Using Malware Analysis to Improve Security Requirements Case Study

Nancy R. Mead, Carnegie Mellon University
Jose A. Morales, Carnegie Mellon University
Gregory P. Alice, Carnegie Mellon University

April 2021

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

Using Malware Analysis to Improve Security Requirements Case Study

Background

Several approaches for incorporating security into the software development lifecycle (SDLC) have been documented. Most of these enhancements have focused on defining enforceable security policies in the requirements gathering phase and defining secure coding practices in the design phase. Although these practices are helpful, cyberattacks based on core flaws have persisted.

Major corporations such as Microsoft, Adobe, Oracle, and Google have made their security lifecycle practices public (Lipner & Howard, 2005; Adobe, 2014; Oracle, 2014; Google, 2012). Collaborative efforts such as the Software Assurance Forum for Excellence in Code (SAFECode) (Simpson, 2008) have also documented recommended practices. These practices have become de facto standards for incorporating security into the SDLC. These security approaches are limited by their reliance on security policies such as access control, read/write permissions, and memory protection, and their reliance on standard secure code writing practices such as bounded memory allocations and buffer overflow avoidance. These processes are helpful in developing secure software products, but—given the number of successful exploits that occur—they fall short. For example, techniques such as design reviews, risk analysis, and threat modeling typically do not incorporate lessons learned from the vast landscape of known successful cyberattacks and their associated malware.

We propose that current SDLC models can be further enhanced by including misuse cases derived from malware analysis. Our focus is on the vulnerabilities resulting from design flaws. We also propose an open research question: Are specific types of systems prone to specific classes of malware exploits? If this is the case, developers can create future systems that are more secure, from inception, by including use cases that address previous attacks.
The extensive and well-documented history of known cyberattacks (Beuhring, 2014; Bisiaux, 2014; McMahon, 2014; Sood, 2013; Tankard, 2011) can be used to enhance current SDLC models. Specifically, a known malware sample can be analyzed to determine if it exploits a vulnerability. The vulnerability can be studied to determine whether it results from a code flaw or a design flaw. For design flaws, we can attempt to determine the overlooked requirements that resulted in the vulnerability. We make this determination by documenting the misuse case that corresponds to the exploit scenario and creating the corresponding use case. Such use cases represent overlooked security requirements that should be applied to future development to avoid similar design flaws that lead to exploitable vulnerabilities. This process of applying malware analysis to ultimately create new use cases and their corresponding security requirements can help enhance the security of future systems.

We recommend a process for creating malware-analysis-driven use cases that incorporates malware analysis into a feedback loop for security requirements engineering on future projects and not merely into patch development for current systems. Such a process can be implemented in the following steps and is illustrated in Figure 1.

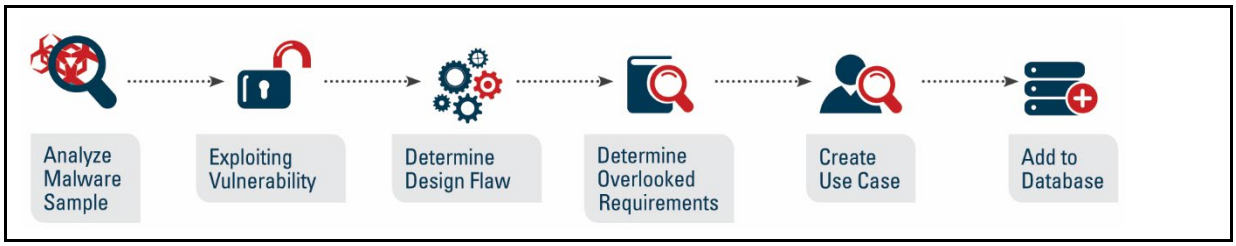

Figure 1. Malware-Analysis-Driven Use-Case Creation

1. A malicious code sample is analyzed both statically and dynamically.
2. Analysis reveals the malware is exploiting a vulnerability that results from either a code flaw or a design flaw.
3. In the case of a design flaw, the exploitation scenario corresponds to a misuse case that should be described. The misuse is analyzed to determine the overlooked use case.
4. The overlooked use case corresponds to an overlooked security requirement.
5. The use case and corresponding requirements statement is added to a requirements database.
6. The requirements database is used in future software development projects.

Steps 1 and 2 include standard approaches to analyzing a malicious code sample. The specific analysis techniques used in Steps 1 and 2 are beyond the scope of this paper. In Step 2, the analysis is used to determine whether the exploited vulnerability is the result of a code flaw or a design flaw. Typically this can be determined by detailed analysis of the exploit code. Of course, this presumes that the malware is detected, which in itself is a challenge. Step 2 illustrates the advantage of malware analysis by leveraging the exploit code to determine the flaw type. Standard vulnerability discovery and analysis without malware analysis excludes exploit code and may make flaw type identification less straightforward. Step 3 details how the exploit was carried out in the form of a misuse case, which provides the needed information to determine the overlooked use case that led to the design flaw.

In Step 4, the overlooked use case is the basis for deciding what may have been the overlooked requirement(s) at the time the software system was created that led to the design flaw. These are the requirements that should have been included in the original SDLC of the software system, which would have prevented creation of the design flaw that led to the exploited vulnerability. Steps 5 and 6 record the overlooked use case and corresponding requirement(s) for use in future SDLC cycles. This process is meant to enhance future SDLC cycles in a simplified manner by providing known overlooked requirements that led to exploited vulnerabilities. Including these requirements in future SDLC cycles helps avoid the creation of exploitable vulnerabilities, resulting in software systems that are more secure.

It should be noted that the new security requirement(s) may conflict with or render null other requirements that either already appear in an existing software product or are under consideration for a new software product. This means that the specification needs to be revisited and tradeoff analysis may be needed to determine which requirements to add/delete/modify.

Case Study Overview

This case study explores the proposed process of analyzing a malware sample. Using a sample of malware that steals data from Android mobile devices, we determine the exploitation scenario used by the malware exploit. Our investigation into the consequences of this malware exploit reveals a design flaw in a mobile application that could compromise user data. We scrutinize the design flaw to determine the applicable misuse cases and use those misuse cases to ascertain the missing security requirements to be used on future mobile applications for the Android platform. We explored the proposed malware-analysis-driven use-case creation method by studying impacts of a variation of malware for Android and its potential exploit of an open source Android email client called K-9 Mail. The steps outlined in Figure 2 resulted in the following activities and findings:

Step 1: A malicious code sample is analyzed both statically and dynamically.
To identify malware that can target the K-9 Mail application, a survey of Android malware was conducted using publicly available exploit databases from major computing security firms. The databases were examined for Android malware which compromises the security model of the application layer for a viable candidate. The malicious code sample selected for this case study is DroidCleaner, which is a Trojan variety of malware. Since application security practitioners may not be experts in malware analysis, literature from industry experts who conduct static and dynamic analysis of malicious code was studied to determine the impacts of the malware on the application's security posture.

Security researchers at Kaspersky Lab analyzed this malware and found that, while it masquerades as a utility to free memory on Android devices, it secretly sends premium-rate SMS messages and carries the capability to infect PCs when the Android device is connected. One of the key features of DroidCleaner is its capability to upload all of the contents of the device external storage directories to a remote server under the control of the malware designers (Paoli, 2013). Dynamic analysis of the malware conducted by FortiGuard's Threat Research and Response shows the commands issued by DroidCleaner to the Android OS, including a request to send a directory listing of the External Storage area of the device to a remote server followed by a repeated transfer requests for each file in External Storage to the remote server (Fortinet, 2013). The capability to upload the device external storage to a hacker's server has significant ramifications for mobile application developers.

Step 2: Analysis reveals the malware is exploiting a vulnerability that results from either a code flaw or a design flaw.
Though DroidCleaner was not designed to directly target the K-9 Mail application, it is a data-stealing application. What the malware designers did with the contents of the uploaded data is unknown, but the data was likely examined for valuable private information. Future attacks using a similar attack vector for stealing and extracting value from data are likely.

Android is derived from the Linux kernel and inherits many security features from Linux. Application sandboxing in Android is achieved by running each application as a separate user in the underlying Linux kernel (Google, 2014). This design choice leverages Linux's ability to separate application data, provide secure inter-process communications, and perform process

isolation. Because most Android OS functions operate at application level permissions (Google, 2014), cracking OS libraries does not give an attacker a foothold into other applications. Android provides each application with a data storage area called internal storage, which is controlled by the application's ID (Google, 2014) much in the way a user in UNIX has control over his/her personal Home directory. Android also has a general storage area called external storage, which may be in the form of removable media or an emulated storage area in the device's file system. In contrast to the stringent controls for internal storage, the external storage area is granted permissions at the top level (Ahmad, Musa, Nadarajah, Hassan, & Othman, 2013). That is, if read access is required for a single directory in external storage, read access must be granted to the entire storage area.

At installation time, applications are assigned the permissions requested during the install process. As a result of this permissions mechanism, software is easier to implement, as the application will never have a permission-denied error; however, this mechanism has the disadvantage of forcing users to fully trust the software in order to do an installation (La Polla, Martinelli, & D. Sgandurra, 2013). Users are more likely to grant access to quickly install the application rather than scrutinize the requested permissions, thereby granting malware the permissions it needs to wreak havoc.

From the perspective of the K-9 Mail application, there are three primary methods in which Android's robust security model will fail to protect the application data:

1. The application mistakenly grants broad access to the application's data storage area.
2. The application stores data outside of its application storage area (e.g., in external storage).
3. A process other than the Linux kernel obtains root privileges. This scenario can occur when a user "roots" their device or when a Trojan contains an attack against the OS. Malware designed to obtain root privileges typically uses the same code available for ambitious Android users who wish to root their device.

K-9 Mail does not grant broad access to its application storage area, so it is not vulnerable to the first issue above. K-9 Mail does allow users to save email data in the external storage area, so it is vulnerable to the second issue. The third issue occurs when the entire Android security model has been compromised. There is very little the K-9 Mail application can do when the entire platform on which it is operating is compromised, much in the way fire-resistant fabric will not hold up in a house that is on fire.

DroidCleaner is able to defeat K-9 Mail's security model when the application is configured to store email in external storage. In order to obtain external storage drive-level permissions, DroidCleaner asks the user for permission to access the external storage and network at the time of installation. When permissions are granted, DroidCleaner has access to all the data stored in this storage area (Ahmad, Musa, Nadarajah, Hassan, & Othman, 2013). Internet permissions open up network access, granting DroidCleaner a channel for transferring data to the hacker. For K-9 Mail users configured to use External Storage, the DroidCleaner malware is capable of uploading the K-9 Mail data stored on the device.

Assuming a K-9 Mail user was victimized by DroidCleaner, and the data from his/her External Storage was uploaded to a hacker's computer, the hacker would have access to the data stored by K-9 Mail in external storage. The following section examines the impacts of such an exploit. The exploit was simulated by changing the settings in the K-9 Mail client for sample email accounts to use External Storage, and the newly visible directory in <external storage>/Android/data was observed, copied to a PC, and examined. K-9 Mail stores the following data in External Storage:

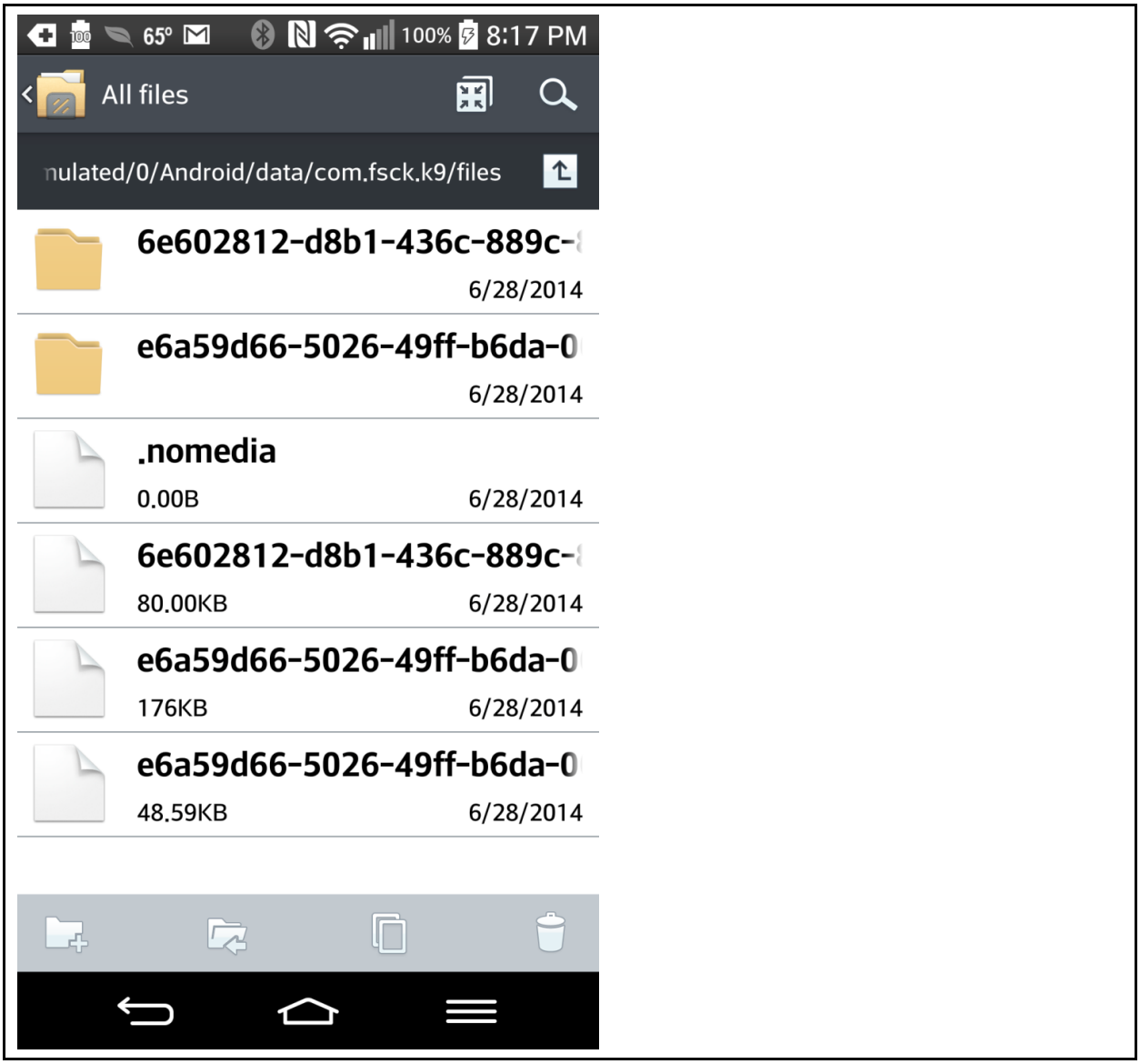

Figure 2. K-9 Mail Local Email Storage Files

Though the data in Figure 2 appears to be cryptic when viewed on the phone, the bottom three files are Sqlite databases corresponding to email accounts. The directories correspond to the databases. Figure 3 shows the contents of one of the directories.

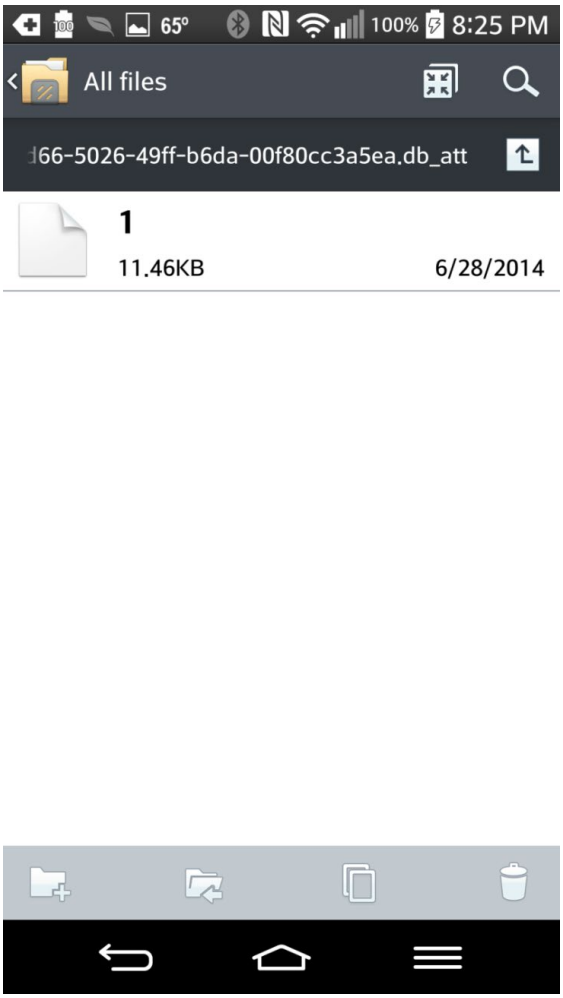

Figure 3. K-9 Mail Local Attachment Files

The directory contains a file with no extension, but examination reveals it corresponds to an opened attachment.

To assess the damage from malware such as DroidCleaner, we examined the contents of the K-9 data directory using a PC. Figure 4 shows the results of this examination.

#	id	d	f...	u...	subject	date	flags	s	t	(	t	html_content
1	1	0	8	1	Three tips to get the most out of	14005	X_DO\	m	gr			<!DOCTYPE html><html><head><meta http-equiv="content-type" content="text/h...
2	2	0	8	2	The best of Gmail, wherever you	14005	X_DO\	m	gr			<!DOCTYPE html><html><head><meta http-equiv="content-type" content="text/h...
3	3	0	8	3	Stay more organized with Gmail's	14005	X_DO\	m	gr			<!DOCTYPE html><html><head><meta http-equiv="content-type" content="text/h...
4	4	0	8	4	Getting started on Google+	14005	X_DO\	nc	gr			<html dir="ltr"><body style="display:block;font-family:Arial;max-width:600px;"><!– X-
5	5	0	8	5	Welcome to YouTube!	14009	X_DO\	nc	gr			<html lang="en"><head><title>Welcome to YouTube</title><style type="te...
6	6	0	4	2	Financial info	14030	X_RE	gr	bi			My account is 12334567. – Sent from my Android device with K-9 Mail. P...
7	8	0	8	6	Secret Message	14030	X_DO\	gr	gr			This is a secret message with an attachment.
8	9	0	8	7	Delivery Status Notification (Dela	14031	X_DO\	m	gr			<pre class="k9mail">This is an automatically generated Delivery Status Notification<
9	10	0	8	8	Delivery Status Notification (Dela	14032	X_DO\	m	gr			<pre class="k9mail">This is an automatically generated Delivery Status Notification<
10	11	0	8	9	Delivery Status Notification (Failu	14033	X_DO\	m	gr			<pre class="k9mail">Delivery to the following recipient failed permanently: <br /
11	12	0	7	K9L(	Secret	14039	X_RE	gr	gr			This is a secret message.
12	13	0	4	4	Secret	14039	X_RE	gr	gr			This is a secret message. – Sent from my Android device with K-9 Mail. P...
13	14	1	9	K9L(	NULL	NULL	NULL	N	N	N	N	NULL
14	15	1	2	2	NULL	NULL	NULL	N	N	N	N	NULL
15	16	0	8	10	Secret	14039	X_DO\	gr	gr			This is a secret message. – Sent from my Android device with K-9 Mail. P...

Figure 4. K-9 Mail Database Contents

The Sqlite database is not encrypted. When the files are transferred to a PC, the contents of the email account display in clear text. Each row in the "html_content" column is a record in the Sqlite database containing the HTML contents of the emails synced to the Android device. The text "This is a secret message" is visible in clear text; the visibility of this test email demonstrates the security flaw present in this design.

Furthermore, when the extension-less file "1" from the attachment directory is loaded onto a PC and opened in its native application (Microsoft Word), it displays in clear text (it is also not encrypted). The ability of K-9 data files to be copied from Android External Storage to another location by DroidCleaner or other malware and read in plaintext represents a confidentiality vulnerability. K-9 Mail users who are victims of DroidCleaner or a similar data-stealing Trojan are at risk of having their email contents exposed.

The storage of the email contents and attachments in the external storage area of Android without any protective measures is a design flaw since Android does not provide operating system level enforcement of directory permissions for the external storage area. Trojans can simply request read access to the external storage area to gain access to all of the locally stored contents for K-9 Mail.

Step 3: In the case of a design flaw, the exploitation scenario corresponds to a misuse case that should be described. The misuse is analyzed to determine the overlooked use case. There are two exploitation scenarios exposed by DroidCleaner. The first exploitation scenario is the ability of a hacker to upload and view email stored on the Android device. The second exploitation scenario is the ability of a hacker to upload to their own server and view attachments stored on the Android device. In both exploitation scenarios, the user sets K-9 Mail to store data in the external storage area, and a data-stealing Trojan or other attack is conducted against the device.

Misuse Case 1 – The contents of emails stored on the device are stolen. The misuse case in Figure 5 was developed based on the exploitation scenario that allows a hacker to view the contents of emails stored on the Android device.

In this misuse case, the user keeps his/her email on the phone's storage drive. The hacker gains access to the phone's storage by compromising the operating system. A common way for the hacker to gain access to the phone is by tricking the user into installing a Trojan, which is then granted access to the drive via the user during the install process. Once the Trojan gains access to the drive, the Hacker is able to use the Trojan to download files, including the email contents file.

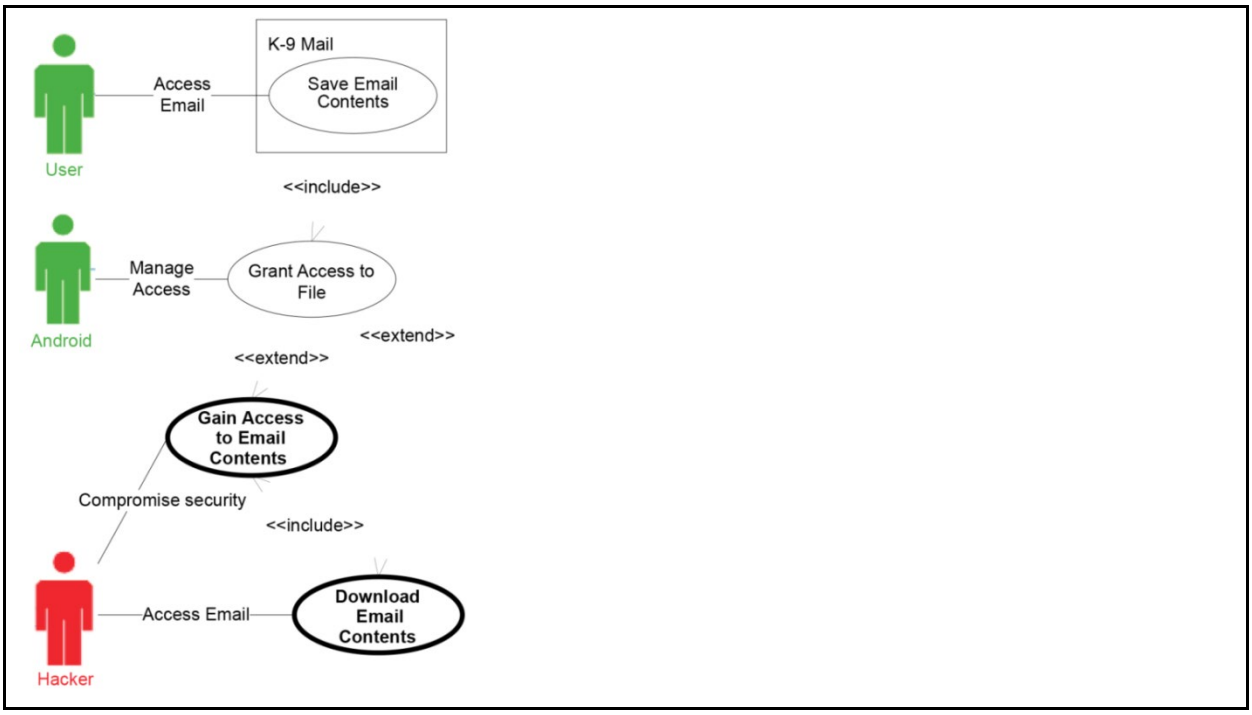

Figure 5. Misuse Case 1

Misuse Case 2 – Attachments stored on the device are stolen. In the misuse case shown in Figure 6, the user stores his/her attachments, which contain confidential information, on the external storage of their device. The hacker exploits the Android operating system to gain access to the attachments in storage. This can be accomplished through a data-stealing attack. The hacker then uploads the attachments to his/her own computer using the exploit on the Android device.

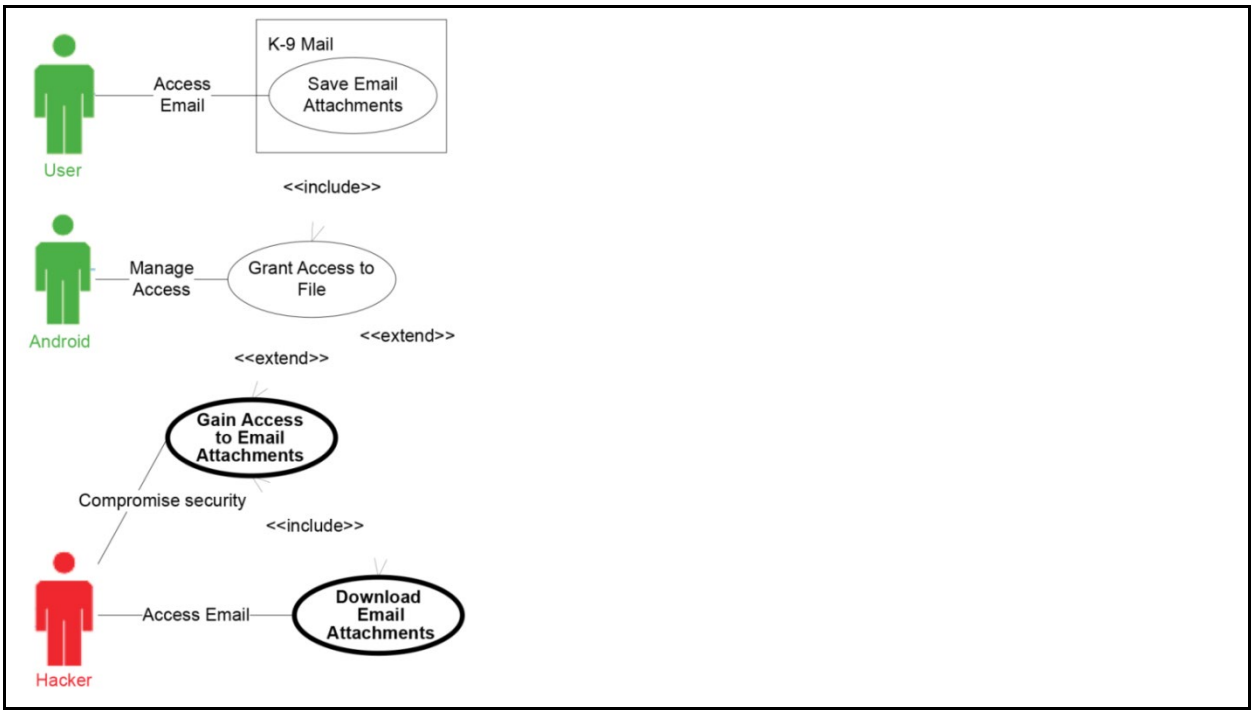

Figure 6. Misuse Case 2

Step 4: The overlooked use case corresponds to an overlooked security requirement.
We analyzed the misuse cases and developed the security requirements in Table 1.

Table 1. Security Requirements Derived from Use Cases

Requirement 1	1.1 Email contents shall be protected from unauthorized access. Email contents shall be stored in an area available only to the application (Android internal storage default configuration) and/or protected through encryption that cannot be decrypted using data available in Android external storage. 1.2 Processes with access to external storage shall not have the ability to view K-9 Mail contents in clear text. If External Storage is selected, a warning message or mitigation such as encryption is recommended.
Derived From	Misuse Case 1
Requirement 2	1.1 Attachments shall be protected from unauthorized access. Attachments shall be stored in an area available only to the application (Android internal storage default configuration) and/or protected through encryption that cannot be decrypted using data available in Android external storage. 1.2 Processes with access to External Storage shall not have the ability to view K-9 Mail attachments in clear text. If external storage is selected, a warning message or mitigation such as encryption is recommended.

Derived From	Misuse Case 2

Step 5: The use case and corresponding requirements statement is added to a requirements database.
The scope of the case study covers only the development of security requirements from malware analysis, so the developed requirements are not captured in a requirements database.

Step 6: The requirements database is used in future software development projects.
The case study does not have an associated mobile development project. There is no plan to use requirements developed in this case study in future software development projects, however the K-9 owners were notified of the flaw. There was no response to the email notification, so an existing change request for encrypting data potentially exposed by the flaw was updated to request a higher priority based upon the current existence of exploits.

Student Instructions

In your assignment you were given references to an actual system and malware that successfully attacked it. With these artifacts, work through all the steps of the malware analysis process. Document the results of each step. Discuss whether the new requirements should have been identified before the system was successfully attacked. Also discuss how the requirements, if implemented properly, could prevent successful attacks against similar systems.

Instructor notes

This case study was developed by Greg Alice (Alice & Mead, 2014), when he was a student doing an independent study with Nancy Mead (Mead & Morales, 2014; Mead, Morales, & Alice, 2015) at Carnegie Mellon University. It could be used as a classroom example, complete with the solution. Alternatively, a student assignment could be extracted from the early part of the case study, and the students could be asked to work through all steps of the analysis process. Note that if they are given this as an assignment, students should not be given the SEI Report (Alice & Mead 2014) or the IJSSE article (Mead, Morales, & Alice 2015) until they have completed the assignment, because it contains a solution.

However, a better exercise for students would be to pick a recent published malware example, a system that was successfully attacked by it and then have the students work through all steps of the analysis process shown in Table 1, keeping in mind that they are being asked to generate overlooked security requirements, vs. writing a patch for the hacked system. Most likely the case study work would be best assigned to a team over several weeks, or to an individual student as part of an independent study activity. A student who was already in the workforce, could potentially work on a case study in an area that was important to the student's organization.

Example solution

A solution for the K-9 case study is provided with the Case Study description. Once again, the entire case study could be used as a classroom example OR the students could learn the method and then be given the general parameters of the case study. They would then work through the steps to come up with a solution.

References

Adobe Systems, Inc. (2014). Security/Proactive Efforts. Retrieved November 12, 2014 from http://www.adobe.com/security/proactive-efforts.html

Ahmad, M., Musa, N., Nadarajah, R., Hassan, R. & Othman, N. (2013). Comparison Between Android and iOS Operating System in terms of Security. *8th International Conference on Information Technology in Asia (CITA)*, Bangi, Selangor, Malaysia.

Alice, Gregory & Mead, Nancy. Using Malware Analysis to Tailor SQUARE for Mobile Platforms (CMU/SEI-2014-TN-018). Software Engineering Institute, Carnegie Mellon University, 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=425994

Beuhring, A. & Salous, K. (2014). Beyond Blacklisting: Cyberdefense in the Era of Advanced Persistent Threats. *IEEE Security & Privacy,* 12.5 (2014), 90-93.

Bisiaux, J-Y. (2014). DNS threats and mitigation strategies. *Network Security* 2014.7 (2014), 5-9.

Fortinet, Inc. (2013, February). FortiGard Encyclopedia: Virus: Android/Claco.A!tr. Retrieved November 12, 2014 from http://www.fortiguard.com/encyclopedia/virus/#id=4585895

Google. (2012). Google's Approach to IT Security: A Google White Paper. Retrieved November 12, 2014 from https://cloud.google.com/files/Google-CommonSecurity-WhitePaper-v1.4.pdf

Google. (2014). Android Security Overview. Retrieved November 12, 2014 from https://source.android.com/devices/tech/security/

La Polla, M., Martinelli, F., & Sgandurra, D. (2013). A Survey on Security for Mobile Devices. *IEEE Communications Surveys & Tutorials*, 446-471.

Lipner, S. & Howard, M. (2005, March). The Trustworthy Computing Security Development Lifecycle. Retrieved November 12, 2014 from http://msdn.microsoft.com/en-us/library/ms995349.aspx

McMahon, J. (2014). An Analysis of the Characteristics of Cyber Attacks. *Discovery, Invention & Application* 1 (2014).

Mead, N.R., Morales J. A., Using Malware Analysis to Improve Security Requirements on Future Systems, Evolving Security & Privacy Requirements Engineering (ESPRE) Workshop, IEEE International Requirements Engineering Conference Proceedings, August 25, 2014, pp. 37-42

Mead, N.R., Morales, J. A., Alice, G. P., A Method and Case Study for Using Malware Analysis to Improve Security Requirements , International Journal of Secure Software Engineering, IGI Publishing, 6(1), pp.1-23, January-March 2015 Oracle. (2014). Software Security Assurance / Secure Development / Secure Coding Standards. Retrieved November 12, 2014 from http://www.oracle.com/us/support/assurance/development/secure-coding-standards/index.html

Oracle. (2014). Software Security Assurance / Secure Development / Secure Coding Standards. Retrieved November 12, 2014 from

http://www.oracle.com/us/support/assurance/development/secure-coding-standards/index.html

Paoli, C. (2013, February). New Android Malware Aims to Infect PCs. *Redmond Magazine*. Retrieved November 12, 2014 from http://redmondmag.com/articles/2013/02/06/android-malware-aims-to-infect-pc.aspx

Simpson, S. (ed). (2008, October). Fundamental practices for secure software development: a guide to the most effective secure development practices in use today. *SAFECode*. Retrieved November 12, 2014 from http://www.safecode.org/publications/SAFECode_Dev_Practices1108.pdf

Sood, A. K., Enbody, R.J. & Bansal, R. (2013). Dissecting SpyEye–Understanding the design of third generation botnets. *Computer Networks* 57.2 (2013), 436-450.

Tankard, C. (2011). Advanced Persistent threats and how to monitor and deter them. *Network security* 2011.8 (2011),16-19.