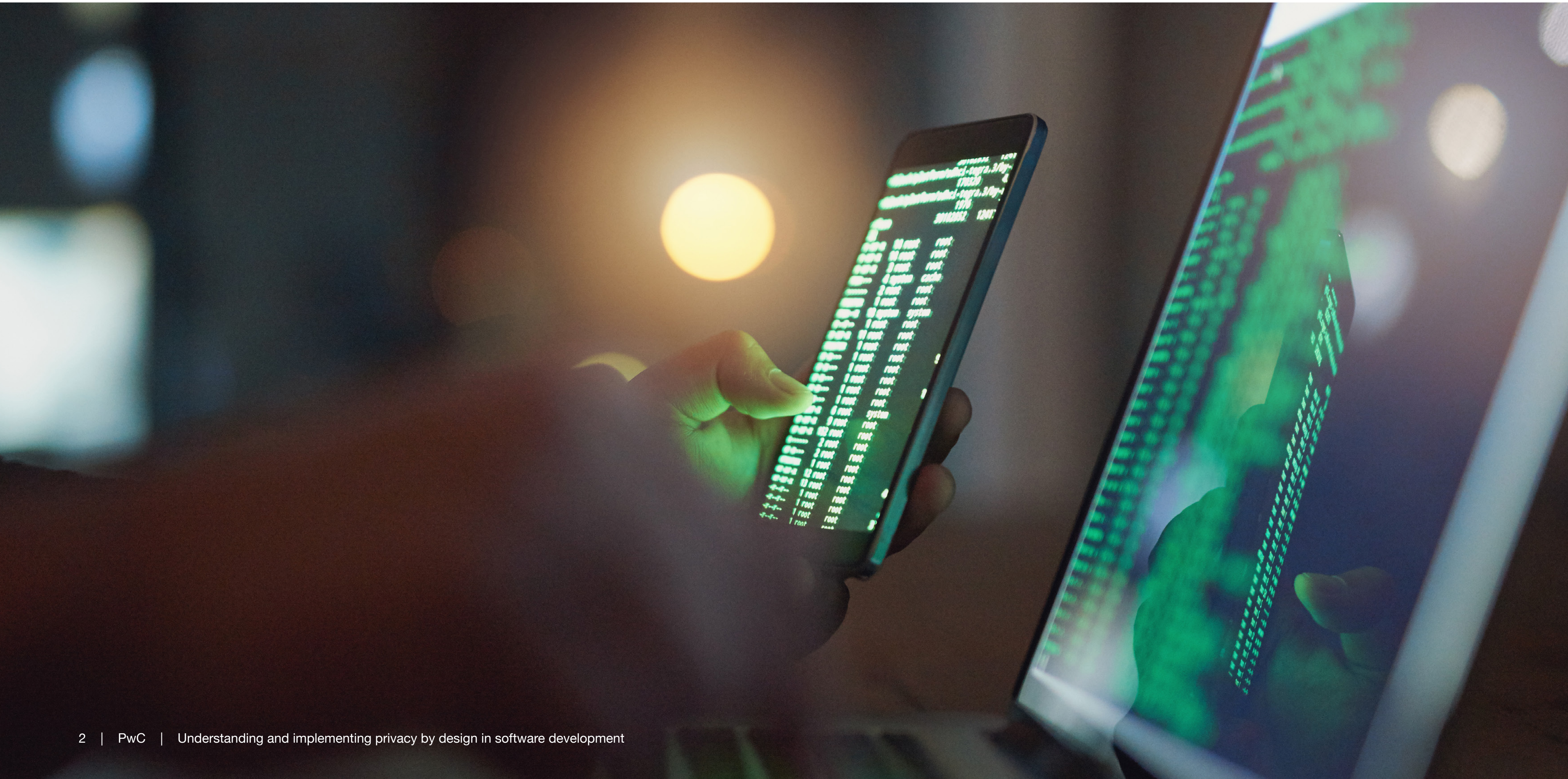
Understanding and implementing privacy by design in software development

March 2021

Today, countries and organisations fully recognise the ever-increasing value attached to personal data and the equally high risks associated with it. Software, be it online, mobile, desktop, or even IoT-based, has emerged as the most powerful and scalable way to personalise services and understand customer needs and, in turn, to gather vast amounts of user data.

The growing demand to monetise information and derive increasingly new benefits from it have led to concerns around unethical data collection and usage. Consequently, this landscape of extensive data collection and preservation has created an urgent need for preventing misuse of personal information. While companies need not stop data collection/creation altogether, they have to find a way to balance user privacy and business interests.

The concept of privacy by design (PbD) can help organisations strike the right balance.



What is PbD?

PbD is a proactive measure that aims to embed the concept of privacy in all data-processing activities right from the outset. Thus, it is not a reactive measure or countermeasure taken in response to a breach. Unlike privacy by default, PbD is not limited to publicly accessible software but also extends to internal IT systems, business practices and network elements. PbD requires that privacy be firmly established across the requirement assessment, design and operations of an organisation's tech culture and not limited to a product or software.

Seven principles of PbD

While the origins of the concept of privacy can be traced back to the 1970s (owing to the onset of mass surveillance), its foundation principles started to take shape in the 1990s and the late 2000s.[1] With the EU including data protection by design and default in its General Data Protection Regulation (GDPR) guidelines, PbD has become an important part of discussions on data privacy since 2012.

The seven PbD principles[2,3] are as follows:

1. Proactive and preventive approach to privacy risks –privacy is no longer a remedial measure.
2. Systems and procedures to be set to protect privacy of end users, by default.
3. Privacy should be fully integrated into the IT architecture and practices, making it a key component in the functionality of any system.
4. No trade-offs between privacy and functionalities.
5. Data should be protected and secure throughout its life cycle – from collection to deletion.
6. Technology and practices must be transparent and open to auditing/verification by relevant stakeholders.
7. A 'user-first and user-friendly' approach to enforce utmost privacy protection and notification standards.

The burning question – how does one go about incorporating these principles in software development?

PbD takes a design thinking approach to managing individual control over personal data flow, incorporating it into systems and technologies by default. People generally treat this as an 'engineering issue', while we believe it to be a **strategy** that needs to be implemented at the grass-roots level.

What, then, is a suitable strategy for an organisation to implement PbD-compliant software?

1. https://www.scandinavianlaw.se/pdf:/47-18.pdf Bennett, Colin J. (1992). Regulating privacy: Data protection and public policy in Europe and the United States. Ithaca: Cornell University Press
2. https://www.dzlp.mk/sites/default/files/doc_id_2.1.4.3.pdf
3 https://www.aepd.es/sites/default/files/2019-12/guia-privacidad-desde-diseno_en.pdf

01 Incorporate the seven principles during the scoping and planning phases

During the initial stages of every software development life cycle (SDLC), development teams should begin by defining and constraining requirements for personal data in view of the long-term strategic data objectives. The following guidelines can be useful for any scoping activity:

1. **Data minimisation and collection with a purpose:** When it comes to personal data, there should be a 'less is more' approach. The scoping team should clearly identify and define the data elements that would be gathered through the product. Each user story or epic pertaining to data collection should describe two key aspects: (a) the purpose behind the collection (b) use of the data collected. These aspects will ensure that developers and architects understand and appreciate the sensitivity and purpose of the information collected from users and enable them to incorporate the relevant safety measures in subsequent development phases.

2. **User consent:** Each step of data collection within the product should be preceded by a clear notification to the user and, in many cases, a clear option for the user to provide her/his consent. Such controls should be planned and embedded in tasks and sub-tasks for each epic. Also, as dictated by the foundation principles, this option of user consent should be set by default.

3. **Individual choice:** Project managers and architects should build user stories, solution architecture, user interface (UI) and other elements of the product around the tacit notion that individuals have the right to request for deletion or updating of their data at any stage.

02 Protect data throughout its life cycle and incorporate effective controls while developing the solution

Data collected by software is not only varied and expansive in nature, but may also have a very long life cycle. Therefore, the core principles of PbD should apply at each juncture of data generation, transformation and usage.

Secure data throughout its life cycle

a. **Data generation:** In most cases, a software's data is generated by users via UIs or interactions. Such models of data ingress should be thoroughly guarded. When entering data, the user should be able to toggle between showing and hiding their personal information on the UI itself, especially to prevent shoulder hacking. While such controls are typically used for sensitive information like passwords, they can easily be extended to other data points.

b. **Data in transit:** While the transport protocol nowadays is fairly secure against man-in-the-middle attacks and sniffers, an additional layer of encryption for data packets containing user information is a good privacy measure to adopt. Information such as phone numbers and email IDs typically navigate between the UI and the back-end server in the form of free-text strings. Encoding such elements can ensure enhanced privacy protection control.

c. **Data at rest:** Traditionally, data stored in a system has been the most vulnerable to privacy breaches, and naturally so – owing to the sheer number of personnel that have access to it. To minimise the risk of privacy breaches, while designing a software application, the solution architects should mechanise the encryption-at-rest protocols for both user data as well as documents. Using a simple symmetric encryption method, data can be encrypted while being stored, and decrypted when being readied for use. The encryption keys can be stored in secure locations away from the hosting environment itself.

d. **Data marked for deletion:** With the emergence of the principles of 'right to vanish' and 'right to be forgotten', any user may demand to have her/his information deleted from the software. This includes deletion or 'blanking' application logs, transaction records, chat logs and other data buckets containing titbits of user information. The 'forget' operation should be programmed into the architecture of the software and every such instance should be closely monitored by a specific set of individuals, including the data privacy/protection officer (DPO) of the organisation. Each request for deletion should be logged, signed off on, monitored and addressed in a secure and stipulated fashion.

Validate bulk or document uploads

It is a common practice to embed relevant controls in the UI elements to detect, process and validate user inputs. But development teams, at times, fail to perform similar checks during bulk upload operations. Due to the asynchronous nature of such operations and the need to minimise the wait time for the user, developers may choose a lower degree of validation during document uploads. Data uploaded through documents should undergo the same amount of scrutiny and validation as that uploaded through the UI, and appropriate measures should be embedded while developing the software.

Implement ethical data usage policies

Every organisation that processes or chooses to process user information should chalk out a detailed ethical data usage policy. The policy can have two subdivisions:

1. for public or customer viewing – ensuring transparency around the data usage policy
2. for staff – to be complied with by all staff working on a product or software.

The policy should contain guiding principles for responsible data handling, transparency, fairness and safeguarding user privacy. In addition, the staff version may contain penalty clauses, zero tolerance and permitted legal action in case of privacy breaches.

Further, staff working in software development should be trained and certified on data ethics and privacy periodically and at the time of induction into the organisation.

Check the network elements, audit trails and trackers

Tertiary data and metadata form a major component of the information collected by software. While such data cannot be considered as private data directly, when linked with additional information, it can potentially disclose a person's identity. Therefore, while developing software, architects and developers should recognise the need and risks associated with collection of IP addresses, location and other metadata.

The following considerations must be taken into account while collecting metadata:

1. **Session-oriented or transient browser-based data:** Certain metadata can be stored in the form of cookies or session variables to make it available only to the browser or user session. Even if the metadata is stored at the back end, once the user is logged out (or over a period of time), the cookie or session variable can be removed.

2. **Metadata separation:** If metadata is required for tracking and analytics purposes, it can be stored in a separate location from the user/application data. This approach will introduce a firewall between the two datasets, making it difficult for any intruder/aggressor to establish linkages.

Include the DPO in the initial approval hierarchy

Typically, organisations and development teams seek approval and advice from the DPO towards the end of a product's SDLC. And such approvals are, in most cases, nominal or formal. But in order to comply with PbD, it is imperative for product owners to seek advice and guidance from the DPO during the planning and initial development phases. While each story or epic needn't be authorised by the officer, the architecture, security measures and overall design of the software need to be vetted and approved by the DPO.

```
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

    #selection at the end -add back the deselected mirror modifier obje
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the active ob
    #mirror_ob.select = 0
```

03 Monitor the product's data sinks

Involuntary data collection

Often, software tends to collect information that either (a) it is not meant to collect, or (b) that is in addition to the information that it is tasked to collect from the user. For instance, while a simple chatbot generally tries to help out a user based on the limited information provided, it doesn't restrict a user from entering her/his own personal information in the chat window. If a user chooses to enter a phone number or has understood a question wrongly and enters unsolicited information (unintentionally or mistakenly), the chatbot would store the same in its database.

Such unforeseen scenarios are too many to provision for during the development phase. Hence, it is wise to perform regular and periodic checks on the data sinks of the software. These checks can help in:

1. detecting and formulating deterrents to unwarranted data ingestion
2. redefining effective input controls, such as changing a particular question to achieve a more accurate and suitable response from the user.

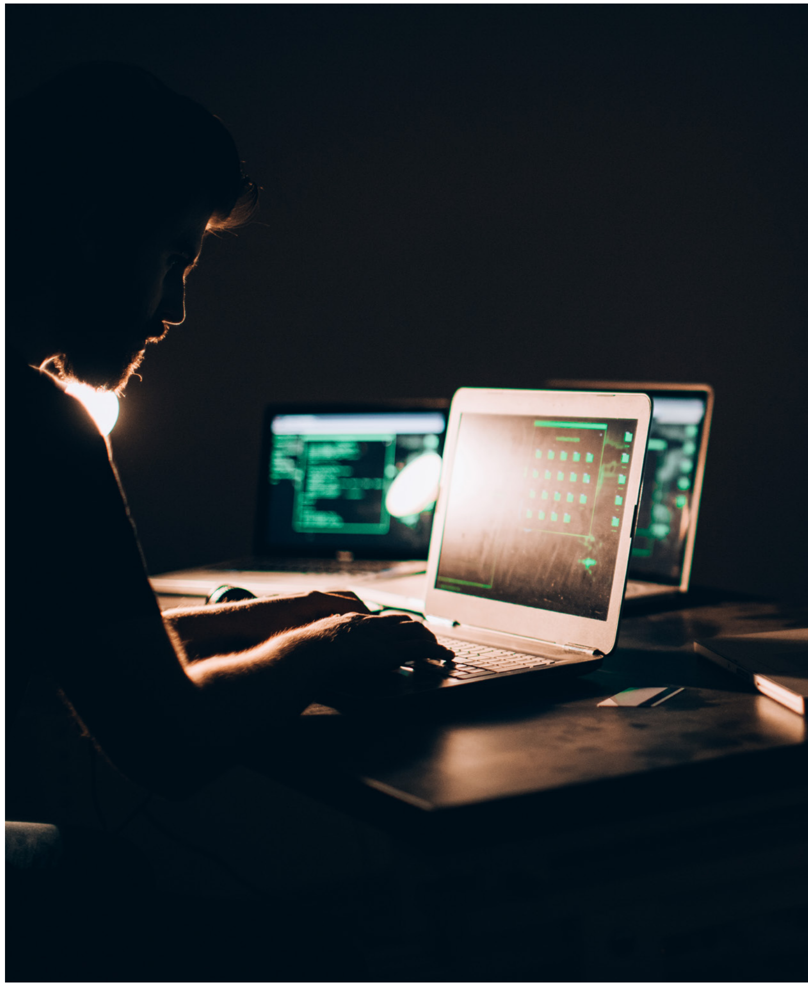
Data lineage and usage reports

During the life cycle of software, data requirements may change periodically or eventually. Certain new data elements may be added, while a few old ones may go unused or become obsolete in day-to-day operations. Periodic audits of the data dictionary vis-à-vis intent of use may yield data points that are not critical to the functioning of the application, and can therefore be eliminated from the collection process. This methodology conforms to the less is more approach of data collection to ensure privacy.

Controller-processor architecture

Certain software, especially that internal to organisations, forms a part of an elaborate data grid or lake. Several applications or 'downstream' systems may rely on or use the data provided to them. In such cases, software that contains private information should adopt a strict data processing 'contract' with consumer mechanisms. There should be a mechanism, similar to that of a controller-processor contract between two parties, set in place to seek and receive access to information stored in sensitive systems. A data processing template and periodic data audit guidelines for the processor should be defined before providing access to the controller's data.

Calls from application programming interfaces (APIs) and microservices should be validated and authorised before addressing them, and the data exchanged should be restricted to the attributes set forth in the data processing contracts between the two systems.



04 Duly scrutinise change requests, versions and modifications

Each change request should go through the same rigorous mechanism outlined in the aforementioned sections. User stories and epics should contain clearly defined new data parameters (if any) and their usage. Storage and protection policies of additional data should be discussed, documented and authorised by relevant stakeholders. Changes in UI should comply with the seven principles. Further, all major change requests that entail feature modification or data collection should be approved by the DPO.

05 Provide privacy control back to users

Inform users about data collected and obtain consent

Before collecting information, inform users about the following aspects:

- why or for what purpose their data is being collected
- how their data is being stored and protected
- for how long the organisation or software will store the data
- advice on what type of data should be entered in the user input fields, mitigating unsolicited data collection
- user consent to be acquired before form submission or page loads.

These pointers can be implemented at the form level (before loading or submitting) and on a separate screen (form or page) prior to the data collection template.

Create a user-friendly privacy policy and not a long-drawn legal document

Explaining privacy policies, measures and consent or notifying the end user about them should not warrant a ‘wall of text’, nor should the user be bombarded with legal/technical jargon. As dictated by the seven principles, the privacy policy and related documentation should be designed in a user-centric fashion.

The UI for the policy should:

- be clean
- be broken down into sections
- contain images and call-outs to explain the policy visually to users
- not skimp on the content itself.

Update users about changes to the policy

Data privacy laws and norms are constantly changing, and hence, a software application’s privacy policy would require additional clauses or modifications. Such changes can be compiled in an email or a document and be sent to relevant users. This need not be a manual process, and a mechanism within the software can be used to enable such communication to the user. Further, push notifications can be used to alert users about policy changes and redirect them to a document or webpage containing the modifications.

Each change to the privacy policy should be followed by seeking of consent from the user, before submission of data or transacting with the software.

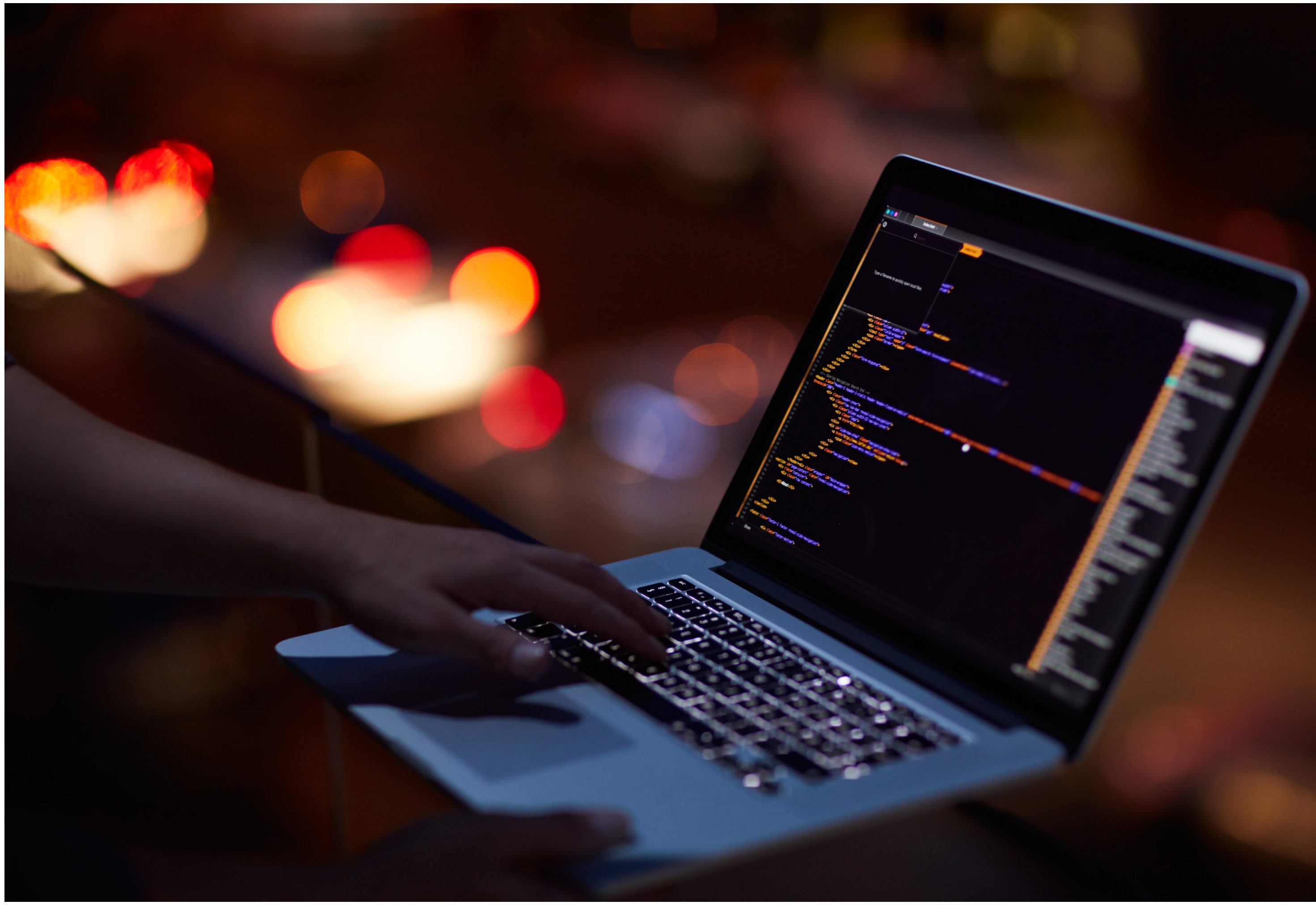
06 Run analytics to identify data leeches

Product owners should monitor the data pipeline (data at rest and in transit) to identify programs, trackers and bots that are 'feeding' on the information contained in the software. While encryption can protect the data from being misused, periodically analysing data traffic can serve as an added layer of privacy protection.

07 Secure all decommissioning and migration processes

Decommissioning and migration protocols should be defined within the software life cycle to streamline the 'sundowning' process of a software application or version. From a data privacy perspective, this is a critical juncture and should comprise the following:

- A secure methodology should be defined within the software to migrate, retain or remove user data. The data should be packaged for transport or deletion and should be encrypted. The migration/deletion process should be monitored by critical stakeholders and relevant evidence (logs, documentation and screenshots) should be gathered to catalogue the various steps of the procedure for future reference/audits.
- Users should be notified of the process, especially during decommissioning; and they should be reminded of the organisation's data retention policies. They may then choose to have their information deleted from the software.
- The data retention policy should also comply with the PbD principles and minimise the amount of data collected to the extent possible, maintaining a balance between user privacy and policies.
- While shutting down or reusing the hosting infrastructure, adequate data erasing measures should be employed to preclude information extraction using 'data carving'.
- The DPO should be notified of the process and should be part of the approval hierarchy within the protocol.



08 Perform due diligence before implementing third-party components/technologies

All third-party or open source components, modules, APIs and libraries should be analysed for vulnerabilities before being implemented in the code.

Periodic static code analysis and component analysis should be performed to identify deprecated modules which need to be updated or removed from the software.

Further, communication between third-party components via tokens should be secured and data shared between them should comply with the aforementioned controller-processor architecture.

09 Minimise data access to support teams

Provide obfuscation or anonymisation controls for admins

While designing software, the role of support personnel should be critically considered. Support teams such as infra teams, database admins, network support and DevOps support should not be provided access to user data, either from the front end or in the data layer. Obfuscation and anonymisation controls can be used for the log-ins provided to the support teams. Even if a task requires the support personnel to peek into the data, private information can be masked on the UI.

Further, support personnel should have minimal access to decryption keys. Only a few members (managers and above) should be entrusted with the responsibility of maintaining encryption codes and other key elements on the production infrastructure.

Keep developers' and admins' access to environments separate

It is imperative to restrict production access to developers. While the development team can provide guidance to the support personnel for troubleshooting bugs, they should not be allowed direct access to the production environment. This measure ensures that no single team member has 'keys to all doors' of a software application.

Another way to secure production data is for the support team to change the encryption keys, environment variables, connection strings and other relevant elements after going live with the software. This mitigates privacy breaches via backdoors that are typically used by internal aggressors.

Way forward

PbD is not a new concept in itself, and leverages established principles. In essence, it brings in a mindset shift towards applying privacy considerations at each step of software development. It can be used both as a preventive and detective control for data protection. From the software development perspective, it protects both the developer and the organisation from potential liabilities as it helps organisations to design products by complying with privacy standards and regulations.

For any organisation to walk the path of PbD, the following steps can be of help:

- **Assess your current PbD processes and operations:** Do you have a PbD policy? Do you have a guideline document in your organisation? Do you have any guideline documents/SOPs within your organisation that incorporate PbD principles?
- **Develop privacy guidelines:** These policies should be specifically created for application owners and developers – e.g. are there specific sets of guidelines for project managers, application developers and data custodians?
- **Know the data:** This includes data processed and collected by the software product, tools and platform. Have you classified the data collected, including metadata? If the software is off the shelf, have you assessed it from the PbD perspective? Do you know how much and what kind of derived data is produced by such software?
- **Take help from privacy experts:** Are the risk and harm that could arise out of this processing being assessed by a privacy expert who understands the law and is trained in engineering software?
- **Build accountability:** Is privacy a default setting within the software business? Do you have a privacy governance structure and board involved? Are software engineering privacy risks tracked and taken to closure?
- **Create awareness and foster a privacy culture:** How often are your teams trained on privacy? Are your engineers sensitised to the importance of user privilege? The focus should be not just on compliance but also on the 'individual' and 'doing the right thing'.

About PwC

Contact us

Dhritimaan Shukla
Partner and Privacy Leader
PwC India
Mobile: +91 98990 38326
dhritimaan.shukla@pwc.com

Karthik Addanki
Director, Forensics
PwC India
Mobile: +91 77020 99600
karthik.venkata@pwc.com

Sonali Saraswat
Associate Director, Privacy
PwC India
Mobile: +91 96207 01515
sonali.saraswat@pwc.com

Acknowledgements

This article has been researched and authored by **Sonali Saraswat**, **Karthik Venkata** and **Dhritimaan Shukla**.

pwc.in

KS/March 2021-M&C9878