Short Paper: CHIPS: Content-based Heuristics for Improving Photo Privacy for Smartphones

Jiaqi Tan
Carnegie Mellon University
Pittsburgh, PA, USA
tanjiaqi@cmu.edu

Utsav Drolia
Carnegie Mellon University
Pittsburgh, PA, USA
utsav@cmu.edu

Rolando Martins
Carnegie Mellon University
Pittsburgh, PA, USA
rolandomartins@cmu.edu

Rajeev Gandhi
Carnegie Mellon University
Pittsburgh, PA, USA
rgandhi@ece.cmu.edu

Priya Narasimhan
Carnegie Mellon University
Pittsburgh, PA, USA
priya@cs.cmu.edu

ABSTRACT

The Android permissions system provides all-or-nothing access to users' photos stored on smartphones, and the permissions which control access to stored photos can be confusing to the average user. Our analysis found that 73% of the top 250 free apps on the Google Play store have permissions that may not reflect their ability to access stored photos. We propose CHIPS, a unique content-based fine-grained run-time access control system for stored photos for Android which requires minimal user assistance, runs entirely locally, and provides low-level enforcement. CHIPS can recognize faces with minimal user training to deny apps access to photos with known faces. CHIPS's privacy identification has low overheads as privacy checks are cached, and is accurate, with false-positive and false-negative rates of less than 8%.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access control*

Keywords

Android; Photo Privacy; Access Control

1. INTRODUCTION

Smartphones are becoming increasingly ubiquitous, and they are carried and used everywhere. Users are likely to capture and store photos of the people and events in their daily lives on their smartphones, creating collections of potentially privacy-sensitive photos. At the same time, smartphone platforms such as iOS and Android allow users to install third-party applications (apps), which can interact with other apps and data on the smartphone, including retrieve photos stored on the smartphone, and connect to the Internet to exfiltrate data. This creates a privacy concern where photos which a user considers private can be inadvertently extracted by an app and exfiltrated without the user's knowledge. In this paper, we show concrete, real-world evidence that there are privacy issues with how the Android platform controls access to stored photos. Then, we hypothesize that photos containing faces of persons known to the user, e.g. family or close friends, are privacy-sensitive, as the user would not want these photos to be inadvertently leaked. Using this hypothesis as a heuristic, we develop a *unique content-based approach, CHIPS, to provide finer-grained access control for stored photos for the Android platform.*

Contributions. Our contributions in this work are: (i) we review the privacy risks to stored photos due to Android's architecture, (ii) we analyzed the top 250 free apps on the Google Play store to evaluate how current apps may pose unintended privacy risks to stored photos, (iii) we design and implement a novel content-based access control system for photos: we present CHIPS, a practical face recognition-based, fine-grained, run-time access control system for stored photos which denies unauthorized apps access to photos containing faces of user-specified persons, (iv) we show in this initial prototype that using an off-the-shelf face recognition algorithm, we can attain good accuracy with low training requirements (optimal accuracy with 4 training images per face) on frontal faces in reasonable lighting conditions, and (v) we focus on our novel system architecture for privacy enforcement for photos, which provides strong enforcement with acceptable runtime overhead. *To the best of our knowledge, CHIPS is the first technique to apply face recognition to photos locally on smartphones to automatically determine if they are privacy-sensitive (based on prior user input), and to implement access control at run-time to deny untrusted apps access to sensitive photos.*

2. BACKGROUND

2.1 Motivation

Android's Photo Permissions. To understand how Android controls access to stored photos, we have to understand how photos are stored, how apps access photos, and the permissions system of the Android framework.

Photos captured by the camera of an Android device are stored in the external storage directory, or "SD card", which

WiSec'14, July 23–25, 2014, Oxford, United Kingdom.

http://dx.doi.org/10.1145/2627393.2627394.

is either a physical removable memory card, or a logical area in the device's memory. Android is based on Linux, and it uses Linux file owners and permissions to control access to files. Android provides per-app isolation for data files belonging to different apps. However, this per-app file isolation does not apply to stored photo files, as photos are stored in the external storage location, `/sdcard`. All files in the external storage location of an Android device are owned by the `system` user and `sdcard_rw` group, and they are group-readable and group-writable by default.

An Android app can access stored photos in a number of ways. (i) An app can list the files in the system-wide photo directory and open the file storing the photo. (ii) An app can use the Android MediaStore API to retrieve a list of available photos and their metadata (e.g. full path, thumbnails). Then, the app can open the file for the photo at its given path. (iii) An app can delegate photo selection to the system by using the Android framework's "Image Picker" interface (in the `com.android.gallery3d` package), and the framework returns an internal URI (Uniform Resource Identifier). Then, the app can use the framework's `ContentResolver` object to obtain an Java `InputStream` to read the photo, and the framework opens the file containing the requested photo on behalf of the app. Hence, in each of the ways that an Android app can access stored photos, filesystem permissions control access to each photo.

There are two problems with Android's access control for stored photos. First, permissions controlling access to photos stored in a smartphone are currently coarse-grained. Android allows users to choose whether to allow an app to access photos stored on the SD card; however, users can only choose to allow the app access to all stored photos, and cannot selectively deny access to individual photos, because access to the SD card location is controlled by the `READ_EXTERNAL_STORAGE` system permission. While an app might present users with an "Image Picker" interface, an Android app can access any photo, and not just the user-selected ones. Hence the app will be able to read all stored photos. Second, while `READ_EXTERNAL_STORAGE` controls access to stored photos on Android, this may be unintuitive to the average user. The `CAMERA` permission is described as allowing the app to "take pictures and videos", while the `READ_` and `WRITE_EXTERNAL_STORAGE` permissions are described as allowing the app to "modify or delete the contents of your SD card". Hence, it may appear to the average user that the permission which controls access to photos is the `CAMERA` permission, as it is the only permission whose description mentions photos. Unfortunately, this is not the case, and only the `READ_EXTERNAL_STORAGE`, but not the `CAMERA`, permission is required to access stored photos, which we verified empirically.

Evidence of Unintended Photo Access. Next, we identify apps which appear to not access stored photos based on their requested permissions, but which in fact have access to stored photos. We analyzed the top 250 free apps from the Google Play store (e.g. Facebook, Pinterest, etc.) to identify apps which requested the `READ_EXTERNAL_STORAGE` permission, but which did not request the `CAMERA` permission. We believe that a novice user, when shown this combination of permissions, would not believe that these apps are accessing any stored photos from the device, when it can in fact do so. In this group of apps, we also further distinguish between apps which requested the `INTERNET` permission, which would be able to exfiltrate stored photos, as compared to those which did not. We used the Androguard analysis tool [1], which allowed us to extract the permissions requested by each app from its Manifest. We found that 183 out of 250 apps (2 apps excluded due to bytecode analysis difficulties) requested `READ_EXTERNAL_ STORAGE` but not `CAMERA` permissions, and 181 of these apps also requested the `INTERNET` permission. Based on their requested permissions, 73% *of the top 250 free apps on the Google Play store have unexpected and complete access to the photos stored on a user's smartphone, and can even exfiltrate these photos* (further analysis is needed to determine if these apps actually exfiltrate photos without the user's knowledge). Next, we analyzed the bytecode of these 181 apps using Androguard, and found that 120 of these apps launched the Android-supplied Image Picker, in which case users would know that the app was accessing stored photos, even though the app's permissions may not reflect so. Nonetheless, these apps can still access all stored photos, regardless of which photos users picked. Thus, there is a need for stronger, finer-grained access control for stored photos.

2.2 Problem Statement and Limitations

Goals. Our goals for the design and implementation of CHIPS are: (i) not require any changes to the way third-party apps access photos, or to the way the Android platform stores photos, (ii) minimize the false-negative and false-positive rates of our face recognition for privacy identification, (iii) run all face detection and recognition locally on the smartphone to preserve privacy, (iv) minimize the user assistance required during the training for privacy identification, (v) have the training process for privacy identification complete in a reasonable amount of time (e.g. < 1 hour for 500 images on an average smartphone), (vi) have the dynamic privacy enforcement decision be reached in a reasonable amount of time (< 1 second on an average smartphone), (vii) third-party apps should not be able to circumvent CHIPS's decision to deny access to a stored photo.

Assumptions. We make a number of assumptions about users' apps and smartphones, although most of these assumptions are artifacts of our current implementation rather than necessitated by our design, and we intend to address some of these assumptions in our future work. First, we assume that file extensions accurately describe file contents, and we only perform our privacy checks on files with image extensions (e.g. `.jpg`). This technique is not robust to user apps renaming files to non-image extensions before accessing them, and we plan to implement file content checking in future versions. Second, we assume that the Android core libraries, framework, and operating system are isolated from user apps, and that user apps are unable to modify the core libraries, framework, or operating system. This assumption is necessary as our privacy enforcement relies on the Linux kernel (with our added modifications) to report file accesses to the CHIPS system service, and our CHIPS system service is implemented as part of the Android framework.

Threat Model. Our goal is to guard against unauthorized retrieval of potentially private photos on users' smartphones by untrusted apps. We assume that the smartphone OS and framework are trusted, and are not circumvented by malicious attackers. Hence, all access to a smartphone's stored photos must be via the authorized means described in §2.1. We also assume that the CHIPS user app is not

exploited by malicious users, and that the data of the CHIPS user app cannot be modified by other apps.

Limitations. We do not develop novel face detection or recognition algorithms. We use the implementation of the well-known Eigenfaces [14] algorithm in the OpenCV library [4] as an initial proof-of-concept to show that a standard face recognition algorithm can be run entirely locally on average smartphones today while achieving good performance, while focusing on the system architecture (i.e. mediation mechanisms) necessary for enforcing photo privacy. Hence, we do not address face recognition challenges, such as occluded faces, poor lighting, and side poses. As such, our initial evaluation of the face recognition accuracy of CHIPS uses a dataset (§4) of frontal faces without difficult lighting conditions, side poses, nor occluded faces. We also currently require changes to the Linux kernel so that our privacy enforcement cannot be circumvented by low-level means e.g. using the Java Native Interface (JNI) to access photos.

3. DESIGN AND IMPLEMENTATION

3.1 Design

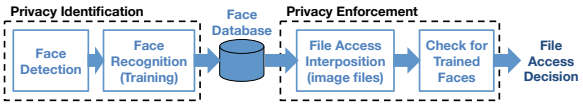


Figure 1: Overall approach of CHIPS

In place of Android's current all-or-nothing permissions for photo access by apps, CHIPS consists of a 2-step process (shown in Figure 1) which empowers users to indicate their privacy preferences for stored photos at a finer granularity.

3.1.1 Privacy Identification

CHIPS's privacy identification consists of an offline training phase, and a background classification phase. The offline training phase is carried out before privacy-sensitive photos are accessed: users assist CHIPS in training a face database. This database contains sample faces of persons whom the user considers privacy-sensitive (e.g. family, close friends). The background classification phase involves extracting faces from stored photos, and checking if these faces match any faces in the trained database.

Both the offline training and background classification phases begin with face detection to extract rectangular regions in a photo containing faces. This ensures that our face recognition works only on faces, and does not need to work with other unrelated parts of each photo. We used the Local Binary Patterns (LBP) [9] classifier to detect faces. Next, we use the well-known algorithm, Eigenfaces [14], for face recognition. The training step builds a database of sample faces for each person, so that during the classification step, the algorithm can tell if a given face belongs to a person whose face was in the training database, and which person the face belongs to. As Eigenfaces is a supervised algorithm, we require that for each person that the user would like CHIPS to later recognize, the user assists CHIPS by selecting faces of that person as training data.

3.1.2 Privacy Enforcement

Next, we describe the design of CHIPS's privacy identification and enforcement mechanisms for photo privacy. We need to interpose on all file accesses to identify when image files are being accessed. Then, when an image file is accessed, we need to check if there is privacy-sensitive content (i.e. faces of persons in the training database from §3.1.1), before deciding whether to block access to the file.

Kernel Interposition. As photos are accessed as regular files in Android rather than through a specialized interface (§2.1), we must interpose on all file accesses to identify accesses to image files for which our privacy checks must be applied. We need to interpose on file accesses in the kernel, rather than in the Android framework (e.g. `java.io` classes), because apps can access files using the Java Native Interface (JNI) to make C library calls, or directly invoke system calls, which would bypass any framework or library interposition. CHIPS's kernel interposition also checks if the accessed file has an image format extension, e.g. `.jpg`, before deciding whether to perform further security checks. If the accessed file is not an image, CHIPS's kernel interposition allows the kernel's regular file permission checks to proceed. CHIPS needs to decide in the kernel whether the accessed file is an image which requires additional access control (i.e. detect and recognize faces) so that access to non-image files can proceed with minimal latency.

System Service for Access Decisions. Next, CHIPS runs a system service as part of the Android framework (runs in user-space, but as the special Android framework user) to receive notifications of file accesses from the kernel and provide access decisions for photos. This system service performs initial checks against an app whitelist, moving the burden of maintaining app whitelists out of the kernel, so that we can minimize kernel modifications. Deciding whether an app is whitelisted to access all images is a security-sensitive operation, hence we place this functionality in the Android framework so that the CHIPS system service is at the same level of protection as the Android framework [12]. The CHIPS system service communicates with the CHIPS user app to request face recognition in accessed photos.

User-level Privacy Checks. Finally, we place the privacy checks (§3.1.1) in a user app, so that the most complex part of the privacy checks (face detection and recognition) are outside the protected Android framework's process to prevent privilege escalation attacks in the event of crashes. Image files accessed by apps not in CHIPS's whitelist are forwarded to the CHIPS user app which performs the face detection and recognition. CHIPS checks the image to determine if there are any trained faces. If there is no match, the file access is allowed; if there is a match, then access to the image file is blocked, and the offending access is logged to allow the user to make a decision later. The CHIPS user app also maintains a cache of face recognition results for photos stored in the smartphone. This is to reduce the latency that would result from having to perform face detection and recognition on-demand on a photo when it is being accessed.

Whitelists. CHIPS also allows users to specify which apps are to be given access to all stored photos without privacy checks, and to allow users to specify which stored photos are not privacy-sensitive, so that all apps can access them.

3.2 Implementation

We implement CHIPS for Android 4.2, and our implementation has three components (Figure 2). The first component is the kernel-level file access interposition. We added 1.2 KLOC (thousands of lines of code) to the Linux kernel used

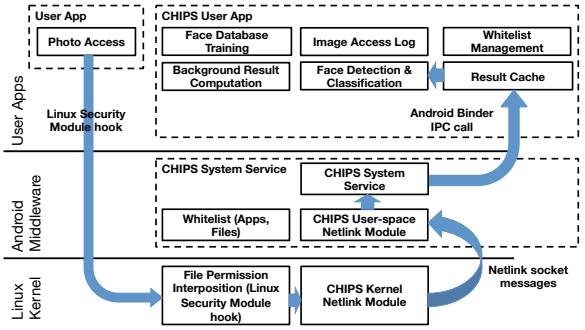


Figure 2: Architecture of CHIPS

in Android. The second component is the CHIPS System Service, which manages file access decisions, and is implemented as an Android System Service, and runs as part of the Android Framework. The system service consisted of 2.4 KLOC. The third component is the CHIPS user app, which is an Android app which allows users to train the face database, and which performs face recognition on accessed files, and also allows users to view a log of photos accessed by other apps, and to whitelist photos or apps. The CHIPS user app consisted of 8.5 KLOC.

3.2.1 *Face Detection and Recognition*

CHIPS's privacy identification component uses implementations of well-known face detection and recognition algorithms from the OpenCV [4] library. We used pre-trained LBP classifiers [9] included with the OpenCV library for face detection. To improve runtime, we scaled images down to a size of 600×600 pixels before face detection. To improve detection accuracy, we first used the frontal-face classifier to extract faces, and then used LBP classifiers for eyes, noses, and mouths on the extracted faces as a sanity check. We ensured that only one nose, two eyes, and one mouth are detected, and that their relative positions in the face are correct. Next, we used the OpenCV [4] implementation of the Eigenfaces [14] algorithm on faces from the face detection step for training and classification. To improve the runtime of Eigenfaces, we first scaled down detected faces to 100×100 pixels, and we found no degradation in accuracy.

3.2.2 *Privacy Enforcement: Access Control*

To demonstrate the privacy enforcement of CHIPS, we modified the Android framework. We used the CyanogenMod [3] distribution of the Android Open-source Project (AOSP), which includes device-specific code e.g. device drivers. We used CyanogenMod 10.1 (based on Android 4.2).

Kernel File Access Interposition. We interposed on all file accesses in the Linux kernel used in Android by implementing a Linux Security Module (LSM) [15]. To interpose on file accesses, we implemented our own `file_permission` security hook, which is called on every file read and write. We also implemented a kernel module for communicating with the CHIPS system service using Netlink sockets.

CHIPS System Service. The CHIPS system service acts as an intermediary between the CHIPS kernel-level file interposition, and the CHIPS user app which runs face recognition on accessed photos. The CHIPS system service is implemented as part of the Android framework. The CHIPS system service receives kernel notifications when files with image extensions are accessed, and it checks against its whitelist of filenames and the user IDs of accessing apps to determine if access should be allowed, or if the image needs to be checked. In the latter case, the system service requests for a privacy check on the image filename accessed.

CHIPS User App. The CHIPS user app provides the privacy checking functionality for privacy enforcement. The user app is started when the smartphone boots up. It also provides an interface to manage the whitelists of apps (allowed access to all photos) and photos (allowed access by all apps) in the CHIPS system service. When the CHIPS user app receives a file access notification, it checks its cache (described next) for a face recognition result for the accessed photo, and if there is no result present, the user app proceeds to carry out face recognition. If any faces in the trained database are present in the photo, the user app instructs the system service to deny the app access to the photo.

Result Caching and Background Scanning. To improve the performance of CHIPS's privacy enforcement, the CHIPS user app runs a background service to run face recognition on all stored photos and cache the results. This is to eliminate the need to run face recognition when the user is accessing photos. The cache is invalidated when the user retrains or updates the face recognition model, or when a stored photo is modified. This background face recognition runs only when the smartphone is connected to an external power source to prevent battery drainage.

4. EVALUATION

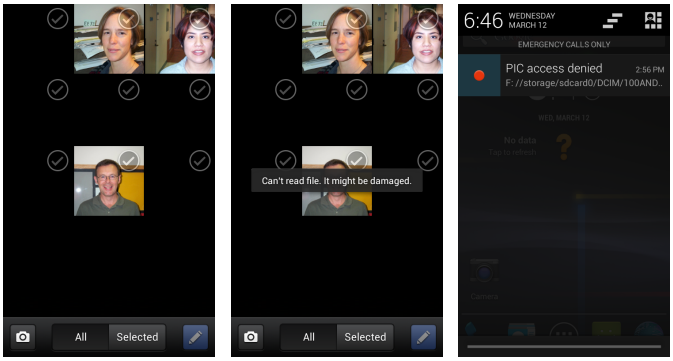
To evaluate CHIPS, we used the Caltech Faces 1999 dataset [2], which contains 450 color photos of frontal faces of 27 individuals with different lighting conditions, expressions, and realistic backgrounds. We believe this dataset bears some similarity to photos that average users would capture of people on their smartphones. Our experiments used a Google Nexus S (1 GHz Cortex-A8 processor, 16 GB storage, 512 MB RAM) smartphone, with our modified version of Android 4.2.

4.1 Case Studies of Enforcement

We demonstrate CHIPS's photo privacy enforcement on the Android app for Facebook. We trained a database containing the faces of 19 (out of 27) of the persons from the Faces 1999 dataset, and loaded a number of photos from the dataset onto our test phone. These photos contained both persons whose faces were in and not in the trained database. Figure 3 shows CHIPS's photo privacy enforcement blocking the Facebook app's access to stored photos containing trained faces. We tried to share a photo in the Facebook app. Figure 3(a) shows the Facebook app's internal image picker. There are 9 stored photos, and 5 of the photos have no displayed thumbnails. This is because CHIPS has detected faces from the trained database in these photos, and denied access to them, while the other 4 photos have faces not in the trained database. Figure 3(b) shows the error message that occurs when a user selects one of these blocked photos due to CHIPS's kernel interposition denying access to the accessed photo. CHIPS also alerts the user that access to the photo has been denied (Figure 3(c)).

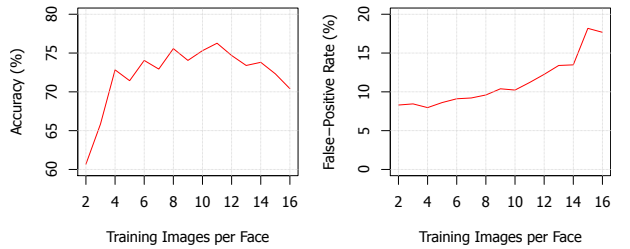
4.2 Face Recognition Performance

Face Detection. We evaluated CHIPS's face detection on the Caltech Faces 1999 dataset. A false-positive occurs

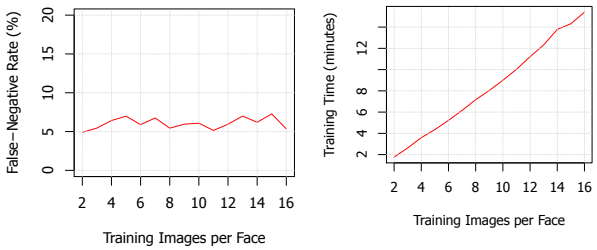


(a) Image Picker (b) Blocked image (c) System alert

Figure 3: CHIPS enforcement on the Facebook app. (Stored photos © California Institute of Technology [2])



(a) Face recognition accuracy. (b) False-positive rate (privacy identification).



(c) False-negative rate (privacy identification). (d) Time taken for database training

Figure 4: Performance for face recognition and privacy identification, and time taken for training.

when our algorithm reports a face in a region of the photo which is not a face, and a false-negative occurs when our algorithm reports that there are no faces in a photo that contains a face. We found that our face detection algorithm had no false-positives, and a false-negative rate of 2.0%.

Face Recognition Accuracy. The accuracy of CHIPS's face recognition is the percentage of faces in the test set for which the face recognition algorithm returns the correct label for a given face in the training database, and of faces not in the training database for which CHIPS returns "unknown". We varied the number of training images per face from 2 to 16. In addition, the test set included a number of faces for which we provided no training images. We report the average accuracy over 5 iterations of the experiment. From Figure 4(a), we can see that the face recognition accuracy improves significantly as we increase the number of training images per face from 2 to 4. Any additional training images per face only improves the face recognition accuracy slightly. The optimal face recognition accuracy is obtained with 11 training images per face, although the accuracy varies between 72% and 76% when there are between 4 and 15 training images per face. Hence, we can see that: (i) CHIPS's face recognition algorithm performs adequately, achieving more than 70% accuracy, and (ii) CHIPS's face recognition does not require significant amounts of training data, and CHIPS can achieve near-optimal (within 4%) face recognition with just 4 training images per face.

Privacy Identification. The goal of CHIPS's privacy enforcement is to identify whether a given face is in its database rather than to correctly label every face. Hence, it is sufficient for CHIPS to identify if a face is in the database, even if the algorithm misidentifies the face as belonging to a different person in the database. Thus, a false-positive for CHIPS's privacy identification occurs when a photo does not contain any faces in the trained database, but CHIPS labels it as one of the faces in the trained database, while a false-negative occurs when a photo contains a face in the trained database, but CHIPS labels it as not belonging to the database. We report the false-positive and false-negative rates for CHIPS's privacy identification over 5 iterations for each number of training images per face. The false-positive rate increases with more training images per face, and the optimal false-positive rate occurs at 4 training images per face, with a 8% false-positive rate (Figure 4(b)). This is likely due to overfitting when there are too many training images. On the other hand, the false-negative rate stays below 8% (Figure 4(c)) regardless of the number of training images per face. Thus, CHIPS's privacy identification performs well, with low false-positive and false-negative rates. CHIPS can block access to privacy-sensitive photos about 92% of the time, and it wrongly blocks access to non-sensitive photos not more than 8% of the time when there are at most 4 training images per face.

4.3 Performance

Training Time. Figure 4(d) shows the time taken to train CHIPS's face recognition for a 19-person database, averaged over 5 runs. All timing experiments ran on the Google Nexus S smartphone. The time taken for CHIPS's model training increases linearly with more training images per face. For optimal privacy identification, CHIPS should use 4 training images per face (§4.2), and CHIPS takes slightly under 4 minutes to train a database using 4 training images per face. Also, CHIPS can complete training in under 10 minutes when there are 10 or less training images per face for a 19-person database. Hence, CHIPS's face recognition training can be completed in a reasonable amount of time.

	Photo Access Scenario	Time
1	No kernel interposition (Baseline)	95.8 ms
2	App/photo not whitelisted, results not cached	3677 ms
3	App/photo not whitelisted, results cached	190.0 ms
4	App whitelisted	117.0 ms

Table 1: Average access times to stored photos

Enforcement and Classification Time. Next, we measure the time taken by CHIPS to enforce photo access con-

trol decisions, and to classify photos to determine access control decisions. We measure the time taken by an Android app to access stored photos in various scenarios. Table 1 shows the times taken to access each photo, averaged over 50 random photos from the Caltech Faces 1999 dataset. Scenario 1 establishes the baseline time taken to access each stored photo without CHIPS. While it takes 3.6 seconds to perform a privacy check for a photo without result caching (Scenario 2), we believe that the majority of photo access decisions will have been precomputed and cached by CHIPS's background service. Hence, for the majority of image file accesses for which photo access decisions have been cached, CHIPS would add 94.2 ms (98% overhead) to the critical execution path of accessing the photo (Scenario 3). Finally, Scenario 4 shows that for whitelisted apps, CHIPS adds only 21.2 ms (22% overhead) to the critical path of photo access.

5. RELATED WORK

Android Permissions. Apex [10] and Jeon et al. [7] propose finer-grained permissions for Android, but do not specifically target stored photos, unlike CHIPS. AppFence [5] modified the Android framework to preserve privacy by covertly substituting shadow data for sensitive data. AppFence protects only the camera, and does not protect stored photos, unlike CHIPS. Aurasium [16] mediates third-party apps using intercepts at the C and Java library level, whereas CHIPS uses mediation in the kernel to prevent apps from directly invoking system calls to bypass mediation.

Photo Privacy. P3 [11] protects the privacy of photos stored on third-party Photo-sharing Service Providers (e.g. social networks, photo-sharing sites). Darkly [6] is a privacy-preserving computer-vision library based on the OpenCV library [4], and it protects users from privacy loss due to continuously-sensing perceptual applications. PlaceAvoider [13] proposed new image analysis techniques for recognizing sensitive places in video streams from first-person cameras. PlaceAvoider focuses on image analysis, whereas CHIPS focuses on the systems architecture needed for enforcing stored photo privacy. Klemperer et al. [8] designed a series of user-studies which evaluated the effectiveness of using user-assigned tags to build access control rules for photos.

6. CONCLUSION AND FUTURE WORK

We have presented CHIPS, a fine-grained, face-recognition-based run-time access control system for stored photos on Android smartphones, which overcomes Android's current all-or-nothing access model for stored photos. We have demonstrated that CHIPS's privacy enforcement prevents unauthorized access to privacy-sensitive photos in unmodified real-world Android apps (Facebook), and that this enforcement imposes acceptable overheads of just 94.2 ms (98% overhead) per accessed photo when results are cached. We have also demonstrated that existing face detection and face recognition algorithms are sufficiently accurate, so that we can identify if a given face belongs to a trained database with a false-negative rate of 8%, and with a false-positive rate of 8%, and that they require minimal training, attaining optimal performance with just 4 training images per person.

In future, we intend to expand the CHIPS framework to support other types of media such as audio/video, along with other algorithms. For instance, we can extend CHIPS to run optical character recognition (OCR) algorithms on accessed photos to search for sensitive information, such as credit-card numbers and addresses, to proactively block access to photos containing such information. We also intend to explore the use of content-type checks to robustly identify files requiring privacy checks without relying on file extensions.

Acknowledgements

This research is funded in part by CMU-SYSU Collaborative Innovation Research Center and the SYSU-CMU International Joint Research Institute. We would like to thank the anonymous reviewers and our shepherd, Apu Kapadia, for their comments and constructive feedback. We would also like to thank Anupam Datta for his feedback on earlier versions of this work.

7. REFERENCES

[1] Androguard. https://code.google.com/p/androguard/.
[2] Computational Vision at CalTech. http://www.vision.caltech.edu/archive.html.
[3] CyanogenMod. http://www.cyanogenmod.org.
[4] OpenCV. http://opencv.org/.
[5] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In *ACM CCS*, 2010.
[6] S. Jana, A. Narayanan, and V. Shmatikov. A Scanner Darkly: Protecting User Privacy From Perceptual Applications. In *IEEE Security and Privacy*, 2013.
[7] J. Jeon, K. Micinski, J. Vaughan, A. Fogel, N. Reddy, J. Foster, and T. Millstein. Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In *IEEE SPSM*, 2012.
[8] P. Klemperer, Y. Liang, M. Mazurek, M. Sleeper, B. Ur, L. Bauer, L. Cranor, N. Gupta, and M. Reiter. Tag, You Can See It! Using Tags for Access Control in Photo Sharing. In *ACM SIGCHI*, May 2012.
[9] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Li. Learning Multi-scale Block Local Binary Patterns for Face Recognition. In *International Conference on Biometrics (ICB)*, 2007.
[10] M. Nauman, S. Khan, and X. Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *ASIACCS*, 2010.
[11] M. Ra, R. Govindan, and A. Ortega. P3: Toward Privacy-Preserving Photo Sharing. In *NSDI*, 2013.
[12] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google android: A comprehensive security assessment. *IEEE Security and Privacy*, March 2010.
[13] R. Templeman, M. Korayem, D. Crandall, and A. Kapadia. PlaceAvoider: Steering First-Person Cameras away from Sensitive Spaces. In *NDSS*, 2014.
[14] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1), 1991.
[15] C. Write, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *USENIX Security Symposium*, Aug 2002.
[16] R. Xu, H. Saidi, and R. Anderson. Aurasium: Practical Policy Enforcement for Android Applications. In *USENIX Security Symposium*, 2012.