Android Apps Security

Mitigate Hacking Attacks and Security Breaches

—

Second Edition

—

Sheran Gunasekera

Apress®

Android Apps Security

Mitigate Hacking Attacks and Security Breaches

Second Edition

Sheran Gunasekera

Apress®

Android Apps Security: Mitigate Hacking Attacks and Security Breaches

Sheran Gunasekera
Singapore, Singapore

ISBN-13 (pbk): 978-1-4842-1681-1 ISBN-13 (electronic): 978-1-4842-1682-8
https://doi.org/10.1007/978-1-4842-1682-8

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484216811. For more detailed information, please visit http://www.apress.com/source-code.

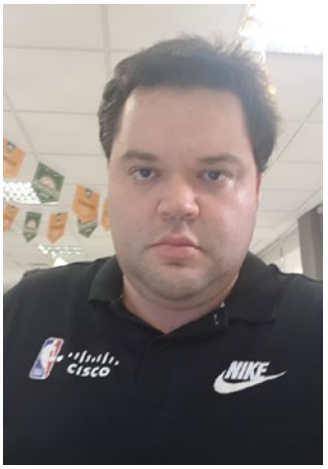
Printed on acid-free paper

Table of Contents

About the Author

Sheran Gunasekera is a security researcher and software developer. He is cofounder and Director of Research for Madison Technologies, a product development company in Singapore, where he advises the in-house engineering team in both personal computer and mobile device security. He is also one of the co-founders of RedStorm, an Information Security Bug Bounty Platform. Sheran's foray into mobile security began in 2009 when he started with BlackBerry security research. Since then, he has been in leadership roles in both engineering and security at several startups in Asia. He publishes research that he has done on his blog at `https://sheran.blog`.

About the Technical Reviewer



Thiago Magalhaes is a professional with more than a decade of experience in the information technology area with a wide experience in designing and dealing with large-scale distributed production environments. He is also a skilled DevOps engineer supporting, automating, and optimizing mission-critical deployments. A Linux lover by nature, he has a broad area of responsibility focused on security, high availability, reliability, and troubleshooting. He has hands-on experience in the administration and maintenance of application services like DNS, OpenLDAP, Samba, Mail, HTTP, Apache Tomcat, Squid, DHCP, SMTP, FTP, IMAP, NIS, and NFS. Last but not least, he is responsible for the ongoing maintenance, growth, and development of large-scale servers running Unix. In his spare time, he loves cooking for his friends and watching Netflix.

Acknowledgments

I would like to thank my family for their support and understanding during the authoring process. To my wife Tess, thanks for putting up with my absence and for waiting on me hand and foot while I wrote this book. To my daughter Shoshana, thank you for those times of laughter that I desperately needed and for your adultlike understanding of why I was doing what I was doing. My two furry, canine babies Zeus and Morpheus were instant stress reducers whenever they would somehow manage to get past Tess' guard and nudge my legs. They aren't at a high enough reading level to appreciate this note, but I'm sure they will get the message somehow.

I would also like to thank my friend and cofounder Prabu for essentially running our company entirely by himself as I dropped off the face of the earth to devote time to this book. Prabs, I don't tell you enough how much I appreciate what you have done for the company. Thank you for shouldering not only your but my responsibilities in running the company while I wrote this book.

To my friends and cofounders at RedStorm, Ariesto and Ele, thank you for keeping the InfoSec ship running smoothly while I was MIA.

To Thiago, my Technical Editor, thank you! I enjoyed your notes and insights as you reviewed the chapters in this book. It was great fun working with you.

Last but not least, team Apress. Thank you all for your help in making this second edition a reality. Mark Powers, I really enjoyed working with you. You know how to get the best out of an author, and I want to thank you for igniting the spark for me to keep researching even after this book. Steve Anglin, thank you for your patience and persistence. It took almost a decade, but it's finally done. The rest of the folks behind the scenes that I may have not met or spoken with, thank you!

Introduction

This book was a long time coming, and yet I can only feel that now was the perfect time to write it and publish it. Much like a young stand-up comedian who when just starting out has to collect all his life experience with which to deliver as humor, this second edition is a collection of personal experiences and research done along the way. This book is intended as a reference tool rather than an in-depth, granular teaching tool. It is a better friend to those developers and security researchers who are in their early-mid career than those just starting out. It is a collection of how I have done things and the reasons why I chose to do them the way I did.

In this book, I approach Android security from an offensive standpoint. If the first edition were the Blue Team, then this one is definitely the Red Team book. The principle I try to stand by in this book is that the best way to test your app is by breaking it and breaking it into as many pieces as you can. A true test of your app will be if it can withstand some of the techniques that we use in this book because it is a collection of techniques that are being used out there today. To this end, you will find a lot of information about how to intercept network traffic, how to break SSL and SSL Pinning, how to root your device, and then how to figure out that security is a lot more than looking for that silver-bullet piece of tech. It is never the case. You have to do the work. You have to research; you have to test and you have to understand the behavior – of apps and people. There is no silver bullet to security; you have to spend countless hours and, yes, sleepless nights worrying about it.

This book is also a work in progress I feel. As I wrote the chapters, I felt myself taken in different areas that I could not afford to explore. I hope to revisit some of those topics in the future and who knows? Maybe there will be another book. I do hope you find the book useful and that you learn to look at security from a different perspective. If there's one thing I want you to take away from this book, it is that you can't have security on autopilot. It is a topic you have to think about and consciously make decisions about at every step of the way. The bad guys out there will not rest, so that means less time to celebrate your wins and more time to spend looking at worst-case scenarios in your very own bubble of paranoia.

CHAPTER 1

Introduction

A little over seven years ago, I wrote the first edition to this book, and to be honest, I could have done better. So, when Apress reached out to me to refresh this book, I was elated – elated because not only do I get the opportunity to do better but also because in the space between the first and the second edition, I had the opportunity of taking a fantastic journey. It's a journey that had many highs and lows – one part where I led a team of software builders and one where I led a team of software breakers. I will give you an insight into how it feels to build and secure a product in a company which went from 40 people to over 3500 people seemingly overnight – a company that, as of writing this book, has a valuation of ten billion US $.

Before I take you on this journey, I would like to thank Steve Anglin at Apress who gave me that second chance at doing better and who still believed in me enough to reach out to. Together with me on this book is Mark Powers who I have worked with closely in making sure all parts of this book are good to go. Also, I'd like to thank all the folks on the team who I will not interact with directly, but who I know are there making sure this book looks its best. A big thank you to all of you.

The Startup Landscape

Most folks who have not been living under a rock are keenly aware of how the "startup ecosystem" has witnessed meteoric growth. I attribute this growth to the VCs and investors that have been pumping in hundreds of millions of dollars into round after round of fundraising. With more capital being available, it seemed like more startups popped up out of thin air. At the center of each startup was almost always an app. Be it an iOS or Android app, it existed and was mostly touted as a solution to a problem at your fingertips. Now this was not happening only in the United States, although it really took root there. Word of startups that built apps spread around the globe. Headlines proclaiming what seemed like colossal sums of money being raised by these startups

S. Gunasekera, *Android Apps Security*, https://doi.org/10.1007/978-1-4842-1682-8_1

made everyone everywhere sit up and take note. Asia was no exception. It saw the success that the startups in the United States had and raced to keep pace. The startups were different, though. Asia (excluding China – because China's startup ecosystem is considerably different) had a different set of challenges, and thus the startups that were born there had a unique set of operating conditions that made it unique to Asia alone. Some challenges were a lack of stable infrastructure, poor handset quality and thus handset performance, and a more reserved culture when it came to trust, payments, and app usage.

Through it all, however, one key datapoint has remained a crucial success factor for an app waving startup: the user base, or the number of users of that product or app. For startups to attract large funding rounds, it had to show growth. One key metric that defined growth was weirdly not revenue. Instead, it was the number of users that interacted with the app. The more users you had, the more valuable you could become in the eyes of VCs. The formula was simple: gain more users, gain the attention of the bigger VCs that would fund you well. The startup founders did just that; they went after the "critical mass" of users to help kick off the success of their creation. With this, the flow of VC money poured into Asia and even gave rise to VCs grown from within Asia. In an article[1] written by the *Nikkei Asian Review* in October of 2019, the numbers for Asian-focused VC funds sat at 323 billion US $ under management, while the numbers for US-focused VC funds were at 397 billion US $. The numbers were taken at the end of 2018, and the article projected an eventual overshadowing of US VC funds by the ones in Asia.

So, what does this all mean? Well, the currency of startups these days is not actual currency. No. The majority of these startups are not even breaking even. The currency of startups in present day is users – users, glorious users. Users with their personal information, credit card numbers, addresses, email addresses, and unencrypted passwords or password hashes, it's quite the buffet for anyone that wishes to feast. The bigger the funding and the company, the more upmarket your buffet. I would even go so far as to say if you are tracking your attacks on your infrastructure regularly and see a rise to a new, higher norm, chances are you're making it big in the startup industry. Pat yourself on the back and go get yourself a CISO if you haven't one already.

[1] https://asia.nikkei.com/Business/Business-trends/Asia-set-to-eclipse-US-as-world-s-venture-capital-powerhouse

Between Two Books

A lot has transpired between the two editions of this book. Startups were formed, users were gained, users were hacked, and users were lost. Let's take a quick look at the security landscape between the years of 2013 and 2019. One thing that you may notice when you go back looking at breaches is that apart from news of Android malware, everything else seems like a hacking attempt. Let's look at the malware first.

What Is Malware?

I will use the description that I used in my first edition here. It hasn't changed:

> *Malware is defined as any piece of malicious software that lives on a user's computer or smartphone whose sole task is to cause destruction of data, steal personal information, or gain access to system resources in order to gain full control of the device it is on. Malware is written with the sole intent of causing harm; and usually, malware authors will write the malware to target a specific weakness in an operating system or platform. Often, the malware author will want to maximize the spread of her malware and will look to implement a mechanism where his software can copy itself to other, similar devices.*

Now, previously, I used the word "lives" on a user's computer or smartphone. Let's dissect how the malware gets to live on the device in the first place. Obviously before the malware starts living on the phone, it has to be introduced. To explore further, let's look at a piece of Android malware that has managed to steal over a million Google accounts. It's named Gooligan. Gooligan was first discovered as a mobile malware campaign that was not as malicious as it became.

Researchers from Check Point discovered the malware that came bundled with an application called SnapPea as shown in Figure 1-1.

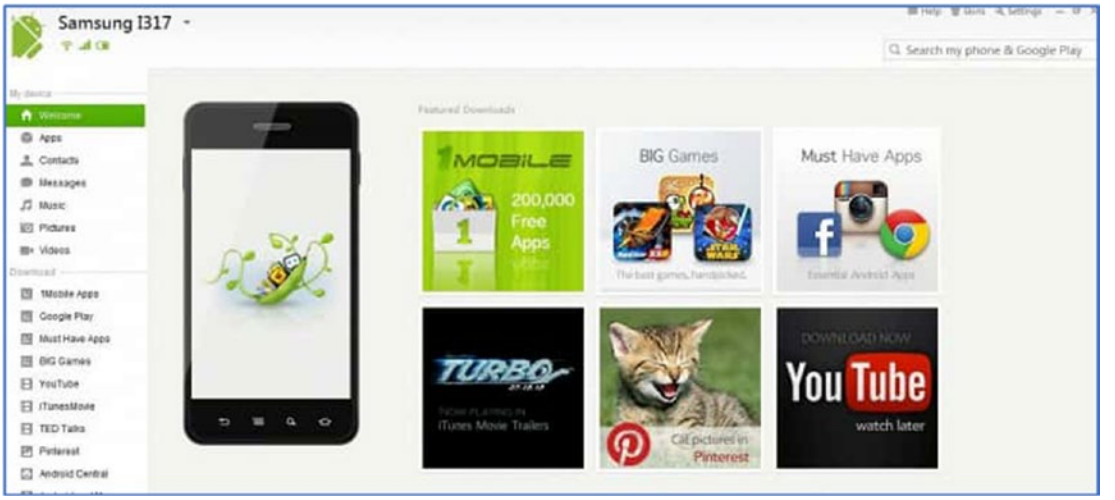


Figure 1-1. *The user interface for SnapPea*

Now SnapPea, it was found, would try to root your Android device with 12 different exploits. Check Point has remarked that it was the most amount of Android root attempts that they had witnessed in the wild. Two of the exploits were found to be the vRoot [CVE-2013-6282] exploit and the TowelRoot [CVE-2014-3153] exploit. These two are the only exploits that have CVE IDs in the database. When the rooting of the device was complete, the malware would install several other malicious apps that would further install subsequent ad-laden apps. The malware would then connect to a central server and await further instructions. It seemed like the primary purpose of this malware was to install other adware apps that made all infected devices connect to advertising sites and make it appear like legitimate users were viewing the ads. More views meant more payment for the app owners, and so greater infections yielded greater profits.

One thing interesting about the infection mechanism is that if SnapPea was installed from the Google Play Store, then there would be no infection. The app had to be downloaded from other sources that were most likely less stringent in how they checked the apps that they published. Also of note is that the user had to install the PC version of SnapPea and then connect his Android device to his PC in order to kick off the infection process. A blog post from Check Point shows the infection flow as depicted in Figure 1-2.

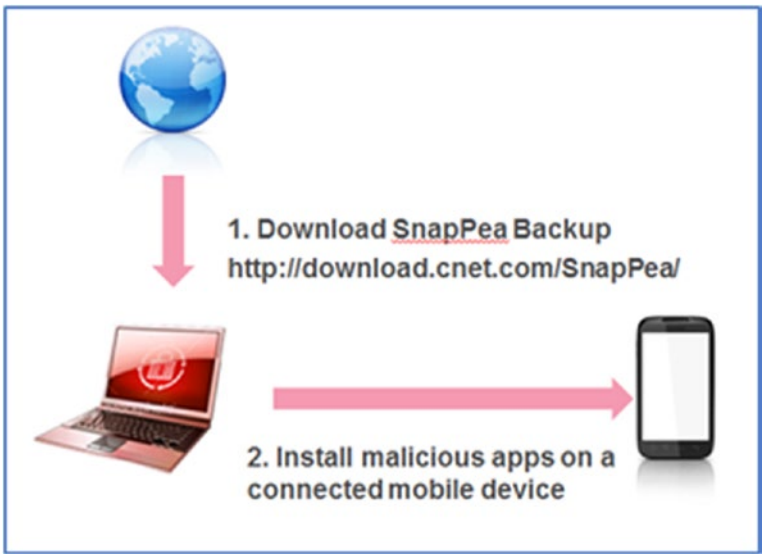


Figure 1-2. *How the malware made its way onto a device*

Subsequent iterations of this malware, which then became known as Gooligan, used similar rooting mechanisms, but had far more sinister goals in mind. After the original malware authors released their adware/malware campaign with SnapPea in 2015, they retreated and disappeared not to be heard from again until June of 2016 with Gooligan. Once again, apps that were available on third-party app stores carried the malicious payload. The apps were marketed as free versions of popular paid apps. So essentially, the draw this time was that you're getting a free version of an app which you would normally have to pay for – piracy.

Once installed, the app tried to root the device that it ran on. Then, it would tell a central command and control server information about the device and the fact that it was rooted. Then it would download a whole slew of rootkits and proceed to steal accounts and even authentication tokens for Google Photos, Play, Drive, Docs, and Gmail. A summarized flow of how Gooligan worked was found on the Check Point blog post[2] and was reproduced here in Figure 1-3. Gooligan left a trail of compromised accounts in its wake, and the number of accounts compromised went up from 400,000 in September of 2016 to 1 million by November 2016. In addition to stealing Google accounts, Gooligan would also download, install, and then review other seemingly ordinary apps and would leave a positive review on the Google Play Store for those apps.

[2] https://blog.checkpoint.com/2016/11/30/1-million-google-accounts-breached-gooligan/

The review text was received from the command and control server. Think of it this way, in November of 2016, there were approximately 1 million Android phones, downloading possibly useless apps from Google Play, installing them, and leaving positive reviews. It is highly likely that the Gooligan authors advertised a service in which app authors can get five-star reviews for a fee – similar to how you can find a service that guarantees to increase the count of your Twitter or Facebook followers.

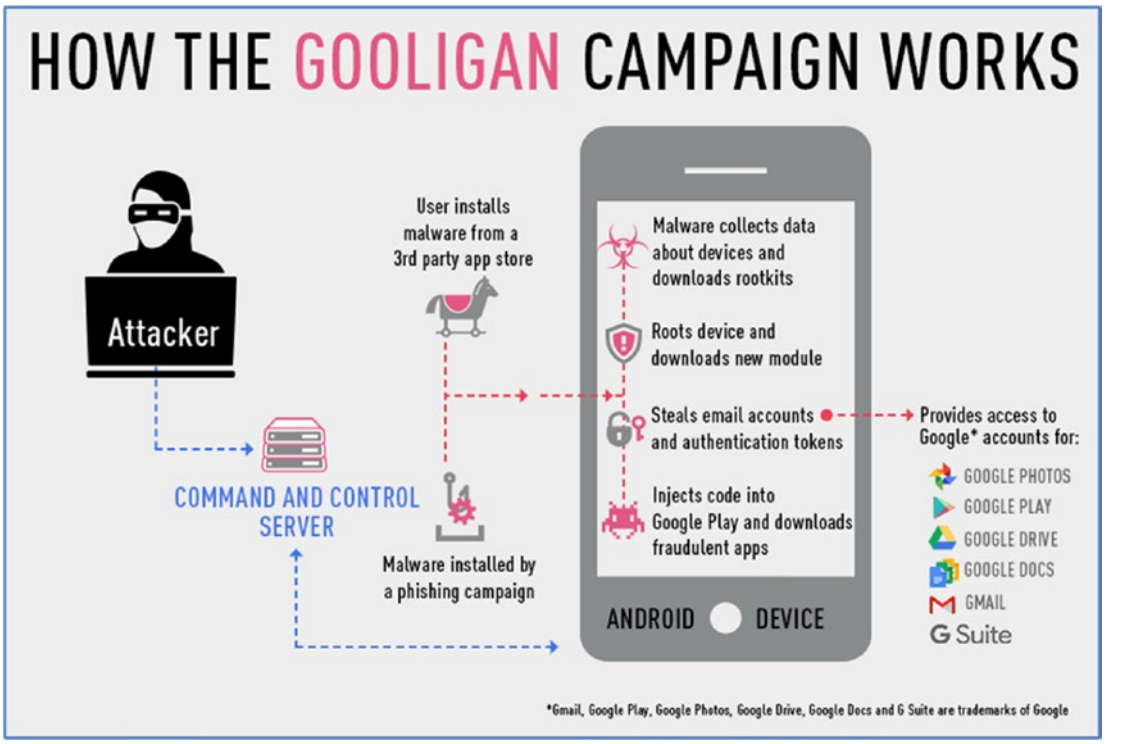


Figure 1-3. *How Gooligan worked*

Launching Attacks via Phones

Barring a situation similar to Gooligan earlier, it is important to keep in mind that attacking a phone gives you far less reward for your effort. The logistics of attacking a phone directly in that manner are beyond the reach of the solo hacker. Typically, organized malware attacks such as Gooligan are the work of larger, organized groups.

While the details of a lot of breaches still remain well hidden, my theory on how such large numbers of data records get stolen is that hackers launch their attacks from a semitrusted source. Bear with me on this one. If you consider the company that

builds an application of some sort from scratch, they will almost always have to build or integrate a back end with an API layer. They then build the mobile app to communicate with this API layer. Now an interesting thing happens that I have seen among teams working on these projects. The fact that both the back end and the mobile app are built by the same team gives a heightened sense of trust to the requests originating from the app side. The idea that all the data comes from the app that was also built in-house means that a slightly more relaxed security posture can be maintained at the back end. Let's analyze this first. Consider how the back-end developer may think about it:

- My API URL is an unknown, unpublished one
- My API can only be accessed by our mobile app that was built in-house

These assumptions are fairly normal ones to have in general if you have not had to think about the security of the overall product. Lucky is the paranoid developer that considers his code unleashed into hostile waters the minute he publishes it. The two preceding assumptions usually will lead to more lax security that can take on the form of sending too much data between phone and server, relying on the phone to carry the load of authentication, offloading some processing onto the phone, or storing sensitive data on the phone. The assumptions are wrong. Imagining for even a second that your data in transit between the app and the back end is safe from prying eyes is an illusion. I will show you later on in this book how to look through the data traversing between phone and server.

Note You may have seen me use the app and product earlier. By app, I mean just the application that is running on your smartphone. By product, I mean both the app and the back-end component which contains the API that the app will talk to.

OK, let's keep in mind that some developers may make the assumptions listed earlier and proceed to build and launch a product. We now have a product that makes assumptions about security. The product is released and the public loves it. Your startup begins to build a group of loyal users. Joy of joys you even begin to see repeat users when you do your cohort analysis! You begin to attract the attention of investors, and then the media starts writing about you. Let's hit the pause button here. Similar to the Hollywood movie effects, our actors all come to a halt with a sound that a record player makes when you physically stop the record with your finger.

What you have here is a pivotal moment when things can go colossally wrong, or you hire that CISO and they yell at you for getting this far without paying attention to security. Since we're tracking the security incidents between 2013 and 2019, let's go with colossally wrong. Let's unpause our scene.

Because of all the attention you have been getting from users, investors, and media, you also begin to attract the attention of bored or motivated hackers. A bored hacker who has a little too much time on his hand scrolls down past the news article about your product, then scrolls back up again. He thinks, "Hmm, I wonder what those guys are sending back and forth between back end and the app…." He downloads your app onto his rooted Android phone, plugs it into his laptop, then fires up his favorite set of tools on that laptop, and within minutes he begins to see all the traffic you send back and forth. He smiles, "Someone decided to send phone and credit card numbers in requests that only needed to send a name. Oh, that was particularly lazy of you to not filter out those bits of data." His smile widens as he whispers, "I have you now." End scene.

My gigantic ego would immediately associate myself with the superior skilled, elite hacker in the preceding story, if not for one thing: that was a true story of me when I tested an app in September of 2019. Yes, yes. As you will soon see, I have a flare for the dramatic. If you ask my daughter, she would even say melodramatic. Moving on to my theory, I firmly believe that a major portion of breaches involving a loss of large quantities of personal data stems from a weakness found in an app – a weakness similar to what we discussed earlier. The underlying assumption that all is well in the security of the product ecosystem can give rise to some stressful times that can make or break your business.

So, what truly happened between the first and the second edition of this book? Not a whole lot of change sadly. I've been testing app security as recently as two weeks before I started writing this book, and I still find the same attack vectors that I found over six years ago. Yes, this is an incredible unhelpful thing to say and probably goes on to even alienate some of the developers out there. I'm from the tough love school, and so, I feel that the best way to think about security, especially in a startup worth millions of dollars, is that you only get one chance. This is another reason why I am taking a crack at writing another book, so that we can all go through some of my experiences together and try to come out of it with better security. Right? Great! Let's now move on to some of that journey I've been promising you earlier in this chapter. Let's first start with some of my lows.

Hello, I'm Your CTO

As I am apt to tell whoever I am speaking with or addressing, I was extremely fortunate enough to join a startup in Southeast Asia, specifically Indonesia, called Gojek. Now if you haven't heard of Gojek, that's fine. Let me give you a brief idea of what we set out to do. I used to tell folks that were unfamiliar with Gojek that we were an Uber for the two wheelers aka motorbikes. To address your still burning questions, we must turn to the transportation infrastructure in Indonesia. If you ask an average Indonesian living in the capital city of Jakarta, they would say that the transportation system, regardless of what mode of transport you took, involved sitting stationary for large chunks of time. The word in Indonesian is called *macet* (mah-chet) which means "traffic jam." Macet was so much a part of the vernacular that a group of people you were meeting would nod sympathetically if you walked in 20 minutes late. Even if you had an actual legitimate reason or emergency for being late that didn't involve a traffic jam, your business counterpart would look up at your disheveled appearance and offer up a polite "Macet?" before you would start your meeting. Now because of macet, four wheelers aka cars would stand no chance in making up any lost time between your two meetings for the day. What emerged was an informal group of motorbike riders that thought, "Hang on, we can zip through macet and make some cash ferrying people who were late between their two meetings for the day." Essentially was born the culture of the motorbike taxi. It isn't entirely new and is usually found in cities that were developing fast and didn't have the infrastructure to cope. Thailand has one called the *motorcy*. In Indonesia, they called themselves *ojek*. Ojeks were great! They could get you back and forth fairly quickly but came with a slight problem: price haggling. If you wanted to get around, you would find an ojek asleep on his bike on a street corner, then tell him where you needed to go and why you were refusing to pay his ridiculous asking price. You can see where this is going, can't you?

Two friends of mine, Nadiem and Kevin, thought, "Hey, let's make an app for these ojeks! People don't like the fact that they have to haggle all the time so let's make that process simpler and build an app to hail and ride ojeks." Although Gojek was founded in 2010 as a rudimentary web page with a call center acting as an ojek dispatch, in 2015 they launched a mobile app for it. A rebirth if you will. The guys offered me to be one of the cofounders and picked me to be their first CTO, and you might think I jumped at the chance. Well, you would be wrong. I was skeptical to say the least when I heard the concept. I wondered where all the technology was that needed a CTO. But Nadiem, in his usual persuasive manner, asked me to try it for three months, and if I didn't like it,

then we could part ways. I ended up staying for almost four years. (One thing you should probably know is that a Gojek year was roughly equivalent to 3 normal years. I am deadly serious.) We went from a small startup of about 40 people in 2015 to one that is well over 3500 people today. An app that started with three core features then, transitioning to a super app with more than 20 features now. A company with minimal funding then, now valued at 10 billion dollars.

As the CTO, I was tasked with keeping the product running, building the product, building a team, and ensuring all our technology needs were met. To say that it was challenging would be an understatement – mostly because a lot of the things I had to do, I had to learn on the job. No prior role of mine prepared me for the complexity and stress that came with being the CTO of this new hot startup out of Southeast Asia. On most days, I felt like I had failed. We had so much growth, so few developers, and so many system failures that caused downtime. Our tech team was five people in those early days, and I would rack my brain trying to find and hire more developers while simultaneously helping with coding the back end, optimizing APIs and infrastructure maintenance. We went on like this for some time, and we slowly improved. Our product got better, we launched features fast, and we got funded. I was out of my element though. I was struggling because the role for me was one that I was diametrically opposed to: here I was building things when all I was used to doing was breaking things for over 15 years. I pushed on and learned a lot. I learned about building teams, about communication, and about accepting that good enough is good for now. I had never seen a scale or speed like that in my life, and it felt quite lonely because there was no one around that I could talk to about it. Yet I was happy, I was doing something I thoroughly enjoyed, I was challenged on a daily basis, and I didn't have time to think or take a break: bliss.

We were chugging along as usual balancing the product features, performance, and growth every day for roughly one year when I came to the realization that I was probably in over my head. While I knew that I could handle the role of CTO given enough time to catch up, this particular business was ruthless and unforgiving in the time that it gave you. I also knew that with our continued trajectory, we would desperately need to get a handle on security. Early on I had started to build a small security team that would sit there and hack our product to uncover any security flaws that could affect our users' privacy or affect the business negatively, but I knew that I would have to increase the size of this team and provide leadership in that area. With that, I had separate conversations with both Nadiem and Kevin and transitioned my role to be the CISO.

This was a bit of a low point in my time at Gojek because I was in a learning mode and doing something I had not done before, and being challenged at it was immensely satisfying. But in reality, there was a desperate need as we grew to make sure that security was tight, and I knew, given my background, I had to step up and deliver for the good of the company. As time passed, we even coined the phrase to signify important moments like this: "It's not about you."

The next low point was some time after I had transitioned from the CTO role to the CISO role; we received several emails from both well-meaning individuals and glory and profit-seeking individuals alike that told us they had successfully hacked our product and gave details on what they had uncovered. To strengthen my theory that I mentioned earlier, it turned out that they had breached our API back end by hacking our Android app. They discovered the communications patterns between the app and the back end and were able to compromise our product including collecting our customer data.
In the past, I have also been able to report flaws in products to companies including BlackBerry, and I always appreciated a company that reacted and responded fast. Still, nothing quite prepares you to receive an email like that – one that says that someone out there has managed to successfully compromise your product and offers you proof.
I felt my stomach lurch and a wave of nausea washed over me. Here I was, part of an organization that was on the receiving end of the hacking for a change; usually I was the one hacking others. Nevertheless, I took a deep breath, notified our leadership team, and took this to the tech team so that it could be rectified quickly.

Getting hacked is truly a terrible feeling. Having it unfold in public while hyped by the media for sensationalism makes that feeling worse. I empathize with anyone that has been through a scenario like that. It is a lesson that is swift and often one that you will not forget. It can make or break a company or a career. It leaves you feeling vulnerable and exposed. It is a feeling that I want to help you avoid. It is a mistake I want to prevent you from making. Therefore, I'm going to pack as much of this book as possible with real-world instances from which I learned some important lessons. Now there will most likely be different ways of handling a particular situation, but I will tell you how I did it.
In the first half of this book, I will discuss and document many of the areas you should be looking at as a builder of software when you are part of a startup. I will outline some techniques and tips that you can consider when you have to build something at scale. Your teams may be large and diverse, your communications may not be as streamlined as you would like, but I will talk about some practical mechanisms that you can adopt to improve your security as a builder. Not all of it will be a narrative as my stories in this

chapter; this **is** a technical book after all. Yet, I may occasionally wax poetic and regale you with my tales of the time I spent at this startup that is close to my heart, named Gojek. As for the second half of this book, I dedicate this to my people, the breakers…

Hello, I'm Your CISO

And I will help you to break apps so you can be better informed about how to fix them and avoid feeling what I felt when I got hacked.

A startup CISO's job is still fairly new. Acceptance has just begun because the number of articles I read on the topic is increasing steadily. Where to put your CISO, who should he report to, and what should his job description be are all discussed more frequently. Yet, the answers you will get will vary by and large by the person answering it and his agenda. I had plenty of discussions when I started in my role as CISO at Gojek – endless debate on whether I should report to the CTO or to the CFO or even the CEO. As you may have guessed, we had moved on a little from our startup roots at Gojek and were now emerging into a more structured operating model.

Let's tackle these questions in this section because I think they are important. Let's try to look at the role of the CISO from a few different perspectives. Let's start with the CEO and founder.

Reporting to the CEO

The founder and CEO of a startup does everything in the beginning. No task is too small; no job is beneath him. A founder and CEO will agonize over the longevity and survival of his idea and his company. How can I hire the best? How can I accelerate faster? How do I stay ahead of my competition? Where can I find money to pay my employees and keep the business running? These are some of the many questions that likely bounce around in his head. If we try to break down the role of a founder and CEO, it is one of removing friction at all costs. All his tasks and his purpose in the company are to remove friction and enable growth, sustainability, and survival of the idea, vision, and company. Thus, to report to a CEO, the CISO must be willing to be an enabler of growth, of technology, and of features. A CISO reporting to a CEO cannot be too restrictive, but at the same time, a CISO reporting to a CEO has to be an educator. He has to be someone that can demonstrably show the risks involved when setting out on a particular strategy. Communication is key in this relationship. More than that, it is important to be able to

understand your CEO and his goals so that you can effectively highlight the risks you may see on the path to these goals. For example, if your CEO is unaware of fraud that is taking place due to a flaw in the logic of your product, then learn to present the problem in terms that he would respond to. If he is a number or a data person, then quantify the fraud in the form of either a dollar value loss or a number of users lost datapoint. If he is more of a socially inclined person, then present how the fraud could cause a loss of productivity for the large number of users of your product. Your CEO is highly likely to be someone that has a bigger vision so understand that and try to adapt your narrative to show how what you propose will help drive toward that bigger goal.

Reporting to the CFO

This relationship, while rare, can still be one that you find yourself in. In this dynamic, the role of the CISO would most likely be weighted more onto the regulatory and compliance side of the spectrum. Technology security, such as red teaming and hacking your own product, may come second to ensuring that sufficient and sensible security policies and operating guidelines are in place. A CISO in this relationship may find that he has more leverage over individual teams or departments. The mantra of "do it this way or we are not in compliance" may follow this CISO and with good reason. Financial startups or startups that answer to regulators such as central banks have a strict set of guidelines on security that the startup must comply with. Often, noncompliance can lead to large fines or even a loss of an operating license that could bring the business to a crashing halt.

Reporting to the CTO

Don't.

I kid, I kid. This relationship is one that is mostly fraught with the greatest conflict of interest. The CTO's role in a startup is to deliver a feature-rich product that iterates fast and is performant. Finding a flaw or selecting a strategy that is more secure may directly affect the CTO's ability to do his job. We can all theorize about carving out a small team to fix any flaws discovered by the security team, but when it's crunch time and engineers are limited and the investors are breathing down your neck and the competition is seemingly moving faster, it becomes near impossible to slow down and address security flaws. This is a significant problem especially if the CTO isn't well versed in or does not come with a security mindset. Should a CISO find himself in this relationship, then

pre-agreeing with the entire leadership team about a structure of transparency should be something to consider. The worst position for a startup to be in is one where none of the security flaws discovered either in-house or externally are being brought to the leadership's attention. While there may be some short-term respite for the tech team, it rarely bodes well for the entire organization in the long run.

In my role as the CISO, I reported to Nadiem, the CEO. I structured my team in three broad areas of specialization: the Red Team, who would mount the offense on our product and infrastructure; the Blue Team, who would work to actively secure the infrastructure; and the Compliance Team, who would build, maintain, and disseminate policies and guidelines. The Compliance Team would also work together with the regulators on any licensing requirements that were necessary.

Reviewing What Gets Published

Being a fast-growing startup, one of the bigger challenges we faced as a security team was trying to figure out what products existed and which ones were launched newly. Essentially, this is inventorying, because you can't secure what you don't know about. Working closely with the infrastructure team, we took control of the process of adding a new set of APIs to our load balancer. This created a sort of gate that allowed us to question the requesting team whether they had conducted a preliminary penetration test on their system. If they hadn't, then we would not proceed with the release of the APIs until they did. Since we had announced this and continued to communicate this fact, it was rare that we encountered a team that requested an API addition having not completed a penetration test.

Another such "gate" scenario you can consider is to place a source code security review service as part of your software build process. Usually, tech teams will have build processes known as CI/CD or continuous integration/continuous delivery. Essentially, this boils down to having several teams contributing code that is merged to one main development branch and having shorter development cycles so that software can be deployed or delivered faster. Sometimes we would release software three or four times a day. Things like new features or bugfixes could go out very fast thanks to this process. One prerequisite to deliver the software was that the "build should pass." This means that prior to a release, all the source code that has been contributed is built. Once built, the software can be released. The build process will have a set of tests that it runs to ensure the software does what it is supposed to. For example, the component that added

a food item to your order was tested to see if when the item was added, the correct item was added, and that the correct quantity was added. If any of the tests would fail, especially on the critical tests, then the entire build process would stop with a failure notice. This meant that the software could not be released until the failures or bugs were addressed. By adding the source code review system to the path of the build process, your security team can ensure that any code committed to the main branch passes secure coding practices (at least as effective as the product used for source code review). If the source code reviewer found a flaw or an instance of insecure coding, then it would trigger a build failure. Thus, it would be possible to prevent insecure software from being released. Now keep in mind that this would just catch the low hanging fruit of the insecure source code. Although source code checkers are evolving, they are still not up to a level where they can prevent the majority of attacks.

Did I Just Waste My Time Reading All This?

I don't think so. Yes, it is a lot of narrative and very little to do with Android apps or anything deeply technical, but I think it sets the context for this book. I feel that by contextualizing some of the technical topics that come later in this book, it can add a little bit more value as to the "why" of doing things and doing things a certain way.

Startups, especially technology product startups, are here to stay. The mobile device is here to stay. It would be very interesting to get a glimpse into how all this will further evolve, but for now, looking at the tech and security through a startup lens adds a lot of value in my opinion. As the chapters unfold and go from defensive to offensive security, having this context in mind should serve you well. Just for the heck of it though, the offensive chapters should be fun to read without context either as I will write them as a handy reference guide that you can directly flip to, get setup, and go from there.

With all that being said, let's head over to our next chapter and review some of what we learned previously with regard to secure software development principles. See you there!

CHAPTER 2

Recap of Secure Development Principles

In Android coding with Kotlin or Java, you may have to go out of your way a little to write insecure code. I mean, there are the obvious ones like leaving your private key hardcoded, but in general you should do fine on the code front. Kotlin has made several improvements to Java, which was one of the only ways to write Android code natively in the past. Some notable, among the several, Kotlin improvements have been the addition of null safety and lesser quantity of code written. The null safety check in Kotlin means that if you were to access a variable that points to a null reference, then the system doesn't throw a `NullPointerException` or `NPE` like how Java used to. Kotlin reduces the amount of exception checking because it has no exception checking. This means that in cases where an interface throws a specific exception, you don't need to keep checking for exceptions each time you use one, for example, Java's `StringBuilder's append` method. Kotlin makes both try and throw expressions, so they can return a value which gives rise to code that looks like

```
val a: Int? = try { parseInt(input) } catch (e: NumberFormatException) { null }
```

or

```
val s = person.name ?: throw IllegalArgumentException("Name required")
```

But I'm not going to dwell at all on the language used to write Android apps in this chapter. Instead, I am going to look at instances where code written can pass on vulnerabilities to other parts of the system, like the back-end servers. I will also talk about some of the things you need to keep in mind with regard to your user's safety and privacy. So, let's get started. Let's look at privacy first.

S. Gunasekera, *Android Apps Security*, https://doi.org/10.1007/978-1-4842-1682-8_2

Privacy

Data privacy has become a big deal lately. I mean, it always should have been, but the past few years have given us such wonderful headlines that privacy is almost always in the news. As a security practitioner, I'm thrilled that more people give a damn. As a security practitioner, I am also appalled at the amount of blatant violations of privacy still going on. Privacy is such a big deal that in May of 2018, the EU implemented GDPR or the General Data Protection Regulation. The law itself is fairly straightforward and logical, but interestingly, it allows for warnings and fines to the tune of up to 4% or 20 million Euros based on certain infringements. To date, 34 fines have been issued to various institutions including a hefty 50 million Euro fine to Google. Whether these have been paid or not are not part of this discussion, although I know Google was appealing their fine. The point is, however, to understand that we're getting serious about privacy.

If you're a developer that requests for and stores a user's private information, then you have a duty to keep that data private and safe. Depending on who you ask, the information one would consider private varies, but for a blanket definition, we typically define private information as information about a user that is generally not shared with the majority of the folks he meets. So, things like home address, gender, marital status, age, bank or credit card information, and phone number are usually considered the major components of information that a person would want to keep private. But let's analyze why first. I mean, why would you want to keep this information private? One can argue that by having that information about someone, there is precious little that can be done directly to affect him other than sending unsolicited pizza or a mob to his house or prank calling him until he disconnects his phone number. I'd like to pause here and talk about that mob I just mentioned. While I played it off as more or less innocent, there is one specific case where people have even died when mobs have been dispatched to their house. What I'm referring to here is the practice of swatting.

Swatting

Swatting takes place when someone malicious that knows your home address calls up emergency services and then deceives them into believing that there is a life-threatening emergency taking place at your house – typically, an armed person that has already hurt someone or a terrorist suspect. The resulting visit to your home by the SWAT or emergency response team can sometimes have catastrophic results. There was a case in 2017 where a 28-year-old father of two was shot by a police officer. The victim was not

even related to the group that initiated the swatting incident. He was an innocent victim that had the misfortune of having his address used for the purposes of winning a bet.

Let's go a little deeper into what happens when an emergency response team heads to a victim's house. They head there on full alert prepared to confront an armed and dangerous individual. The victim has no idea why someone has just broken into his home. He doesn't know if it is a friend or a foe. During heated times like that, sometimes a victim may resist or at least take on a defensive role. To the emergency response team, this may look like a challenge or threat, and if cooler heads don't prevail, then someone is taking a shot at the perceived aggressor. This is one way to look at why safeguarding your users' home addresses is important.

Aside from swatting, there is another key reason for keeping your user data private. And no, I'm not talking about the nasty headlines written about how badly you screwed up and allowed 100,000 user accounts to get leaked. I'm talking about social engineering. The more personal, private information that a social engineer has on a victim, the more plausible he can make his story. Humans are wired to trust. They are also wired to default to the truth in most cases. By using subtle cues of private information on a victim, it is a lot easier to push them into believing you and your story. "Oh this guy already knows about my information, so I guess he IS calling from that bank to help me with my bank account" would be a response from a hapless victim on the other end of the phone.

So, then, what's the secret to keeping a user's data protected? It's not really a secret. You have to protect your own infrastructure where the data is stored. One of the first steps I would advise would be to collect as little private information from a user as possible. It may be tempting to say "Let's just ask for this data now and figure out what to do with it later," but I think that is a bad idea. If you have a genuine need for the data, then ask for it – especially if it is something you can't do without, like a shipping address. The concept is that if you have the bare minimum amount of data, then you can rest easy knowing that the fallout from a breach isn't going to be that catastrophic, especially for the user.

The next piece of advice I would give is to separate the data into a so-called "Cold Storage" database. I've outlined how that may look like in Figure 2-1.