zipabout

the secret of every successful journey

Data Security in a Shared Data Environment

As the collection, processing and exploitation of “big data” becomes widespread in industry, the cost to business of maintaining individual “silos” is becoming apparent: from the raw economics of managing equipment, to the threat of a breach, and the opportunity cost of missed insights and collaborations. This paper outlines the security aspects of a *centralised* data-processing platform (the “data platform”), from a technical and legal standpoint; paying particular attention to the trade-off between the benefits offered (for efficiency, management and sharing) versus potential risks and their mitigations. There is no single optimum when striking this balance, so a spectrum of use-cases are explored, with their relative risks compared through analysing the capabilities they provide to different actors (data controllers, the platform operator, other platform users, external collaborators, external attackers and software agents).

The only way to *guarantee* data security is not to collect it in the first place; this can be the correct choice, but the platform exists for those data deemed necessary, valuable and useful. Its purpose is to extract the most insight and utility, whilst minimising the possibility of risks.

The Problem

Data “silos” are those individual databases, document stores and servers maintained by separate entities (business, academic and governmental), which are disconnected from each other and the wider world, except for some narrow (often bespoke) interface. Isolating data in this way can provide a security advantage, and compliance with relevant legislation such as the EU’s General Data Protection Regulations (GDPR) (European Commission, 2016) and its implementation as the UK’s Data Protection Act 2018. However, this only works so long as equipment, practices and expertise are maintained. As more organisations follow this approach, especially those not specialised in information technology, standards can slip and problems arise[1]. A centralised data platform can offload this maintenance burden from individual organisations and amortise the cost across many users.

A further problem comes from hiding data in the first place, since this also hides potential insights it may provide and hence opportunities for commercial advantage or research opportunities. Whilst big data platforms, such as Hadoop, can expose some of this latent knowledge, their use requires even more in-house expertise and maintenance cost. Such tools are also unable to exploit patterns *between* silos, which hinders opportunities for collaboration. For example, finding overlaps in delivery schedules and capacity allows multiple businesses to combine their journeys, reducing costs and carbon emissions. Insights like this are made possible by centralising data into one platform, where boundaries are a matter of policy rather than necessity, allowing the controlled exposure of selected data sets, summaries or events to chosen collaborators or the wider world.

The Solution

Zipabout's centralised data platform is a scalable, managed solution to the task of data storage, processing and controlled sharing. Its flexibility provides a high level of control for users and organisations, from both an integration and security point of view: spanning from simple operations on small, static data sets; to complex, multi-stage processing of high-throughput streams in real-time; from blind storage of client-encrypted data; to selective sharing of differentially-private summary statistics; to global publishing of research datasets.

The Data Platform

A high-level overview of the data platform's architecture is given in Figure 1, showing the prominent role of the real-time database and analytics engine. The platform's operation is best understood as a "pipeline": external inputs are encrypted with a user's key, stored to a permanent archive and streamed into the real-time database. Both of these components

[1] This is particularly acute in the growing "Internet of things" (IoT), where personal data breaches are common as traditional product manufacturers attempt to implement and run their own infrastructure (Alaba, 2017).

keep the location/identifier of a data set hidden (see the section on *capability-based security* below). Those with whom access has been shared can subscribe to the relevant stream or fetch static data from the archive; in effect using the platform as cloud publishing/storage solution. This is the most basic use-case for the data platform: it can be the most secure, but also provides the fewest opportunities for offloading processing costs or gaining insights.[2]

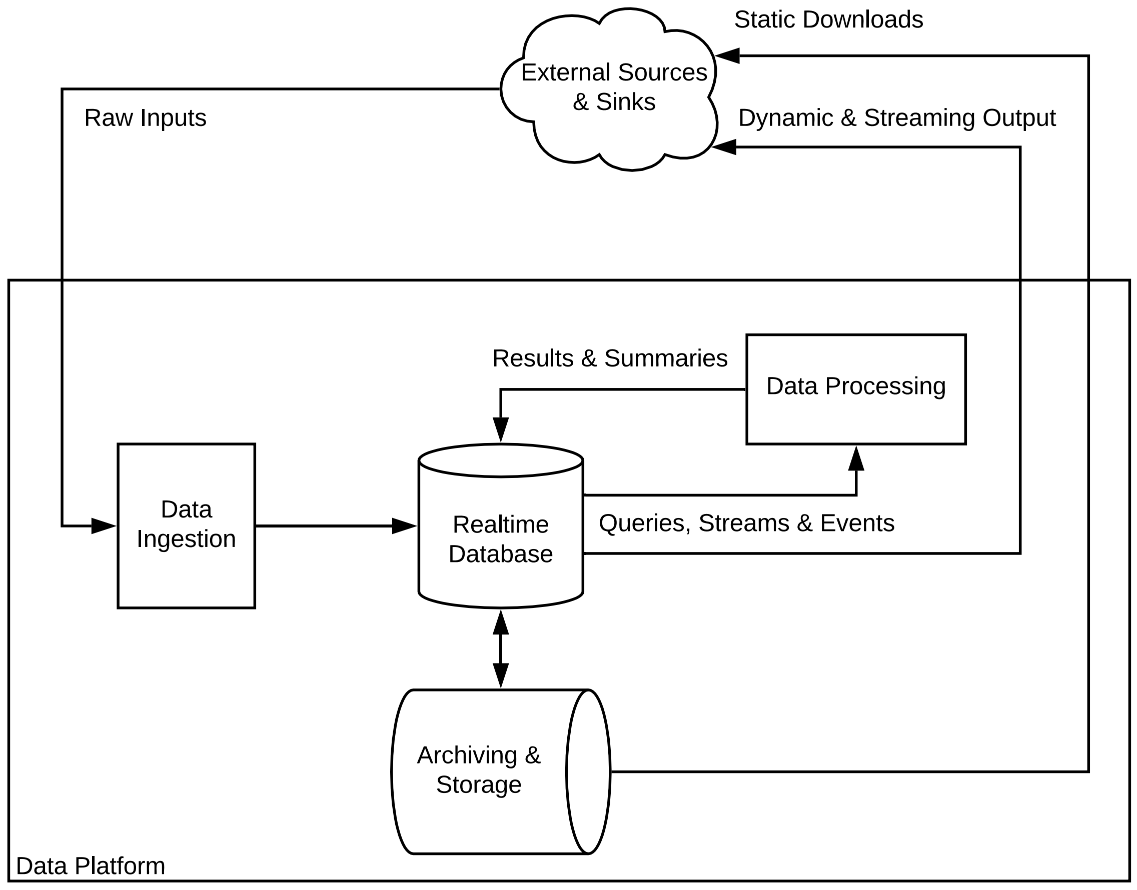

Figure 1 High-level architecture of the data platform, and its relation to external entities.

More sophisticated scenarios require the input to be readable by the platform (although it is still encrypted "in flight"; see the section on *encryption* for more details). In this case, a

[2] Encrypting data at the source, before uploading to the data platform, ensures that nobody else is able to read it. Whilst secure, this prevents the platform from performing any processing, so it can only act as dumb storage, or a coordinator for "external" processing stages.

processing pipeline can be established to perform (almost) arbitrary computation.[3] From a security perspective, the stages of a pipeline are of three types: *direct*, *indirect* and *external*:

- **Direct** stages feed their results straight into the next processing stage. Security-wise, a series of direct processing stages is no different to a single stage, although this subdivision may be useful for reducing the scope of required capabilities (e.g. which data sets can be accessed by each step)[4]. This is the most secure, since the resulting data is *ephemeral:* it never exists outside of the producing and consuming stages, it is not stored, and hence cannot be queried by an attacker or accidentally shared. The drawback is that such data cannot be re-used (although it can be *recalculated*), so is usually specific to a particular pipeline.
- **Indirect** steps feed their results back to the real-time database. This lets multiple pipelines consume the same data, and ensures it is archived to allow future querying. This comes at a security cost, as such results may be exposed in future leaks.
- **External** stages take place outside the data platform: effectively downloading the data and uploading the results. Although this data is encrypted in flight, the platform can make no guarantees about the security of the external system. Such data is, by necessity, subject to archiving and querying, hence the security concerns for indirect stages also apply here. Although external processing makes the transmitted data more vulnerable, it can provide an overall security benefit by moving the processing into a more trusted environment. For example, if a processing step requires auxiliary data which an organisation does not wish to upload, they can keep it offline and perform that step on their own machines. This also applies to encryption keys, allowing the platform to orchestrate the processing of data which it is unable to read (either in whole, like in the cloud storage use-case; or in part, such as particularly sensitive fields of a data set).

[3] Subject to time and memory feasibility. In particular, calculations using "recent" data will perform better than those requiring "historic" data (far back in the stream). This can often be mitigated with streams of intermediate "summaries"; e.g. directly calculating an average is costly, but is trivial using a counter and a running total.
[4] Direct processing stages are mostly useful for their modularity and composability; breaking down a task into manageable steps, which are easier to reason about and debug.

Pipelines must end with an indirect or external stage, so the resulting data, stream or events are available for use elsewhere in the platform, or exported to another system or tool such as a real-time visualisation.

Governance and Ownership

Uploading data to a centralised platform, whether solely for use by the originating organisation or for sharing with others, requires an understanding of the relevant stakeholders, oversight, legislation and governance; some of which are outlined here.

Data Controller

The **data controller** is the person or entity which "determines the purposes and means of the processing of personal data" (European Commission, 2016, p. Article 4(7)). The decision to use a data platform does not alter the data controller: the person or organisation acting as data controller for a data set remains in that role if that dataset is subsequently processed via the data platform. This arrangement might subsequently be altered (assuming some agreement between the relevant parties) if the sharing capabilities of the data platform are employed. This can introduce **joint controllers**, who are each responsible for the appropriate use of the data, but may have their own, separate purposes.

Data Processors and Technology Providers

As developer of the data platform, Zipabout Ltd is a **technology provider**, and is hence responsible for the integrity of the software itself (for example, the absence of malicious features like back-doors). AWS are also a technology provider, since they develop the cloud computing technology underlying the data platform. AWS was chosen as a technology provider due to its leading role in the cloud computing sector; its implementation, and in some cases invention, of rigorous state-of-the-art security practices; and the trust it has earned from many high-profile organisations around the world.

The data platform is "software as a service" (SaaS), meaning that Zipabout Ltd also *runs* the software on behalf of users and customers, who connect via the Internet. Similarly, AWS offers its cloud computing facilities as SaaS. This makes Zipabout Ltd and AWS **data**

processors, since their role is to act "on behalf of the [data] controller" (European Commission, 2016, p. Article 4(8)). This role is enforced by having all content in the data platform private by default, such that only the user account which uploaded or created a data set can view and manage it. A third-party encryption service (Amazon KMS) ensures that all access by Zipabout Ltd, even outside the data platform's security model, is logged for auditing purposes.

Personal Data and Privacy

Recent legislation, such as the EU's GDPR, places strict requirements on the handling of personal data; that is, data which pertains to or identifies a specific individual (European Commission, 2016). In particular, informed consent must be given by those individuals, the purpose of the data must be stated, as well as those the data may be shared with. The same standards that protect personal data under GDPR are just as applicable to commercially sensitive data (which may or may not include personal data). The technology described in this white paper describes the protection methodology for both types of data.

Care must be taken when collecting and processing such personal data, regardless of whether that is implemented (in whole or in part) via the data platform. If the data platform is applied to personal data, special attention should be given to its sharing and exploratory data-mining capabilities, as these would generally require explicit consent under such regulations. The section of this document regarding *differential privacy* may also be relevant in such situations.

Technical Details

Encryption

The data platform employs encryption for two purposes. Static data, such as data files or historical archives, are encrypted when not in use (termed *at rest*). This limits the potential for data leaks, since these files are unreadable without their associated decryption key. A new encryption key is generated for each platform user, limiting the scope of a compromised key or user account. This encryption is provided by Amazon's Key Management Service (KMS), which uses special-purpose security hardware to generate and store keys, ensuring they are never visible (to users, Zipabout or even Amazon) and that all uses of these keys are logged, to allow monitoring and auditing.

The second use of encryption in the data platform is for data in transit, which uses the latest version of the Transport Layer Security protocol (TLS, version 1.3 as of January 2020). This prevents eavesdroppers from reading the data, and is the industry standard for secure Internet traffic such as online banking. Fresh keys are generated for each transfer to ensure *forward secrecy*: that a potential breach in the future cannot compromise past traffic (e.g. if eavesdroppers saved a copy).

Capability-based security

Permissions in the data platform follow a capability model. This enforces access controls by restricting the information needed to perform an action, known as “capabilities”, such that those without permission are unable to even state what the action is. For example, access to data is restricted by storing it at secret, unguessable (*cryptographically-secure*) locations. The location itself acts like a special-purpose password for accessing the data, since only those who have been told the location will be capable of making a request (Fabry, 1974).

Capabilities provide a very general security model, encompassing common patterns like users, roles, passwords, API keys, database connections and remote shells. The data platform manages access on a more fine-grained level, with separate capabilities for reading, appending-to and managing each dataset, with further restrictions enforced via *proxies*. A proxy delegates requests to a capability, whilst keeping it hidden; for example, the read capability for a dataset can be hidden in multiple proxies, one for each user who needs access. This keeps the “true” location secret and allows each user’s access to be revoked by disabling their proxy. Proxies can also alter or discarded requests, for example only fetching a limited range of data, or logging all accesses to a database, or ignoring requests made outside certain hours. Since proxies themselves are (restricted) capabilities, they can *also* be proxied, forming hierarchies like the example in Figure 2.

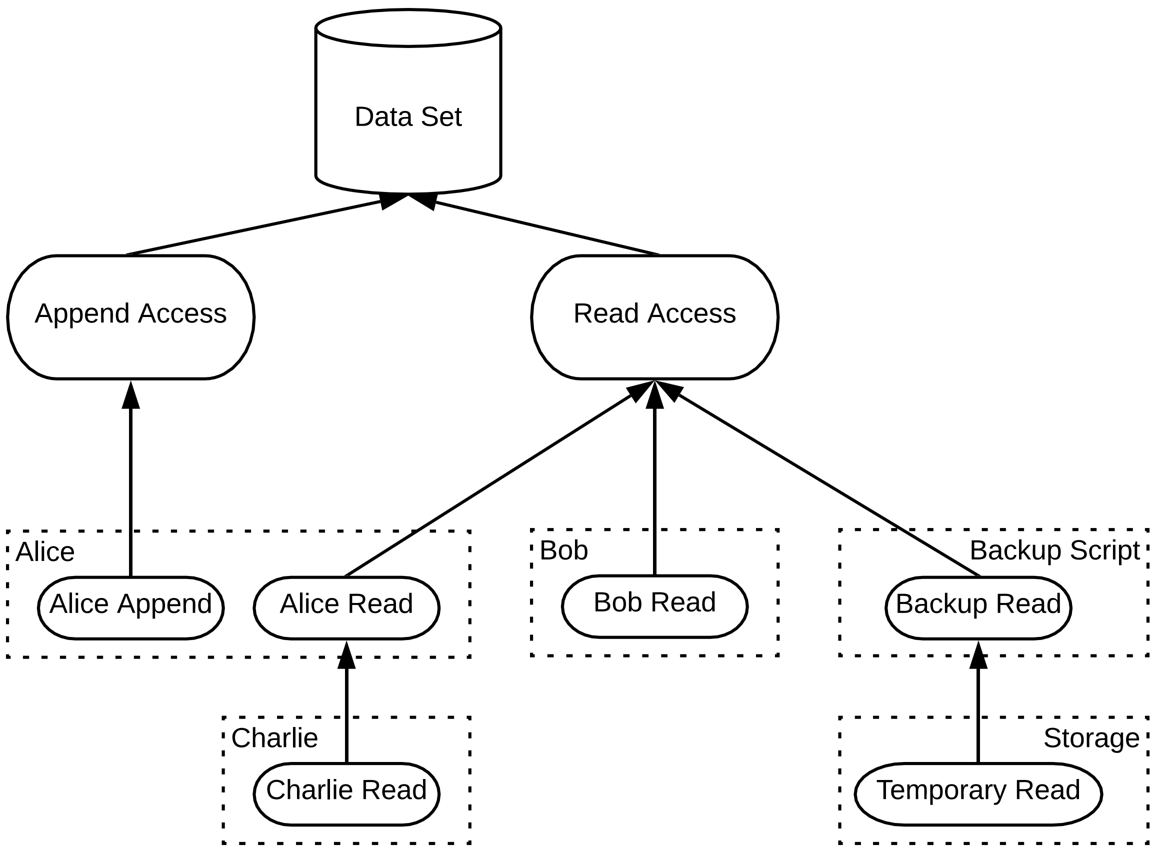

Figure 2 Example of hierarchical capabilities, shown as rounded boxes with arrows indicating proxying. Dotted boxes represent external entities (people, software and services), via the capabilities given to them.

All platform users can create, share and revoke proxies for their capabilities. This discourages insecure practices, such as password sharing, which is commonly used to bypass less flexible security models[5]. For example, in Figure 2 Charlie can read the data set because Alice has given him a proxy for her read capability. Preventing Alice from doing this would incentivise her to give Charlie a more powerful capability: the password for her user account. That would incur many security disadvantages: there would be no record of Charlie in the system, all of his activities would be logged as Alice's, he would gain more capabilities than required (e.g. append access), and his access could not be revoked separately to Alice's. Since capabilities are general and, by definition, *required* to perform tasks, they cannot be "worked around" and there would be nothing to gain from trying.

[5] For example, systems with access-control lists (ACLs) allow granting access in two ways: altering an ACL or sharing a password. The former is usually restricted, which incentivises the latter.

Capabilities also extend seamlessly to software, including automated tasks and third-party integrations. Figure 2 shows a backup script with its own capabilities, which creates short-lived proxies (i.e. temporary access URLs) to give a third-party storage service access for backing up the data. This prevents the service abusing its access, e.g. by spying on the latest data at any time, and also allows that service to perform the backup task however it deems appropriate: for example, by passing the capability along to a machine which is geographically closer. This "passing along" of capabilities not only allows greater flexibility and compatibility for integrating systems, compared to permission-based models like whitelists; it can also be more secure, by preventing tricky situations like the perennial "confused deputy".[6]

Differential Privacy

Some data sets cannot be shared without anonymising personal- or commercially-sensitive records. This must be performed in a principled way to prevent the elided data being reconstructed (or "de-anonymised") in the future; possibly by cross-referencing some other data set (Shmatikov, 2006). For this reason, the data platform provides methods for *differential privacy:* a mathematically-rigorous approach to jittering data, which obscures the contribution of each individual record to the total, bounds the amount of information revealed even across many queries, and provides plausible deniability that an entry exists in a data set (Nissim, 2017). There are two ways to introduce such jitter: it can be applied to the individual data entries (known as *local differential privacy*), which can be done before uploading and hence requires no trust in the data platform; or it can be applied when sharing summary statistics such as averages.

Data Integrity

All data in the platform is immutable: once specified, it cannot be changed (although new *versions* can be created, alongside the original, and access to old versions can be revoked).

[6] A common problem in permission-based security models. A privileged component (the "deputy") holds many permissions and performs certain actions on behalf of unprivileged components. Malicious components may perform unpermitted actions by "confusing" the deputy into doing it for them. An example is cross-site request forgery, where a browser is the deputy and its user's login sessions are exploited (OWASP Foundation, 2019).

Immutability simplifies distributed- and parallel-processing, but also provides integrity guarantees: if data has changed in any way, it can only be due to corruption or tampering, and can be flagged as such (e.g. for investigation and/or restoration from a backup).

Storing extra copies of data just to perform such comparisons would be costly. Instead, the platform calculates a *cryptographic hash* for all new data, which is a number of a certain length which acts like a “fingerprint” for that data [7]. If two datasets differ, even by just a single bit, they can have completely different hashes[8]. Hashes provide a fast, compact mechanism to detect whether data in the platform differs from its original value. Users can also calculate hashes themselves, to cross-check against the platform’s (this also ensures that uploads succeeded without error), and store their own copies to ensure that the platform never changes the hashes it reports. For data streams, old entries don’t have to be included in the hash calculation every time: since hashes act like “fingerprints” for data, they can be used *instead of* that data for subsequent hash calculations.[9] This allows even high-volume streams to have an accompanying stream of hashes to prove its integrity (known as a “Merkle tree” (USA Patent No. 4,309,569, 1982), or “blockchain” due to its use in Bitcoin (Crosby, 2016)).

[7] The platform uses the SHA256 algorithm, which produces hash values of 256 bits, or about 78 digits (National Institute of Standards and Technology, 2015).
[8] With 256 bits there are only 2^{256} possible hashes, so data bigger than 256 bits (32 bytes) will inevitably have “collisions”, where different data values result in the same hash. For *cryptographically secure* hash algorithms, like SHA256, these collusions are unpredictable, which makes it infeasible for an attacker to alter data whilst still producing the old hash (known as a *second-preimage attack* (Hoffman, 2005)).
[9] A stream x1, x2, x3, … can produce a stream of hashes h1=hash(x1, 0), h2=hash(x2, h1), h3=hash(x3, h2), … The next hash in the stream can be calculated from only the new datum and the previous hash (or 0, initially), yet that single hash guarantees the integrity of *the entire stream up to that point*. In practice, multiple entries will be hashed at once for efficiency; such “blocks”, linked by their hashes, forms a “blockchain”,

References

Alaba, F. A. (2017). Internet of Things security: A survey. *Journal of Network and Computer Applications*, 10-28.

Crosby, M. a. (2016). Blockchain technology: Beyond bitcoin. *Applied Innovation*, 71.

European Commission. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council. *Official Journal of the European Union*.

Fabry, R. S. (1974, July). Capability-Based Addressing. *17*(7), 403-412.

Hoffman, P. a. (2005). *Attacks on cryptographic hashes in internet protocols.* Internet Engineering Task Force.

Merkle, R. C. (1982). *USA Patent No. 4,309,569.*

National Institute of Standards and Technology. (2015). *Secure Hash Standard.*

Nissim, K. a. (2017). Differential privacy: A primer for a non-technical audience. *Privacy Law Scholars Conf.*

OWASP Foundation. (2019). *Cross Site Request Forgery (CSRF)*. Retrieved from owasp.org: https://owasp.org/www-community/attacks/csrf

Red Hat, Inc. (2018). CVE-2018-14665. cve.mitre.org.

Shmatikov, A. N. (2006). How To Break Anonymity of the Netflix Prize Dataset. *arXiv:cs/0610105*.