Limited Use Cryptographic Tokens in Securing Ephemeral Cloud Servers

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

Keywords:
Security, Moving Target Defence, Cryptography, Cloud Architecture

Abstract:
Many enterprises and consumers today are dependent on services deployed on Infrastructure as a Service (IaaS) cloud providers. Such cloud deployments can have hundreds of virtual servers running. Each virtual server needs to have access to sensitive information such as database passwords and API keys. In such as scenario, verifying that a large number of servers have not been compromised is an arduous task. In this paper we propose an architecture which limits the extent to which an attacker can exploit a compromised server in a large scale cloud deployment. To achieve such a limitation we propose the use of hash chains as an authentication mechanism for virtual server with a Central Trusted Authority (CTA) acting as a proxy to sensitive resources. This architecture shifts the requirement of security validation from hundreds of public facing servers to a few servers without public interfaces which comprise the CTA. Since hash chains offer an inherent limitation in their use, our architecture leans towards using ephemeral virtual servers, thus also providing a moving target defence.

1 Introduction

According to Microsoft [16], more than 85% of work done by enterprise IT departments is towards infrastructure maintenance. Cloud providers offer enterprises a unique advantage by minimizing the effort needed to purchase and maintain physical hardware. As a result, enterprises now consider hosting their infrastructure with cloud providers as a necessity rather than merely a competitive advantage [27].

Hosting infrastructure on the cloud comes with its own set of unique challenges and one of the primary concerns is security. Many cloud providers offer computational and storage resources on virtualized shared hardware to maximize utilization. The essence of securing cloud systems is using multiple layers [30] of security to increase an attacker's cost for taking over the system. One of the emerging layers of security is the use of moving target defences [10].

One of our focuses is to increase the cost to the attacker of launching a zero-day attack. According to Bilge & Dumitras [2], on average, zero day attacks remain undetected for 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 delay 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 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 defence using temporary (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 authenticate ephemeral servers for a limited period of time.

2 Background

2.1 Cloud infrastructure

Cloud infrastructure today is characterised by three different classes of products. Infrastructure as a Service or IaaS, Platform as a Service or PaaS and Software as a Service or SaaS.

IaaS providers such as Amazon's AWS, Google's Cloud platform and Microsoft's Azure offer on-demand compute, storage and network resources which allows organisations to build out their cloud infrastructure to their needs. IaaS providers take on the responsibility of maintaining and securing physical hardware while organisations using IaaS are expected to maintain and secure their virtual hardware and also their application code.

PaaS providers such as Heroku and Google's AppEngine, offer services on top of an IaaS provider. PaaS providers assume the responsibility of maintaining and securing both physical and virtual hardware while the organisations assume the responsibility of merely securing application code.

SaaS providers offer complete applications hosted towards their customer's need and take on all the responsibility of maintaining and securing hardware and application code.

Our research in this paper is primarily focused on providing a mechanism to improve the security for organisations who use IaaS providers.

2.2 Threats

According to OWASP, "Sensitive data exposure" is the one of the most critical types of security threat in web applications as of 2013 [37]. Sensitive data exposure refers to the unintended exposure of sensitive information such as passwords, social security numbers and date of birth. In the context of a cloud systems sensitive information may also include credentials to access a database, email server and REST API keys. These credentials are used to authenticate cloud servers with third party services within or outside the private cloud network. Credentials are usually stored as part of configuration files or in a server's memory. Such credentials have a very long life time and are sometimes valid of years. This increases the risk of data exposure when an attacker has a lengthy period of time within which they can discover and exploit such credentials.

A report by Risk Based Security [33] [36] notes that 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 credentials as mentioned earlier, could potentially grant access to malicious entities within protected environments such as a corporate network. Sensitive data exposure can also be a consequence of other threats such as cross site scripting (XSS) [25] and Code Injection both of which are considered highly critical threats based on OWASP's classification [37].

2.3 Hash Chains

Leslie Lamport [21] first proposed the use of hash chains in his paper on a method for secure password authentication over an insecure medium. One of the properties of hash chain-based authentication is that it has a limited use lifespan. We leverage this property in our system to limit the damage that any single compromised machine can cause.

"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 demonstrated in figure **??**.

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 $x_i = H(x_{i-1})$.

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

Our proposed architecture consists of three primary components. The Client Facing Server (CFS), the Central Trusted Authority (CTA) and the Sensitive Resources that we're trying protect as shown in figure 1.

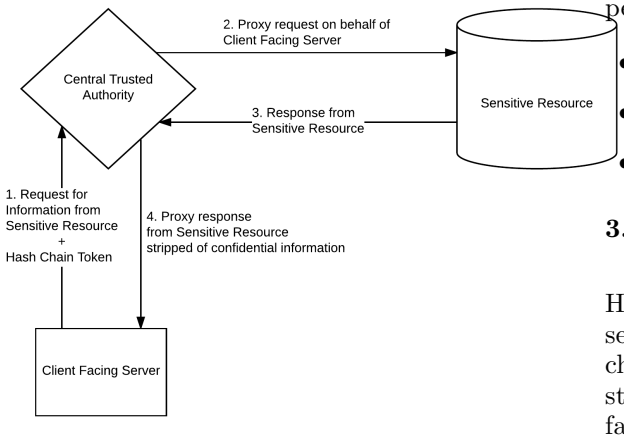

Figure 1: Architectural overview of the system. This figure describes the three primary modules involved. The client facing server, The central trusted authority and a sensitive resource.

Client Facing Servers are any servers which interact with entities on a public network such as Load balancers, reverse proxies and application servers. CFSs are assumed to be the most vulnerable to exploits by malicious entities. To offer moving target defence and increase the cost to an attacker, CFSs have a limited lifetime based on the length of the hash chain. When a hash chain expires the CFS is expected to shutdown because it can no longer authenticate itself with the CTA.

The *Central Trusted Authority* is an entity within the corporate or private network which acts on behalf of a CFS when a sensitive resource is requested. To act on behalf of a CFS the CTA proxies requests to resources such databases and external APIs and includes authenticating information required by the resources. The authenticating information such as DB Passwords and API Keys are not available to CFS. Because the CTA is not publicly available and there are orders of magnitude less CTAs in a system, it is assumed that the CTA is trustworthy as it will be easier to harden the system and monitor the limited types of activity on it. The CTA verifies the identity of a CFS using a hash chain as explained in section 3.1 and figure 7 elaborates on the architecture of the CTA.

3.1 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 performs three roles within the system, which are

- Create new hash chains
- Verify hash chains
- Proxy requests to resources

3.1.1 Creating a 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.

Data: Hash Chain secret K and Hash chain length N
Result: Hash Chain $H^N(K)$
1 $i \leftarrow 1$;
2 $H^1(K) \leftarrow H(K)$;
3 **while** $i <= N$ **do**
4 $\quad i \leftarrow i + 1$;
5 $\quad H^i(K) \leftarrow H(H^{i-1}(K))$;
6 **end**
7 **return** $H^i(K)$ *where* i *equals* N ;
Algorithm 1: Generating a Hash Chain

3.1.2 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.

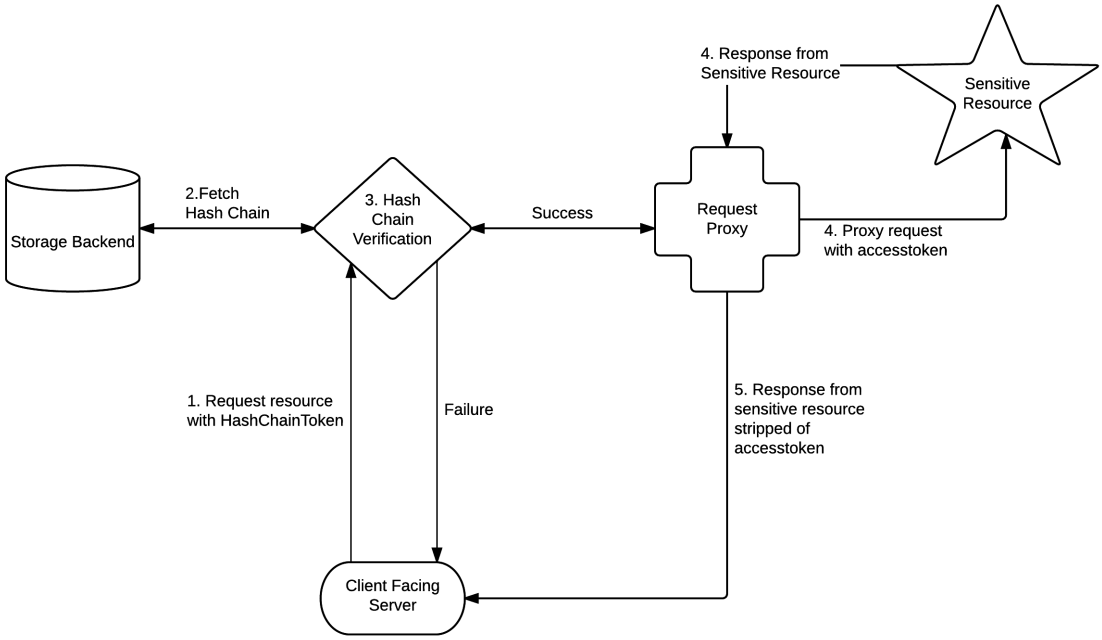

Figure 2: Architecture of the Central Trusted Authority. The CTA consists of a storage backend, a hash chain verifier and a request proxy.

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 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 Performance

4.1 Hash chain creation

In our testing hash chain initialization showed a linear increase, $O(n)$, in time complexity based on the length of the hash chain as illustrated in figure 3. This is in line with our expectations for hash chain implementations.

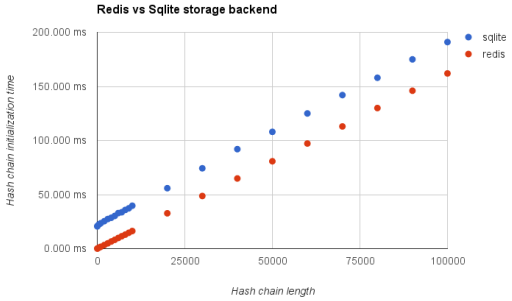

Figure 3: Linear time complexity of the hash chain initialization. Execution platform Intel i5-5200U 2.2Ghz, 12GB RAM, SHA-256 based on python 2.7

In our proposed architecture hash chain length affects CFS uptime based on the equation,

$$Uptime = \frac{Hash\ chain\ length}{Requests\ per\ second}$$

where $Uptime$ is the maximum time in seconds that a CFS can effectively operate, $Hash\ chain\ length$ is the size of the hash chain provided to the CFS at startup while $Requests\ per\ second$ is the average number of requests that a CFS needs to make to a CTA.

4.2 Hash chain verification and request proxy

Hash chain verification is a significantly faster process, $O(1)$, when compared to hash chain creation. Thus performance of the CFS is impacted merely by the delay introduced due to the request proxy.

Our testing reveals that an additional delay in the range of 0.05 to 0.15 seconds is introduced by using Hash chain verification and a simple implementation of request proxy as illustrated by figures 4 and 5.

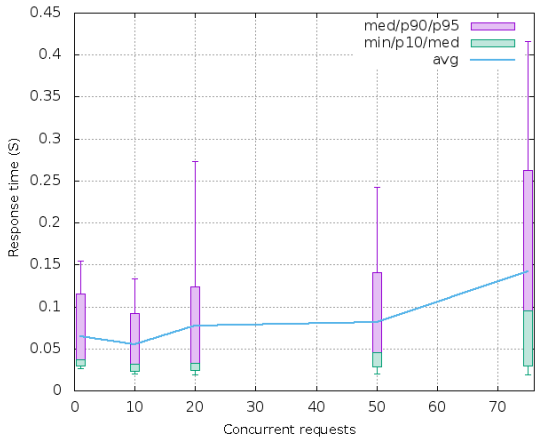

Figure 4: Response time for requests made to a resource through the CTA with a SHA-256 hash chain. The X-axis is the number of concurrent requests made and the Y-axis describes the response time in seconds

4.2.1 Testing environment

Our testing environment was based in AWS. The CTA was a pool of EC2 machines behind a elastic load balancer. Load tests were conducted using multiple CFSs located in the same AWS region as the CTA. The payload requested by the CFS was a file located on S3.

4.3 Server Pool

Creating new virtual servers to act as CFSs can be a slow and expensive process; however, by creating a pool of unused CFSs to act as quick replacements, we can minimize delay for customers. The size of the CFS pool needs to be balanced based on the time to initialize a new virtual server, the size of the hash chain, and the rate at which requests are made.

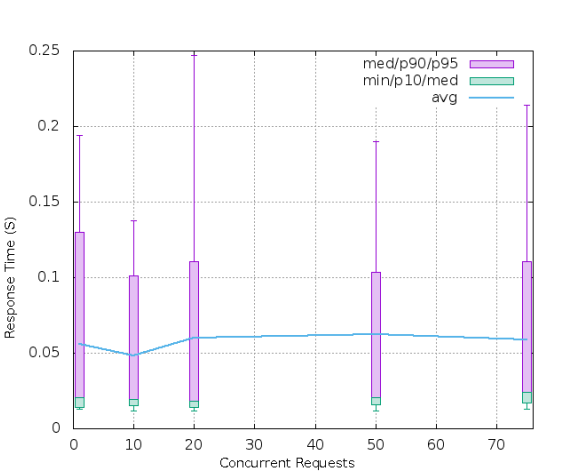

Figure 5: Response time for requests made to a resource directly without authenticating with the CTA. The X-axis is the number of concurrent requests made and the Y-axis describes the response time in seconds

n	Length of hash chain
r_{req}	Request rate
r_{boot}	Boot rate
r_{pool}	Pool size change rate

Table 1: Table of terms for equation 1

The relationship between pool size, hash chain length and requests per second can be illustrated by equation 1.

$$r_{pool} = r_{boot} - \frac{r_{req}}{n} \tag{1}$$

Equation 1 describes how the change in pool size relates to the length of the hash, the boot rate, and the request rate. When the number of requests that have been authenticated using a hash chain reaches the length of the hash chain, the VM using it is no longer functional and must be shut down and replaced. Our system keeps a pool of VMs in reserve so that this process is transparent to the user and no delay is noticed. System designers can use this equation to minimize the number of VMs that have to be held in reserve to sufficiently handle a variable load over the course of time. Boot rate is defined as the average number of VMs that can be instantiated in a particular period of time. Request rate is the average number of requests that come in during a time period. This value can change overtime as services will have different peak usage times during a day.

As an example consider that a company receives 50,000 requests per minute on an average

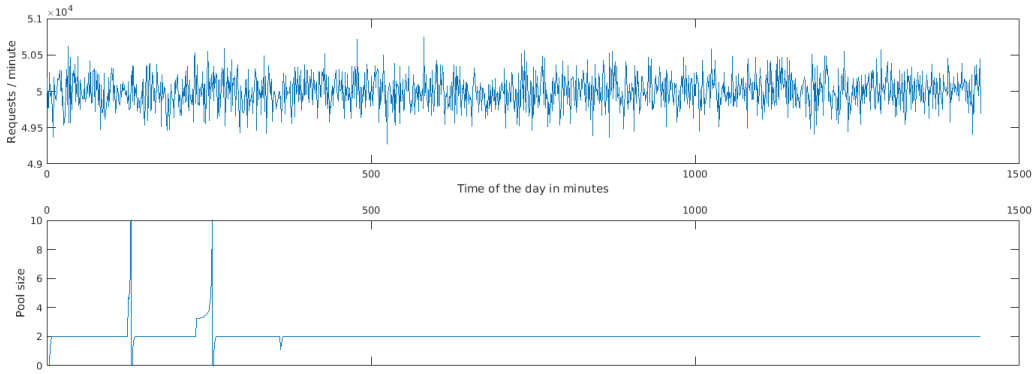

Figure 6: The upper plot shows a random request rate generated by a poisson distribution is with a lambda of 50,000 and the lower plot describes the pool size based on equation 1

day. They want to replace their VMs every 10 minutes, so each VM should have a hash chain with a length of 5000. They would have to boot 1 VM every 5 minutes. Figure 6 shows how our system would keep a small pool of VMs that would not consume significant resources, but would also never run out when the request rates are sampled from a poisson distribution with a lambda of 50,000.

Part of our future work is to design an adaptive algorithm that solves the multi-criteria optimization problem that manages the creating of new VMs based on learned traffic patterns.

5 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 defences 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.

5.1 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 despite the our earlier argument that this case is unlikely.

Chen et al [5] 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 [31].

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 [31].

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 [23, 28] on preventing reboot-attacks can be leveraged to impede the attacker's

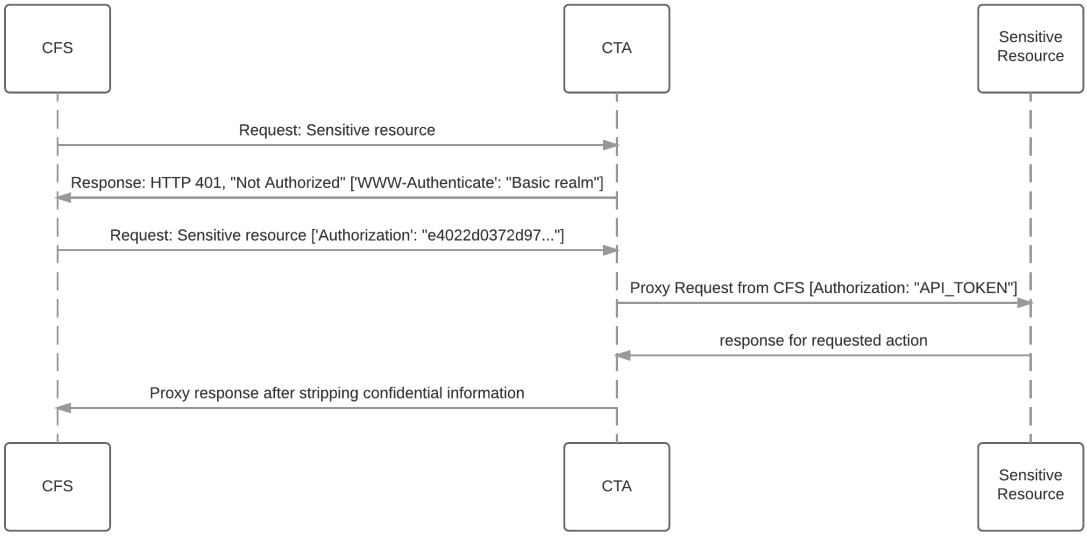

Figure 7: Sequence diagram describing the authentication and proxying capabilities of the CTA. CFS refers to Client Facing Server while CTA refers to Central Trusted Authority

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.

5.2 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, 3, 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] .

6 Discussion and Future work

6.1 Providing moving target defence

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

Unpredictability Cryptographic hash functions [34] 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 a vast state space as the output could be any of 2^{256} different values. As a result, the likelihood of an attacker guessing the correct value is negligible.

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 revocabil-

ity 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.

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, 24, 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 defence 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 [29], 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 learning algorithms to create an adaptive balance between server lifespan and hash chain length to optimize computational resources in a cloud system, this can be derived from the equations we have derived in this paper and current work into load balancing in cloud systems [32]. Such an adaptive system could be used to choose an ideal hash chain size and CFS pool in real time based on the performance requirements of a cloud system.

Timed Release Cryptography [4] can be integrated 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 middleware can potentially provide hierarchical authentication authorization capabilities to the CTA as illustrated by Yi & Wang [38]. 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.

6.5.1 Performance improvements

There are various avenues for performance improvements in our implementation, some of which are highlighted in this section.

Hash chain generation and verification is a well researched area. There have been various algorithms proposed to improve the performance of hash chain generation from O(n) time complexity to O(log n) [6, 17, 19, 35, 41]. We plan to implement these algorithms to significantly improve the delay introduced by hash chain generation at CFS startup. Fischlin and Yum et al [11, 40] have proposed various techniques for improving performance of hash chain verification, implementing these techniques would enable our implementation to perform only a fraction of the

work needed to perform hash chain verification thus reducing the delay introduced during CFS requests for a resource.

Our implementation is built in the python programming language, by taking advantage of the asynchronous event-driven programming paradigms we can significantly improve performance [20] of our request proxy server. Other options to significantly improve performance include integrating into existing reverse proxy servers such as Apache, Nginx or HAProxy. These reverse proxy servers are written in C / C++ and migrating our implementation as a C module might also offer significantly performance improvements.

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}) 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 [18], 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 defence.

Active authentication proposed by [39, 1, 22] 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 defence 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 [26] 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 system lacks the hash chain based approach that we use and as a result does not provide the additional moving target defence that raises the adversary's cost of launching an attack. Another limitation 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 which increases the cost of exploiting vulnerabilities for an attacker with minimal impact on performance by providing moving target defence using a pool of Client Facing Servers (CFS) which only operate for a relatively short period of time. Our architecture also significantly hampers an attacker's ability to steal sensitive configuration information from by centralizing configuration within a Central Trusted Authority.

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 defences that limit the effectiveness of an attacker, even when they are utilizing a zero-day attack.

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] 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.

[3] 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.

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

[5] 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.

[6] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *International Conference on Financial Cryptography*, pages 102–119. Springer.

[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 *Cryptographers 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] M. Jakobsson. Fractal hash sequence representation and traversal. 2002:1.

[18] 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.

[19] S.-R. Kim. Improved scalable hash chain traversal. In *International Conference on Applied Cryptography and Network Security*, pages 86–95. Springer.

[20] K. Kinder. Event-driven programming with twisted and python | linux journal.

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

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

[23] B. Libert and D. Vergnaud. Tracing malicious proxies in proxy re-encryption. In S. D. Galbraith and K. G. Paterson, editors, *Pairing-Based Cryptography 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] Y. Sella. On the computation-storage trade-offs of hash chain traversal. In *International Conference on Financial Cryptography*, pages 270–285. Springer.

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

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

[38] 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.

[39] 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.

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

[41] D. H. Yum, J. W. Seo, S. Eom, and P. J. Lee. Single-layer fractal hash chain traversal with almost optimal complexity. In *Cryptographers Track at the RSA Conference*, pages 325–339. Springer.