. There is an obvious difference in *scale* between the two eras: what was a single, powerful mainframe connected to dumb clients through a local, high-speed, switched network, has become a cluster of consolidated, managed machines (either virtual or physical) connected to clients through public, broadband, best-effort routed networks. Despite these differences, applications taking advantage of the cloud computing paradigm have striking similarities with the old-fashioned centralized software.

The case of web services is a particularly expressive example, as they are, to some extent, the unit of execution in software delivered as a service, much in the same way as the primitive functions of traditional operating systems. In fact, functions take arguments (e.g., numeric values, file descriptors, strings) and, when invoked by a process, perform a certain operation then, optionally, return a result. Similarly, services are invoked by clients and react according to the input given (e.g., SOAP or JSON data). A careful examination reveals that a web service is just *incidentally* a web application (i.e., merely from a technical point of view), since it relies on web-oriented technologies, but on a global scale it actually reminisces more of a local function call.

Ignoring this observation, protection mechanisms like HTTP-aware IDSs would appear a natural solution to protect web services. Unfortunately, these IDSs are aimed at blocking attacks directed toward classic web applications. But as we noticed, it is common practice to compose services together to provide richer functionalities. From a global viewpoint, these mash-ups of different services are likely to offer subtler exploitation opportunities. For instance, single services may be invoked in a benign manner, such that an IDS deployed to protect the web application (which supports the service) would not recognize any anomalous activity. However, said services could be invoked in such a way to create a malicious action that is visible only globally. In some sense, this is the dual of mimicry attacks [42], a stealthy evasion technique in which a process is violated by scrupulously altering the data passed to one or more single system calls, while the characteristics of the CFG are preserved, in order to fool security checks performed by an IDS. In the type of attacks that we envision against web services, the idea of mimicry attacks is reversed, that is, the "global CFG" (where functions nodes are replaced by services) is altered, while each elementary service is invoked benignly. This observation is further discussed in Section 4.2. In addition, in Section 4.1 we show a case where current approaches exhibit shortcomings in detecting client-side code injections even against a single services.

Observation In the light of the aforesaid points, would Internet security come to a point where it needs to be *re-thought*? Would we need to revamp our knowledge on protecting local operating systems and scale it for the next generation, global operating systems[3]?

In the remainder of this section, we detail the aforesaid cases by means of examples, to stimulate other questions and critiques.

4.1 The HTTP is the TCP

The practice of using application protocols, mainly HTTP, to encapsulate a wide spectrum of data types (e.g., binary files, streams of videos, chunks of data with well-defined semantics such as RDF) is becoming very popular. In some sense, HTTP is playing the role of a transport layer, that is, encapsulating a payload and sending it. Complex (even proprietary) protocols offer transparent communication layers between services over HTTP. A simple example is the WebDAV protocol (adopted, for instance, by the popular Google Calendar web service), which relies on HTTP to interface calendar clients and servers (actually, services). As we all know, HTTP sits on top of a real transport

[3]Note that operating systems delivered as mash-ups of web services actually exist in the real world. Notable examples are `http://www.silveos.com/`, `http://cloudo.com/`, `http://eyeos.org/`, `http://ghost.cc/`

layer; but since TCP is so transparent, reliable and highly-available, to some extent it can be considered as a network (or even physical) layer. In our opinion, the spreading of web services and cloud computing modifies the way the Internet networking stack is used by software, as shown in Figure 1.

From a security perspective, this observation suggests that a further layer of inspection is desirable to effectively detect those threats that leverage the actual communication protocol employed by a service. In fact, since the networking stack is evolving, the protection mechanisms (especially those that inspect the application layer) should step up as well, as exemplified in the following.

Running Example In Section 3, we described how a protocol-aware IDS analyzes HTTP messages and checks for their validity with respect to normal usage of the protected web application. Let us assume that a SOAP-based authentication system (also vulnerable to injections) replaces the old-fashioned web application. The HTTP requests have the following template:

```
POST /authenticate HTTP/1.1
Host: www.example.com
Content-Type: text/xml
Content-Length: ...

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ... >

  <SOAP-ENV:Header>
    <h:BasicAuth xmlns:h="http://soap-authentication.org/basic/2001/10/"
       SOAP-ENV:mustUnderstand="1">
      <Name>administrator</Name>
      <Password>broccoli</Password>
    </h:BasicAuth>
  </SOAP-ENV:Header>
  ...
</SOAP-ENV:Envelope>
```

In this example, the payload of the HTTP request is the whole SOAP envelope, which is just a sequence of characters. Suppose, however, that the IDS has a fall-back procedure that, in absence of GET or POST parameters, extracts the alphabet (`[.:="/<>a-zA-Z0-9]`) and the length's mean and variance[4]of the request body.

Let us complicate this example a little bit and assume that the IDS also employs a more sophisticated technique to extract the *syntax* of the body and encode it as a probabilistic grammar or a Markov model, a technique commonly used to detect attack vectors that alter the syntax of a string [31]. After training on some samples, such models can calculate the likelihood of a string with respect to the grammar learned. So, for example, given the sequence of symbols into an HTTP request body, it could tell XML, JSON or plain text apart, because they contain different symbols and also their syntax is dissimilar. Note that, however, this is far from having a multi-purpose parser capable of extracting the real parameters (i.e., `name`, `password`) that influence the behavior of the service. Thus, analyzing the entire SOAP block with the aforesaid approach is insufficient to distinguish between messages that contain "`administrator`" *vs.* "`<script src=//j.mp/xss>`", because, intuitively, these values are well "buried" by the extra content. Also, the vector's alphabet, `[.<>/=a-z]`, is perfectly compatible with `[.:="/<>a-zA-Z0-9]` as it contain no extra characters. For similar reasons, the length model is of little help, since injection vectors can be notably (almost arbitrarily) short.

Obviously, more accurate models may be devised to deal specifically with the simple case illustrated in this example; however, it is just as obvious that such an approach would be difficult to generalize. In fact, such models would require a language-specific parser to extract relevant content from the messages processed by the (custom) application. In addition, the structure of these messages must be known in advance, case by case, because it is impossible to automatically derive the actual variables (while this is doable for HTTP parameters).

As observed in this example, the ongoing change in the networking stack suggests that protocol-aware protection system should account for the actual protocol used by the services *on top* of HTTP. Otherwise, new threats that exploit the upper-layer protocols (e.g., legitimate SOAP content that hides malicious parameters) are difficult, or completely impossible, to detect. To make once again a comparison, anomaly detectors that inspect the payload of IP packets to recognize attacks [24, 27, 43, 45] are inaccurate at detecting most of those vectors that are malicious only by means of the application layer's semantic (e.g., a POST parameter with anomalous content). From the viewpoint of a lower-layer protocol, code injection can be "confused" with payload that encodes regular strings.

Observation To what extent the attacks against the future application (i.e., service) layer are recognizable at the future transport layer (e.g., HTTP)? Is it just a matter to perfecting existing tools to allow deeper examination, or the technical obstacles hide more issues?

4.2 The services are the functions

In the previous section we discussed the changes in the networking aspect of computing caused by service orientation and cloud computing. Similarly, in this section we discuss

[4]Mean and variance could be calculated assuming that valid values for the `name` are exactly the same as in the previous run of the example, and assuming that the SOAP messages have a recurring structure, which is perfectly realistic. The variance is indeed identical.

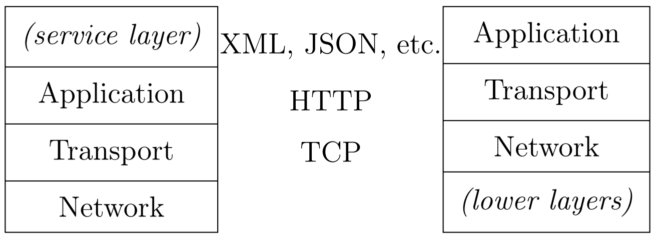


Figure 1: As described in Section 4.1, the change in the networking stack is noticeable from the traditional application layer (left) that, in the case of HTTP, is playing the role of a transport protocol (right) to encapsulate upper layer protocols (e.g., SOAP, JSON, XML), typical of modern web services.

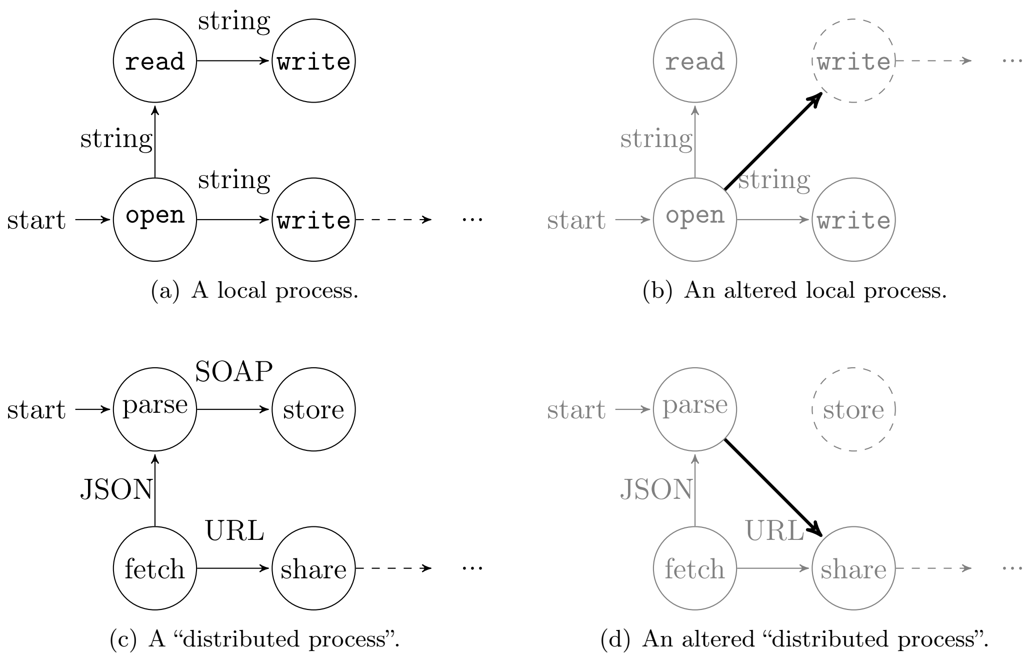
how the cloud computing paradigm is impacting the way applications are constructed and executed.

In a local system, processes originate whenever a certain program (e.g., a calendar application, an e-mail client) is executed. Basically, these programs are built upon a set of primitive functions, exposed by a certain programming language[5]. Cloud computing extends that model to a larger scale. Indeed, the development of applications is increasingly drifting toward the reuse of services (which have well-defined APIs, as classic functions do!), as opposed to the simple reuse of code.

In a traditional operating system, processes can be modeled by means of their CFG. A simple example is shown, using a simplified graphical notation, in Figure 2: (a) the `read` system call receives ans integer value, `open` is invoked to open a file which content is passed to `read`, and finally wrote to the standard output. Assuming that the process has a race condition vulnerability, for instance, an attacker may leverage such flaw to cause an unexpected transition in the CFG (b). In a similar vein, "distributed processes" of modern applications rely on several services (e.g., parse, build, share) to perform certain tasks and to achieve a global goal. In the example drawn in Figure 2 (c), we imagine a service that parses data submitted by a user through a form (e.g., `http://www.formspring.com/`, `http://www.formsite.com/`), serializes it onto a remote storage (e.g., `http://aws.amazon.com/s3/`, `http://drop.io`) and shares (e.g., `http://www.wuala.com/`) a link to it on a social network. Similarly to the case of local process exploitation, an attacker may leverage logic vulnerabilities to cause, for instance, a malicious redirection in a "distributed process" (d).

From a security perspective, in both cases, malicious behaviors can be exemplified as deviations from the expected sequence. On local processes, violations of the CFG can be detected with simple checks. It is, however, more difficult to envision a similar approach to recognize violations of a "global CFG". Since services are logically very similar to functions, we believe that the same techniques used to recognize bad-behaving processes may inspire new approaches to mitigate stealthy attacks against service mash-ups.

[5]Note that, although higher level functions and concepts such as libraries, objects, or classes are available to the programmer, from the operating system's perspective, running programs is all about invoking function in a certain sequence and passing data across functions.



(a) A local process. (b) An altered local process.

(c) A "distributed process". (d) An altered "distributed process".

Figure 2: Example of snippets of processes in a traditional operating system (a), modeled by means of the system calls they invoke. Data is passed across functions via parameters. In a similar vein, "distributed processes" (c) of modern applications rely on several services (e.g., parse, build, share) to perform certain tasks and to achieve a global goal. In this example, we imagine an application that parses data submitted by a user through a form, serializes it onto a remote storage and shares a link to it on a social network. In both the cases, malicious behaviors can be exemplified as deviations from the expected work-flow. On local processes, violations (b) of the CFG can be detected with simple checks. As discussed in Section 4.2, it is however more difficult to think of a similar approach to recognize (d) when a "global CFG" is altered.

Observation Event correlation techniques [40] originated from the need of detecting related alerts across several IDSs. However, these tools have provided no effective solutions to detect large-scale, slow attacks, which is indeed a very difficult problem. Interestingly, this type of attacks resemble the aforesaid global mis-behaving processes. Could correlation techniques attract more research efforts than in the past and finally mitigate a longstanding issue?

5 Conclusions

In this paper, we have discussed some key points that, in our opinion, motivate a constructive reconsideration of the current security measures.

The simple observation that paradigmatic changes (e.g., from thin client connected to a mainframe, to powerful workstations, to, once again, thin clients connected to a cloud) induce parallel changes in the security world, suggests a broad approach to the "novel" security issues. In the approach that we envision, the stack offered by the cloud computing paradigm needs to be mapped to the well-known hardware and software stack. In principle, this would help at mapping also the patterns of the traditional security issues onto the new stack. Examples of the insights that we believe this will make possible are outlined in Sections 4.1 and 4.2. Obviously, this mapping will not, by itself, lead to a complete description of the new threats. Rather, it will point out key areas to develop and refine, in a much similar way to what the periodic table did for the discovery of unknown chemical elements. Similarly, for some areas, this approach will indicate that many issues can be solved through an appropriate "porting" of the traditional security countermeasures to the cloud computing paradigm.

Even if exploring new business models opened up by cloud computing falls entirely outside of the scope of this paper, it is undeniable that the fast-growing underground economy has already embraced the cloud model (in fact, botnets are an embryonic distributed malicious infrastructure [7]). The fact that business-to-business interactions will also embrace this paradigm makes the problem even more evident and alarming, leading to a number of potential frauds on pay-per-use services.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[2] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, et al. Web services security (WS-Security). `http://www-106.ibm.com/developerworks/library/ws-secure`, 2002.

[3] C. Balding. Cloud security. `http://cloudsecurity.org/`, 2009.

[4] K. Bhargavan, C. Fournet, A. D. Gordon, and G. O'Shea. An advisor for web services security policies. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 1–9, New York, NY, USA, 2005. ACM.

[5] B. Brenner. Defining cloud security: Six perspectives. `http://www.networkworld.com/news/2009/092909-defining-cloud-security-six.html`, 2009.

[6] J. Brodkin. Gartner: Seven cloud-computing security risks. `http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853`, 2008.

[7] J. Carr. *Inside Cyber Warfare: Mapping the Cyber Underworld.* O'Reilly Media, Inc., 2009.

[8] Y. Chen, V. Paxson, and R. H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010.

[9] P. R. Clearinghouse. A chronology of data breaches. Technical report, Privacy Rights Clearinghouse, July 2009.

[10] C. Criscione, F. Maggi, G. Salvaneschi, and S. Zanero. Integrated detection of attacks against browsers, web applications and databases. In *Proceedings of the 5th European Conference on Computer Network Defense 2009 - EC2ND 2009*, 2009.

[11] D. Dao, J. Albrecht, C. Killian, and A. Vahdat. Live debugging of distributed systems. In *CC '09: Proceedings of the 18th International Conference on Compiler Construction*, pages 94–108, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] S. Das. Privacy and security top cloud concerns. `http://information-security-resources.com/2009/09/29/privacy-and-security-top-cloud-concerns/`, 2009.

[13] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. *Lecture Notes in Computer Science*, pages 335–350, 2003.

[14] D. Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, 13(2):222–232, 1987.

[15] D. Geels, G. Altekar, S. Shenker, and I. Stoica. Replay debugging for distributed applications. In *USENIX Annual Technical Conference*, volume 2006, pages 2–3, 2006.

[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, New York, NY, USA, 2009. ACM.

[17] Z. Godschalk. Cloud security alliance continues rapid growth. `http://www.cloudsecurityalliance.org/pr20090825.html`, 2009.

[18] S. Gordeychik. Web application security statistics. `http://projects.webappsec.org/Web-Application-Security-Statistics`, January 2010.

[19] A. Gordon and R. Pucella. Validating a web service security abstraction by typing. *Formal Aspects of Computing*, 17(3):277–318, 2005.

[20] D. Harris. Experts get serious about cloud security. `http://gigaom.com/2009/03/31/experts-get-serious-about-cloud-security/`, 2009.

[21] P. Kafka. The twitterhack is cloud computing's wake-up call: Time for security that works. `http://mediamemo.allthingsd.com/20090715/the\-twitterhack-is-cloud-computings-wakeup\-call-time-for-security-that-works/`, 2009.

[22] B. Krebs. Amazon: Hey spammers, get off my cloud! `http://blog.washingtonpost.com/securityfix/2008/07/amazon_hey_spammers_get_off_my.html`, July 2008.

[23] C. Kruegel, W. Robertson, and G. Vigna. A Multi-model Approach to the Detection of Web-based Attacks. *Journal of Computer Networks*, 48(5):717–738, July 2005.

[24] C. Kruegel, T. Toth, and E. Kirda. Service-Specific Anomaly Detection for Network Intrusion Detection. In *Proceedings of the Symposium on Applied Computing (SAC 2002)*, Spain, March 2002.

[25] K. Lamb. Virtualization and security. `http://blogs.iss.net/archive/virtblog.html`, September 2007.

[26] F. Maggi, M. Matteucci, and S. Zanero. Detecting intrusions through system call sequence and argument analysis (preprint). *IEEE Transactions on Dependable and Secure Computing*, 99(1), 2009.

[27] M. V. Mahoney. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 19th Annual ACM Symposium on Applied Computing*, 2003.

[28] P. McFedries. The cloud is the computer. `http://www.spectrum.ieee.org/computing/hardware/the-cloud-is-the-computer`, August 2008.

[29] B. Miller. Yes, 'security in the cloud' is the way to go. `http://www.networkworld.com/columnists/2006/021306faceoffyes.html`, 2006.

[30] B. Miller. Cloud security alliance. `http://www.cloudsecurityalliance.org/`, 2009.

[31] D. Mutz, F. Valeur, C. Kruegel, and G. Vigna. Anomalous System Call Detection. *ACM Transactions on Information and System Security*, 9(1):61–93, February 2006.

[32] M. Naedele. Standards for XML and Web services security. *Computer*, pages 96–98, 2003.

[33] Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono. Model-driven security based on a web services security architecture. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 7–15, Washington, DC, USA, 2005. IEEE Computer Society.

[34] P. Reynolds, C. Killian, J. Wiener, J. Mogul, M. Shah, and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *Symposium on Networked Systems Design and Implementation*, pages 115–128, 2006.

[35] N. Roiter. Mcafee, verizon business team up on cloud security. http://www.searchsecurityasia.com/content/mcafee-verizon-business-team-cloud-security, 2009.

[36] M. Savage. Security challenges with cloud computing services. http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1368905,00.html, 2009.

[37] A. Stern. Update from amazon regarding friday's s3 downtime. http://www.centernetworks.com/amazon-s3-downtime-update, July 2008.

[38] T. A. S. Team. Amazon s3 availability event: July 20, 2008. http://status.aws.amazon.com/s3-20080720.html, July 2008.

[39] D. Turner, M. Fossi, E. Johnson, T. Mark, J. Blackbird, S. Entwise, M. K. Low, D. McKinney, and C. Wueest. Symantec Global Internet Security Threat Report – Trends for 2008. Technical Report XIV, Symantec Corporation, April 2009.

[40] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Trans. Dependable Secur. Comput.*, 1(3):146–169, 2004.

[41] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[42] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, New York, NY, USA, 2002. ACM Press.

[43] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*. Springer-Verlag, September 2004.

[44] S. Wilson. Appengine outage. http://www.cio-weblog.com/50226711/appengine_outage.php, June 2008.

[45] S. Zanero and S. M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 412–419. ACM Press, 2004.