Evidence-Based Security Configurations for Cloud Datastores

Frank Pallas, David Bermbach, Steffen Müller, Stefan Tai
TU Berlin
Information Systems Engineering Research Group
Einsteinufer 17
10587 Berlin, Germany
{fp, db, sm, st}@ise.tu-berlin.de

ABSTRACT

Cloud systems offer a diversity of security mechanisms with potentially complex configuration options. So far, security engineering has focused on achievable security levels, but not on the costs associated with a specific security mechanism and its configuration. Through a series of experiments with a variety of cloud datastores conducted over the last years, we gained substantial knowledge on how one desired quality like security can have a significant impact on other system qualities like performance. In this paper, we report on select findings related to security-performance trade-offs for three prominent cloud datastores, focusing on data in transit encryption, and propose a simple, structured approach for making trade-off decisions based on factual evidence gained through experimentation. Our approach allows to rationally reason about security trade-offs.

CCS Concepts

•**Information systems** → **Cloud based storage;** *Parallel and distributed DBMSs; Database performance evaluation;* •**Security and privacy** → **Distributed systems security; Database and storage security;**

Keywords

Cloud storage, data in transit security, security configurations, performance benchmarking, trade-offs

1. INTRODUCTION

Cloud storage services and systems (cloud datastores) are core building blocks of modern distributed systems. Explicitly designed for maximizing performance, elastic scalability, and availability at the same time, they are a cornerstone of big data applications and the technological basis for today's largest Internet platforms and most innovative startups alike.

Like any other component of a distributed system operated in potentially hostile environments and interconnected across untrustworthy channels, cloud datastores must be appropriately protected against unauthorized access and manipulation to reduce business risks or simply to achieve regulatory compliance and thus practical applicability at all (e.g. when personally identifiable information is stored and processed). In particular, this refers to data flowing to and from an application's storage backend operated in the cloud. Cloud datastores therefore offer different mechanisms for data in transit encryption.

However, any security measure comes with costs attached, particularly in the form of impacts on usability, achievable productivity, system performance, or even realizable functionality. Due to these costs, not any measure that heightens security is advisable to be employed. Instead, any achievable gain in security (or, respectively, reduction of risk) must be weighed against the detrimental impacts before using a certain security measure. Furthermore, security measures can typically be used in a variety of different configurations (e.g., different key-lengths used for traffic encryption) which, in turn, result in differently severe impacts on other system qualities. The choice of a certain security configuration must therefore also be subject to a conscious weighing between the achieved security level and the adverse effects that have to be accepted. Striving for maximum security will only rarely be a rational outcome of such trade-offs.

In the context of cloud datastores, the choice of a security mechanism and its concrete configuration primarily has to trade-off security aspects against performance-related system qualities like achievable throughhput and latency.[1] Making a reasonable trade-off decision, then, requires reliable knowledge about the actual impact that particular security mechanisms and configurations have in these regards. Unfortunately, such knowledge does often not exist.

In lack of reliable knowledge about the actual impact of security mechanisms and their different configuration options on other system qualities, however, managing respec-

SAC 2017, April 03 - 07, 2017, Marrakech, Morocco

 978-1-4503-4486-9/17/04…$15.00
http://dx.doi.org/10.1145/3019612.3019654

[1]However, secondary effects on system qualities like data consistency must be kept in mind, too. See below for a slightly more detailed discussion.

tive trade-offs rationally becomes impossible. In practice, security-related configuration decisions are therefore typically made on the basis of so-called "best practices", security practicioners' experience and intuition, and rather stochastic ad-hoc considerations, obviously leading to suboptimal outcomes.

Over the last years, we conducted a large number of experiments with a diversity of cloud systems and in particular with cloud datastores, to experimentally study system quality trade-offs [6–10, 12, 22, 23]. In this paper, we report on select findings related to security-performance trade-offs for three prominent cloud datastores, focusing on data in transit encryption, and propose a simple, structured approach for making trade-off decisions based on factual evidence gained through experimentation.[2]

Starting with a brief summary of relevant background and related work in section 2, our approach is presented in section 3. In section 4, we then present results gathered for three widely used cloud datastores (Amazon DynamoDB, Apache Cassandra, Apache HBase) following this approach before concluding in section 5.

2. BACKGROUND AND RELATED WORK

On the one hand, our considerations rest upon research on the need for making rational (instead of gut-level) trade-offs between conflicting system properties in the field of information security and on existing, mostly theoretical approaches for doing so. On the other hand, we also draw upon existing, evidence-based approaches to making trade-offs between conflicting quality characteristics in modern distributed systems engineering in general, particularly with regard to the storage backend. Both strands shall be briefly outlined:

2.1 Trade-Offs for Security-Related Configurations

In the field of information security, the need for making trade-offs between different system qualities is omnipresent. As the choice of security measures and their configuration often also affects non-security factors, respective impacts must therefore be factored into security-related decisions on system design and configuration. Furthermore, different goals of security itself – like, for example, confidentiality and availability in the case of data encryption – must often be traded off against each other and even different configurations of the same security mechanism can have significantly different impacts. In this light, making trade-offs in practice has for a long time mostly been perceived as an art rather than a truly rationalizable process [13].

During the past years, however, intentions for a more rational practice of making security-related trade-offs have gained momentum. Especially the research domain of information security economics [2, 26] played a particular role in this regard. Amongst others, approaches have been proposed for trading off between organizational security policies and employee productivity [4, 5], for optimizing software patching practices [20], or for aligning configurations of partially interdependent security measures [15]. While theoretical models have reached notable maturity, however, one of the most severe obstacles for rational decisionmaking in the field of security is still a lack of reliable empirical evidence and of approaches for gathering such evidence (see also [14]).

Any approach trying to counteract this lack – and thus to foster more rational security configuration trade-offs – must pay regard to the multitude of security parameters potentially relevant in a given setting, their manifold and often indirect impacts on other system qualities (see below), and the diverse dependencies on factors like the typical application load or the employed infrastructure. A purely theoretical, model-driven approach will hardly be able to cover all these (often unknown) interdependencies. Similarly, quantitative measurements will also not suffice to encompass all relevant factors and interdependencies, including non-security qualities.

Instead, we herein thus propose an experiment-driven approach to optimizing security-related configuration trade-offs which strongly rests upon established concepts for evidence-based trade-offs in the design and configuration of distributed storage systems in general.

2.2 Evidence-Based Trade-Offs in Distributed Storage Systems

Making trade-offs to optimize different, conflicting system qualities against each other in the light of given application requirements is a philosophy inherent to most modern cloud datastores. In order to maximize performance, elastic scalability, and availability at the same time, for example, cloud datastores typically relax other quality properties like consistency [18]. In contrast to traditional, relational database systems, cloud datastores therefore often do not provide strict consistency guarantees but instead only ensure "eventual consistency" [25]. While this concept facilitates better performance, scalability, and availability, it entails disadvantages in matters of data staleness (i.e., the risk of reading outdated data from a node that did not yet receive a certain update) and ordering (e.g., two replicas may execute two updates in a different order, thus, leading to conflicts). Trading off these concistency-related properties against performance- and availability-related ones can then be done through the choice of replication-related configuration parameters, depending on the concrete requirements of the overall system or application to be built [6].

However, the trade-off between consistency and availability on the one hand and between consistency and latency on the other hand [1], are not the only trade-offs in (distributed) datastores. There are also trade-offs between available transactional features (mainly isolation guarantees) and performance or trade-offs between durability and performance (keeping things in memory vs. instantly persisting changes to disk). Furthermore, in such a situation with a multitude of trade-offs, there are also indirect, transitive trade-off relationships between qualities [6]. For instance, an increased security level could be countered by relaxed consistency settings to keep the performance level constant.

[2]Other security mechanisms aimed at different attack vectors (data at rest encryption, logging, etc.) are beyond the scope of this paper but can basically also be analyzed following the approach presented herein.

Typically, these quality levels are not strict guarantees but rather a system behavior that is caused by a combination of application workload and particular settings. For example, setting a consistency level in Cassandra [21] to QUORUM will result in some undefined staleness and ordering behavior which furthermore highly depends on the incoming application workload, e.g., a read-only workload will never encounter any inconsistencies and a write-only workload will never see its resulting inconsistencies.

The only way to gain reliable information on resulting quality levels is thus through benchmarking experiments – the resulting knowledge in turn is the necessary requirement for making informed trade-off decisions.

3. BENCHMARKING SECURITY CONFIGURATIONS

In this section, we will give an overview of how we typically gained our experiment results in the past. For this purpose, we first sketch-out our benchmarking approach before briefly discussing resulting limitations and disadvantages.

3.1 Benchmarking Approach

In benchmarking, we aim to gather reliable and relevant evidence for making security configuration trade-offs while reducing experimental efforts as far as possible. For this purpose, we propose the following 6-step process, which we have continuously refined through a large number of past experiments. While these steps seem straightforward and somewhat trivial, they are – as experience shows – not. In benchmarking, so many things can go wrong that not following a systematic approach seems courageous at best.

1) Relevant security parameters and trade-offs: In a first step, we identify all parameters and their configuration options, i.e., groups of security configurations, that are subject to the trade-off of interest. We also determine which system qualities beyond the desired security levels these parameters are likely to affect. However, even parameters and system qualities that – analytically – seem to be isolated should not be neglected as unforeseen effects always have a certain likelihood through complex interdependencies in modern distributed systems.

2) Reduction of the parameter design space: Building on the potentially large parameter space from step 1, we identify hard constraints based on the intended application domain, e.g., corporate policies or relevant regulatory requirements that disallow or require specific cipher suites in TLS. Configurations not meeting these criteria can then be disregarded as irrelevant, thus, effectively reducing the design space of potential options. To further reduce benchmarking efforts, we compare the still available options and disregard options that either seem unlikely to have an effect or that will have a very predictable impact. For this, we build on experiences from past experiments but also resort to plain analysis. While this helps to reduce efforts and, thus, cost of benchmarking, it always comes with the risk of missing entirely unexpected behavior.

3) Preference ordering: While it is hard to quantify a level of security for a given configuration on a cardinal scale, this is – at least for the context of cloud datastores – typically less difficult on an ordinal scale. By doing so, we compile a strict preference order of different configurations from a security perspective. Afterwards, we take the opposite point of view: Based on the pre-identified quality trade-offs (which, notably, might also be incomplete), we identify a similarly ordered scale for non-security properties like performance, data staleness, etc. (see above).

4) Experiment planning: As opposed to security, most relevant non-security qualities of cloud datastores can be measured experimentally. For doing this in a way meaningful to the intended application context, we have to identify typical interaction patterns between applications and datastores. All configuration options identified in step 2 then have to be analyzed experimentally. For this purpose, we resort to established benchmarking approaches for cloud datastores, e.g., [17], or develop a new one if there is no feasible option. For the benchmarks themselves, it is of utmost importance that the workload, i.e., the way in which the benchmarking tool creates stress on the system, is as close to the real application workload as possible. The more it resembles the application workload, the more are results meaningful through accurate prediction of expected quality levels.

5) Experiment execution: In the actual experiment phase, we run exactly the same benchmark against the cloud datastore once per configuration setting. This allows us to identify the actual impact that a security configuration has on other qualities, e.g., performance or data staleness, which are in a direct or indirect trade-off relationship to security. Typically, this will lead to a large number of experiments which we additionally repeat several times to ensure repeatability but also to study stableness of our results. Obviously, this is only feasible if the experiments are at least semi-automated to require as little human intervention as possible.

6) Result analysis: Finally, in the analysis phase, we carefully weigh different security configuration options with regards to their security guarantees and their experimentally determined impact on other qualities to make rational trade-off decisions. Typically, the final recommendation will be a "middle" solution. However, as we will see in section 4.1, sometimes experiments also show that security may come for free – at least for the user of the cloud datastore – and that maximum security can be a perfectly rational decision, too.

3.2 Discussion

The approach presented above may seem tedious and time-consuming, even though careful consideration of key parameters can be used to reduce the time and cost effort. However, results are priceless: Depending on whether the "gut feeling" decision would have been in favor of (typically) performance or security, cloud application developers can either gain additional security guarantees at very little or no additional performance impacts or they can gain significant performance improvements by disregarding rare or only theoretical threats while still meeting mandatory security requirements. Also, as we will see below, such results may also suggest to reconsider the entire security subsystem of cloud datastores due to catastrophic performance impacts, even compared to other cloud datastores.

Obviously, our approach still depends on a certain level of experience and does not work "automatically and out of the box". In particular, step 2 and – to a certain extent – step 1 are not fully operationalizable if the resulting parameter space should be meaningful and sufficiently small to experiment with. Also, the preference ordering in step 3 will typically not be fully deterministic for a given setting because of all too often rather tacit than explicit preferences. Nonetheless, our approach still strongly reduces the need for "gut feelings" and paves the way for significantly more rational and structured security configuration trade-offs.

Another issue is the expected lifetime of benchmarking results, i.e., the period of time until measurement results will have become obsolete. Generally, experiments become obsolete after a while if either software or hardware updates are made to the deployment platform, i.e., results for an on-premise datastore where neither software nor hardware are changed will remain relevant. For this, we need to distinguish a cloud datastore service like Amazon DynamoDB and a self-hosted cloud datastore system like Apache Cassandra deployed on a compute cloud. In case of cloud services, changes to software and hardware will be frequent and results will therefore become obsolete very fast. However, at the same time such services offer only very limited security configuration options so that repeating experiments is relatively inexpensive. For a self-hosted cloud datastore, in contrast, there are much more configuration parameters so that experiments can be time-consuming and expensive. However, here the application developer has a much higher degree of control over the deployed system stack: software is only updated explicitly by the application developer, i.e., he can also decide to not update, and underlying hardware is updated rarely.

Still, the probability of the benchmarking results being obsolete should be carefully weighed against the costs and efforts of rerunning the experiments. Fortunately, applications are bound to notice performance changes through monitoring. This may provide an indicator for the point in time when changes have been made: while monitoring results are stable, the current configuration may no longer be the "best" option, but it will not be worse than at the time when it was selected.

4. EXEMPLARY CASE STUDIES

To illuminate the benefit of such experimentally gathered evidence, we present three case-studies focusing on three widely used cloud datastores below: Amazon's provider-maintained DynamoDB service, Apache Cassandra, and Apache HBase. For each case study, the considered cloud datastore is briefly introduced, followed by an outline of the available mechanisms for achieving data in transit encryption and the respective configuration options available. Based on this, we then describe the experiments we conducted and present selected results. As the main focus of this paper is on the general approach of evidence-based security trade-offs, we mainly refer to results for maximum throughput achieved for a mixed read-/write-workload here and also abstain from elaborating on in-depth details of the experiment settings. However, all experiments were conducted in accordance with benchmarking best practices [11], e.g., ensuring that the measurement client did not become a bottleneck in the experiment.

4.1 Amazon DynamoDB – Free Lunch

DynamoDB is a managed cloud datastore provided by Amazon as part of the AWS ecosystem. It is explicitly designed for handling large amounts of data and for providing high throughput in writing and accessing these data. As a managed service, DynamoDB is procured on the basis of a given throughput to be provided and scales automatically to achieve this throughput.

Data in transit encryption for DynamoDB is realized by means of TLS-based HTTPS connections. Generally speaking, TLS can be operated in a multitude of different configurations (protocol version, key-length, operation mode, etc.) which are negotiated during connection handshake and which carry different computational efforts on the client as well as the service side. Without significant tweaks on the client side implementation, however, DynamoDB only allows to either use TLS or not, with concrete configurations varying across regions and over time.[3] In practice, security engineers thus primarily have to decide whether to use data in transit encryption or not when using DynamoDB. By changing the employed region, however, even different TLS configurations may be chosen indirectly.

As outlined above, making a rational trade-off between these options must weigh the resulting security level against the respectively achievable performance. Following former experiments with a comparably weak TLS configuration[4] [22], we thus conducted experiments for DynamoDB operated in an AWS region that provides reasonable TLS security[5] and compared a configuration with data in transit encryption turned on to one without. In all these experiments, we benchmarked throughput, read latency and write latency.

In none of these dimensions, however, we observed statistically significant deviations and thus any performance impact of using TLS-protected communication at all, neither for the previously benchmarked, rather insecure configuration nor for the current setting with reasonable security (see figure 1). Leaving aside an increased computational load on the client side, Amazon is thus shouldering the whole bill of provider-side computational overhead and DynamoDB provides the provisioned throughput independently from data in transit encryption turned on or not. At least in matters of storage-side performance, there is thus no reason to abstain from data in transit encryption at all or to choose a less secure configuration for performance-related reasons. Without the evidence provided by our experiments, intuitively made trade-offs would presumably have come to different conclusions for a multitude of "not so critical" cases.

4.2 Apache Cassandra – It's in the Details

Apache Cassandra is a highly scalable and fault-tolerant distributed datastore often used in distributed systems with large datasets. A common use-case is to deploy a Cassandra cluster on cloud compute instances (like, e.g., Amazon

[3]At the time of our experiments, for example, AWS Ireland used TLS_RSA_WITH_AES_128_CBC_SHA, while AWS Germany offered significantly higher security with TLS_ECDHE_WITH_AES_256_GCM_SHA384.
[4]SSL_RSA_WITH_RC4_128_MD5
[5]TLS_RSA_WITH_AES_128_CBC_SHA.

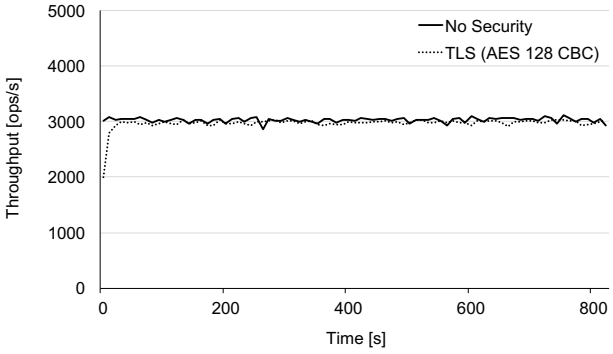

Figure 1: Experimentally determined throughput impact of TLS encryption for DynamoDB

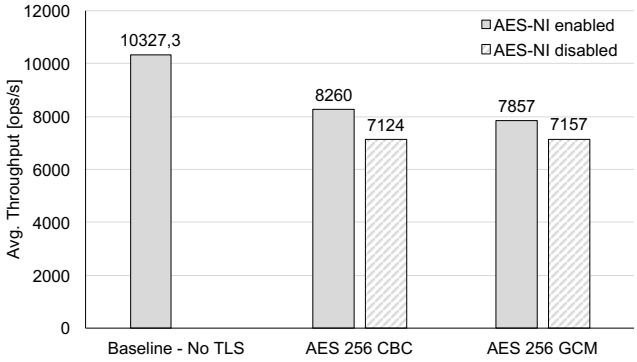

Figure 2: Throughput impact of AES operation mode and AES-NI hardware support for a TLS-protected Cassandra cluster with 3 nodes

EC2 instances) and to scale the number of instances according to dynamically fluctuating loads. Due to this model of self-operation, any encryption overhead has to be borne by the user of a Cassandra-cluster – either in the form of reduced performance or through increased costs for procuring additional instances to achieve similar performance.

Data in transit encryption is realized through TLS in Cassandra. As Cassandra is typically self-operated, available TLS configurations are not restricted to a small, pre-selected set of combinations but just limited by the TLS implementation used on the client- and the server-side. Any user of a self-deployed Cassandra cluster will thus have to make rational choices for such diverse parameters as the protocol version, the protocol to be used in the handshake phase, the algorithm to be used for bulk data encryption together with its keylength and operation mode, or the used Hash algorithm.

Furthermore, the hardware-software-stack that Cassandra is deployed on can play a significant role for the performance impact of a given configuration: If it allows to utilize cryptographic hardware extensions like AES-NI, the expectable performance overhead of supported encryption algorithms is significantly lower than for stacks with purely software-based encryption. Besides, the used JVM may also influence the performance impact resulting from certain configurations because of the different TLS implementations.

Finally, a self-operated, distributed datastore like Cassandra requires to consider data in transit encryption for both, external communication with client applications (application-replica communication, AR) and cluster-internal communication between different nodes (replica-replica communication, RR). Depending on the concrete deployment, it might, for example, be perfectly reasonable to abstain from protecting cluster-internal traffic in exchange for a rather small performance gain as long as internal traffic is appropriately protected by other means (e.g. a virtual private cloud).

Even after sorting out cipher suites with insufficient security or only marginal adoption (esp. RC4-, MD5- or ChaCha20-based combinations) according to step 2 of our approach, a security engineer is thus still confronted with an overly broad set of potential configurations to be evaluated. Following step 2 of our approach even further, we thus narrowed down our experiments to a subset of cipher suites with reasonable parameter combinations. Of the still extensive experiments we conducted, we only report on two parameters here: The operation mode of AES, where we tested "Cipher Block Chaining (CBC)" against "Galois/Counter Mode (GCM)", and the availability of AES-NI extensions for hardware-supported encryption.

For a cluster of 3 nodes deployed on Amazon EC2 (m1.large) with only external (application-replica) communication protected by TLS, we observed significant performance differences resulting from different settings of these parameters (see figure 2): Compared to the baseline with no encryption enabled, the configuration with 256 bit AES led to a nearly similar 31% performance decrease for both operation modes with AES-NI deactivated.[6] Given that GCM is usually deemed more secure than CBC (see step 3 of our approach), this suggests it to be preferred in any case. With AES-NI hardware extensions available, however, CBC led to a performance drop of 20% while GCM produced an overhead of 24% – Depending on the concrete use case, this might be a reason for preferring CBC over GCM. Besides highlighting the impact of available hardware extensions for the achievable performance in general, this also illuminates how different configuration parameters may interrelate in this regard.

With cluster-internal (replica-replica) encryption also activated, the performance impact of encryption was slightly higher without hardware support (around 33% for both operation modes as compared to no encryption at all), while the activation of AES-NI had only marginal or even no significant effect (30% for CBC and still 32% for GCM). Again, this demonstrates the manifold interdependencies between different configuration parameters and the resulting need for carefully benchmarking different configurations.

4.3 Apache HBase – Be Prepared

HBase is another distributed datastore that is often self-deployed on multiple compute cloud instances as the storage backend of large distributed systems and applications.[7]

[6]GCM performed only marginally better than CBC.

[7]There are, however, also different offerings for provider-maintained HBase deployments that can be used "as a ser-

HBase and the underlying HDFS originally were reimplementations of the Google storage stack comprising BigTable [16] and GFS [19]. As HBase is a core component of the Hadoop ecosystem, it is widely used in the big data domain.

Different from DynamoDB and Cassandra, data in transit encryption is not done through TLS connections in HBase but rather employs separate encryption mechanisms embedded into native protocols.[8] In particular, we have to distinguish between an "HBase layer" with communication based on HBase RPC and an underlying "HDFS layer" encompassing Hadoop RPC and the HDFS data transfer protocol. External communication is only realized on the HBase layer while inter-node communication comprises both, HBase and HDFS layer.

For these two layers, data in transit encryption can be activated and configured separately [24, p. 209-210]. Activating encryption for external communication on the HBase layer, in turn, automatically enables HBase-internal encryption in this layer, too. Assuming a distributed HBase deployment in a public cloud, security engineers will typically have to decide whether to activate each of these mechanisms. The performance decrease to be expected – and thus to be factored into the respective trade-off decision – is suggested to be around 10% by the official HBase documentation [3, section 58.3].

To verify this magnitude, we benchmarked an HBase cluster with 6 data nodes deployed on Amazon EC2 (m4.large) in two different configurations first: No data in transit encryption enabled and data in transit encryption enabled on both layers. As it can be seen from figure 3, results were much worse than expected according to the official documentation. Instead of 10%, we actually observed an overall throughput decrease of 47%. For a cluster with 3 data nodes, however, we got a performance decrease of only 7% while for 12 data nodes, throughput dropped by 37% [23]. Especially for larger, more realistic cluster sizes, the actual impact observed through our experiments is thus way beyond what the official documentation suggests.

Again, this illuminates the importance of experimentally gathered evidence for making rational security-related trade-offs: While a performance decrease of 10% might be deemed a reasonable price for securing some less-critical business data against eavesdropping, for example, a decrease of 47% (or, vice versa, a rise of costs by up to 90% for achieving similar performance with a respectively increased cluster size) will certainly lead to significantly different trade-off decisions.

In subsequent experiments with different cluster sizes and instance types, these results basically held true. Through benchmarking additional configurations, we were furthermore able to attribute the largest share of the performance impact (between 66 and 80%) to the HBase- and not to the HDFS layer. Besides fostering better-founded configuration decisions, our approach thus also provides a clear indication for a careful re-examination of respective implementation parts within HBase's security subsystem.

vice".

[8]The details of the HBase security subsystem are quite complex and don't matter much here. Most importantly, however, it heavily employs Kerberos and Java's Simple Authentication and Security Layer (SASL).

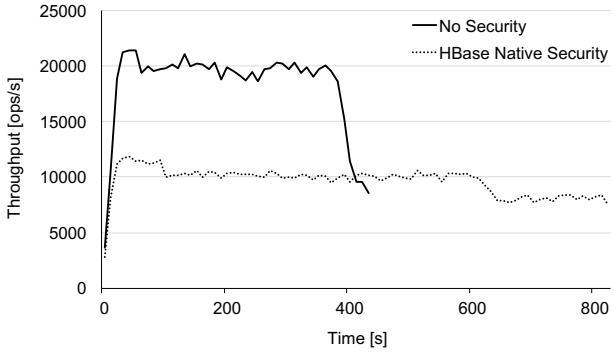

Figure 3: Experimentally determined throughput impact of HBase wire encryption for a cluster with 6 data nodes

5. CONCLUSION

Today's cloud systems offer a diversity of security mechanisms, each with potentially complex configuration options. So far, research on security engineering has focused on achievable security levels, but not on the costs associated with a specific security mechanism and its configuration. Performance engineering, on the other hand, typically opts for the fastest (but often weakest) security configuration – for instance, until a few years ago even online banking sites used RC4-based SSL cipher suites. However, in most scenarios a "middle ground" would be the best option for an application developer: cloud systems that offer reasonable performance while ignoring exotic or theoretical threats. Identifying such a middle solution, however, requires detailed knowledge on the impacts that specific security configurations have on other qualities such as performance or data consistency.

Through a series of experiments with a variety of cloud datastores conducted over the last years, we gained substantial knowledge on how one desired quality like security can have a significant impact on other system qualities like performance. Building on these experiences, we proposed in this paper a simple, structured approach for making trade-off decisions based on factual evidence gained through experimentation. Our approach allows to rationally reason about security trade-offs. Using this approach, we then reported on select findings related to security-performance trade-offs for three prominent cloud datastores, focusing on data in transit encryption. We showed that, depending on the cloud datastore in question, enabling security may have severe impacts in all setups or only for specific configurations and deployments. We also showed that security may, in fact, come for "free" – at least for the cloud user.

In future work, we plan to work on adaptive communication middleware systems that automatically change security configurations of systems such as Apache Cassandra based on their current state to meet both performance and security goals. Furthermore, we intend to apply our approach to other security measures beyond data in transit encryption and to examine their interdependent impact on non-security system properties such as performance but also data consistency.

Acknowledgments

The authors would like to thank Amazon Web Services who provided research grants for the experiments.

6. REFERENCES

[1] D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2):37–42, 2012.

[2] R. Anderson and T. Moore. The economics of information security. *Science*, 314(5799):610–613, 2006.

[3] Apache Software Foundation. Apache HBase reference guide, 2016. https://hbase.apache.org/book.html#security.example.config.

[4] A. Beautement, S. Parkin, I. Becker, K. Krol, and A. Sasse. Productive security: A scalable methodology for analysing employee security behaviours. In *12th Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[5] A. Beautement, M. A. Sasse, and M. Wonham. The compliance budget: Managing security behaviour in organisations. In *Proceedings of the workshop on ew security paradigms (NSPW)*, pages 47–58. ACM, 2009.

[6] D. Bermbach. *Benchmarking Eventually Consistent Distributed Storage Systems*. PhD thesis, Karlsruhe Institute of Technology, 2014.

[7] D. Bermbach, J. Kuhlenkamp, B. Derre, M. Klems, and S. Tai. A middleware guaranteeing client-centric consistency on top of eventually consistent datastores. In *Proceedings of the 1st International Conference on Cloud Engineering (IC2E)*, pages 114–123. IEEE, 2013.

[8] D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of amazon s3's consistency behavior. In *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing (MW4SOC)*, MW4SOC '11, pages 1:1–1:6. ACM, 2011.

[9] D. Bermbach and S. Tai. Benchmarking eventual consistency: Lessons learned from long-term experimental studies. In *Proceedings of the 2nd International Conference on Cloud Engineering (IC2E)*, pages 47–56. IEEE, 2014.

[10] D. Bermbach and E. Wittern. Benchmarking web api quality. In *Proceedings of the 16th International Conference on Web Engineering (ICWE)*, pages 188–206. Springer, 2016.

[11] D. Bermbach, E. Wittern, and S. Tai. *Cloud Service Benchmarking*. Springer, 2017.

[12] D. Bermbach, L. Zhao, and S. Sakr. Towards comprehensive measurement of consistency guarantees for cloud-hosted data storage services. In R. Nambiar and M. Poess, editors, *Performance Characterization and Benchmarking*, volume 8391 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2014.

[13] F. Bjorck. Institutional theory: A new perspective for research into is/it security in organisations. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS)*, 2004.

[14] R. Böhme. Security metrics and security investment models. In *International Workshop on Security (IWSEC)*, pages 10–24. Springer, 2010.

[15] H. Cavusoglu, S. Raghunathan, and H. Cavusoglu. Configuration of and interaction between information security technologies: The case of firewalls and intrusion detection systems. *Information Systems Research*, 20(2):198–217, 2009.

[16] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, OSDI '06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[17] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st Symposium on Cloud Computing (SOCC)*, pages 143–154. ACM, 2010.

[18] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of 21st Symposium on Operating Systems Principles (SOSP)*, pages 205–220. ACM, 2007.

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the 19th Symposium on Operating Systems Principles (SOSP)*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[20] C. Ioannidis, D. Pym, and J. Williams. Information security trade-offs and optimal patching policies. *European Journal of Operational Research*, 216(2):434 – 444, 2012.

[21] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[22] S. Müller, D. Bermbach, S. Tai, and F. Pallas. Benchmarking the performance impact of transport layer security in cloud database systems. In *Proceedings of the 2nd International Conference on Cloud Engineering (IC2E)*, pages 27–36. IEEE, 2014.

[23] F. Pallas, J. Günther, and D. Bermbach. Pick your choice in HBase: Security or performance. In *Proceedings of the 2016 IEEE International Conference on Big Data (BigData 2016)*. IEEE, 2016.

[24] B. Spivey and J. Echeverria. *Hadoop Security - Protecting Your Big Data Platform*. O'Reilly, 2015.

[25] W. Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, Oct. 2008.

[26] WEIS. Workshop on the economics of information security. http://econinfosec.org/.