Task Hijacking auf Android Smartphones

Patrick Brenner - 28. Februar 2021
Spezielle Themen mobiler Kommunikationssysteme
Hochschule der Medien Stuttgart

Zusammenfassung

Das Android Multitasking ermöglicht Anwendern zwar Benutzerfreundlichkeit und App-Entwicklern Flexibilität, Angreifern jedoch auch die Möglichkeit andere Apps zu übernehmen. Diese Übernahme, das sogenannte Task-Hijacking, nutzen auch die kürzlich veröffentlichten Android-Schwachstellen StrandHogg und StrandHogg 2.0, wodurch nahezu jede Android App angegriffen werden kann. Die Vorgehensweise von Android Task Hijacking wird anhand dieser Schwachstellen und mit Blick auf mögliche Gegenmaßnahmen in dieser Arbeit beschrieben.

Abstract

While Android multitasking allows usability for users and flexibility for app developers, it also allows attackers to take over other apps. This method of taking control over apps, known as task hijacking, is also exploited by the recently published Android vulnerabilities StrandHogg and StrandHogg 2.0, which allows to attack any Android app. This paper describes Android Task Hijacking in the context of these new vulnerabilities and discusses possible countermeasures.

Keywords: Android; Smartphone Vulnerability; Task Hijacking; StrandHogg

1 Einleitung

Die Bildschirmzeit, die wir Menschen täglich auf unserem Smartphone verbringen, erreicht von Quartal zu Quartal neue Rekorde [1]. Bezahlungen werden mit dem Smartphone erledigt, Social-Media Apps besucht und unterschiedlichste Messenger Dienste verwendet. Innerhalb einer Minute ruft der Benutzer dabei oft mehrmals neue Apps auf und nutzt dadurch die Multitasking Funktionen des Betriebssystems. 74,4 % aller aktiv in Deutschland verwendeter Smartphones laufen im Dezember 2020 auf dem Google Betriebssystem Android [2] und damit dem Betriebssystem mit den meisten Schwachstellen-Meldungen (Common Vulnerabilities and Exposures (CVE)) im Jahr 2019 [3]. Eine solche Schwachstellen-Veröffentlichung sorgte im Dezember 2019 für mediales Aufsehen, als eine Android Schwachstelle alle der 500 meistverbreiteten Apps und alle Android-Versionen bedrohte [4]. Der Schwachstelle *StrandHogg* folgt nur wenige Monate später ihr Nachfolger *StrandHogg 2.0* [5]. Beide Ansätze nutzen das Android-Multitasking, also die Navigation zwischen den Apps, um diese anzugreifen. Diese Art von Angriff, bereits seit 2015 bekannt [6], wird auch als Task Hijacking bezeichnet und bedroht Smartphones und deren Benutzer auf verschiedene Arten. Sensible Daten wie Passwörter, Fotos und Kontakte können durch Phishing und gefälschte Berechtigungsabfragen (Permissions-Escalation) gestohlen oder ganze App-Interaktionen überwacht werden (Task-Monitoring) [6]. Die folgende Arbeit beschreibt den grundsätzlichen Ansatz des Task Hijackings (Kapitel 2), ordnet die Schwachstellen StrandHogg und StrandHogg 2.0 in diesen ein (Kapitel 3) und diskutiert mögliche Gegenmaßnahmen (Kapitel 4).

2 Android Task Hijacking

Im Zusammenhang mit dem Smartphone Betriebssystem Android wurde die Problematik Task Hijacking erstmals an der USENIX Konferenz 2015 in Washington D.C. und anhand des Papers "Towards Discovering and Understanding Task Hijacking in Android" veröffentlicht [6]. Task Hijacking wird dort als Designschwachstelle des Android Multitaskings (Kapitel 2.1) beschrieben, welche eine Malware durch bestimmte Konfiguration der Manifest.xml (Kapitel 2.2) nutzen kann, um die Zustandsübergänge beim Wechseln zweier Apps (Kapitel 2.3) zu manipulieren. Der dadurch entstehende Zustand wird als Hijacking State (Kapitel 2.4) bezeichnet.

2.1 Android Multitasking

Ein *Task* wird in der Android Developer Dokumentation [7] als eine Sammlung an *Activities* beschrieben, welche es dem Benutzer ermöglichen durch eine App zu navigieren, diese zu verlassen oder in diese zurückzukehren. Eine *Activity* ist dabei eine App-Komponente, die die grafische Benutzeroberfläche auf dem Bildschirm des Smartphones bereitstellt [8]. Android Multitasking bedeutet demnach, dass durch das Starten, Pausieren, Wiederaufnehmen und Verwerfen von Activities die Oberflächen einer oder mehrerer Apps aufgerufen werden [9]. Die Activities werden dabei in verschiedenen *Task-Back-Stacks* nach einer "First in, last out"-Objektstruktur angeordnet, wodurch sich Activities folgender Bezeichnung ergeben [7]:

Root-Activity
: Die Root-Activity ist die erste und damit unterste Activity eines Task-Back-Stacks. Wird eine App neu geöffnet, wird eine festgelegte Main-Activity zur Root-Activity.

Foreground-Activity
: Die auf dem Bildschirm sichtbare Activity eines Tasks ist die Foreground-Activity, weshalb dessen Task als Foreground-Task bezeichnet wird. Wird im Foreground-Task mit der Zurück-Taste zurücknavigiert, wird die Foreground-Activity verworfen und die darunterliegende Activity des Task-Back-Stacks zur neuen Foreground-Activity.

Background-Activity
: Background-Activities sind alle in den Task-Back-Stacks vorhandenen Activities, die keine Foreground-Activities sind. Background Activities sind für den Benutzer also nicht direkt sichtbar. Tasks, die ausschließlich aus Background-Activites bestehen, werden als Background-Tasks bezeichnet.

Da Android das Entwickeln von Single-Activity Apps empfiehlt [10], werden Activities vermehrt einmalig aufgerufen und die Navigation zwischen den Seiten einer App (z.B. der Aufruf einer E-Mail aus dem Posteingang) durch *Fragments* gesteuert. Trotzdem ist mindestens eine Activity notwendig, um Apps aufzurufen [9]. Diese Navigation zwischen Activities und Tasks zeigt Abbildung 1. Durch das Starten der App Gmail (1) wird die Main-Activity ConvList als Root-Activity des Gmail Tasks gestartet und dadurch zur Foreground-Activity (2). Durch den anschließenden Aufruf einer E-Mail erscheint zwar deren Inhalt und damit eine neue Oberfläche (3), es wird jedoch keine neue Activity aufgerufen. Dies ist auf den Single-Activity Stil, den Gmail an dieser Stelle implementiert, zurückzuführen. Durch Aufruf eines externen Links innerhalb der Mail wird die Webbrowser-App Google-Chrome aufgerufen. Die Activity Chrome wird zur Foreground-Activity, jedoch im Task-Back-Stack Gmails (4). Ob eine Activity in einem neuen Task gestartet wird oder, wie im Beispiel, auf dem Task-Back-Stack des bisherigen Tasks platziert wird, konfiguriert der Entwickler der

App in der Manifest.xml (Kapitel 2.2). Durch verschiedene Attribute kann das Verhalten der Activities an dieser Stelle definiert und von bösartiger Malware auch ausgenutzt werden.

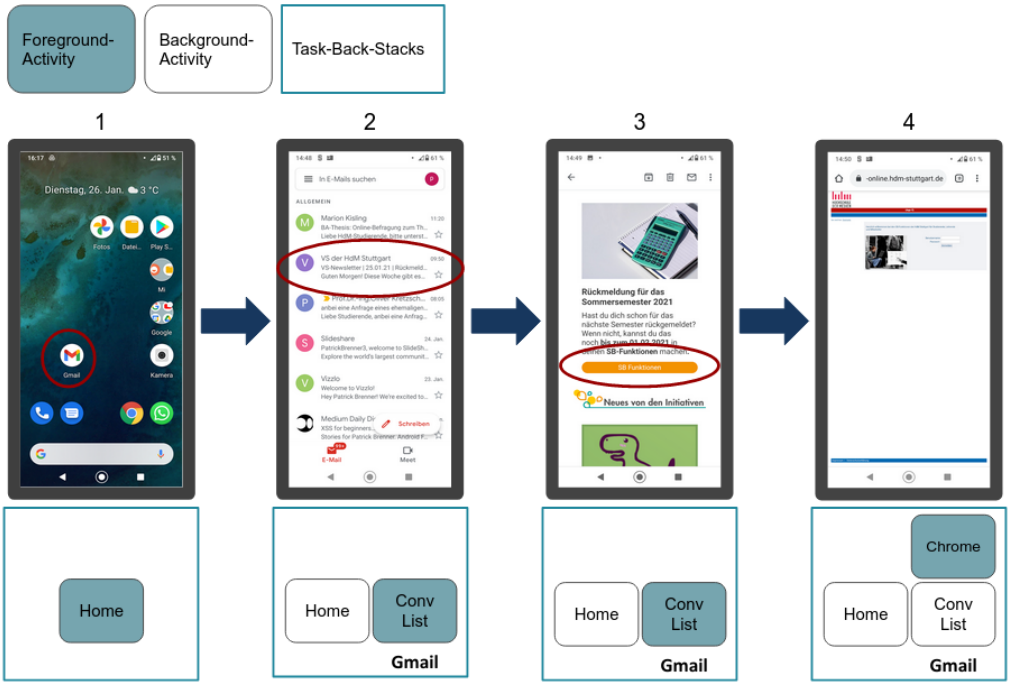

Abbildung 1: Android Multitasking Beispiel eines E-Mail Aufrufs

2.2 Manifest.xml

Jede entwickelte Android-App benötigt nach Android Developer Guide eine Manifest.xml im Wurzelverzeichnis des Projekts [11]. Diese Datei beschreibt wesentliche Informationen der App für die Android-Build-Tools, das Android-Betriebssystem und Google Play [11]. Auch die Activities, die innerhalb einer App gestartet werden, sind in der Manifest.xml aufgeführt und werden unter anderem durch folgende Attribute in ihrem Startverhalten konfiguriert [11]:

LaunchMode

Der *LaunchMode* gibt an, in welchem Modus die Activity gestartet werden soll. Durch das Setzen von Activity-Flags wird festgelegt, in welchem Task und mit welchen Bedingungen die Activity aufgerufen wird. Die unterschiedlichen Modi werden in Kapitel 2.3 als Bedingungen γ beschrieben [7].

TaskAffinity

Die *TaskAffinity*, die als String übergeben wird, definiert eine Beziehung der Activity zu einem Task. Dadurch gibt sie den Task-Back-Stack einer Activity an. Die TaskAffinity ist ein zentrales Attribut des Task Hijackings, weil dadurch auf Task-Back-Stacks anderer Apps zugegriffen werden kann [7].

AllowTaskReparenting

Durch *AllowTaskReparenting* wird einer Activity vorgeschrieben, ob in einen externen Task gewechselt werden kann. Beim Wert "true" wird das Wechseln erlaubt, beim Wert "false" verboten. Es ist dadurch zwar möglich die Activities einer App

innerhalb des eigenen Tasks zu halten, jedoch nicht das Verhalten fremder Activities einzugrenzen [7].

2.3 Task State Transition Model

Bereits Abbildung 1 verdeutlicht, welche Unterschiede sich beim Start einer Activity ergeben. Die Gmail Activity ConvList wird als neuer Task, die Activity Chrome als Activity einer anderen App im selben Task, gestartet. Würde der Benutzer im Beispiel auf einen anderen Link der E-Mail klicken, der beispielsweise zu einer weiteren App führt, ergeben sich neue Task-Back-Stacks. Um diese Variablen zu berücksichtigen und den zeitlichen Verlauf der Task-Zustände zu modellieren, wird nach Chuangang et al. [6] ein Task State Transition Model mit folgenden Komponenten definiert:

Task State S
Der *Task State S* definiert den Zustand, in welchem sich die Task-Back-Stacks, also die Foreground- und Background-Activities befinden. Verändert sich dieser Zustand nicht, sind auch die Zustände der Tasks unverändert.

Transition T
Die *Transition T* beschreibt den Zustandsübergang, der eine Veränderung des Task States auslöst. Das Event e und die Bedingungen γ steuern dabei, wie sich die State Transition auf die Task-Back-Stacks auswirkt.

Event e
Das *Event e* beschreibt den Auslöser einer State Transition. Folgende Auslöser werden dabei im Verlauf der Arbeit verwendet [8]:

startActivity():
Eine Activity wird gestartet.

startActivities():
Mehrere Activities werden aufeinanderfolgend gestartet. Die letzte gestartete Activity des Aufrufs wird zur Foreground-Activity.

back():
Der Benutzer navigiert über die Back-Taste zurück. Standardmäßig wird dadurch die Foreground-Activity verworfen.

Bedingungen γ
Durch Angabe der *Bedingungen* γ wird das Verhalten einer vom Event e ausgelösten State Transition beschrieben. So wird definiert, auf welche Art eine Activity gestartet oder das Zurücknavigieren konfiguriert wird. Diese Konfigurationen werden als Attribute der Android Manifest.xml (Kapitel 2.2) oder als Intent-Flag der Klassen startActivity() oder startActivities() definiert [9]. Die folgenden Bedingungen für das Starten einer Activity finden dabei im weiteren Verlauf der Arbeit Verwendung [7]:

default
Das System platziert die Activity im Task-Back-Stack des Activity Aufrufs (Foreground-Task). Wird eine Activity aus dem Home (Android Launcher) gestartet, wird überprüft, ob die in der TaskAffinity definierte Task als Background-Task existiert. Ist dies nicht der Fall, wird ein neuer Task-Back-Stack erzeugt.

newTask
Das System startet mit *newTask* einen zusätzlichen Task. Existiert bereits ein durch die TaskAffinity definierter Background-Task, platziert das System die Activity auf diesem Task-Back-Stack.

singleTask
: Das System erstellt durch *singleTask* eine Root-Activity eines neuen Tasks. Befindet sich jedoch eine Instanz derselben Activity in einem Background-Task, leitet das System die Activity an diesen Background-Task weiter.

singleInstance
: Das System wendet eine identische Vorgehensweise zu singleTask an, erlaubt allerdings keine anderen Activities in diesem Task zu starten. Die Activity ist immer das einzige Mitglied ihres Task-Back-Stacks.

Abbildung 2 stellt das Multitasking Beispiel aus Kapitel 2.1 im Task State Transition Model und mit folgenden Transitions T dar:

T1: (S0, S1, $e^{Home:startActivity(ConvList)}$, $\gamma^{default}$)
Aus dem Home wird das Event startActivity(ConvList) - die Root-Activity der App Gmail - ausgelöst. Dadurch wird ein neuer Task-Back-Stack Gmail erzeugt und die Activity ConvList gestartet.

T2: (S1, S2, $e^{ConvList:startActivity(Chrome)}$, $\gamma^{default}$)
Durch das Klicken des Links wird das Event startActivity(Chrome) ausgelöst, welches durch die Bedingung default eine Activity Chrome innerhalb des Gmail Task-Back-Stacks platziert. Der Aufruf der E-Mail, den Gmail durch die Verwendung von Fragments ohne zusätzliche Activity umsetzt, wird hier ignoriert, da diese Interaktion zwar die Oberfläche ändert, jedoch keine Auswirkungen auf das Task State Transition Model hat.

T3: (S2, S1, $e^{Chrome:back}$, $\gamma^{default}$)
Durch das Zurücknavigieren wird die Activity Chrome verworfen.

T4: (S1, S0, $e^{ConvList:back}$, $\gamma^{default}$)
Durch das Zurücknavigieren wird die Activity ConvList verworfen.

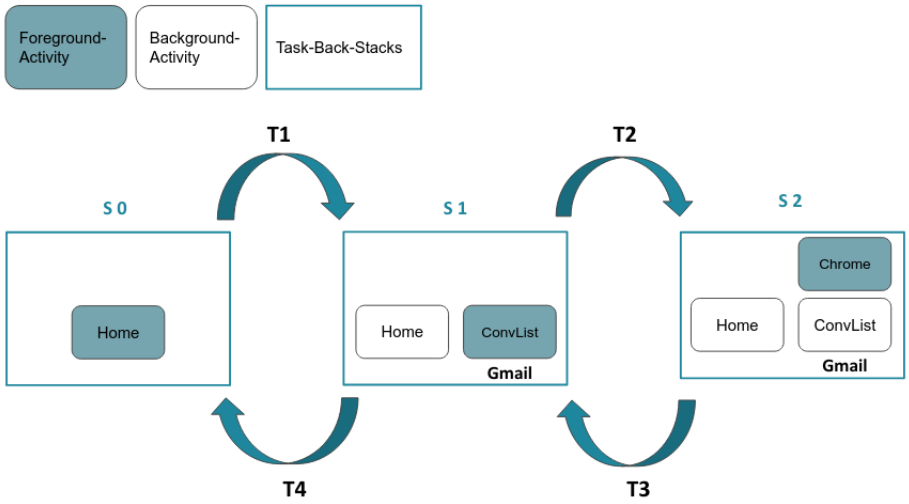

Abbildung 2: Task State Transition Model eines E-Mail Aufrufs

2.4 Hijacking State Transition

In den bisherigen Abschnitten wird das Android Multitasking als benutzerfreundliche Funktion beschrieben. Der mit wenig Aufwand und für jede App zu konfigurierende Aufruf von Activities anderer Apps kann aber nicht nur gutartig, sondern auch bösartig verwendet werden. Gelingt es einem Angreifer einen Task-Back-Stack mit einer gutartigen und einer bösartigen Activity zu besetzen, tritt ein Hijacked Task State ein [6]. Ein Zustand, welcher das Multitasking ausnutzt und damit Benutzer auf Malware-Activities umleitet. Der Zustandsübergang (State Transition), der einen solchen Hijacked State auslöst, wird als Hijacking State Transition (HST) bezeichnet und in zwei Arten unterteilt [6].

HST 1: Die bösartige Activity liegt auf dem Task-Back-Stack der Opfer-App.

HST 2: Die gutartige Activity liegt auf dem Task-Back-Stack der Malware.

In beiden Fällen kann ein Angreifer den Hijacked Task State ausnutzen und den Benutzer täuschen. Ein Beispiel eines HST2 wird in Abbildung 3 veranschaulicht. Dabei verwendet die Malware M1 folgende Attribute in der Manifest.xml, um den Hijacked State auszulösen:

```
<activity
    android:name="M1"
    android:taskAffinity="com.android.chrome" />
```

Der Benutzer ruft in der App PayPal eine externe Website und damit eine Google-Chrome Activity (1) auf. Diese erzeugt durch die Bedingung γ newTask einen neuen Task bzw. sucht einen bereits gestarteten Task identischer TaskAffinity. Da ein Angreifer einen solchen durch die manipulierte TaskAffinity zuvor als Background-Task startet, landet die Google-Chrome Activity auf dem Task-Back-Stack der Malware (HST2). Aus dem Hijacked-State (2) wird durch das Zurücknavigieren nicht wie gewünscht die PayPal-Activity A1, sondern die Malware-Activity M1 zur Foreground-Activity. Dadurch kann ein Angreifer beispielsweise eine Phishing Seite einblenden und Benutzerdaten stehlen (3). Diese Art der Attacke wird wegen des Zurücknavigierens, welches zur Malware-Activity führt, auch als Back-Hijacking bezeichnet [6].

3 StrandHogg Schwachstellen

Obwohl die Problematik des Android Multitaskings 2015 veröffentlicht wurde [6], gibt es nur wenig bekannte Beispiele von Malware, die diese Schwachstellen für Angriffe ausnutzten. Erst als im Dezember 2019 das norwegische App-Security Unternehmen Promon über die Schwachstelle StrandHogg berichtet, werden 36 bösartige Task-Hijacking Apps auf verschiedenen Android Geräten entdeckt [4]. Da nahezu alle Android Apps dadurch angreifbar werden, findet StrandHogg medial schnell Beachtung. Der “böse Zwillling” StrandHogg 2.0 folgt nur wenige Wochen später im Mai 2020 [5]. Während StrandHogg durch TaskAffinity Einträge in der Manifest.xml bösartige Activities auf dem Task-Back-Stack der Opfer-App platziert, wird bei StrandHogg 2.0 ein Android-Bug ausgenutzt, der dies ohne TaskAffinity Eintrag ermöglicht. Bei beiden Schwachstellen wird eine HST1 eingesetzt und der Benutzer dadurch mit Phishing, Permissions-Escalation sowie Task-Monitoring bedroht.

3.1 StrandHogg

StrandHogg verwendet einen bereits beschriebenen Ansatz (Abbildung 3) und gelangt durch die Manipulation der TaskAffinity in den Task-Back-Stack einer Opfer-App [4]. Jedoch wird

Abbildung 3: Back Hijacking Attacke (HST2) auf PayPal

der Hijacked State durch eine HST1, also das Platzieren der Malware-Activity auf dem gutartigen Task-Back-Stack, erreicht. Wird die Opfer-App zuvor gestartet und befindet sich im Task-Back-Stack, platziert sich die Malware-Activity als Oberste des Stapels und wird zur Foreground-Activity. Dadurch kann die Malware Berechtigungen in falschem Namen erfragen (Permissions-Escalation) oder mit einer identisch zur Opfer-App aussehenden Anmeldeseite sensible Daten stehlen [4]. Eine solche Phishing-Malware, welche die StrandHogg Schwachstelle ausnutzt, zeigt Abbildung 4 an einem Beispiel. Dabei verwendet die Malware M1 folgende Attribute der Manifest.xml und die aufgeführten Transitions T, um den Hijacked State auszulösen [4]:

```
<activity
    android:name="M1"
    android:taskAffinity="com.paypal.android.p2mobile"
    android:allowTaskReparenting="true" />
```

T1: (S0, S1, $e^{Home:startActivities(M1,Ablenkung)}$, $\gamma^{newTasks}$)
Die Activities M1 und Ablenkung werden als gutartig getarnte App (z.B. eine Wetter-App) durch startActivities() gestartet. Da die letzte gestartete Activity dieses Aufrufs zur Foreground-Activity wird, zeigt der Bildschirm des Benutzers diese harmlos wirkende Ablenkung (Wetter-App) an. Im Hintergrund platziert sich die Activity M1 durch die manipulierte Angabe der TaskAffinity unbemerkt auf dem Task-Back-Stack PayPals.

T2: (S1, S2, $e^{Ablenkung:back}$, $\gamma^{default}$)
Durch das Zurücknavigieren und Verwerfen dieser Ablenkung, landet der Benutzer wieder auf dem Home-Screen. Durch einen erneuten Aufruf der App PayPal könnte die oberste Activity des Task-Back-Stack (M1) zur Foreground-Activity werden. Diese Malware-Activity kann, wie in Abbildung 3 dargestellt, eine Phishing-Seite abbilden und dadurch sensible Benutzerdaten stehlen.

Je nach Ziel der Malware ist es durch StrandHogg jedoch auch möglich, den Task-Back-Stack des Opfers zu leeren und anschließend als Root-Activity neu zu starten. Dadurch wird die Interaktion des Benutzers mit der App durch Task-Monitoring überwacht [4]. Da StrandHogg grundsätzliche Funktionen des Android-Multitaskings verwendet, wurde bisher (Stand 28.02.2021) kein Patch gegen die Schwachstelle veröffentlicht [4].

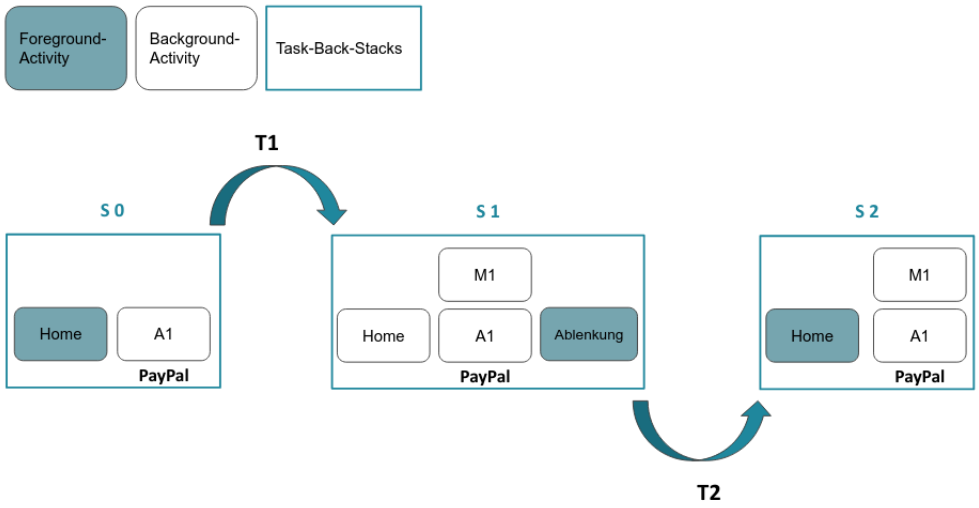

Abbildung 4: Task State Transition Model einer StrandHogg Attacke

3.2 StrandHogg 2.0

Parallel zur StrandHogg-Veröffentlichung berichtet Promon Google durch einen Report-Issue im Dezember 2019 vom gefährlicheren Nachfolger StrandHogg 2.0 [5]. Der "böse Zwilling" basiert auf einem Android-Bug der Versionen 9 und geringer, wodurch zum Zeitpunkt des Reports 91,8 % aller Android Geräte betroffen sind [5]. Da keine TaskAffinity Angaben in der Manifest.xml notwendig sind, ist Malware, die die StrandHogg 2.0 Schwachstelle ausnutzt, für Malware-Detektoren schwer erkennbar und kann simultan mehrere Apps angreifen [5]. Ein Bug der Android Klasse startActivities() führt dazu, dass Malware ein in Abbildung 5 veranschaulichtes Szenario mit folgenden Transitions T ausnutzen kann:

T1: (S0, S1, $e^{M1:startActivities(A1,M2,A2,M2,Ablenkung)}$, $\gamma^{A1,A2,Ablenkung:newTask;M2:default}$)
Durch den Malware-Aufruf startActivities(A1,M2,A2,M2,Ablenkung) platziert der Angreifer seine Malware-Activity M2 auf den Task-Back-Stacks der Apps Facebook und PayPal. Nur die explizit mit der Bedingung newTask gestarteten Activities werden bei startActivities() als neuer Task gestartet(A1,A2,Ablenkung), weshalb die Activity M2 jeweils auf den zuvor gestarteten Task-Back-Stacks platziert wird ohne dessen TaskAffinity anzugeben. Durch das Starten einer Ablenkung, die zur Foreground-Activity wird, wird dem Benutzer das Task-Hijacking verborgen.

T2: (S1, S2, $e^{Ablenkung:back}$, $\gamma^{default}$)
Navigiert der Benutzer zurück und öffnet in der Folge die Background-Tasks Facebook oder PayPal, wird M2 zur Foreground-Activity und der Angreifer kann beispielsweise eine Phishing Seite einblenden oder Berechtigungen unter falscher App-Identität erlangen (Permissions-Escalation). Je nach Ziel der Malware ist es auch bei StrandHogg 2.0 möglich, den Task-Back-Stack des Opfers zu leeren und anschließend als Root-Activity zu überwachen [4].

Der Bug, welcher das Einfallstor der StrandHogg 2.0 Schwachstelle ist, wurde für die Android-Versionen 8, 8.1 und 9 inzwischen behoben [5]. Nach Systemupdates der genannten Versionen werden Activities (M2) des startActivities() Aufrufers (M1) nicht mehr auf dem Stapel anderer Tasks (Facebook und PayPal) platziert [12]. Für die Android Versionen 7 und geringer ist diese Schwachstelle jedoch weiterhin vorhanden [5].

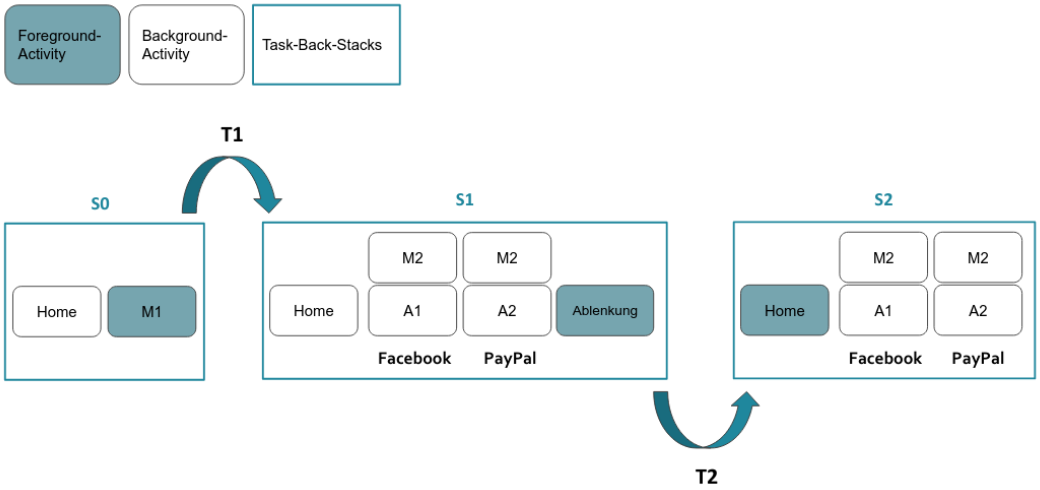

Abbildung 5: Task State Transition Model einer StrandHogg 2.0 Attacke

3.3 Gegenmaßnahmen

Da Task-Hijacking Schwachstellen grundlegende Funktionen des Betriebssystems ausnutzen, ist es für Benutzer und Entwickler nur bedingt möglich entgegenzuwirken. Wie an StrandHogg 2.0 zu sehen, können Schwachstellen oft nur auf Betriebssystemebene geschlossen werden [6]. Denkbare Ansätze dieser Ebene werden in Kapitel 4 thematisiert. Trotzdem können sich auch Benutzer und Entwickler durch ein bestimmtes Vorgehen schützen.

Benutzer

Da nicht auf die Bedingungen einer Activity oder die Attribute der Manifest.xml eingewirkt werden kann, ist es für einen Benutzer nur bedingt möglich sich gegen StrandHogg und StrandHogg 2.0 zu schützen. Selbst Mobile-Security-Software, die Malware erkennen oder die Google Malware-Abwehr “GooglePlay Protect” [13] bieten hier keinen garantierten Schutz. Vor allem Hostile-Downloader werden nicht zuverlässig von diesen Schutzmechanismen erkannt [4]. Diese Hostile-Downloader installieren, wie im Beispiel StrandHogg, die Malware, welche aufgrund bestimmter Merkmale (z.B. fremde TaskAffinity) nicht in den Google PlayStore gelangt [13]. Trotzdem kann der Benutzer durch aufmerksames Nutzerverhalten Task-Hijacking bzw. die ausnutzende Malware erkennen. Werden beispielsweise Anmeldedaten einer App, in welche der Benutzer bereits eingeloggt war, nach Wiederaufnahme erneut abgefragt oder eine unstimmig wirkende Anmeldeoberfläche erkannt, sind dies Indikatoren [4]. Zudem sollte bei Berechtigungsabfragen (Kamera, Bilder, Kontakte etc.) aufmerksam auf den Namen der App geachtet und diese Freigaben über die Android Systemeinstellungen regelmäßig überprüft werden. Taucht ein Verdacht auf, kann durch Android Debugging Tools [14] überprüft werden, welche Tasks und Activities im Task-Back-Stack gestartet sind, um dementsprechende Malware zu erkennen und zu deinstallieren. Gegen

System-Bugs wie im Fall StrandHogg 2.0, ist ein Schutz nur durch Installieren von Systemupdates möglich. Diese sollten regelmäßig und zeitnah nach der Veröffentlichung installiert werden.

Entwickler

Auch App-Entwickler können sich nur bedingt gegen Schwachstellen wie StrandHogg und StrandHogg 2.0 schützen, da zwar die eigenen Activities gesteuert, aber die App nicht ausreichend gegen paketfremde Activities abgesichert werden kann. Wie den StrandHogg Veröffentlichungen [[4, 5]] zu entnehmen, konnten sich 100 % der 500 meistverbreiteten Apps nicht ausreichend schützen. Trotzdem bieten Attribute der Manifest.xml Schutzmechanismen. Gegen StrandHogg empfiehlt Promon für alle Activities einer App die TaskAffinity mit einem leeren String in der Manifest.xml anzugeben, damit die Activities der gutartigen App keine Beziehung zu einer anderen App haben. Dies mildert das Risiko bis zu einem gewissen Grad, kann jedoch, wie im Beispiel StrandHogg 2.0 zu sehen ist, Task-Hijacking nicht immer verhindern [4]. Gegen beide StrandHogg Varianten empfiehlt Promon den App-Entwicklern Activities mit den Bedingungen singleTask oder bestenfalls singleInstance zu starten. Dadurch ist es einer Malware-Activity nicht mehr möglich, den Bug auszunutzen und auf den Task des Opfers zu gelangen [5]. Jedoch wird durch all diese Attribute und Bedingungen auch das Android Multitasking sowie die Benutzerfreundlichkeit einer App stark beeinträchtigt.

3.4 Übersicht StrandHogg und StrandHogg 2.0

	StrandHogg	**StrandHogg 2.0**
Betroffene Versionen	Alle Android Versionen (kein Patch veröffentlicht)	Android Version 9 oder geringer (Patch für Version 8, 8.1, 9 im Mai 2020 veröffentlicht)
Betroffene Anwendungen	100 % der 500 meist verbreiteten Apps	100 % der 500 meist verbreiteten Apps
Simultaner Angriff	Nein	Ja
Manifest.xml	TaskAffinity der Opfer-App	-
Events e	startActivity() startActivities()	startActivities()
Bedingungen γ	newTask	Opfer-Activity: newTask Malware-Activity: Default
HST	HST1	HST1
Bedrohungen	Phishing Permissions-Escalation Task-Monitoring	Phishing Permissions-Escalation Task-Monitoring
Malware	36 Malware Apps entdeckt	-
Gegenmaßnahmen Benutzer	Aufmerksame Nutzung (nur bedingter Schutz)	Android Systemupdate der Versionen 8, 8.1 und 9
Gegenmaßnahmen Entwickler	TaskAffinity = leerer String (nur bedingter Schutz) singleTask oder singleInstance	singleTask oder singleInstance

Tabelle 1: StrandHogg und StrandHogg 2.0 im Vergleich

4 Ausblick

Task Hijacking ist als Android Problematik seit knapp 6 Jahren bekannt und zeigt an den Schwachstellen StrandHogg und StrandHogg 2.0, dass die Grundfunktion des Android-Multitaskings auch weiterhin Angriffsfläche bietet. Sollten zukünftig bösartige Hacker solche Schwachstellen entdecken, würde dies weitreichendere Folgen für den Benutzer haben. Um den bei StrandHogg 2.0 genutzten Bug mit Updates zu beheben, benötigte Google 6 Monate [5]. Zeit, die bösartige Angreifer nutzen könnten, um zahlreiche Android Benutzer und nahezu alle Apps über Monate zu bedrohen. Um das System dagegen abzusichern, muss eine gute Balance zwischen sicherem Task-Management und benutzerfreundlichem Multitasking gelingen. Vor allem die Möglichkeit der Task-Übernahme durch Einträge in der TaskAffinity muss dabei eingeschränkt werden. Denkbar wären beispielsweise zusätzliche Attribute in der Manifest.xml, welche die Tasks kontrollierbarer machen. Beispielsweise könnten paketfremde Activities durch ein Boolean-Attribut abgewiesen oder nur Apps desselben App-Herstellers erlaubt werden. Solche Attribute wurden bereits 2015 empfohlen [6], jedoch in keiner vergleichbaren Form im Android Betriebssystem implementiert. Damit wurde die Möglichkeit verpasst, StrandHogg und StrandHogg 2.0 präventiv zu vermeiden. Dies wäre auch mit Zertifikaten denkbar, die die TaskAffinity Beziehungen zweier Apps mit einer gegenseitigen Authentifizierung absichern. Google könnte nur Apps, die bereits für den Play Store durch “GooglePlay Protect” geprüft wurden, Zertifikate ausstellen und Multitasking erlauben. Für Malware, die wie bei StrandHogg über Hostile-Downloader auf die Smartphones gelangt, wäre demnach kein Task Hijacking durchführbar. Eine zusätzliche Sicherheit kann zudem durch ständig optimierte Malware-Erkennung erreicht werden, wodurch Schadsoftware nicht in den PlayStore gelangen und zusätzlich auf den Smartphones erkannt werden kann. Hier ist Android mit “GooglePlay Protect”, welches sich durch Machine-Learning Prozesse automatisch an neue Malware anpasst, auf einem guten Weg [14].

Literatur

[1] App Annie. *Mobile App Usage Worldwide*. URL: https://www.appannie.com/de/insights/market-data/mobile-app-usage-surged-40-during-covid-19-pandemic/.

[2] Kantarworldpanel. *Smartphone os market share*. URL: https://www.kantarworldpanel.com/global/smartphone-os-market-share/.

[3] CVE-Details. URL: https://www.cvedetails.com/top-50-products.php?year=2019.

[4] Promon. *StrandHogg*. Techn. Ber. URL: https://promon.co/security-news/strandhogg/.

[5] Promon. *StrandHogg 2.0*. Techn. Ber. URL: https://promon.co/security-news/strandhogg-2-0-new-serious-android-vulnerability/.

[6] Chuangang Ren u. a. "Towards Discovering and Understanding Task Hijacking in Android". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, S. 945–959. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ren-chuangang.

[7] Android Developers Documentation. *Understanding Tasks and Back Stack*. URL: https://developer.android.com/guide/components/activities/tasks-and-back-stack.

[8] Android Developers Documentation. *Activity*. URL: https://developer.android.com/reference/android/app/Activity.

[9] Android Developers Documentation. *Introduction to Activities*. URL: https://developer.android.com/guide/components/activities/intro-activities.

[10] Android Developers Blog. *Use Android Jetpack to Accelerate Your App Development*. URL: https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html?m=1.

[11] Android Developers Documentation. *App Manifest Overview*. URL: https://developer.android.com/guide/topics/manifest/manifest-intro.

[12] GoogleGit. URL: https://android.googlesource.com/platform/frameworks/base/+/a952197bd161ac0e03abc6acb5f48e4ec2a56e9d.

[13] Google Developers. *Play Protect*. URL: https://developers.google.com/android/play-protect.

[14] Android Studio. *SDK Platform Tools*. URL: https://developer.android.com/studio/releases/platform-tools.