System Design Artifacts for Resilient Identification and Authentication Infrastructures

Diego Kreutz, Oleksandr Malichevskyy
LaSIGE/FCUL, Portugal
{kreutz, olexmal}@lasige.di.fc.ul.pt

Eduardo Feitosa, Kaio R. S. Barbosa and Hugo Cunha
IComp/UFAM, Manaus, Brazil
{efeitosa, kaiorafael, hugo.cunha}@icomp.ufam.edu.br

***Abstract*—The correct and continuous operation of identity providers and access control services is critical for new generations of networks and online systems, such as virtualized networks and on-demand services of large-scale distributed systems. In this paper, we propose and describe a functional architecture and system design artifacts for prototyping fault- and intrusion-tolerant identification and authentication services. The feasibility and applicability of the proposed elements are evaluated by using two distinct prototypes. Our results and analysis show that building and deploying resilient and reliable infrastructure services is an achievable goal through a set of system design artifacts based on well-established concepts from security and dependability. We also provide a performance evaluation of our resilient RADIUS service compared with the long standing FreeRADIUS.**

Keywords*—*System design; fault and intrusion tolerance; identification and authentication services; network access control.

I. Introduction

The growth of Authentication and Authorization Infrastructure (AAI) services is motivated by the fact that users are allowed to transparently access different services (e.g., Facebook, Google, Twitter, and Amazon) with a single credential or authentication session. These services rely on Identity Providers (IdPs) or Authentication, Authorization, and Accounting (AAA) protocols to identify and authenticate the user before granting him access to the requested resources or services. OpenID [1] and RADIUS [2] are examples of such services.

Despite the importance of AAIs for service infrastructures such as clouds and virtual networks, there are still open questions regarding their availability and reliability. This can be supported by recent work showing that digital attacks and data breach incidents are growing [3]. Additionally, advanced persistent threats [4] are becoming one of the top priorities of security specialists. Therefore, security and dependability properties should be the top priority of future AAIs.

Most of the existing RADIUS-based services and OpenID-based IdPs do not completely address security and dependability properties such as confidentiality, integrity, and availability. This can be observed on the services' online documentation and deployment recommendations [5][6][7][8][9]. Some implementations and deployments provide basic mechanisms to improve the service's reliability and robustness, such as SSL communications and simple replication schemes to avoid eavesdropping and tolerate stop failures, respectively. Hence, there are opportunities for further research with the ultimate goal of designing more resilient solutions which are able to deal with new threats and cyber attacks.

To the best of our knowledge, this paper proposes the first set of system design artifacts and functional architecture to design and deploy fault- and intrusion-tolerant identification and authentication services. Two distinct prototypes, RADIUS and OpenID, are used as proof of concept to demonstrate the applicability of the functional architecture and system design artifacts. We also briefly, we briefly discuss components and essential building blocks for implementing more robust and secure services. We conclude with an analysis of the approaches and requirements for deploying resilient services.

The next section introduces the motivation of our work. In addition, the functional elements and system design artifacts to develop robust and reliable AAI services are described in Section IV. Thereafter, the results are analyzed and discussed in section V. Lastly, Sections III and VI comprise of the related work and final remarks.

II. Motivation

AAI solutions are often based on protocols like OpenID and RADIUS. However, both of them are not designed with features for robust security (e.g., strong confidentiality) and dependability (e.g., high availability), being frequent targets of attacks and data theft attempts (e.g., user credentials).

OpenID is a framework to build identity providers [1]. It is based on open Hypertext Transfer Protocol (HTTP) standards, which are used to describe how users can authenticate on third party services through their own IdP. There are two main advantages of this approach. First, it allows identification and authentication protocols to be transported over standard Web protocols. Second, users need only one single credential to access different services provided that service providers accept external IdPs.

In spite of allowing the user to have a single credential to access multiple domains, there are different security issues on the OpenID identification scheme and service availability. Recent research has shown that the discovery and authentication steps are vulnerable to cross-site request forgery attacks, phishing and man-in-the-middle [10]. Also, its availability is vulnerable to denial of service attacks and protocol handling parameters [11][12].

RADIUS is an AAA protocol. The authentication verifies user's identity prior to granting access to the network or service. Authorization is used to determine which actions a user can perform after a successful authentication. Accounting provide methods for collecting data about the network or service usage. Collected data can be used for billing, reporting and traffic accounting. Therefore, RADIUS is commonly used to provide AAA features for infrastructures such as corporate networks and carrier grade provider networks.

The main security issues of RADIUS are in the protocol

 ISBN: 978-1-61208-330-8

specification and poor implementations [13]. Regarding flaws in the protocol, RADIUS does not validate the integrity of some packages (`Access-Request`) and does not provides mechanisms against reflection attacks. As well as this, existing implementations of RADIUS are also susceptible to dictionary, man-in-the-middle and spoofing attacks.

Fault and intrusion tolerance. We can choose two different approaches when designing secure and resilient systems. First, one can assume that it is possible to build robust and secure enough systems. However, as it is well known, a system is as secure as its weakest link. Moreover, it can be considered as secure until it gets compromised. Hence, the second approach is to assume that eventually the system will fail or be intruded. With this in mind, one can design highly available and reliable systems by leveraging mechanisms and techniques capable of enabling them to operate under adversary circumstances, such as non-intentional failures and attacks.

Our system design artifacts and functional architecture support the second approach, i.e, we do not intend to solve all security and dependability problems of AAI's services. Yet, by taking advantage of advanced techniques and resources from different domains we can build fault- and intrusion-tolerant systems capable of ensuring essential properties such as integrity, confidentiality, availability and reliability.

III. Related Work

Despite the existence of different solutions and components that can be used to improve the security and dependability of AAI services, such as advanced replication techniques, virtualization, proactive and reactive recovery techniques and secure components, there are no methodologies, functional architectures or a set of system design artifacts that are capable of demonstrating how different elements can be orchestrated to build highly available and reliable systems. Existing approaches and solutions are designed for specific scenarios or a particular system. One example is to use TPMs to create trustworthy identity management systems [14]. While the solution allows one to create trustworthy mechanisms for issuing tickets, it is not designed for high availability or fault and intrusion tolerance. Another example is a cooperative coordination infrastructure for Web services [15], which proposes security mechanisms that allow reliable coordination of services even in the presence of malicious components. It works as an integration infrastructure for Web services, being supported by a set of service gateways and one resilient tuple space. This solution offers fault and intrusion tolerance capabilities for the coordination infrastructure. However, it does not represent a generic or adaptable architecture that can be applied to improve the robustness and security of different systems. Furthermore, it only protects the data confidentiality of a particular service through authentication and encryption mechanisms. Therefore, such scenarios indicate the need of more general architectures and system design artifacts that can be combined in a more systematic way to develop and deploy services with higher security and dependability properties.

IV. System Design Artifacts

A. Overview of functional elements

Figure 1 shows a simplified representation of the four main functional elements: (a) client; (b) service; (c) gateway; and (d) Critical Infrastructure Service (CIS). This is the typical functional architecture of computing environments where identification and authentication solutions are separated services. In addition, the fifth element is a secure component, which can be used in conjunction with any of the previously mentioned elements. Its purpose is to provide additional support for ensuring properties such as confidentiality, integrity, and timing, when ever required.

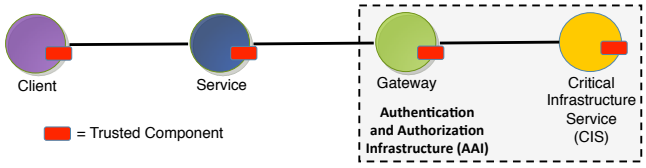

Fig. 1. Main functional elements.

A client can be a user trying to access the network or a networking element. In other words, it represents a generic element whose definition depends on the target scenario.

In a typical network, the service may represent elements such as wireless routers or Ethernet access switches. For Web applications based on OpenID, a service can be a relying party or an access control subsystem of online shopping Web sites.

A gateway provides connection between the service and the critical infrastructure service, a.k.a., back-end service. It has two basic functions. First, it handles multiple protocols from both sides, acting as a protocol gateway. The second function of this element is to mask the replication protocols and mechanisms used to deploy resilient back-end services, providing transparent backward compatibility with AAI protocols such as RADIUS and OpenID.

The back-end service is the critical element of the infrastructure, which is assumed to require higher levels of security and dependability assurances. A CIS can be part of the local domain or provided by third parties as an on-demand service, for instance. It is assumed that these services must tolerate different types of faults such as those caused by unexpected behavior or attacks, and work correctly in case of intrusions. OpenID providers and RADIUS services are examples of critical AAI systems for networked infrastructures and online services. Therefore, failures on these services can potentially impact a corporation's systems and business.

Lastly, secure components can help to ensure properties such as data confidentiality, integrity checks, and timing assurances to specific parts of the system. As an example, user keys can be stored on a smart card. Similarly, server keys and authentication tokens can also be securely stored in secure components. Furthermore, all critical cryptographic operations can be safely executed by these trusted components, without compromising sensitive data in the case of intrusions. Currently, server and self-signed Certificate Authority (CA) keys are stored in the server's file systems. Hence, any system administrator has easy access to them, representing potential security threats.

B. Design artifacts for resiliency

Figure 2 illustrates architectural elements with added system design artifacts for augmented security and dependability properties. The architecture allows different fault thresholds or

 ISBN: 978-1-61208-330-8

replication techniques in a per element basis. For instance, the service, gateway and CIS elements can have distinct characteristics when ever they need to ensure specific reliability and availability requirements, such as crash faults, arbitrary faults, or resist to resource depletion attacks.

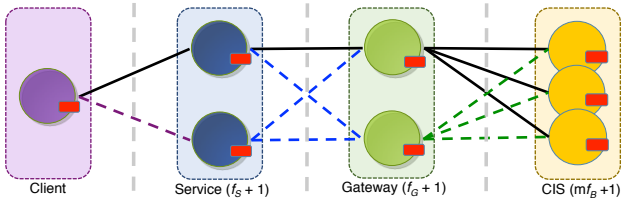

Fig. 2. Architectural components overview.

We assume that the service and gateway elements are designed to tolerate crash faults and detect some specific problems such as message integrity or authenticity violations while tolerating up to f_S and f_G simultaneous failures, respectively. Secure components can be used to verity message integrity and authenticity if sensitive data and procedures are required to accomplish the task. Otherwise, simple software-based verification methods can be sufficient to do the job.

Clients connect to any service replica, while a service can connect to any gateway. The connection to the replicas can be controlled using simple lists, which happens in AAA protocols, or round-robin for load balancing, for instance. However, in functional architecture there is no strict need for load balancing since the main goal is to provide fault tolerance. Hence, it is reasonable to assume that components are configured with at least a simple circular list of replicas.

On the other hand, the CIS does not support distinct methods to choose which replica to connect to. Gateways have to know all replicas required to support the number of faults assumed in the system. Using as an example a system that requires $2f + 1$ replicas to tolerate f faults, gateways need to known at least $2f + 1$ replicas to ensure that arbitrary failures on the CIS are going to be masked as long as $f + 1$ replicas are correct.

As a fault- and intrusion-tolerant infrastructure, the CIS is implemented with protocols to tolerate arbitrary faults. Gateways receive the responses from all (or at least enough to ensure a safe voting) back-end replicas and decide which one is the correct response that should be forwarded to the service or client. To achieve this goal the back-end service requires $mf + 1$ replicas, where m refers to the specific BFT algorithm [16] in use (e.g., $m = 2$, $m = 3$).

C. Essential building blocks

The main building blocks are technologies and components that make it possible to conceive resilient and more secure services based on the proposed functional architecture.

Virtual machines provide flexibility and agility to deploy systems and services. In highly resilient systems, mechanisms like proactive-reactive recovery and diversity can leverage functionalities provided by hypervisors, such as fast start, stop, suspend, resume and migration of virtual machines.

Replication protocols represent one of the major building blocks of resilient services. State machine replication and quorum protocols are common approaches to mask arbitrary faults on dependable systems. Replicas allow the system to tolerate up to f simultaneous faults without compromising its operation. Additionally, complementary ingredients for high availability and robustness are proactive and reactive recovery mechanisms and techniques to increase the diversity of the system [17].

Secure components are small and reliable pieces of software, and/or hardware, capable of ensuring critical properties or functions of the system, such as integrity control, timing and data confidentiality. They can be used in different parts of the functional architecture. For instance, in an OpenID-based authentication solution both end user and the server can trust their sensitive data (e.g., certificate, keys) and crypto functions to a trusted component. Hence, a compromised server will not leak confidential data like private keys or user's tokens.

Secure end-to-end communication. It is necessary to achieve confidentiality and privacy of user data. Protocols such as Transport Layer Security (TLS) and Tunneled Transport Layer Security (TTLS) can be leveraged to provide reliable channels, mutual authentication and server authenticity verification. These functions can be helpful to avoid attacks like man-in-the-middle and eavesdropping.

D. Requirements and components

Table I shows the properties and requirements for designing and deploying services with different levels of resiliency and trustworthiness. We use the notion of trust to indicate whether the system is capable of ensuring data confidentiality of sensitive data such as private keys. Most of the existing identification and authentication services belong to the first three classifications, where "−−" means only primary-backup replication to tolerate crashes of the master server. In other words, those services are less secure and not highly resilient. For instance, an attacker can sequentially compromise all servers since there are no advanced recovery mechanisms in place, such as proactive recovery, to ensure the system's liveness and reliability.

TABLE I
SERVICE PROPERTIES AND REQUIREMENTS/COMPONENTS.

Properties	Secure comp.	Replication protocol	Recovery mechanism	Wormhole model	Intrusion-tolerant
1. Untrusted	no	no	no	no	no
2. Trusted but not resilient	yes	no	no	no	no
3. Resilient (−−) but not trusted	no	yes	no	no	no
4. Resilient but not trusted	no	yes	yes	no	no
5. Resilient but not trusted	no	yes	yes	yes	no
6. Resilient and trusted	yes	yes	yes	yes	yes
7. Resilient and trusted (++)	yes++	yes	yes	yes	yes

Our system design artifacts and functional architecture are expected to contribute to the development of services with properties of classes 4 to 7, where "++" indicates multiple verification points (e.g., the correctness and authenticity of a message could be verified within any element of the functional architecture, potentially reducing the request-response time by taking further actions as soon as possible). Property 4 does not use the wormhole model [18], while 5 does. This model

 ISBN: 978-1-61208-330-8

proposed hybrid distributed systems comprised of two parts: the payload (main parts and functions of the system) and a tiny subsystem with stronger properties for ensuring minimal timing requirements of the system, such the synchrony those needed to ensure the finalization of consensus protocols. It means that the former can not support an asynchronous system since there is no way to assure that consensus protocols, which are one of the basic building blocks of replication protocols, will finish their execution.

A system of class 5 is not intrusion-tolerant because it does not use secure elements to ensure data confidentiality. This is precisely one of the state machine replication and BFT limitations. These protocols are designed to ensure integrity and availability, but not data confidentiality. Therefore, additional mechanisms, such as secure components, are required to ensure the system's sensitive data confidentiality. Furthermore, the worm hole model is required to ensure minimal synchrony requirements of consensus protocols if the system if assumed to be asynchronous.

Lastly, systems of type 6 and 7 need a wormhole to rely on specially designed trusted components to ensure the minimal and critical system properties, which are required if the system is assumed to be asynchronous. However, if the system is synchronous or partially synchronous, then a wormhole is not required for timing purposes, for instance.

Another difference between systems of type 6 and 7 lies in the more extensive use of secure components. While in type 6, trusted components are used only in the client and back-end service, a system of type 7 requires secure components in other elements as well, such as gateways and services. One use of these additional secure components, on different architectural elements, is to safely earlier detect corrupted messages.

E. Deployment scenarios

Figure 3 shows the main trade-offs of service deployments. Despite the performance gains when using shared memory replication solutions such as Intrusion Tolerance based on Virtual Machines (ITVM), which uses a single physical machine and shared memory for communication purposes between virtual machines [19], these services can suffer from resource depletion attacks and be affected by infrastructure incidents [17]. Nevertheless, depending on the needs and requirements of the target environment, an ITVM-based system can be the most adequate solution. On the other hand, resilient systems using techniques provided by replication frameworks, such as BFT-SMaRt [20], allows us to create more robust services which are capable of achieving higher degrees of availability through multiple physical machines and/or multiple domains. A physically distributed system can leverage the resources and defense mechanisms of multiple domains, but with the burden of lower overall performance. Therefore, one can conclude that there is no unique solution to all problems. The mechanisms and protocols to be employed have to be analyzed and chosen based on requirements and guarantees needed by the target environment. Only with these inputs can one decide which are the best system artifacts for building a particular solution.

Resilient services using distributed machines across multiple domains (e.g., distinct clouds) are capable of tolerating physical hazards of machines and domains (e.g., network connection problems, energy outages and disk failures) as well as logical problems (e.g., misconfigurations of systems/networks and resource depletion attacks) by leveraging the support offered by each infrastructure. In practical terms, it has already been shown that cloud providers can tolerate DDoS attacks of big proportions without incurring losses for customers [21]. One of the resources against this kind of attack are the geographically distributed data centers, which together can form a robust and diversified infrastructure.

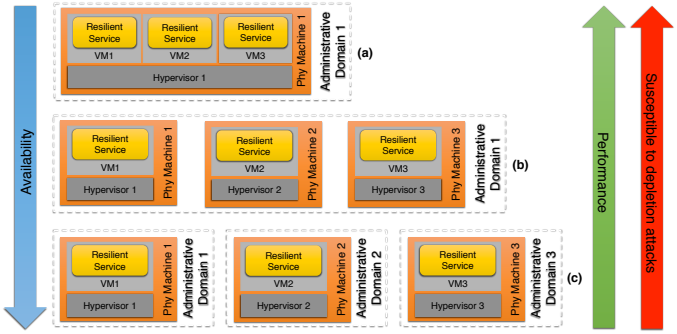

Fig. 3. Service deployment configurations.

V. Results and Discussion

A. RADIUS and OpenID prototypes

To evaluate the functional architecture and system design artifacts, two fault- and intrusion-tolerant prototypes were developed, one an OpenID provider and another, a RADIUS server. Our prototypes use the BFT-SMaRt [20], an open source and free Java-based library that provides high performance Byzantine Fault-Tolerant (BFT).

BFT-SMaRt can be leveraged to deliver a resilient service architecture that requires higher levels of availability and a lower probability of faults due to depletion attacks (e.g., performance or functionality degradation caused by a resource consumption racing or exhaustion). In addition, the state machine replication framework allows us to deploy replicas in a single physical machine, in multiple physical machines, or in multiple physical machines spread throughout different domains (e.g., multiple clouds). Consequently, replicas need to send and receive data over reliable and authentication channels, as is the case in BFT-SMaRt, to avoid attacks such as eavesdropping and man-in-the-middle.

The two prototypes follow the current standards of OpenID and RADIUS protocols, respectively. This means that any application based on these protocols will work with the resilient and more secure version of the service without requiring any modification. Hence, OpenID providers can offer to their users more reliable and secure services, which are able to support faults and intrusions in a smooth and transparent way. Due to space constraints, we only briefly introduce the OpenID BFT prototype in the following section. It is worth mentioning that most of the system design and implementation aspects apply to both RADIUS BFT and OpenID BFT.

B. OpenID BFT implementation

Figure 4 gives an overview of the OpenID BFT, which is based on the proposed functional architecture and system

 ISBN: 978-1-61208-330-8

design artifacts. Dashed lines indicate alternative paths used in case of failures of the default paths (solid lines). The timeouts are used to detect faults. While timeouts A and B and default properties of the functional architecture, timeout C is a specific requirement of OpenID.

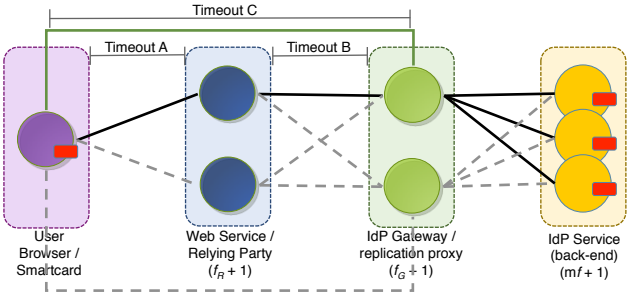

Fig. 4. Overview of the OpenID-BFT functional architecture.

In the OpenID BFT architecture, a client uses a Web service (relying party) that redirects him to his respective OpenID provider (IdP gateway). The user's credentials are required in the identification process, done by the IdP service replicas.

We have implemented the OpenID prototype with the *openid4java* library [22] (version 0.9.8), which supports OpenID 1.0 and 2.0. Our implementation defines OpenID 2.0 as the default authentication scheme. It is a fully fledged identity provider which is capable of tolerating up to f arbitrary replica failures. Moreover, secure elements are employed on the client side and authentication server to ensure the confidentiality of sensitive data such as user and server keys, self-signed CA's private key and session tokens. More information regarding the system's building blocks, design and implementation choices (e.g., state machine replication and secure elements) can be found in [17].

C. Fault thresholds and detection mechanisms

RADIUS BFT and OpenID BFT use the BFT-SMaRt framework, requiring $3f + 1$ replicas in the CIS to tolerate up to f faults or intrusion. We have introduced a third, yet fictitious, prototype called SM Service, where SM stands for shared memory. Its purpose is to describe the requirements of an ITVM like solution, which uses virtual machines and shared memory to tolerate faults and intrusions [17]. SM Service requires $2f + 1$ replicas to tolerate up to f arbitrary faults. However, it works with the best case, i.e., only $1f + 1$ active replicas. When ever the consensus protocol cannot finish due to differences in replicas' responses, the remaining replicas (f) are awakened and used to reach an agreement. One of the main disadvantages of this approach is the fact that it runs only on a single machine, i.e., its availability and operation can be affected by resource exhaustion attacks and infrastructure breakdowns. When high availability is required, or the system is subject to depletion attacks, solutions such as OpenID BFT and RADIUS BFT are needed to address those challenges.

Table II summarizes the fault model and fault threshold of the main component of the architecture. It also identifies example of components in real environments, such as AAA and OpenID infrastructures. As can be observed, it is assumed that services and gateways have a fail-stop (crash) behavior. Nevertheless, they can also detect some arbitrary behavior, such as malformed packets and corrupted messages, as is the case of the gateway element of our resilient RADIUS service. Another interesting potential use case are software defined networks, which still lack security and dependability mechanisms and protocols [24].

Services and gateways support fail-stop and a sub-set of arbitrary faults. Figure 5 shows the fault detection mechanisms. Only abnormal behavior like message corruption or forgering can be detected by clients and services. Yet, gateways are capable of detecting and masking arbitrary faults of the CIS element. Lastly, the CIS tolerates intrusions in up to f replicas once those events can be treated as arbitrary faults.

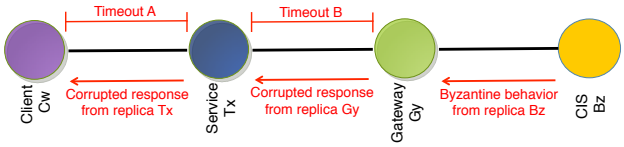

Fig. 5. Fault detection mechanisms.

D. A secure TLS component

Table III summarizes the methods required to implement a secure TLS component. To ensure mutual end-to-end authentication in RADIUS and OpenID, secure components can be used both at the client and server side [25]. These components are key design pieces to ensure properties such as confidentiality, storing sensitive data (e.g., secret key and attributes) and executing critical operations on it in a safe way.

TABLE III
SECURE TLS COMPONENT INTERFACE.

Method	Input	Output
`generate-Random`	Random number `size` (in bytes).	`Random number` with a specific size.
`extract-PreMaster`	Client's `premaster secret`.	`True` if *premaster* is correctly deciphered, `false` otherwise.
`generate-Master`	Random numbers from client and server.	`True` if the *master* secret was generated, `false` otherwise.
`getServer-FinishMsg`	`Hash` of the *record stream*.	Finalization message of the server.

A secure TLS component needs to be designed to provide the required methods to execute a TLS handshake, in case, four methods are needed to accomplish this task. Any outside software component can invoke those methods to execute a handshake between a client and the authentication server. No sensitive data leaves the secure TLS component. For instance, the server's private key, which is required to decipher the premaster key sent by the client and to generate a master key, can only be used inside the secure component. Therefore, an intruder cannot compromise the confidentiality of the server.

We implemented a secure TLS component based on the methods specified in Table III. It is used by RADIUS BFT's CIS to ensure a secure and reliable mutual EAP-TLS authentication between the user (using a certificate) and the AAA authentication server. In the case of OpenID BFT, the secure element ensures the confidentiality of user credentials and server keys and session tokens.

 ISBN: 978-1-61208-330-8

TABLE II
FAULT MODEL, FAULT THRESHOLD AND EXAMPLE OF COMPONENTS.

Component	Fault model	Replicas	Faults	Example of real use case components
Client	—	—	—	End user, network device
Service	Crash	$f_S + 1$	f_S	WiFi router, network management services, OpenFlow switch
Gateway	Crash	$f_G + 1$	f_G	New element interfacing with the target service and CIS
CIS	Byzantine	$mf_B + 1$	f_B	RADIUS AAA server, OpenID server, NIB of an OpenFlow controller [23]
Secure component	Crash	1	0	TLS keys and cryptographic methods

E. Properties, characteristics and performance

In Table IV, we sum up the main properties and characteristics of our prototypes. As can be observed, they are similar for OpenID BFT and RADIUS BFT because both of them leverage the same kind of architectural elements, replication mechanisms and secure components. Again, we have the SM Service for simple comparison purposes.

TABLE IV
PROTOTYPES' PROPERTIES AND CHARACTERISTICS.

Property/support	OpenID BFT	RADIUS BFT	SM Service
1. Multiple physical machines	yes	yes	no
2. Trusted components	yes	yes	yes
3. Hypervisor is trusted and secure	no	no	yes
4. Depletion attacks susceptibility	low	low	moderate
5. Performance (operations/s)	moderate	moderate	high
6. Availability guarantees	high	high	moderate
7. Arbitrary faults tolerance	yes	yes	yes
8. Intrusion tolerance	yes	yes	yes

Susceptibility to depletion attacks is intrinsically related to virtual machines using the same hypervisor. It is moderate to high on solutions based on a single hypervisor because a depletion attack can compromise the performance of non-malicious virtual machines in more than 50% of cases, depending on the specific attack. Examples of such resource exhaustion attacks and their impact on virtual machines of the Xen hypervisor can be found in [17].

Virtual machines on a single hardware platform can use shared memory spaces to execute protocols such as consensus [19], while frameworks such as BFT-SMaRt rely on message communication systems, whose performance depends on the specific algorithms being used and the corresponding implementation details. In OpenID BFT and RADIUS BFT, we use the BFT-SMaRt framework, which implements a set of optimization for state machine replication [16]. Therefore, we can consider its performance as moderate when compared to an ITVM-based solution, which is the best known performance setup for executing a state machine replication protocol. Moreover, other state machine replication implementations using non-optimized protocols would lead the system to lower performance measurements when compared to BFT-SMaRt.

It is well known that fault- and intrusion-tolerant mechanisms introduce some overhead in the system. To give an idea of the overhead, we measured our RADIUS BFT implementation and compared it with FreeRADIUS, which is a well-known and widely deployed implementation of RADIUS. The authentication latency increases by approximately an order of magnitude. Even though, it keeps below 200ms, which is an acceptable (non-perceptible by normal users) value for an authentication system. We also observed a drop in the throughput, i.e., number of authentications per second. In this case the difference narrows down to an overhead of about 5% to 40%, depending on the system's specific configurations and optimization. One of the reasons for the lower impact on the system's throughput is regarding the fact that BFT-SMaRt has a highly optimized batching subsystem for state machine replication, which is able to process a high volume of requests without impairing the system latency.

In summary, system design and development decisions should take into consideration the specific requirements of the target environment. While a solution like SM Service would be more suitable when depletion attacks are unlikely to happen, the hypervisor can be considered a trustworthy element and where high availability (e.g., support operation even under network disruption or other infrastructure disasters) is not a requirement. Yet, services such as OpenID BFT and RADIUS BFT are more indicated when high availability is required, performance is not the most critical issue, the hypervisor cannot be trusted, or when resource exhaustion attacks can happen. These sorts of solutions can resist to different kinds of threats or infrastructure incidents once replicas can be deployed in different physical machines and domains.

Resilient RADIUS versus FreeRADIUS

The throughput of the resilient RADIUS service and the FreeRADIUS server was measured using 2 to 20 simultaneous supplicants. Each client was configured to execute 10.000 sequential authentications using the same credentials. Furthermore, each authentication requires exactly ten packets, which needs to be considered when calculating the number of authentications per seconds. Therefore, we used a C program to measure the number of packets cached by `tcpdump`.

As shown in Figure 6, the throughput of the resilient RADIUS remains almost stable, while it varies for `FreeRADIUS`. This variation is due to the dynamic pool of threads, which automatically increases the number of working threads based on the number of authentication requests. Thus, it causes a slightly decrease in performance from 4 to 10 simultaneous clients. Afterwards, by activating new threads, the system's performance goes up again. The FreeRADIUS was configured with a minimum of 3 active threads and a maximum of 30 threads.

 ISBN: 978-1-61208-330-8

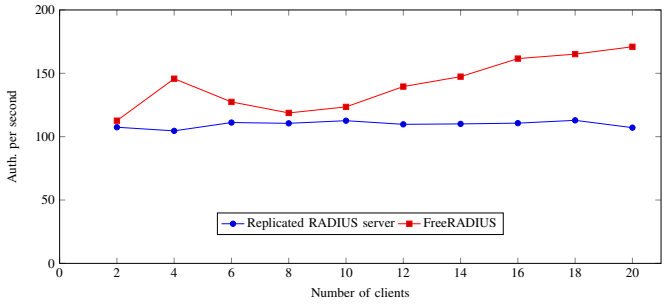

Fig. 6. Resilient RADIUS and FreeRADIUS throughput.

The stable throughput of the replicated RADIUS can be attributed to the gateway's throughput and the AAA replication. It sequentializes the network access servers' requests, sends each request to the replicas and clocks waiting for an answer. A thread pool, similarly to FreeRADIUS, could be used to increase the gateway's throughput. Moreover, the replicated RADIUS server also poses some limits to the system performance since requests must be acknowledged and ordered among all replicas. Nevertheless, the BFT-SMaRt system has a highly optimized batching and parallelization sub-system capable of sustaining high throughputs with higher number of clients (e.g., thousands) [20].

On the other hand, the latency almost doubles, going from nearly 100ms for FreeRADIUS to almost 200ms for the resilient RADIUS. The main problem of the latency lies on the gateway and the trusted component, due to their implementations. Both sub-systems could be re-designed to better explore concurrency and/or parallelism.

VI. Conclusion

This paper presented the first functional architecture and system design artifacts for designing and deploying more robust and reliable identification and authentication services such as OpenID and RADIUS. We believe that this is an important step for developing more systemic countermeasures against new security threats. Our results and evaluations indicate that we are able to build resilient and more secure identification and authentication infrastructures by combining important mechanisms and techniques from security and dependability.

We discussed how a functional architecture can be combined with system design artifacts to build different fault- and intrusion-tolerant services. The same components were successfully applied to build two distinct prototypes.

Interestingly, we believe that the proposed functional architecture and system design artifacts can be also extended to different scenarios. Based on our previous work and observations [17], we could apply the same concepts and components to create secure and dependable control platforms of software defined networks [24], where the CIS is a consistent and fault tolerant distributed data store [23], and resilient event brokers for monitoring cloud infrastructures [26], for instance.

Acknowledgments

This work is supported by EU's 7th Framework Program (FP7), through the project SecFuNet (FP7-ICT-STREP-288349), and by CNPq/Brazil, through grants 590047/2011-6 and 202104/2012-5.

References

[1] D. Recordon and D. Reed, "OpenID 2.0: a platform for user-centric identity management," in 2nd Workshop on Digital IDM, 2006, pp. 11–16.

[2] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "RFC 2865 - Remote Authentication Dial In User Service (RADIUS)," 2000.

[3] Verizon RISK Team, "Data breach investigations report," Verizon, Tech. Rep., 2013, goo.gl/7mIBy, [retrieved: March, 2014].

[4] C. Tankard, "Advanced persistent threats and how to monitor and deter them," Network Security, vol. 2011, no. 8, 2011.

[5] FreeRADIUS, "Documentation," 2012, goo.gl/6g8Qy, [retrieved: March, 2014].

[6] RADIUS Partnerships, "Deploying RADIUS: Practices and Principles for AAA solutions," 2008, goo.gl/fslu7, [retrieved: March, 2014].

[7] Juniper Networks, "Steel belted RADIUS carrier 7.0 administration and configuration guide," 2010, goo.gl/Y5b9k.

[8] OpenID, "OpenID community wiki," 2010, goo.gl/PCASy, [retrieved: March, 2014].

[9] Clamshell, "Clamshell: An OpenID Server," 2013, goo.gl/09pYF, [retrieved: March, 2014].

[10] S.-T. Sun, K. Hawkey, and K. Beznosov, "Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures," Computers & Security, pp. 465–483, 2012.

[11] B. van Delft and M. Oostdijk, "A security analysis of OpenID," in IFIP Advances in Information and Comm. Tech., vol. 343, 2010, pp. 73–84.

[12] M. Uruena, A. Munoz, and D. Larrabeiti, "Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites," Multimedia Tools and Applications, pp. 1–18, 2012.

[13] J. Feng, "Analysis, Implementation and Extensions of RADIUS Protocol," in Conference on Networking and Digital Society, 2009.

[14] A. Leicher, A. Schmidt, Y. Shah, and I. Cha, "Trusted computing enhanced OpenID," in ICITST, 2010, pp. 1–8.

[15] E. Alchieri, A. Bessani, and J. Fraga, "A dependable infrastructure for cooperative web services coordination," in ICWS, 2008, pp. 21–28.

[16] J. Sousa and A. Bessani, "From byzantine consensus to BFT state machine replication: A latency-optimal transformation," in EDCC, 2012, pp. 37–48.

[17] D. Kreutz, H. Niedermayer, E. Feitosa, J. da Silva Fraga, and O. Malichevskyy, "Architecture components for resilient networks," SecFuNet Consortium, Tech. Rep., 2013, http://goo.gl/xBHCNb, [retrieved: March, 2014].

[18] P. E. Veríssimo, "Travelling through wormholes: a new look at distributed systems models," SIGACT News, vol. 37, Mar. 2006.

[19] J. Lau, L. barreto, and J. da Silva Fraga, "An infrastructure based in virtualization for intrusion tolerant services," in ICWS, june 2012.

[20] A. Bessani, J. ao Sousa, and E. Alchieri, "State Machine Replication for the Masses with BFT-SMaRt," DI/FCUL, Tech. Rep., Dec. 2013, http://hdl.handle.net/10455/6897, [retrieved: March, 2014].

[21] M. Prince, "The DDoS That Almost Broke the Internet," 2013, goo.gl/g5Qs1, [retrieved: March, 2014].

[22] OpenID4Java, "OpenID 2.0 Java libraries," 2013, goo.gl/c3kFV, [retrieved: March, 2014].

[23] F. Botelho, F. M. V. Ramos, D. Kreutz, and A. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," in Second European Workshop on Software Defined Networking, 2013.

[24] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in SIGCOMM HotSDN, 2013, pp. 55–60.

[25] P. Urien, E. Marie, and C. Kiennert, "An innovative solution for cloud computing authentication: Grids of EAP-TLS smart cards," in Fifth International Conference on Digital Telecommunications (ICDT), 2010, pp. 22–27.

[26] D. Kreutz, A. Casimiro, and M. Pasin, "A trustworthy and resilient event broker for monitoring cloud infrastructures," in IFIP DAIS, 2012, pp. 87–95.

 ISBN: 978-1-61208-330-8