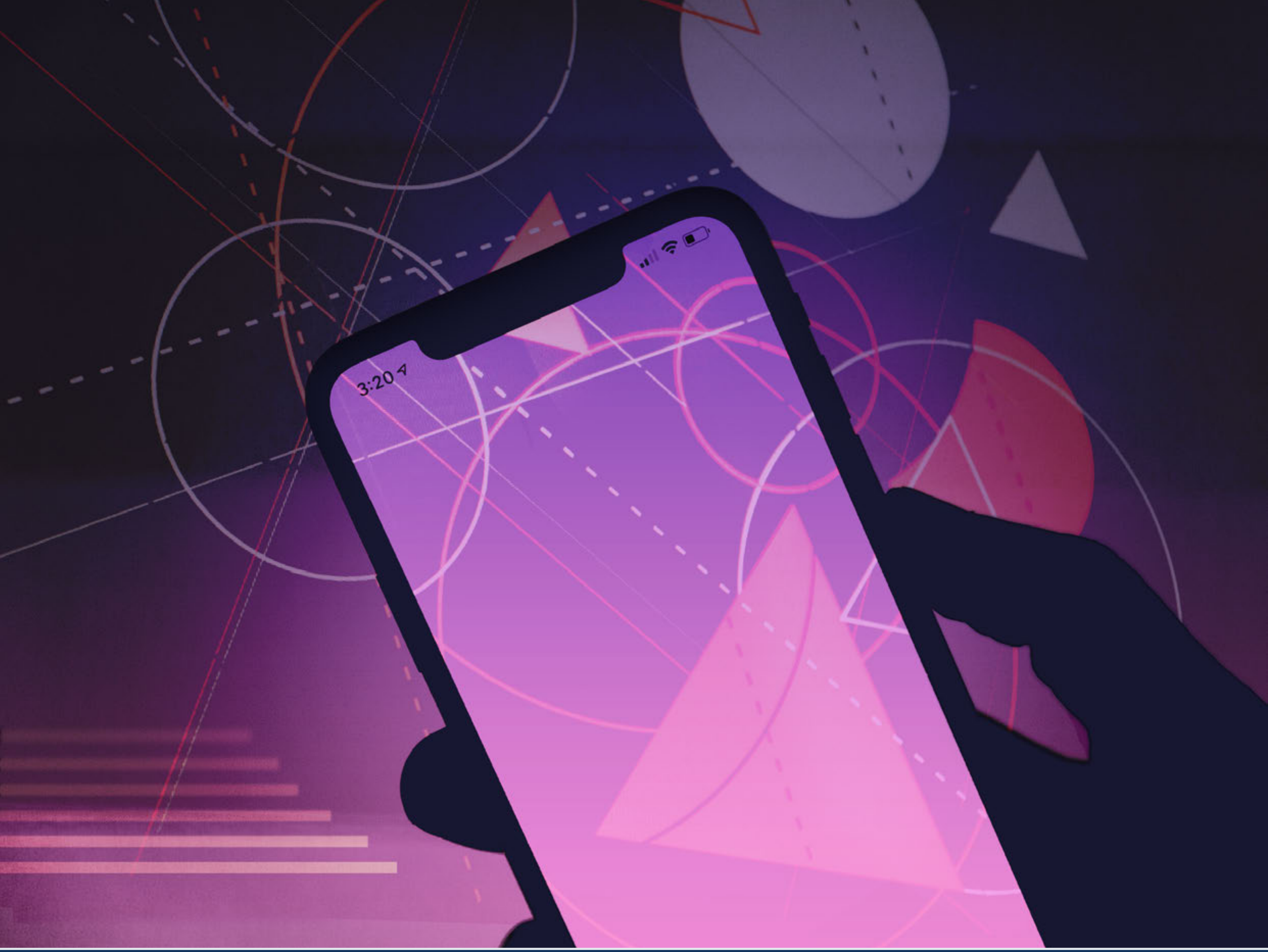


Incorporating Mobile App Security into the Development Lifecycle Without Friction

GUARDSQUARE
Mobile application protection

Incorporating Mobile App Security into the Development Lifecycle Without Friction

Table of contents

The secure software development lifecycle 1

1. Inception........ 3

2. Requirements analysis........ 4

3. Architecture & design........ 5

Mobile app privacy design........ 6

4. Development........ 7

5. Testing.. 9

6. Deployment........ 10

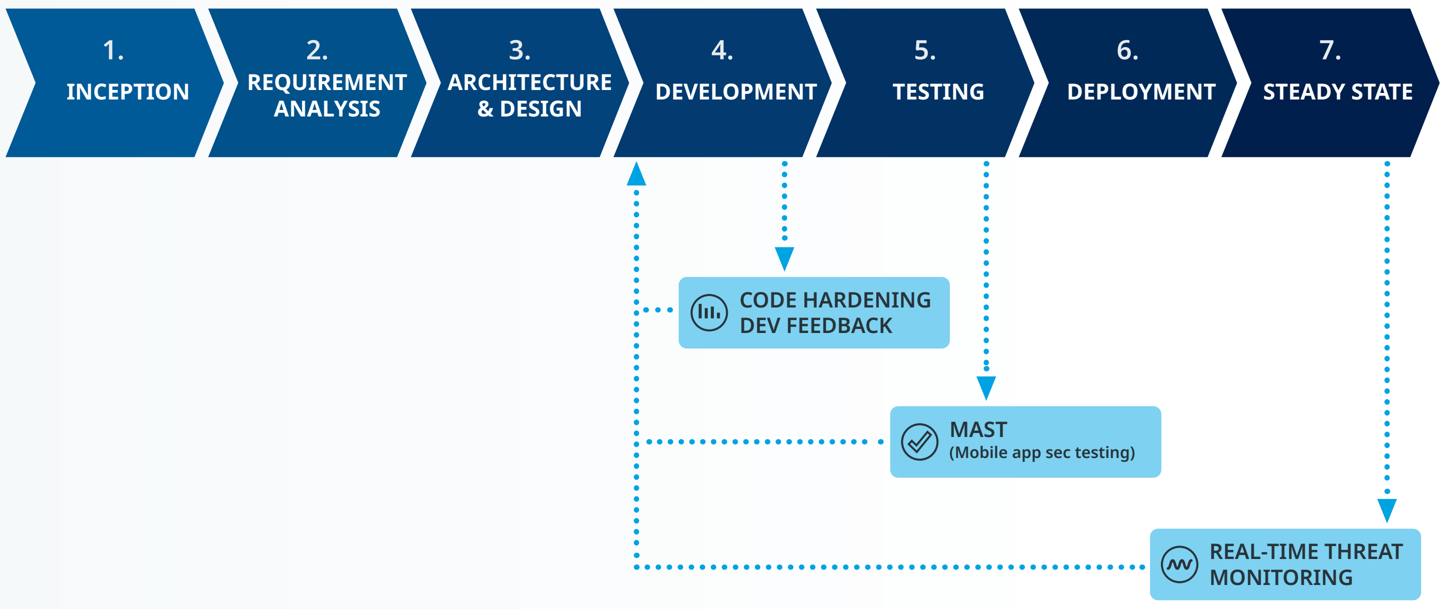
7. Steady state........ 11

Security tools for the SSDLC 12

GUARDSQUARE
Mobile application protection

The secure software development lifecycle

Many organizations are starting to consider security an integral part of the development lifecycle. They follow the concept of a secure software development lifecycle (SSDLC). The phases of the SSDLC vary, but here's the model we will delve into today.



The mobile application development process is iterative. Developers often use one of the many variants of the Agile development methodology, such as Scrum or Lean Software Development. Teams are focused on continuous improvement, since mobile apps are consistently updated for many reasons, including:

- Bug fixes
- New features
- Security updates
- Operating system updates
- And more.

But, often within mobile app development, security is either not considered at all, or is only incorporated very late in the development lifecycle. In most cases, security is reactive, only after a breach has occurred. The reason? Many development teams are racing to market with their apps. Being “first” can often make or break an app publisher.

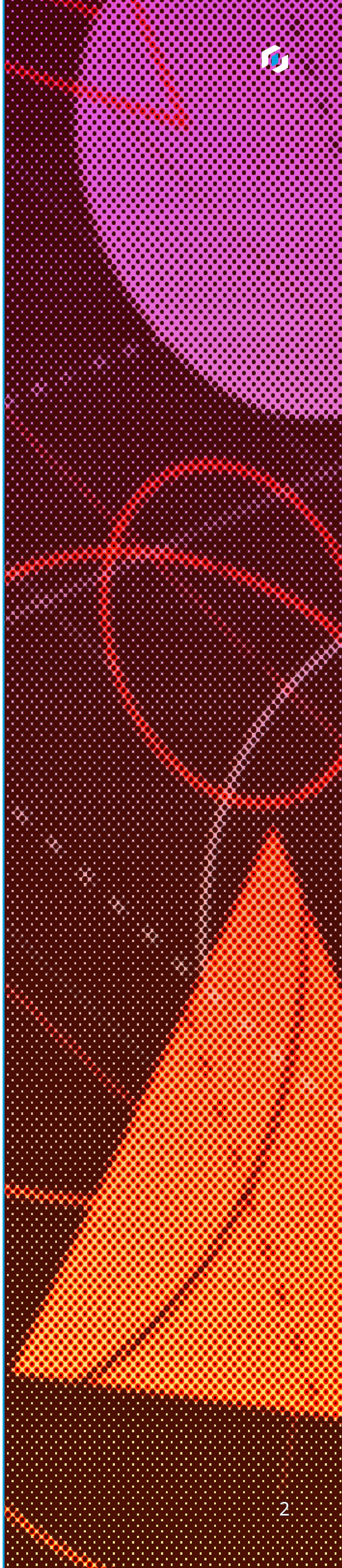
Unfortunately, there’s a perception that security can slow mobile development teams down. The reality is, many development teams lack security expertise. In fact, 82% of organizations say there’s a shortage of cybersecurity skills on their teams. Not knowing how to incorporate security can be a top cause of slowdowns.

Consider this: it is more expensive to find bugs after deployment, versus finding them earlier in the lifecycle through the quality assurance (QA) process. Similarly, security is inherently more expensive if organizations find vulnerabilities after deployment, rather than incorporated into the lifecycle via security assurance (SA). If security isn’t incorporated early and often, major consequences could happen, such as:

- Financial loss
- IP theft
- Data loss (company or customer)
- Reputational damage
- And more.

Luckily, there are proven ways to include security within the iterative development process. Contrary to popular belief, secure development can still happen on compressed timelines. As mobile teams iterate, they go through all the phases of the SSDLC. The cycle repeats for each feature, and security and development teams evaluate the application each time (although this can happen within a matter of hours or days for updates).

This eBook will show you how and where to seamlessly integrate security throughout the entire development lifecycle, without having to slow app development teams down. Let’s take a look at each phase of the SSDLC as it applies to mobile development.



1. Inception

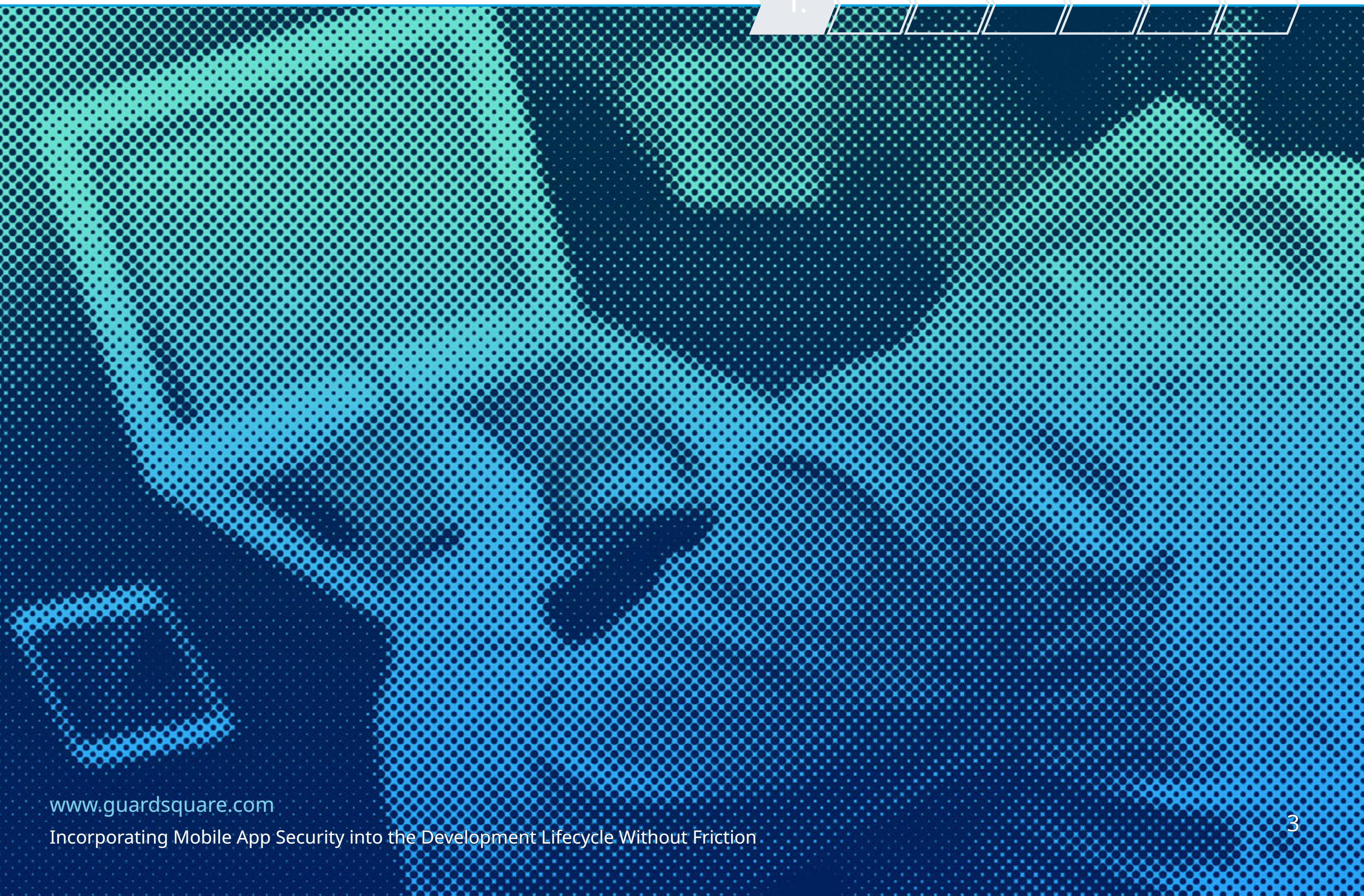
In this first phase, the app is being conceptualized and planned. At this point, teams need to bring security into the SDLC. Security awareness training can help developers hone their skills. It's important to note that not all applications require the same level of security. Training sessions should be adjusted based on these requirements.

For example, some applications will be subject to internal, localized or industry-specific security and compliance regulations. These applications may require enhanced security and risk mitigation measures.

Possible regulations could include:

- Data privacy regulations (GDPR in Europe, CCPA and other state-specific regulations in the U.S., PIPEDA in Canada, etc.)
- PSD2, PCI, or GLBA for financial apps
- HIPAA for healthcare
- SOC 2 for the organization's internal information security
- SOX for internal financial reporting
- And many more.

In the requirements analysis step, the team digs into these types of specifics.



2. Requirements analysis

In this stage, teams come up with a requirements, or list of what should be included in the app. These should include:

- **Business Requirements:** High-level statements of objectives, goals, and needs for the organization.
- **Stakeholder Requirements:** Often based on research, these dictate what the user might expect from the app.
- **Solution Requirements:** These are characteristics of the application, including how it will work for users and meet business needs.
    - **Functional:** Features that will impact how a user interacts with the application, such as external user interfaces (UI) and user experience (UX), authentication, transaction functionality, and more.
    - **Nonfunctional:** App attributes that aren't directly seen by the user, such as security, reliability, performance, maintainability, scalability, and usability.

Specific to security requirements, in this phase, teams start working on threat modeling and risk modeling. In other words, they look at the application and its third-party dependencies to evaluate risk. Some teams develop their own in-house threat models, and reuse them for different products. Others rely on industry standard threat modeling frameworks, such as:

- Microsoft threat modeling frameworks and tools.
- Trike, an open source methodology and tool.
- OWASP's PASTA methodology, or Process for Attack Simulation and Threat Analysis.

In the risk analysis and threat modeling process, some key questions to consider include:

- What compliance standards does the organization need to meet, if any?
- How will the app connect to company servers?
- Will the app store sensitive information from customers or the company?
- How much valuable intellectual property (which gives a competitive edge) will be involved in developing the app?
- What third party libraries or other services does the app rely on? What are the security risks associated with these third-parties?

3. Architecture & design

During this phase, the team will be focused on both the technical design and the UI for the app. People who are doing this design should be trained in security. Most technical decisions should consider security and user privacy. Often there's a tradeoff between whether users or the organization assume more risk.

As an example, let's consider an app that's sharing a user's location with friends. This could happen in two ways:

- **Broadcast location information:** The friend's app connects if the user's location is close. If someone pretends to be a user's friend, this can be a security issue. This decision increases the user's risk.
- **Send data to the company's server:** The app only sends the location and connects if two verified friends are close to each other. This decision limits the distribution of data externally. However, it makes the company's server aware of location. This increases the company's risk.

Beyond technical decisions like these, UI design also must consider security. For example, a social networking app should design features that make it clear when users grant access to a third party. They should also enable the user to easily revoke that access.

Usually, these types of conceptual decisions are made by a CISO, application security team, or security architects, rather than developers. In larger companies, security communicates standards (such as OWASP MASVS), regulations and policies. They detail ways to roll out policies using security checklists. The development team's responsibility is to decide how they'll conform to these requirements.

Finally, teams perform a security design review, which includes:

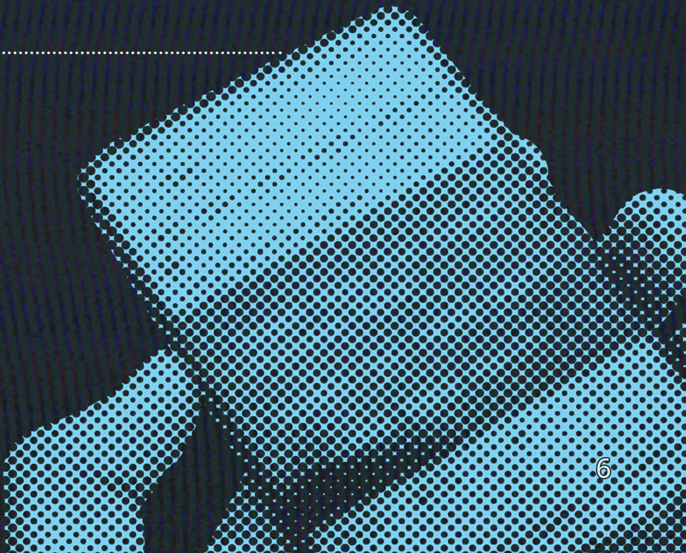
- **Identifying all security assets and their lifetime within the application.** This includes encryption keys, sensitive data, resources, files, and more.
- **Identifying the attack surface of the application.** Considerations include vulnerable code/data locations, unprotected aspects of the communication protocol, and more.
- **Creating an "attack tree."** This is a "what if" chain of attack scenarios for hackers attempting to gain access to security assets.
- **Assessing the application's security vulnerabilities.** From there, the team will recommend mitigations to address vulnerabilities, or explicitly sign off on accepting them. Security is never perfect. It does come at a cost. But, the organization must make sure these decisions are conscious.

Mobile app privacy design

GSMA is a global industry organization that released standards for user privacy in mobile applications. Designing for privacy is important for mobile applications that store sensitive customer data, or operate in regions with strict consumer data privacy regulations.

Key considerations within GSMA's standards include:

- **Gaining active consent from users wherever possible**
- **Making users aware of app updates**
- **Setting data retention and deletion timelines**
- **Protecting children and minors**
- **Informing users of advertising features**
- **And more**



4. Development

First, organizations must prepare for the development phase by making sure secure coding training is mandatory for all developers. During development, secure coding best practices should be followed. Some secure coding basics, according to Carnegie Mellon's CERT, include:

- **Validate input.**
  Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities.
- **Heed compiler warnings.**
  Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws.
- **Architect and design for security policies.**
  Design software to implement and enforce security policies.
- **Keep it simple.**
  Keep the design as simple and small as possible, as complex designs increase the likelihood that errors that could lead to security issues will be made in their implementation, configuration, and use.
- **Use effective quality assurance techniques.**
  Good quality assurance (QA) techniques, such as fuzzing, penetration testing, and source code audits, can help identify and eliminate vulnerabilities.
- **Define security requirements.**
  Identify and document security requirements early in the development lifecycle and make sure that subsequent versions are reviewed for compliance.

Code hardening is also an important part of the development process. Code hardening protects Android and iOS applications and libraries from reverse-engineering and exploitation. It is a minimally invasive process that can be automated, without slowing the development team (or the app itself) down.

Common code hardening techniques include:

Obfuscation:

Rendering code illegible without affecting its functionality. The techniques used to obscure code in this manner vary considerably. They may include:

- replacement of readable names in the code by difficult to decipher alternatives **(name obfuscation)**
- modification of the logical structure of the code to make it less predictable and traceable **(control flow obfuscation)**
- conversion of simple arithmetic and logical expressions into complex equivalents **(arithmetic obfuscation)**.

Encryption:

Ensures the code of the application and the data it contains cannot be accessed while the application is at rest. The encrypted code is decrypted on-the-fly when the application is executed, guaranteeing that it functions as intended. To be effective, the encryption must be applied in various layers. Essential encryption techniques include:

- string encryption
- class encryption
- asset encryption
- resource encryption.

At this point, teams can also automate anti-tampering protection, or runtime application self-protection (RASP). RASP enables Android and iOS applications to defend themselves against analysis at runtime and live attacks. The various RASP mechanisms monitor the integrity of the applications and the environment in which they are running. When a threat is detected, the applications react in a pre-programmed manner. The possible reactions range from the display of a security notification to the termination of the user session and/or the application. In addition, RASP ensures the communication between mobile application and server is secure.

Security processes are important to conduct continuously, since development on a mobile app never stops. Luckily, both code hardening and RASP can be automated within the CI/CD pipeline and applied regularly (e.g. on a nightly basis).

5. Testing

Mobile app testing goes through every step the user might take within the app. This process ensures that nothing is broken, for example:

- An image doesn't load
- An external link doesn't open
- An interaction or series of interactions cause the app to freeze

Just as teams do QA testing on their features at the end of a build, they also should do SA. This process usually includes two major steps:

- **Pentesting:**

  Otherwise known as penetration testing, the objective is to penetrate the application or network security defenses to find vulnerabilities. This can either be done internally by Red Teams or by external platforms or service providers. Pentesting should be done both on obfuscated and unobfuscated code.

- **Automated security testing:**

  This type of testing scans the built application for sensitive keys, strings, and more. Two types of testing, SAST and DAST, are typically used together.

  - SAST, or Static Application Security Testing (also known as "white box testing") allows developers to find security vulnerabilities in the mobile app's source code earlier in the SDLC. It also ensures developers are confirming to coding guidelines and standards without executing the underlying code.

  - DAST, or Dynamic Application Security Testing (also known as "black box" testing) can find security vulnerabilities and weaknesses in a running mobile application by employing fault injection techniques on an app. DAST can also identify runtime problems such as authentication and server configuration issues, and more.

Once the app has passed security testing, it's on to deployment.

5.

6. Deployment

In the deployment phase, developers submit the app (whether net-new or an update) to the app store. The app must comply with operating system-level requirements that are designed to protect the end user. Both Apple and Google make their policies for developers public, as it is in each marketplace's best interest to ensure the integrity of the apps within it.

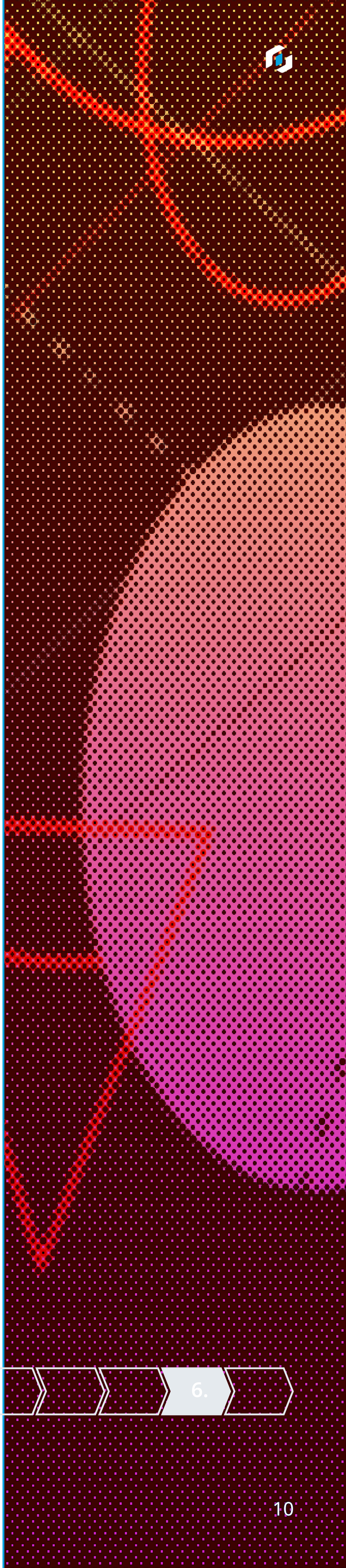
Here, most of the security work will be related to the server side. For most apps, which combine mobile apps and cloud services, there are two deployments going on.

- **Deploying the app to the app store:**

  One of the most important security considerations in this phase is keeping the developer's code signing certificate safe. That way no one can pretend to be that developer.

- **Deploying to the server:**

  Security in this phase involves ensuring that the server system is properly configured and patched, and that server passwords kept safe. Misconfigured cloud servers can expose sensitive data and leave back doors open to potential attacks.



7. Steady state

Just because an application has been published to an app marketplace and downloaded by the user doesn't mean the development process stops. New app updates and security patches should be a regular part of the application lifecycle. This is a process called patch and configuration management. In addition, as new operating system versions emerge, apps should be updated accordingly.

After an Android or iOS app is released, security teams and developers often lack visibility into the most common attack vectors and vulnerable parts of their code. Real-time threat monitoring can help detect threats to both the app and environment. These can include common tactics used by hackers to weaken an app's or environment's security, such as jailbreaking, rooting or hooking. Threat monitoring can also identify suspicious users. From there, teams can adapt their security configurations to protect apps against suspicious activity and users.

Finally, more security assurance testing should be applied at this stage to detect further points of vulnerability in the app. From there, teams should create strategies to respond to incidents (i.e. incident response plans) or potential points of vulnerability. This should involve increasing layered mobile application security defenses (using a combination of code hardening, RASP, and real-time threat monitoring).

Security tools for the SSDLC

Tactically speaking, even resource-strapped teams can make security frictionless across the entire development lifecycle – if they have the right tools integrated into their CI/CD pipeline. These tools can speed up the process of integrating security, so teams do not have to sacrifice security for time-to-market.

Here are just a few of the technologies that can be applied to different phases of the SSDLC.

Development

- **Secure coding assistants:** These solutions help developers and app security specialists build their secure coding skills, get real-time advice when they need it, and monitor skills development over time.
- **Code hardening:** Code hardening protects APKs/IPAs and SDKs for Android and iOS apps from reverse-engineering and hacking by using obfuscation and encryption. These techniques make the code mainly illegible and inaccessible to hackers. Hardened code is resistant to both automated and manual analysis.
- **Runtime application self-protection (RASP):** RASP protects the application against analysis at runtime and live attacks. When a threat is detected, the application reacts in a pre-programmed manner, such as displaying a security notification or terminating a user session. RASP and code hardening are complementary approaches.

Testing

- **Mobile application security testing (MAST)**: These tools help teams that may lack the security resources or skills test their mobile applications to find and fix security vulnerabilities on the client-side, server-side and within third-party libraries.
- **Pentesting tools:** While much of the pentesting process is manual, a pentester may rely on disassemblers or hooking frameworks to try to circumvent the app's defenses.

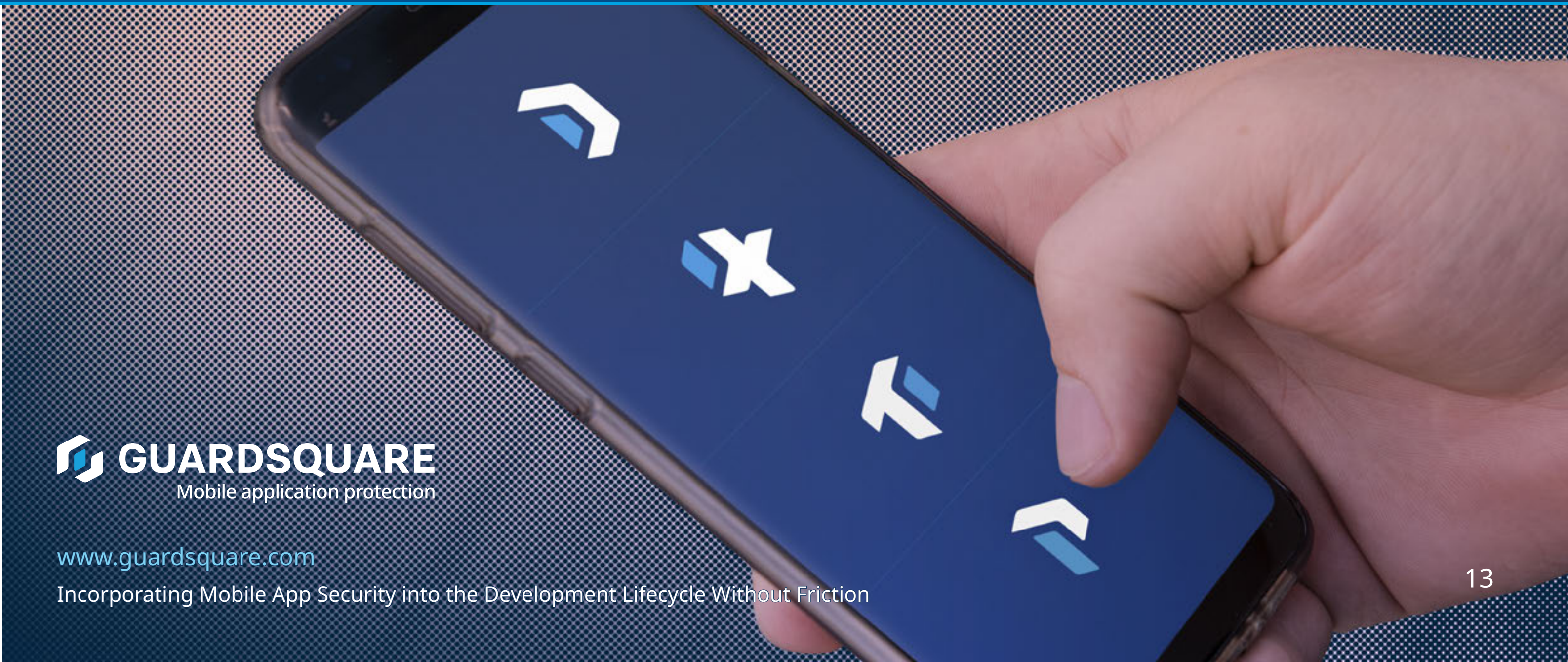
Steady State

- **Real-time threat monitoring:** After an app is released, security teams often lack visibility into the most common attack vectors and vulnerable parts of their code until it's too late. Monitoring threats in real time can help teams adapt their security configurations and protect apps against suspicious activity and malicious users.
- **User-facing tools:** For organizations that rely on mobile devices, there are end user-facing security tools that aren't necessarily a part of the development lifecycle, but can protect against potential threats. These include, but are not limited to:
    - Mobile application management (MAM): These software tools and services provision and control access to internally developed and commercially available mobile applications. These can be useful within organizations that follow bring your own device (BYOD) policies at work.
    - Mobile device security solutions: These solutions protect employees' devices themselves when they're running on corporate networks, and can include (but are not limited to):
        - Endpoint security: monitors files and processes on devices that are connected to a network, scanning for malicious behavior.
        - Email security: detects, blocks and remediates threats to email, preventing data loss via mobile device.
        - Secure gateways: cloud security that operates at the DNS and IP layer, to prevent phishing, malware, ransomware and more.
        - Cloud security access broker (CASB): a gateway between on-premises infrastructure and cloud applications.

A combination of the right knowledge and tools can help mobile app development and security teams apply the security best-practices across every iteration in their development lifecycle.

Want to find out more about integrating security without slowing down?

Learn more >



GUARDSQUARE
Mobile application protection

www.guardsquare.com