Limited use cryptographic tokens in securing cloud servers

Gautam Kumar, Brent Lagesse
Computing and Software Systems
University of Washington Bothell
{gautamk,lagesse}@uw.edu

***Abstract*—Many enterprises and consumers today are dependent on services which deploy their computational resources on Infrastructure as a Service (IaaS) cloud providers such as Amazon's AWS, Google's Cloud platform and Windows Azure. Cloud deployments at scale can have hundreds of virtual servers running and the same time sharing sensitive configuration information such as DB passwords and API Keys. This paper aims to minimize the threat of a potential information leak of such sensitive configuration data using a centralized architecture and the limited use nature of hash chains as an authentication mechanism to provide moving target defence against attackers.**

INTRODUCTION

The essence of securing cloud systems is using multiple layers [16] of security to increase an attacker's cost for taking over the system. One of the possible layer of security is using moving target defences [6].

In this paper we propose an implementation of moving target defence using ephemeral servers and a central trusted authority which acts on behalf an ephemeral server and proxies requests to sensitive resources such as database servers, caching servers and REST end points. Hash chains are used as an authentication mechanism by the central trusted authority. We take advantage of the limited use property of hash chains to secure authenticate ephemeral servers for a limited period of time.

BACKGROUND

Cryptographic hash function [19]

A cryptographic hash function is any one way function which meets the following requirements

- Preimage resistance
- Collision resistance
- Second Preimage resistance

A hash function has preimage resistance if given a hash value h it is computationally infeasible to find any message m such that $h = hash(k, m)$ where k is the hash key.

A hash function is collision resistant if, given two messages m_1 and m_2 it is hard to find a hash h such that $h = hash(k, m_1) = hash(k, m_2)$ where k is the hash key.

A hash function has second pre-image resistance if given a message m_1 it is computationally infeasible to find a different message m_2 such that $hash(k, m_1) = hash(k, m_2)$ where k is the hash key. The second pre-image resistance is a much harder property to achieve for hash functions. This property is closely related to the birthday problem [13].

Hash Chains [9]

Leslie Lamport [11] was first to propose the use of hash chains in his paper on a method for secure password authentication over an insecure medium.

"A hash chain is a sequence of values derived via consecutive applications of a cryptographic hash function to an initial input. Due to the properties of the hash function, it is relatively easy to calculate successive values in the chain but given a particular value,it is infeasible to determine the previous value"

As an example, Let x be the initial password a hash chain of length 2 would be $H^2(x) = H(H(x))$. So a hash chain of n values is denoted as $H^n(x)$ and the i^{th} value in the chain would be computed as $x_i = H(x_{i-1})$.

For a given value in the chain x_i its computationally infeasible to determine the previous value

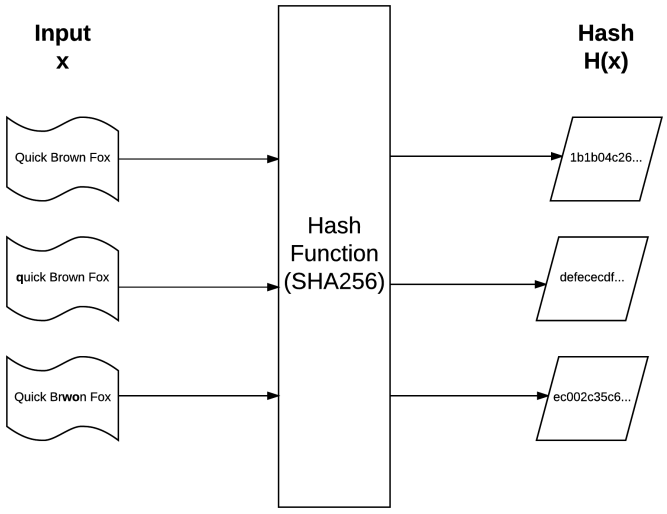

Fig. 1. A simplified view of a hash function which represents its input and potential result. The length of the hash sum always remains the same regardless of the input size. Any small change in the input drastically changes the output.

in the chain x_{i-1}.

Potential threats

According to OWASP's Top 10 security threats, "Sensitive data exposure" is the 6^{th} most critical type of security threat in web applications as of 2013 [23].

Sensitive data exposure simply refers to unintended exposure of sensitive information such as passwords, social security numbers, date of birth and so on. In the context of a cloud systems sensitive information may also include credentials to access a database, email server, REST API keys and so on. These credentials are usually stored as part of a configuration file which cloud servers can use to authenticate themselves with third party services within or outside the private cloud network.

According to a report by Risk Based Security [18] [20] the number of data leaks has dramatically increased from 2012 to 2013, to the tune of $812 million. Though sensitive data exposure in the context of cloud configurations would only constitute a small part of these leaks, leaking of such credentials can potentially lead to massive data leaks or other potential vulnerabilities being exposed to potential attackers.

Sensitive data exposure can potentially be a consequence of other threats such as cross site scripting (XSS) [14], Injection is the most critical threat while XSS is the 3^{rd} most critical threat as classified by OWASP in 2013 [23].

Proposed solution architecture

The proposed architecture to defend against the threat of sensitive data exposure is to use a Central Trusted Authority (CTA) responsible for storing sensitive information. The CTA would act as a proxy and would make requests on behalf of client facing servers, refer Fig. 2.

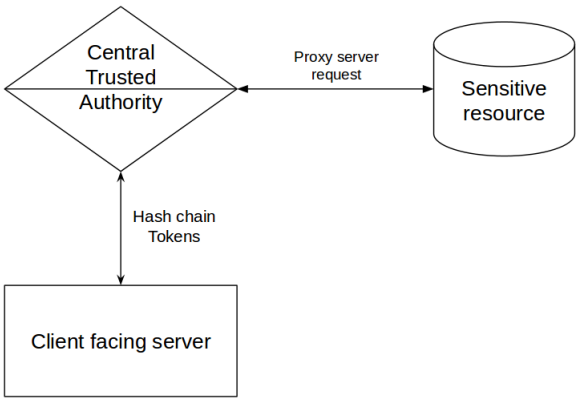

Fig. 2. Architectural overview of the system. This figure describes the three primary modules involved. The client facing server, The central trusted authority and a sensitive resource.

The client facing servers would use hash chains to authenticate with the CTA. Hash chains cryptographically limit the number of times a key can be used. Limited use was intentionally selected to promote the creation of a moving target for attackers.

Assumptions

Client facing servers are the servers which are exposed outside the private cloud network environment. These client facing servers could potentially be load balancing servers, compute servers.

The client facing servers are assumed to be ephemeral. This is common in many cloud deployments [21] and contributes to the moving target nature of the security architecture. Companies such as netflix expect this behaviour with their chaos engineering architecture [1]. This allows for higher reliability of their server infrastructure.

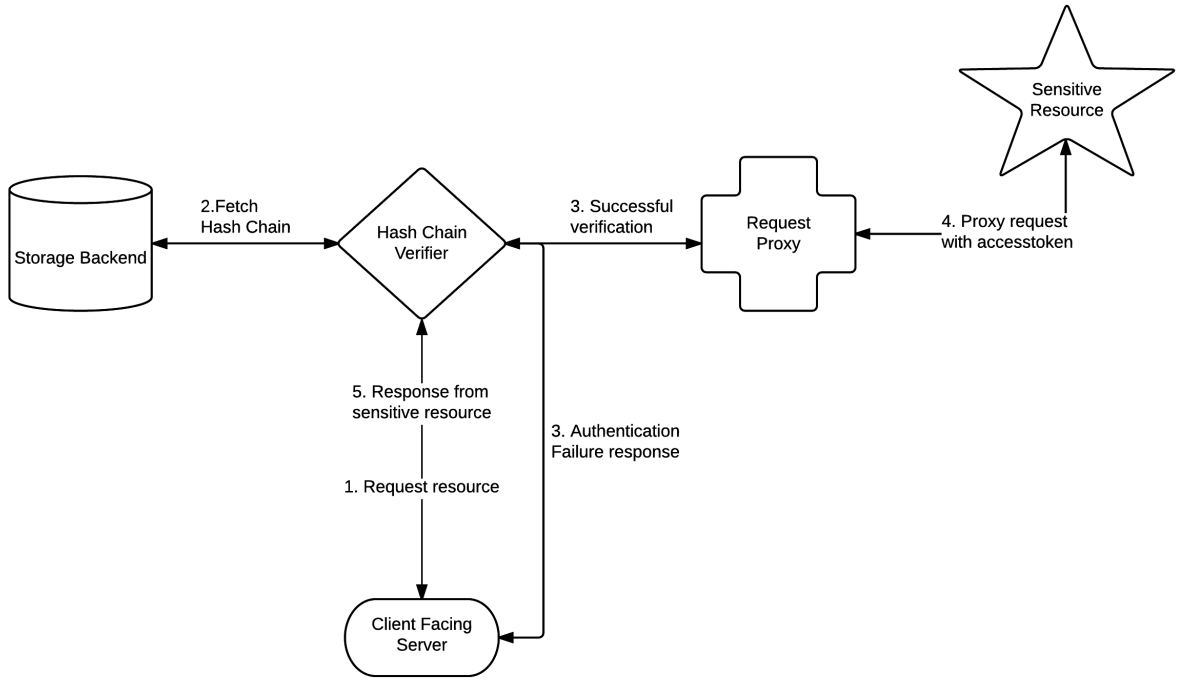

Fig. 3. Architecture of the Central Trusted Authority. The CTA consists of a storage backend, a hash chain verifier and a request proxy.

Client facing servers are expected to shut down after their hash chain expires. This contributes to the ephemeral nature and also to moving target defence of the overall system.

CTA Architecture and Implementation

The Central Trusted Authority consists of three primary components, A hash chain verifier, A storage backend and a request proxy. The CTA generally performs three roles within the system which are

- Create new hash chain
- Verify hash chains
- Proxy requests

Creating new hash chain: Hash chains are created by iteratively hashing a secret key K, n number of times. After the hash chain is created the CTA stores $H^{100}(K)$ in the storage backend and returns K and n to the client facing server. The secret key K is not stored by the CTA.

The client facing server can now use the the secret key n number of times.

Hash chain verification: Hash chains are used to authenticate client facing server requests which require access to a sensitive resource. The hash chain verification process consists of four steps, Receive hash chain secret, compute the hash sum of the chain secret and verify with the storage backend.

The storage backend stores the last verified key H^i of the hash chain where i is the last verified index of the hash chain. When initializing the hash chain $i = n$.

- Receive hash chain secret $H^{i-1}(K)$ from the client.
- Compute H^i by $H^i = H(H^{i-1}(K))$
- If the computed H^i equals the value in the storage backend
  - Replace H^i with $H^{i-1}(K)$.
- Else reject the key and refuse connection.

REQUEST PROXY

The proposed solution architecture of using a Central Trusted Authority (CTA) to proxy requests on behalf of all the clients, places the CTA as a single point of failure. We can overcome this limitation by improving the reliability and trustability of the CTA.

Reliability

A lot of research [5] [22] [24] [10] has been conducted into improving the reliability of cloud systems through vertical, horizontal scaling and automatic provisioning. Much of that work can be leveraged for the purpose of improving the reliability of the proposed architecture. For example, Beltrán [2] proposes an architecture of utilizing multi-tired load balancers to improve the reliability of a service.

Large scale database deployments typically deploy databases behind a reverse proxy for load balancing and Geo distribution [8]. Thus a reverse proxy such as Vitess [8] is ideally suited to act as the CTA in such architectures.

Security and Trust

Trust and security of the CTA is another primary area of concern with a distributed cloud system. The CTA as a central point of access to all sensitive resources would be a prime target for malicious actors and thus cannot be completely trusted. Chen et al [4] propose a solution to this problem of a Malicious proxy using trusted hardware such as Trusted Platform Module (TPM) or the IBM 4758 cryptographic coprocessor [17].

Chen et al assume the proxy, the CTA in our architecture, is malicious but is incapable of modifying the underlying hardware. The CTA executable is also verified by a trusted third party to operate correctly as a proxy as described in [17].

The proposed architecture allows for proxying both HTTP and TCP requests.

HTTP protocol authentication: is currently implemented using the *Authorization* header [7]. The client facing server is expected to send the hash chain key as part of the *Authorization* header to the CTA while making a request.

TCP proxy: authentication can be implemented via HTTP tunnelling, TLS client authentication or IPSec authentication.

Performance testing

In our testing hash chain initialization showed a linear increase in time complexity as shown in figure 5. This is in line with our expectations for hash chain implementations. A hash chain of length 100,000 takes 150 - 200 ms to initialize based on the storage backend used.

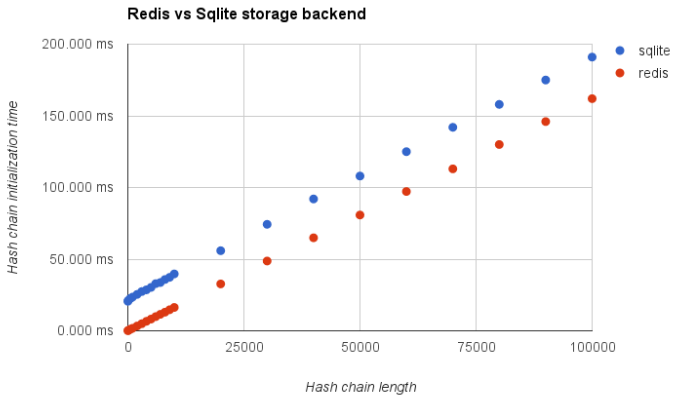

Fig. 5. Linear time complexity of the hash chain initialization. Execution platform Intel i5-5200U 2.2Ghz, 12GB RAM, SHA-512 implemented in python.

Limitations

The proposed architecture and implementation are ideally suited to cloud IaaS providers such as Amazon AWS where ephemeral servers are easily managed. Deploying the proposed system to a cloud provider without the ability to quickly provide large number of ephemeral servers could result is significant performance degradation.

Hash chains inherently posses certain vulnerabilities such as a Hash chain cycle and Hash chain length oracle attacks which can potentially reduce the effectiveness of the proposed architecture [12].

Potential future work

Formulating an ideal balance between server lifespan and hash chain length to optimize computational resources in a cloud system, this can be derived from current work into load balancing in cloud systems [**?**]. Such a formulation can be used by dev ops engineers in choosing an ideal hash chain size based on the performance requirements of a cloud system.

Integrating Timed Release Cryptography [3] as a hash chain renewal mechanism to potentially increase the lifespan of a server after a certain cool off period.

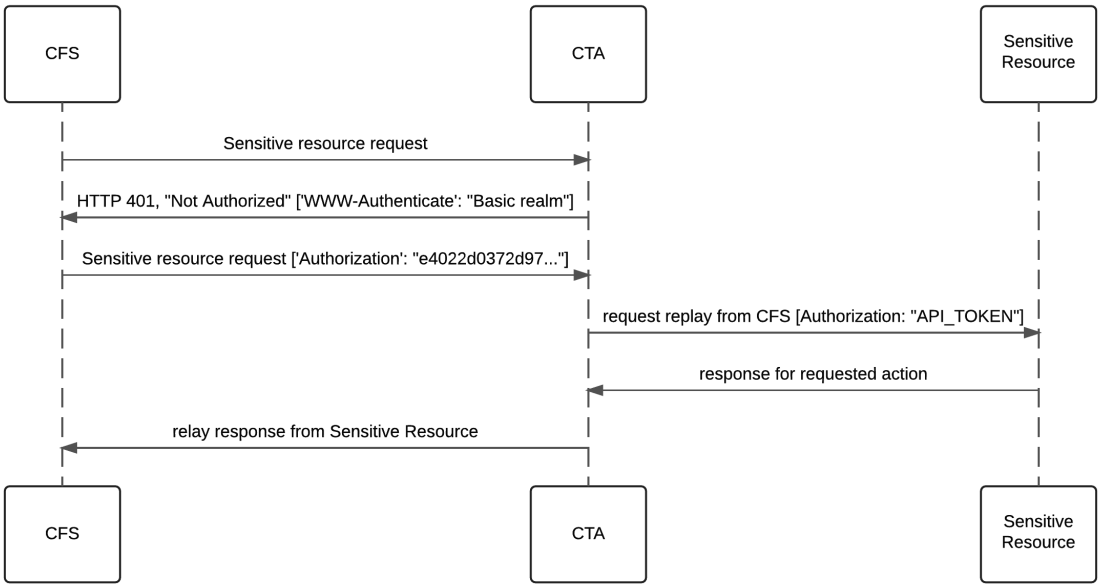

Fig. 4. Sequence diagram describing the authentication and proxying capabilities of the CTA. CFS refers to Client Facing Server while CTA refers to Central Trusted Authority

A Merkle hash tree implementation middle-ware can potentially provide hierarchical authentication authorization [25] capabilities to the CTA.

Related work

Confidant [15] is a library maintained by Lyft, a transportation network company based out of San Francisco. Confidant provides an implementation of the Central trusted authority server with encryption at rest, authentication and authorization handled by AWS's Key Management Service, KMS and a storage backend of DynamoDB. This severely hampers the ability of a potential user to deploy a Confidant instance on a different cloud IaaS provider beside Amazon's AWS.

Confidant also serves as an inspiration for the CTA component of the proposed architecture.

Summary

In this paper we propose an architecture to provide moving target defence and minimize the damage that attackers can cause by centralizing the storage of sensitive configuration information and taking advantage of the limited use property of hash chains to authenticate potentially vulnerable client facing servers.

References

[1] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal. Chaos engineering. 33(3):35–41.

[2] Marta Beltrn. Automatic provisioning of multi-tier applications in cloud computing environments. 71(6):2221–2250.

[3] Konstantinos Chalkias and George Stephanides. Timed release cryptography from bilinear pairings using hash chains. In *Communications and Multimedia Security*, pages 130–140. Springer.

[4] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johanes Gehrke. Towards statistical queries over distributed private user data. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 169–182.

[5] S. Dutta, S. Gera, A. Verma, and B. Viswanathan. SmartScale: Automatic application scaling in enterprise clouds. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 221–228.

[6] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and X. Sean Wang, editors, *Moving Target Defense*, number 54 in Advances in Information Security, pages 29–48. Springer New York. DOI: 10.1007/978-1-4614-0977-9_2.

[7] John Franks, Phillip Hallam-Baker, Jeffrey Hostetler, Scott Lawrence, Paul Leach, Ari Luotonen, and Lawrence Stewart. *HTTP authentication: Basic and digest access authentication*. RFC 2617, June.

[8] Todd Hoff. 7 years of YouTube scalability lessons in 30 minutes. http://highscalability.com/blog/2012/3/26/7-years-of-youtube-scalability-lessons-in-30-minutes.html.

[9] Dwight Horne. Hash chain. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and*

Security, pages 542–543. Springer US. DOI: 10.1007/978-1-4419-5906-5_780.

[10] Senthil SK Kumar and P. Balasubramanie. Dynamic scheduling for cloud reliability using transportation problem. 8(10):1615.

[11] Leslie Lamport. Password authentication with insecure communication. 24(11):770–772.

[12] D. Lee. Hash function vulnerability index and hash chain attacks. In *2007 3rd IEEE Workshop on Secure Network Protocols*, pages 1–6.

[13] Lawrence M. Lesser. Exploring the birthday problem with spreadsheets. 92(5):407–411.

[14] M. T. Louw and V. N. Venkatakrishnan. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *2009 30th IEEE Symposium on Security and Privacy*, pages 331–346.

[15] lyft. Confidant: Your secret keeper. A library to store and retrive senstive configuration within a central trusted authority encrypted at rest using Amazon KMS. https://github.com/lyft/confidant.

[16] A. Panwar, R. Patidar, and V. Koshta. Layered security approach in cloud. In *3rd International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2011)*, pages 214–218.

[17] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. Bootstrapping trust in commodity computers. In *2010 IEEE Symposium on Security and Privacy*, pages 414–429. IEEE.

[18] Risk Based and Security. An executives guide to 2013 data breach trends.

[19] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, number 3017 in Lecture Notes in Computer Science, pages 371–388. Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-25937-4_24.

[20] Xiaokui Shu, Danfeng Yao, and Elisa Bertino. Privacy-preserving detection of sensitive data exposure. 10(5):1092–1103.

[21] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. 41(1):45–52.

[22] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM.

[23] Dave Wichers. OWASP top-10 2013.

[24] Zhang Xuejie, Wang Zhijian, and Xu Feng. Reliability evaluation of cloud computing systems using hybrid methods. 19(2):165–174.

[25] X. Yi and W. Wang. The cloud access control based on dynamic feedback and merkle hash tree. In *2012 Fifth International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 217–221.