What the App is That? Deception and Countermeasures in the Android User Interface

LUCAS COPI

Introduction

- Smartphone and Tablet Usage is becoming increasingly popular
- It has become the primary way of accessing digital media in the US
- Devices carry with them a wealth of confidential user data
- This has created attention from cybercriminals

Introduction continued

- Paper investigates vulnerabilities stemming from devices running multiple apps at the same time
- Most devices allow one app to run in the foreground while multiple apps continue running background processes
- This can lead to malicious background apps hijacking user devices
- Paper investigates specific style of attacks known as GUI attacks
- Create and demonstrate new systems to alert users to potential malicious GUI activity

Background

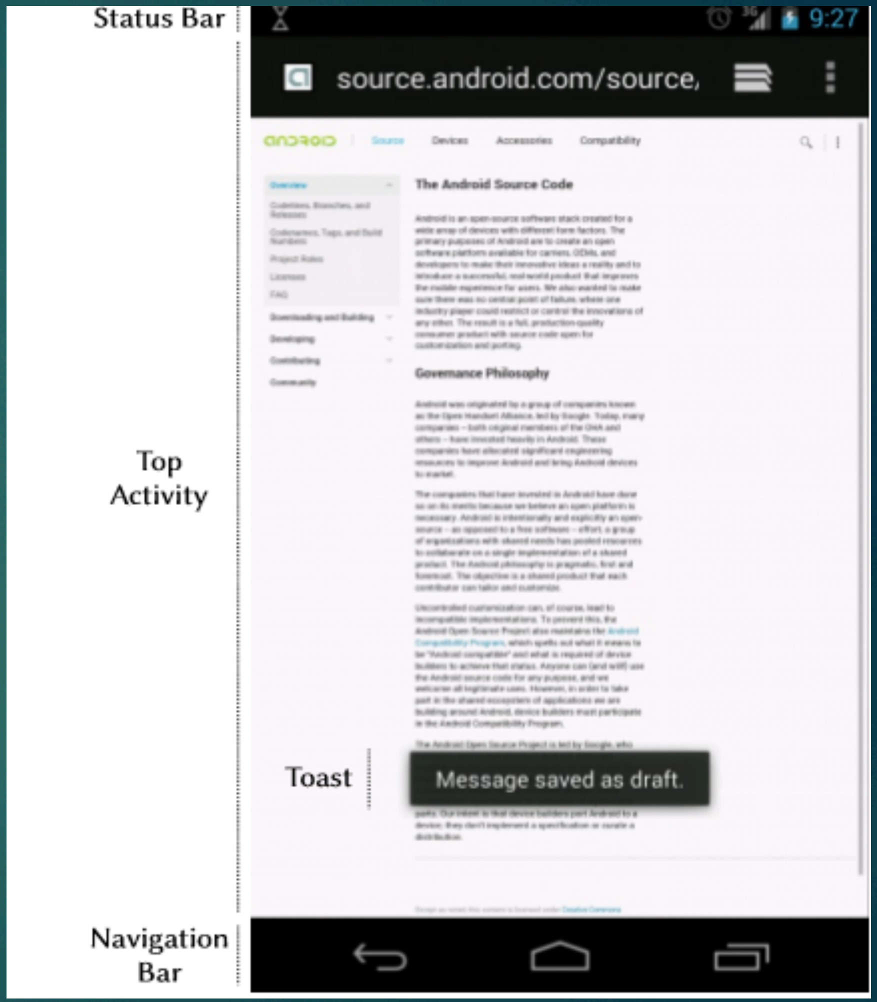
- Android platform is based on the Linux OS and is designed for touch screen devices
- Each app on a device runs in isolation from others except for well-defined communication channels
- Apps are contained in apk files that are signed as a security measure
- Apps are composed of different developer-defined components: activity, service, broadcast receiver, and content provider

Background Continued

- Activity defines a GUI and its interactions with the user input
- Service performs long running tasks in the background
- Broadcast Receiver responds to specific system-wide messages
- Content provider manages data shared with other components (can be within same app or with different apps)
- Permissions:
  - All apps that perform sensitive operations need specific permissions
  - These are granted at the time of installation
  - Some permissions can only be granted to system apps
- Required permissions and other properties are stores in an apps manifest file

Android Graphical Elements

- Apps draw graphical elements by instantiating system components: views, windows, and activities
- A view is the basic UI building block: buttons, text fields, images are all examples
- Activities are controllers that are associated with views and define actions when view elements are activated



Graphical elements continued

- Activities are managed by the activity manager service and implemented with an activity stack
  - The activity on top of the stack is shown to the user
- Each app can reorder the activities it owns
- Users request activity switching by using navigation bar buttons
- Windows are virtual surfaces that host the graphical content contained by the views
  - Windows are normally automatically managed by the window manager system service

GUI confusion attacks

- Attack vectors:
  - Draw on top
  - App switching
  - Full Screen
  - Enhancing techniques

Category	Attack vector	Mentioned in
Draw on top	UI-intercepting draw-over	[3], [5]
	Non-UI-intercepting draw-over	[3], [4], [5]
	Toast message	[3], [10]
App switch	*startActivity* API	[6]
	Screen pinning	—
	moveTaskTo APIs	—
	killBackgroundProcesses API	—
	Back / power button (passive)	—
	Sit and wait (passive)	—
Fullscreen	non-“immersive” fullscreen	—
	“immersive” fullscreen	—
	“inescapable” fullscreen	—
Enhancing techniques	*getRunningTask* API	[5]
	Reading the system log	[11]
	Accessing proc file system	[6], [12]
	App repackaging	[13], [14], [15]

GUI attacks-draw on top

- Malicious code attempts to draw graphical elements over other apps
  - Done by adding graphical elements to a window that is placed over the top activity
  - Windows are opened using addView API which accepts flags
  - These flags determine whether the window intercepts user input or lets it pass through, the type, and the screen region
- Types of possible attacks include: UI-intercepting draw-over with the priority phone flag and non UI-intercepting draw-over which forwards user input to underlying windows

GUI attacks-app switch

- App switching attacks steal focus from the top app and replaces it with an activity from the malicious app
  - Two types: active and passive—active replaces currently running app while passive waits for specific user input
- Several system API's give apps power to modify the activity stack
  - Startactivity, movetaskto, killBackGroundProcesses

GUI attacks-Fullscreen

- Apps have the ability to enter full screen mode which covers the navigation bar
- This can be exploited to create fake navigation bars to fool the user
- Android has some features built in to mitigate such attacks
- However, they can be circumvented with specific flags and input values of GUI-related API’s

GUI attacks-enhancing techniques

- Other techniques can be used along with the previous attacks vectors to increase the effectiveness of the attacks
- Techniques to detect how the user is interacting with the system allow malicious apps to mount more pointed attacks
  - i.e. waiting for a banking app to open
- Apps can read messages in the system logs for clues about the on screen activity
- getRunningTasks API and the proc filesystem give information about the current running apps and activities

Android GUI API

- Researchers designed a tool to explore every possible state of the startActivity API
- As previously noted: startActivity API can be used to open activities on top of others creating the possibility for a GUI attack
- The tool also explored window creating scenarios
  - attempt to find a collection of parameters that would allow the window to cover the entire screen and leave the user no way to close it

startActivity API

- Three things influence how an activity is placed on the stack: type of calling component, launch mode attribute, flags
- Program found three scenarios when an activity can be drawn on top of another:
  - The NEW_TASK flag is used
  - The activity has the single instance launch mode
  - Has a combination of NEW_TASK and CLEAR_TASK flags, NEW_TASK and MULTIPLE_TASK with launch mode that is not single task and CLEAR_TASK flag with single task launch mode

Inescapable full screen window

- Three ways for an app to modify a window to carry out a GUI attack
  - Modify window type
  - Specify flags that determine the windows layout
  - Calling the setSystemUiVisibility API with specific flags
- The tool found combinations using the SYSTEM_ERROR flag could send a window into an inescapable full screen leaving the user to use the navigation bar or close the window

TYPEs	TOAST, **SYSTEM_ERROR**, **PHONE**, **PRIORITY_PHONE**, **SYSTEM_ALERT**, **SYSTEM_OVERLAY**
Layout flags	IN_SCREEN, NO_LIMITS,
System-UI Visibility flags	HIDE_NAVIGATION, FULLSCREEN, LAYOUT_HIDE_NAVIGATION, LAYOUT_FULLSCREEN, *IMMERSIVE*, *IMMERSIVE_STICKY*

Static Analysis

- Researchers designed a tool to study real world implications of GUI attacks
- The tool studied how the previous techniques are used by benign and malicious apps
- The tool was used to automatically detect potentially malicious of the techniques

Tool description

- The tool takes an app's apk file and outputs a summary describing any potentially malicious aspects that could be used to carry out a GUI attack
- Checks app permissions, identifies calls to API's detailed above, applies backward program splicing to check values for said API's
- The tool then analyzes the apps control flow
- Using all of this it determines whether to flag the app as malicious

App Classification

- An app is classified as suspicious based on three conditions
  - The app uses a technique to get information about the device state
  - The app uses an attack vector
  - There is a path in the call graph where condition 1 and condition 2 are met
- Tool was designed to be used during the market level vetting process
- Does not include security checks for app lockers and is meant to be utilized in conjunction with human analysis

Results

- Ran the tool on four sets of apps:
  - A set of 500 randomly downloaded apps from Google Play
  - A set of 500 apps downloaded the top free category on Google Play
  - A set of 20 app described as app lockers in Google Play
  - A set of 1260 apps from the Android Malware Genome project

Results Continued

permission name	*benign1* set		*benign2* set		*malicious* set		*app-locker* set	
GET_TASKS	32	6.4%	80	16.0%	217	17.2%	19	95.0%
READ_LOGS	9	1.8%	35	7.0%	240	19.1%	13	65.0%
KILL_BACKGROUND_PROCESSES	3	0.6%	13	2.6%	13	1.0%	5	25.0%
SYSTEM_ALERT_WINDOW	1	0.2%	34	6.8%	3	0.2%	10	50.0%
REORDER_TASKS	0	0.0%	4	0.8%	2	0.2%	2	10.0%

technique	*benign1* set		*benign2* set		*malicious* set		*app-locker* set	
startActivity API	53	10.6%	135	27.0%	751	59.6%	20	100.0%
killBackgroundProcesses API	1	0.2%	8	1.6%	6	0.5%	4	20.0%
fullscreen	0	0.0%	22	4.4%	0	0.0%	1	5.0%
moveToFront API	0	0.0%	0	0.0%	1	0.1%	1	5.0%
draw over using *addView* API	0	0.0%	9	1.8%	0	0.0%	3	15.0%
custom toast message	0	0.0%	1	0.2%	0	0.0%	1	5.0%
getRunningTasks API	23	4.6%	68	13.6%	147	11.7%	19	95.0%
reading from the system log	8	1.6%	18	3.6%	28	2.2%	8	40.0%
reading from *proc* file system	3	0.6%	26	5.2%	43	3.4%	4	20.0%

TABLE V: Detection of potential GUI confusion attacks.

Dataset	Total	Detected	Correctly Detected	Notes
benign1 set	500	2	2	The detected apps are both app-lockers.
benign2 set	500	26	23	10 chat/voip app (jumping on top on an incoming phone call/message), 4 games (with disruptive ads), 4 enhancers (background apps monitoring and killing, persistent on-screen icon over any app), 2 anti-virus programs (jumping on top when a malicious app is detected), 2 app-lockers, and 1 keyboard (jumping on top to offer a paid upgrade).
app-locker set	20	18	18	Of the two we are not detecting, one is currently inoperable, and the other has a data dependency between checking the running apps and launching the attack (we only check for dependency in the control flow).
malicious set	1,260	25	21	21 of the detected apps belong to the *DroidKungFu* malware family, which aggressively displays an Activity on top of any other.

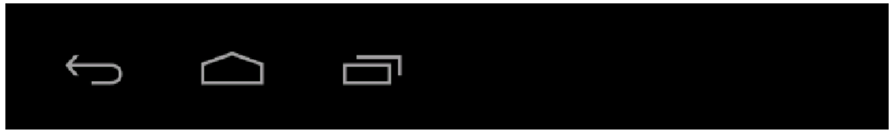
Defense Mechanisms

- Researchers designed a system to alert users to GUI modifications
- Currently no way for users to know which application is being interfaced with, within a GUI
- New system establishes a trusted path to inform the user
- Targets three areas:
  - Understanding which app is being interacted with
  - Understanding real author of the app
  - Displaying this information in an efficient manner
- System based of HTTPS elements in web browsers

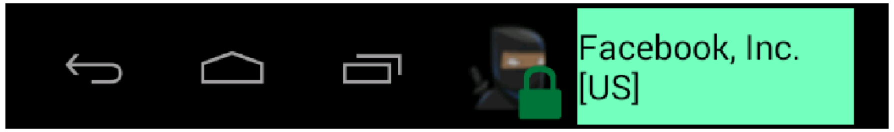
Displaying information

- System uses the unique identifier (found in the apk file) in conjunction with Extended-Validation HTTPS infrastructure
- System also uses a secret user chosen image to protect validity of its notifications

(a) Task B_1 and Task B_2 (real Facebook app)



(b) Task A_{std} (non-fullscreen attack app)



(c) Task A_{full} (fullscreen, defense-aware, attack app)

Implementation

- Prototype is based on the Android Open Source Project
- The target-app detection component of the prototype checks the activity stack and the window manager service to ensure users are only interacting with activities on the top of the stack
- A constantly active service validates and authenticates the installed apps in the device
- The navigation bar is modified to display information about the activity the user is interacting with

Evaluation

- Used human subjects to determine effectiveness of system
- Subjects were split into three groups:
  - Stock android
  - Android with new system without instructions
  - Android with new system with instructions
- Subjects then performed four different tasks:
  - Accessing facebook normal, accessing facebook with full screen attacks and with other GUI attacks

Results

TABLE VIII: Results of the experiment with Amazon Turk users.
Percentages are computed with respect to the number of *Valid Subjects*.

	Group 1: Stock Android	**Group 2**: Defense active. Subjects not aware of the possibility of attacks	**Group 3**: Defense active, briefly explained. Subjects aware of the possibility of attacks
Total Subjects	113	102	132
Valid Subjects	99	93	116
Subjects answering correctly to Tasks:			
B_1 and B_2	67 (67.68%)	70 (75.27%)	85 (73.28%)
A_{std}	19 (19.19%)	60 (64.52%)	80 (68.97%)
A_{full}	17 (17.17%)	71 (76.34%)	86 (74.14%)
A_{std} and A_{full}	8 (8.08%)	55 (59.14%)	67 (57.76%)
A_{std} and B_1 and B_2	4 (4.04%)	51 (54.84%)	73 (62.93%)
A_{full} and B_1 and B_2	6 (6.06%)	63 (67.74%)	76 (65.52%)
A_{std} and A_{full} and B_1 and B_2	2 (2.02%)	50 (53.76%)	66 (56.90%)

Conclusion

- Paper analyzed many GUI attacks
- Developed two level defense system
  - One at market level
  - One at device level
- Performed a user study demonstrating the effectiveness of their system
- All research and implementation was done on Android 4.4 or 4.6
  - Although most of the attacks are similar for 5.0 some implementation for both the attacks and security measures may be different