Limited Use Cryptographic Tokens in Securing Ephemeral Cloud Servers

Gautam Kumar
University of Washington Bothell
Computing and Software Systems
gautamk@uw.edu

Brent Lagesse
University of Washington Bothell
Computing and Software Systems
lagesse@uw.edu

ABSTRACT

Many enterprises and consumers today are dependent on services which are deployed on Infrastructure as a Service (IaaS) cloud providers. Such cloud deployments can have hundreds of virtual servers running. These virtual servers could share sensitive configuration information such as DB passwords and API Keys. Validating the security of hundreds of servers is an arduous task which does not scale as many vulnerabilities are discovered everyday. The goal of this paper is to propose an architecture which limits the extent to which an attacker can exploit a compromised server in a large scale cloud deployment using Hash Chains as an authentication mechanism with a Central Trusted Authority (CTA) as a proxy to sensitive resource. This architecture slides the requirement of security validation from hundreds of servers to a few servers which comprise the CTA. The use of hash chains can be used to limit the amount of time that each virtual server is alive thus providing moving target defence.

CCS Concepts

•Security and privacy → Domain-specific security and privacy architectures; *Security protocols;* **•Software and its engineering → Software architectures;**

1. INTRODUCTION

Cloud infrastructure today is characterised by three difference classes of products. Infrastructure as a Service or IaaS, these are providers such as Amazon's AWS, Google's Cloud platform, Microsoft's Azure who offer on-demand compute, storage and network resource at the click of a button. Platform as a Service or PaaS, these are providers such as Heroku and Google's AppEngine who host a developer's code and take on the responsibility of platform maintenance, i.e maintaining the compute, storage and network resource and scaling them as needed. Software as a Service or SaaS these refer to any web based application which runs on cloud infrastructure, examples include Google Apps and Microsoft's Office 365.

According to a white paper published by Microsoft [16], more than 85% work done by enterprise IT departments is towards infrastructure maintenance. IaaS providers offer enterprises a unique advantage by minimizing the effort needed to purchase and maintain physical hardware. So it stands to reason that enterprises now consider hosting their infrastructure with IaaS providers as a necessity rather than merely a competitive advantage[26].

Hosting infrastructure on the cloud comes with its own set of unique challenges and one of the primary concerns is Security. IaaS providers offer computational and storage resource on virtualized shared hardware to maximize utilization. So the essence of securing cloud systems is using multiple layers [29] of security to increase an attacker's cost for taking over the system. One of the possible layer of security is using moving target defenses [10].

One of our focuses is mitigating the value of zero-day attacks. According to [3] the average zero-day attack lasts about 10 months. Once zero-day attacks are disclosed, malware authors begin to utilize them multiple orders of magnitude more frequently; however, there is often a significant lag between notification and the deployment of patches fixing the zero-day exploit. One of the goals of this project is to reduce the risk of systems when we are not aware of the existence of the vulnerability, but also during the gap of time that the community is made aware, but patches have not been fully deployed.

Consider the situation where a large service provider has a number of public facing servers with access to a private, remote database. In the case that one of those public facing servers is compromised by an adversary, the data in the private database will be vulnerable for the entire duration that the attacker has access until the attack is detected and the vulnerable server patched or removed. The goal of our work is to drastically reduce the quantity and length of time that the attacker has to sensitive information *even when we do not know that the attack has occurred.*

In this paper we propose an implementation of moving target defense 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.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. BACKGROUND

2.1 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 [36].

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 [32] [34] 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) [24], Injection is the most critical threat while XSS is the 3^{rd} most critical threat as classified by OWASP in 2013 [36].

2.2 Cryptographic hash function

A cryptographic hash function is any one way function which meets the following requirements [33]

- 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 [20].

2.3 Hash Chains

Leslie Lamport [18] first proposed 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"

A hash chain in essence is merely the successive computation of a Cryptographic hash function on a given value.

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

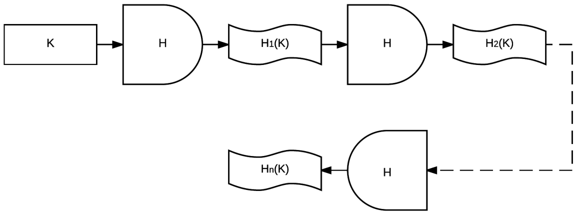

Figure 1: A simplified view of a hash chain where the start message is K with H being the cryptographic hash function. $H_1(K)$, $H_2(K)$ and $H_n(K)$ represent successive messages in the Hash chain

For a given value in the chain x_i its computationally infeasible to determine the previous value in the chain x_{i-1}.

3. PROPOSED 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.

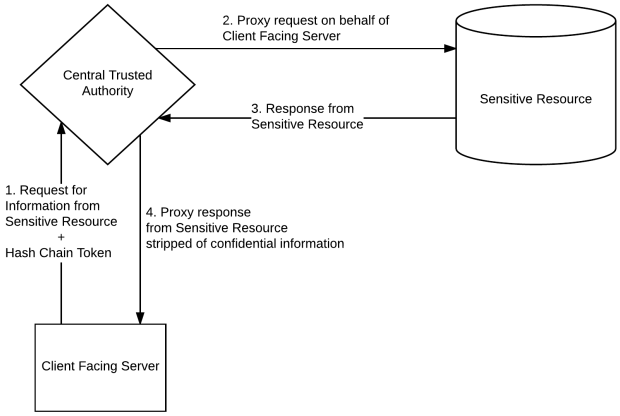

Figure 2: Architectural overview of the system. This figure describes the three primary modules involved. The client facing server, The central trusted authority and a sensitive resource.

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.

3.1 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.

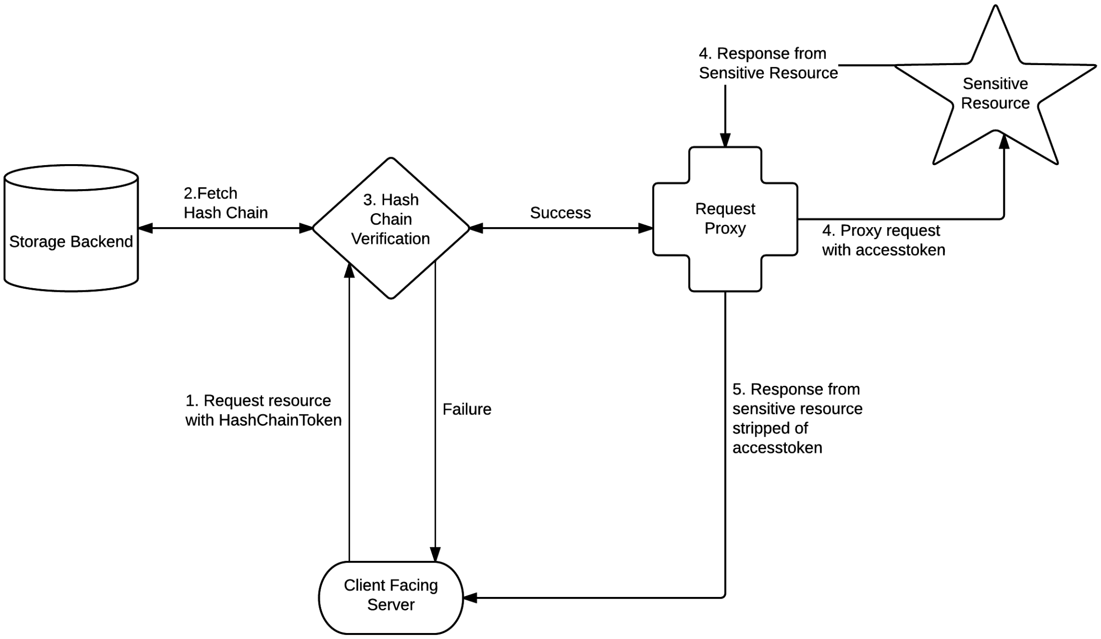

Figure 3: Architecture of the Central Trusted Authority. The CTA consists of a storage backend, a hash chain verifier and a request proxy.

The client facing servers are assumed to be ephemeral. This is common in many cloud deployments [35] and contributes to the moving target nature of the security architecture. Companies such as Netflix expect this behaviour with their chaos engineering architecture [2]. This allows for higher reliability of their server infrastructure.

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

3.2 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

3.2.1 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 $H^n(K)$ 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. This process is detailed in algorithm 1.

3.2.2 Hash chain verification

Hash chains are used to authenticate client facing server requests which require access to a sensitive resource. The hash chain verification is detailed in algorithm 2.

```
Data: Hash Chain secret $K$ and Hash chain length $N$
Result: Hash Chain $H^N(K)$
$i \leftarrow 1$;
$H^1(K) \leftarrow H(K)$ ;
while $i <= N$ do
    $i \leftarrow i + 1$;
    $H^i(K) \leftarrow H(H^{i-1}(K))$;
end
return $H^i(K)$ $\textit{where } i \textit{ equals } N$ ;
```

Algorithm 1: Generating a Hash Chain

```
Data: Authentication key $H^{i-1}(K)$ from the client
Result: Response from sensitive resource
Let $H^i_{client} = H(H^{i-1}(K))$ ;
Let $AuthenticationData = fetch(H^i_{client})$ ;
if $AuthenticationData$ $\textit{exists in storage backend}$ then
    replace $H^i_{client}$ with $H^{i-1}(K)$ in storage backend ;
    fetch and return response from sensitive resource ;
else
    return authentication failure ;
end
```

Algorithm 2: Verification of Hash Chain authentication

4. SECURITY THREATS

The proposed 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 deem this an acceptable trade-off as the CTA is not a public facing system and only serves the purpose of verifying hash chains and proxying requests. As a result, hardening its defenses against possible threats becomes an easier problem. In our current implementation, the CTA is a single server; however, we discuss alternative designs that we are considering in section 6.

Our architecture proposal does not prescribe any particular hardware / VM co-location for any components in the CTA. Thus each component can be implemented on separate machines or the CTA can be instantiated as a whole within a single VM. In this section we detail the potential security threats against each component and possible mitigation strategies.

4.1 Malicious Request Proxy

The request proxy component acts on behalf of the client facing server to make a call to a sensitive resource such as an API endpoint. The request proxy is also responsible for attaching authentication information such as an API Key when contacting the API endpoint. Upon receiving a response from the sensitive resource the request proxy strips out sensitive information such as API Keys and Refresh Tokens before forwarding the response to the client facing server.Under this threat scenario the request proxy is assumed to be untrustworthy and potentially malicious.

Chen et al [6] 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 [30].

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 [30].

There are three primary attacks that a malicious actor may perform on the CTA. First, the attacker may try to expose the sensitive information stored, such as API keys and DB passwords, from the CTA using vulnerabilities in the CTA executable. Second, the attacker could potentially modify the requests / responses which are being proxied by the CTA. Third the attacker could potentially launch a reboot attack to inject a malicious executable after attestation to carry out one of the above attacks.

Prior work [22, 27] on preventing reboot-attacks can be leveraged to impede the attacker's ability to inject a malicious executable. The proposed architecture also allows for the CTA to be placed behind secure corporate firewalls to further limit the risk of a malicious take over by an attacker. Further work is needed to fully secure the CTA against a malicious proxy.

4.2 Insecure Storage back-end

Our proposed architecture can accommodate various kinds of storage backends and thus potential security threats against the storage backend may vary.

Considering the scenario of an RDBMS system, The most common vulnerability is SQL Injection and a lot prior of work has been done to secure RDBMS systems against SQL Injection [14, 4, 15] and those techniques to drastically limit the risk of data leakage or system takeover.

Using a pre-hardened database as a service such as Amazon's RDS or Google's CloudSQL can further improve the security of our storage backend [8] .

5. PERFORMANCE TESTING

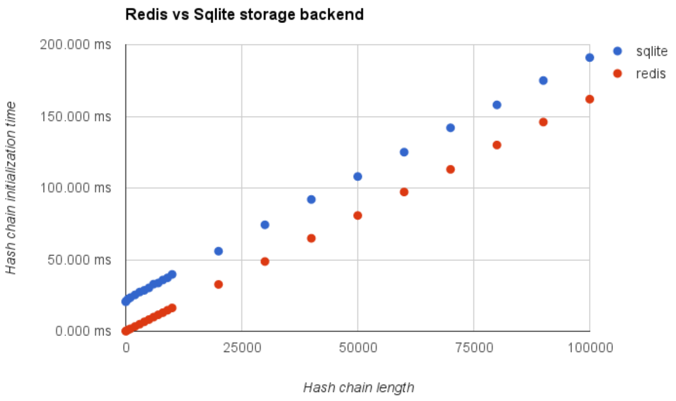

Figure 5: Linear time complexity of the hash chain initialization. Execution platform Intel i5-5200U 2.2Ghz, 12GB RAM, SHA-512 implemented in python.

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.

Using connections on Amazon EC2 US West (Oregon), we connected between machines in the same zone and connected from residential ISPs to EC2 to provide evidence that the increased workload in our proposed protocol will not drastically affect performance. Connections within the same EC2 zone took about 0.3 ms for a round trip submission whereas the round trip time on a typical residential cable modem connection took about 25 ms to the same machine. As a result, for small requests, we expect that the network latency will be increased by a few percent and even less, relatively, for large requests, as bandwidth within the same cloud zone can be orders of magnitude greater than most ISPs provide to both residential and commercial customers. The only additional significant computation that occurs during runtime is the hash chain verification. Hash chain verification is designed to be an efficient operation and the original verification process has been improved upon several times [11, 39]

Creating new virtual machines to act as client facing servers is an expensive process; however, by creating a pool of unused client facing servers (CFS) to act as quick replacements, the client will not notice the moving target defense happening behind the scene. The size of the CFS pool is balanced based on the time to initialize a VM, the size of the hash chain, and the rate at which requests are made. Part of our future work is to design an adaptive algorithm that manages the creating of new VMs based on learned traffic patterns.

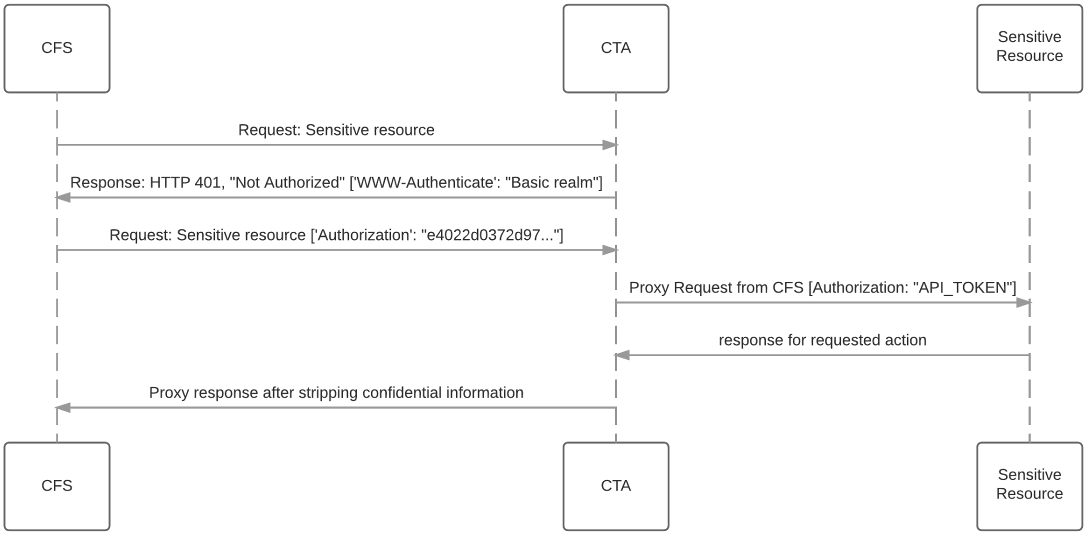
CFS
CTA
Sensitive Resource
Request: Sensitive resource
Response: HTTP 401, "Not Authorized" ['WWW-Authenticate': "Basic realm"]
Request: Sensitive resource ['Authorization': "e4022d0372d97..."]
Proxy Request from CFS [Authorization: "API_TOKEN"]
response for requested action
Proxy response after stripping confidential information
CFS
CTA
Sensitive Resource

Figure 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

6. DISCUSSION AND FUTURE WORK

6.1 Providing moving target defense

Green et al [13] identify Unpredictability, Vastness and Revocability as some of the criteria for evaluating moving target defenses. In this section we try define how our implementation conforms to these criteria.

Unpredictability. Cryptographic hash functions [33] are designed to be collision resistant and comply with the avalanche effect and thus the generated sequence is highly unpredictable in nature. Hash chains can compound this effect by sequential application of the cryptographic hash function.

Vastness. A secure cryptographic hash function such as SHA-256 has 128 bit of security in its worst case scenario which translates to potentially 2^{128} values which can be safely computed without resistance thus offering a vast state space.

Revocability. The proposed architecture allows individual chains or a group of chains to be revoked merely by purging them from the database. Further group / hierarchical revocability can be achieved using Merkle hash tree implementations as a means of Hash chain generation.

6.2 Limitations

The proposed architecture and implementation is designed for cloud IaaS providers such as Amazon AWS where ephemeral servers are easily managed. As a result, the proposed system is not extensible 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 [19].

6.3 Artificial diversity

One concern of our system is that if an attacker is able to compromise one server, then once that server is destroyed, they will move on to another CFS. For the case of some vulnerabilities, this will be true. In those cases, we have at least raised the cost to the attacker as they will have to exploit the vulnerability over and over again. This raises the chance that they will be detected, and if the exploit is costly, then it puts a greater strain on their resources. In some applications, we will be able to implement automated diversity into the CFSs that will mitigate attacks even further. Diversity in computer systems [12, 23, 7] has been studied to demonstrate its effectiveness at raising costs to attackers and we expect further research in the area to continue to improve the effectiveness of the defense we propose in this paper.

The proposed architecture lends itself well to establishing artificial diversity using a Distributed Hash Table to store the hash chains as suggested by Morell et al [28], thus further contributing to the un-predictability of the system and further limiting the risk introduced by a malicious node.

6.4 CTA Alternative

In this paper we have presented our solution with a centralized CTA and argued that in some cases this approach is sufficient. In the case that this solution does not work for an application, we are also exploring a DHT-based proxy though we have not yet implemented it. In the DHT-based solution, there would be a large number of proxies in a DHT. Each proxy in the DHT would be responsible for servicing a subset of requests from CFS. In this solution, the DHT would be keyed on the hash chain hash value, so the requests would be routed to a different proxy every time. By using a DHT-based solution, an adversary would only be able to compromise a subset of the requests, further limiting its ability to intercept targeted information or read widespread sensitive requests.

6.5 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 [31]. 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 [5] as a hash chain renewal mechanism to potentially increase the lifespan of a server after a certain cool off period. Implementing Time Release Cryptography would enable our architecture to accommodate systems where a constant pool of ephemeral servers are not available and existing servers can be populated with a batch of hash chains which can only be accessed after a certain period of time.

A Merkle hash tree implementation middle-ware can potentially provide hierarchical authentication authorization [37] capabilities to the CTA. In the scenario of an information leak, merkle trees would allow administrators to black list either a single or a hierarchy of hash chains to safeguard the system.

7. RELATED WORK

There have been other systems proposed which offer moving target defence using a temporary address or a temporary authentication mechanism. Many technology companies have real world implementations of the Central Trusted Authority architecture similar to our proposal. In this section we present the most relevant proposals and discuss how our proposed architecture differs.

Dunlop et al [9] leverage the vast address space (2^{128}) address space of IPv6 to move the source and destination IP addresses mid-session based on a pre-agreed pattern to limit an attacker's ability to intercept or interfere with a TCP session. The technique proposed by Kampanakis [17], however, operates at the network level by using an SDN's ability to vary the address space and the route taken by packets to increase the cost for an attacker and thus providing moving target defense.

Active authentication proposed by [38, 1, 21] offer a way to verify a user's identity based on their behaviour thus eliminating the need for hard to remember passwords. Active authentication is a form of moving target defense where the authenticating factor is constantly changing in a hard to predict manner, thus significantly increasing the cost for an attacker to reproduce the authenticating factor.

Confidant [25] 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. The limitations of this implementation is the inability to deploy a Confidant instance on a different cloud IaaS provider besides Amazon's AWS.

Our proposed architecture leverages dynamic addressing and varying authentication in the form of ephemeral servers and hash chains to increase the cost for an attacker.

8. CONCLUSION

In this paper we propose an architecture to provide moving target defense and minimize the damage that attackers can cause if they compromise a client facing server 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.

We believe this approach can be used in cloud systems to limit the risk of a compromised client facing server. Security-related software bugs are constantly being discovered and exploited, so while we may not be able to deploy a system that will never be compromised, we can deploy defenses that limit the effectiveness of an attacker, even when they are utilizing a zero-day attack.

9. REFERENCES

[1] Y. Aksari and H. Artuner. Active authentication by mouse movements. In *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*, pages 571–574. IEEE.

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

[3] L. Bilge and T. Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM.

[4] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL injection attacks. In *International Conference on Applied Cryptography and Network Security*, pages 292–302. Springer.

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

[6] R. Chen, A. Reznichenko, P. Francis, and J. 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.

[7] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems: A secretless framework for security through diversity. In *Usenix Security*, volume 6, pages 105–120.

[8] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: A database-as-a-service for the cloud.

[9] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. MT6d: A moving target IPv6 defense. In *2011 - MILCOM 2011 Military Communications Conference*, pages 1321–1326.

[10] D. Evans, A. Nguyen-Tuong, and J. Knight. Effectiveness of moving target defenses. In S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. 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.

[11] M. Fischlin. Fast verification of hash chains. In *CryptographersâĂŹ Track at the RSA Conference*, pages 339–352. Springer.

[12] S. Forrest, A. Somayaji, and D. H. Ackley. Building diverse computer systems. In *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*, pages 67–72. IEEE.

[13] M. Green, D. C. MacFarland, D. R. Smestad, and C. A. Shue. Characterizing network-based moving target defenses. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 31–35. ACM.

[14] W. G. Halfond and A. Orso. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 174–183. ACM.

[15] W. G. Halfond, J. Viegas, and A. Orso. A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE.

[16] R. Harms and M. Yamartino. The economics of the cloud.

[17] P. Kampanakis, H. Perros, and T. Beyene. SDN-based solutions for moving target defense network protection. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6.

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

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

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

[21] F. Li, N. Clarke, M. Papadaki, and P. Dowland. Active authentication for mobile devices utilising behaviour profiling. 13(3):229–244.

[22] B. Libert and D. Vergnaud. Tracing malicious proxies in proxy re-encryption. In S. D. Galbraith and K. G. Paterson, editors, *Pairing-Based Cryptography âĂŞ Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 332–353. Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-85538-5_22.

[23] B. Littlewood and L. Strigini. Redundancy and diversity in security. In *European Symposium on Research in Computer Security*, pages 423–438. Springer.

[24] 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.

[25] 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.

[26] A. McAfee. What every CEO needs to know about the cloud. 89(11):124–132.

[27] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 315–328. ACM.

[28] C. Morrell, R. Moore, R. Marchany, and J. G. Tront. DHT blind rendezvous for session establishment in network layer moving target defenses. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 77–84. ACM.

[29] 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.

[30] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping trust in commodity computers. In *2010 IEEE Symposium on Security and Privacy*, pages 414–429. IEEE.

[31] M. Randles, D. Lamb, and A. Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 551–556.

[32] Risk Based and Security. An executiveâĂŹs guide to 2013 data breach trends.

[33] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. Roy and W. 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.

[34] X. Shu, D. Yao, and E. Bertino. Privacy-preserving detection of sensitive data exposure. 10(5):1092–1103.

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

[36] D. Wichers. OWASP top-10 2013.

[37] 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.

[38] M. L. Yiu, E. Lo, and D. Yung. Authentication of moving kNN queries. In *2011 IEEE 27th International Conference on Data Engineering*, pages 565–576. IEEE.

[39] D. H. Yum, J. S. Kim, P. J. Lee, and S. J. Hong. On fast verification of hash chains. In *CryptographersâĂŹ Track at the RSA Conference*, pages 382–396. Springer.