**EMCS2210: Privacy and Personal Data Protection**

Assignment: Review Org's Cloud Plan & Policy

Brian Russel Davis, [brian\_davis@brown.edu](mailto:brian_davis@brown.edu)

**Introduction**

I am in a unique position to give details on many organization’s cloud privacy policies. I was the Principal Engineer for Cloud Security and Infrastructure at NASA HQ for 1 year. I am currently the CTO of two small start-ups where I am solely responsible for the strategy and implementation. Lastly, I am the Senior Product Manager for Software and Security at PAX which is a cannabis device company. Each of these presents a unique opportunity for me to think about what privacy and data protection mean, and how to implement it based on the culture and size of the organization. That being said, instead of focusing on one organization, I am going to distill my learning and create an outline for a new organization that I have been incubating: jael.ai. jael.ai is AI chat/message service that helps women escape domestic violence. I’ve been invited to install the system for a population of 121k in Wales. As you can imagine a service like this will end up storing some of the most sensitive conversations: Women and men reporting abuse, calls for help, etc. I have used my NASA experience, my startup CTO experience and my experience with a product that is technically Federally illegal to craft ideas that are easy to implement, use bleeding-edge encryption and present a clear and scaleable separation of concerns around the functions and roles of facilitating data privacy.

**Create Privacy with Encryption**

Data can be encrypted in three ways ( three places ). Data at rest ( data sitting in a database, for example ), Data in transit ( TSL connections ) and data “in-use” ( experimental homomorphic encryption ). The last encryption method, although it is the least used and most experimental, can be the most impactful for privacy when the data is inside the organization. Using homomorphic encryption, and having a policy of never decrypting the user data protects the user data at levels most organization have never experienced. By never decrypting the data organization can now mitigate against:

1. Insider threats. A malicious insider can steal the data but it would be useless without the encryption key.
2. Mistakes in data transfer. Developers, in error, often use production data to debug and troubleshoot issues. Using
3. Breaches. Thieves often look for keys to decrypt data or wait for the moment when data is decrypted so it can be manipulated to steal it. If the data is never decrypted they never have this opportunity.

To facilitate an easy way of doing this in the cloud, I have created tools like Cloud Based SIDH. It’s basically a Python version of Supersingular isogeny Diffie–Hellman key exchange that works in an Amazon Serverless State Machine. This provides next level quantum computing proof key-exchange as a no-configure cloud service.

**Separation of Concerns Functional Model**

My separation of concerns functional model is all about applying the idea of DBAC to a staffing model. While the separation of concerns is a concept that usually applied to the creation of classes in an application. In my model the separation of concerns applies to the people who are allowed to access customer data in respect to their function in the organization. Product Managers design the application, Software Engineers create the application and Devops ( Development Operations ) Engineers deploy the application. In some very small organization these roles are all filled by the same person. However, I have observed that even in medium and large organizations there tends to be an aversion to separating the access based on job function. The dangers of not separating access based on job function is apparent however. When engineers have access to customer data we run the risk of :

1. Customer data being seem and abused by engineers
2. Customer data being loaded on a personal ( non-work computer ) with no security protocols.
3. Customer data being manipulated staging or testing environments.

The proper way to do handle this access it to define access based on the least amount of required access.

**Devops Engineers** have access to the keys to deploy the app but not to the source code.

**Software Engineers** have access to the source code but not the keys for production.

**Dummy Data** that mimics the size and complexity of production is use in development, testing and staging.

**Access logs** are automatically and irrevocably created to keep track of all transactions that access, view or change production records.

Implement a WAF ( Web Application Firewall ) to grant access to only

The main idea here is that people who need to have access, have access, those who don’t need it, don’t have access.

**Using AI and ML to Analyze Access Logs**

I mentioned in the previous section that all interactions with customer data needs to be logged. However, logging all these events creates mountains of data. Without a programmatic way to analyze the logs and create alerts based on an intricate set of rules that is constantly evolving, the access log will be ignored and serve no purpose. As NASA I created the cloud infrastructure and also configured the initial set of rules used to analyze the access logs.

**Ongoing Red Team Penetration Testing**

Penetration testing is probably one of the only ways to find faults in security BEFORE a real security event. By actively trying to break into the system and access user data, privacy can be protected before a breach happens.