VCR: App-Agnostic Recovery of Photographic Evidence from Android Device Memory Images

Brendan Saltaformaggio, Rohit Bhatia, Zhongshu Gu, Xiangyu Zhang, Dongyan Xu
Department of Computer Science and CERIAS
Purdue University, West Lafayette, IN 47907
{bsaltafo, bhatia13, gu16, xyzhang, dxu}@cs.purdue.edu

ABSTRACT

The ubiquity of modern smartphones means that nearly everyone has easy access to a camera at all times. In the event of a crime, the photographic evidence that these cameras leave in a smartphone's memory becomes vital pieces of digital evidence, and forensic investigators are tasked with recovering and analyzing this evidence. Unfortunately, few existing forensics tools are capable of systematically recovering and inspecting such in-memory photographic evidence produced by smartphone cameras. In this paper, we present VCR, a memory forensics technique which aims to fill this void by enabling the recovery of all photographic evidence produced by an Android device's cameras. By leveraging key aspects of the Android framework, VCR extends existing memory forensics techniques to improve vendor-customized Android memory image analysis. Based on this, VCR targets *application-generic* artifacts in an input memory image which allow photographic evidence to be collected *no matter which application produced it*. Further, VCR builds upon the Android framework's existing image decoding logic to both automatically recover and render any located evidence. Our evaluation with commercially available smartphones shows that VCR is highly effective at recovering all forms of photographic evidence produced by a variety of applications across several different Android platforms.

Categories and Subject Descriptors

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

General Terms

Security

Keywords

Memory Forensics; Android; Digital Forensics

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
 ISBN 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813720.

1. INTRODUCTION

Photographs and videos have served as essential evidence in criminal investigations and legal proceedings. Further, law enforcement agents rely on photographic evidence as clues during on-going investigations. Today, smartphones provide easy access to a camera at all times, and not surprisingly, photographic evidence from smartphone cameras has become commonplace in real-world cases.

During an investigation, digital forensics investigators extract such evidence from a device. Historically, investigators focused on evidence recovery from non-volatile storage such as disk-drives, removable storage, etc. Investigators make forensic copies (images) of storage devices from a crime scene and perform analysis on the images back at the forensics lab. This analysis recovers a bulk of saved files (such as pictures and videos) that the investigator examines for evidence.

More recently, investigators have realized that non-volatile storage alone only reveals a subset of the evidence held in a system. The *contextual* evidence held in a system's *volatile* storage (i.e., memory) can prove essential to an investigation [4, 9]. Memory forensics research has made it possible to uncover much of an operating system kernel's data from a context free memory image [8, 15, 21, 34]. Other work has focused on recovering data structure instances from applications using known in-memory value-patterns [15, 26, 30] or with the assistance of program analysis [22, 28, 29]. Unfortunately, the rapid pervasion of Android devices has rendered many tools inapplicable to smartphone investigations.

Like other digital evidence, photographic evidence which persists on non-volatile storage also lacks context or simply portrays an incomplete picture of a crime — again requiring memory forensics to fill the gaps. To this end, we have developed VCR[1], a memory forensics technique which integrates recovery and rendering capabilities for all forms of in-memory evidence produced by an Android device's *cameras*. VCR is based on the observation that all accesses to a device's camera are directed through one *intermediate service*. By designing VCR's evidence recovery function to target this intermediate service, VCR can automatically recover all forms of photographic evidence regardless of the app that requests it. This trend of centralizing critical services into intermediary processes (which we term *intermediate service architecture*) is widely used in the Android framework, and this paper examines the digital forensics and security implications of such design with regard to the camera framework.

[1]VCR stands for "Visual Content Recovery" and is a reference to the ancient videocassette recorder device.

VCR's evidence recovery faces challenges, however, because the Android framework (known as the Android Open Source Project or AOSP) is often customized by smartphone vendors. To overcome this, VCR involves novel structure definition inference techniques which apply to the Android vendor customization domain — called *Vendor-Generic Signatures*. To the best of our knowledge, VCR is among the first to handle vendor-customized data structures inline as part of targeted evidence recovery.

Additionally, VCR-recovered evidence must be reviewed, cataloged as evidence, and presented to any (not technically trained) lawyer or official. Thus, VCR must transform the unintelligible in-memory photographic data into human-understandable images. Using an instrumentation based feedback mechanism within existing image processing routines, VCR can automatically render all recovered evidence as it would have appeared on the original device.

We have performed extensive experimentation with VCR using a wide range of real-world commodity apps running on different versions of the Android framework and two new, commercially available smartphones. Our results show that VCR can automatically and generically recover and render photographic evidence from the phones' memory images — a capability previously not available to investigators — with high accuracy and efficiency.

2. MOTIVATION

Smartphone cameras are employed in a variety of apps which we use everyday: taking photographs, video chatting, and even sending images of checks to our banks.

Criminals too have found many uses for smartphone cameras. To motivate the need for VCR, we quote Riley vs. California [1], a United States Supreme Court case involving smartphone photographic evidence:

> At the police station about two hours after the arrest, a detective specializing in gangs further examined the contents of the phone. The detective testified that he "went through" Riley's phone "looking for evidence, because ... gang members will often video themselves with guns or take pictures of themselves with the guns." ... Although there was "a lot of stuff" on the phone, particular files that "caught [the detective's] eye" included videos of young men sparring while someone yelled encouragement using the moniker "Blood." ... The police also found photographs of Riley standing in front of a car they suspected had been involved in a shooting a few weeks earlier. [1]

In the above quote, the detective explains how essential smartphone photographic evidence is to ongoing investigations. Further, our collaborators in digital forensics practice describe many other crimes in which such evidence can prove invaluable. In Section 4, we will consider smartphone photographic evidence in a (mock) case based on an invited talk at Usenix Security 2014 on battling against human trafficking [23].

Let us strengthen our adversary model by considering a more tech-savvy criminal than Riley — someone who deletes the image files from the device's storage or even removes the storage (e.g., external SD-card) and destroys it. Current digital forensics techniques would not recover any photographic evidence in such a case. Luckily, regardless of how tech-savvy the criminal may be, photographic evidence from the camera's most recent use remains in the system's memory. VCR gives investigators access to these last remaining pieces of photographic evidence.

A smartphone camera produces three distinct pieces of evidence: photographs, videos, and preview frames. Photographs are left in a device's memory when a user explicitly captures an image. When a smartphone records a video, individual frames are captured and sent to the requesting app — again leaving frames behind in memory.

Preview frames, however, are of particular forensic interest for a number of reasons. Preview frames are a smartphone's analog to a standard camera's view finder. When an app uses the camera, the app will, by default, display the camera's current view on the screen, allowing the user to accurately position the device for capturing the intended picture. Importantly, *whether the user captures a photo or not* the app will display the preview. This leads to the forensically important feature that: **Any app which only opens the camera, immediately leaves photographic evidence in memory**. Further, preview frames (and video frames) are captured continuously and buffered until the app retrieves them. Thus *many frames will be present* in a memory image representing a *time-lapse* of what the camera was viewing.

Building from the scenario in Riley vs. California, imagine that Riley had carefully removed all photograph files from the smartphone's non-volatile storage or (more likely) was using an app which *does not save photograph files* such as a Skype video-call. In this case, the smartphone's non-volatile storage will not contain any evidence of the car suspected in the earlier shooting [1]. However, investigators could now use VCR to analyze the smartphone's memory image and recover the last images, videos, and preview frames left in the memory, which are likely the evidence the criminal is trying to hide.

Figure 1 shows some preview frames which VCR recovered from a smartphone's memory image. Notice that multiple frames are recovered and show the action of the perpetrator's car driving away (i.e., *temporal* evidence for investigators). Also note that these are *preview frames* and the smartphone user was *not actively recording video at that time*. Simply having the camera-using app open left photographic evidence in this memory image. It's easy to see how such evidence links the smartphone's owner to the car in the images (and hence to the shooting).

Our study reveals that this photographic evidence always persists in the smartphone's memory — without being erased or overwritten — until a new app uses the camera (filling the previous image buffers with new evidence). Thus, VCR will always have some evidence to recover. Note that these buffers are not app-specific, only containing frames from the most recent app which used the camera. More importantly, the buffers storing other media data (e.g., audio) are allocated from separate memory pools than the camera's buffers and thus cannot interfere with photographic evidence. Further, VCR is not specific to suspects' smartphones, investigators can apply VCR to memory images from a witness or victim's Android device as well, for instance to collect proof of the user's whereabouts.

Figure 1: Time-lapse effect in recovered preview frames *without explicitly taking a photo.* VCR recovers and renders these images as they would have appeared on the app's camera preview screen — the smartphone analog to a standard camera's view finder.

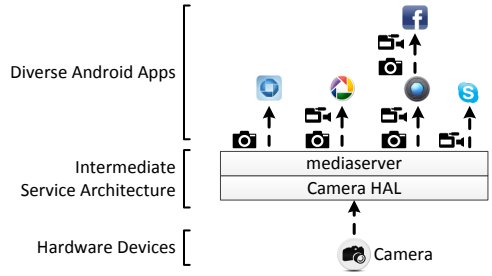


Figure 2: Intermediate service architecture. The mediaserver acts as a mediator between the apps and the camera device. Also, some apps utilize the camera by requesting the default camera app to perform actual image captures (such as the Facebook app shown here).

2.1 Centralized Photographic Evidence

For many core services, Android has adopted an intermediate service architecture. Specifically, accesses to peripheral devices and system services are mediated by an intermediate process. For the camera(s) this process is called the *mediaserver*. Figure 2 presents a high level view of the intermediate service architecture, specifically for the mediaserver's components: Apps, the mediaserver process, and camera hardware abstraction layer (HAL). This high-level intermediate service design makes app development easier and abstract regarding the hardware back-end.

Intermediate services present a standard interface to the apps. Each service is designed to generically handle any vendor/hardware specific implementation beneath it. Most importantly, the AOSP defines generic data structures for the vendor's code to use in order to conform with the standard interface presented to the apps.

The key observation behind VCR's design is that any app which uses the camera *must transitively use the generic data structures to retrieve photographic data from the mediaserver.* This creates a unique opportunity for VCR[2]. By locating and recovering these generic "middleware" data structures, VCR is able to reconstruct and render evidence *without any app-specific knowledge.* More importantly, VCR can remain mostly generic to any hardware-specific implementations because the camera HAL must also use the generic data structures to return photographic data to the apps. This is beneficial to VCR, which can now be designed in a more robust, generic way than tools that must recover data from individual (highly diverse) Android apps.

In fact, the mediaserver also delegates *audio requests* (accesses to speakers and microphones) and most media streaming. We note that photographic evidence is only part of the mediaserver's potential forensic value. VCR can be extended to extract other evidence formats from the mediaserver's memory.

[2]However, as we point out later, this also centralizes privacy-critical components and may benefit attackers as well.

2.2 Assumptions and Setup

VCR assumes that an investigator has already captured a memory image from an Android device. Previous research has designed both hardware [10] and software [33] acquisition tools to obtain a forensic image of a device's memory. VCR operates on memory images captured by any standard memory acquisition tool.

Similar to previous memory forensics projects [21, 26, 29, 34], VCR assumes the kernel's paging structures are intact in the memory image. This is required because VCR operates only on the mediaserver process' memory session. Tools (e.g., [34]) exist to rebuild a process' memory space from a whole-system memory image.

3. SYSTEM DESIGN

VCR consists of two phases: 1) identify and recover photographic data from an input memory image, and 2) transform the unintelligible recovered data into photographic evidence which investigators can review and present.

3.1 Recovering Evidentiary Data

Since photographic image buffers are encoded and indistinguishable from random data, brute-force scanning for the buffers would return countless false results. VCR adopts a more robust algorithm: for each type of evidence (preview frames, photographs, and video frames), VCR locates and recovers a distinct group of interconnected data structures, one of which contains the image data. For simplicity, we refer to such groups of interconnected data structures as "data structure networks."

Ideally, VCR would only need to verify the points-to invariants between the targeted data structures (i.e., each pointer field within each structure points to another structure in the network). In this way, each recovered data structure attests to the validity of the network, thus the located network is not a false positive. However, for key reasons described below, points-to invariants alone are insufficient in this scenario.

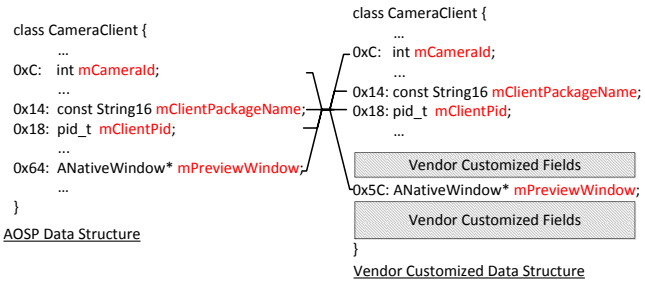


Figure 3: AOSP vs. Vendor Customized Structure.

The structures which VCR recovers form a closed network which is unfortunately too small to derive a viable points-to invariant signature. Instead, VCR must also employ value-invariant signatures for each data structure. However, due to vendor customizations, the structures' field positions and value-invariants cannot be fully known *a priori*.

Nearly every Android device uses a customized (possibly close-source) version of the AOSP. Device vendors make a proprietary copy of the AOSP repository and customize the low level framework (kernel, drivers) and high level utilities (GUI, standard apps). For data structures, vendors may add fields to store custom data, move existing fields to different offsets within the structure, or change the values that existing fields can be assigned (such as adding a new enumeration value). Specific to VCR, vendors modify the camera's allocation pools and internal operation (specific drivers, image processing, etc.). These modifications lead to different definitions of the data structures that VCR must recover.

Luckily, although vendors may customize the data structures, they must still conform to a "gold standard" in order to interact with unmodified portions of the AOSP. We use the term "gold standard" to refer to the many components of the AOSP that are not customizable (e.g., middleware libraries, core functionality, etc.), and thus vendor customizations must not remove structures and data fields **at the source code level** which other components rely on.

As an example, Figure 3 shows the CameraClient class from the AOSP versus the LG vendor customized version. The vendor customizations change the offset of the mPreviewWindow field, but *in order to interact with unmodified AOSP components* all the "gold standard" fields (which VCR relies on) must remain in the structure. In our evaluation we observed vastly different implementations of several data structures which VCR must recover.

After the vendor-customized source code is compiled, VCR loses access to the mapping between source code definitions and binary data structure layouts. Essentially we know that the fields exist, but cannot know *where they are* when locating data structures in a new memory image. To overcome this, VCR is prepackaged with *Vendor-Generic Signatures* (Section 3.2) for the customizable data structures. VCR then dynamically derives *Vendor-Specific Signatures* (Section 3.3) during data structure location and recovery.

Beyond vendor customization, VCR's generic signatures are also robust to changes *between AOSP versions*. When Google updates features in the AOSP, this also leads to changes in data structure layouts. In fact, several fields were added to the CameraClient class between AOSP versions 4.4 and 5.0. VCR's signatures however do not need to be updated because they can adapt to the input memory image. Further, it is easy to add additional signatures in the event that Google fully redesigns some data structure network.

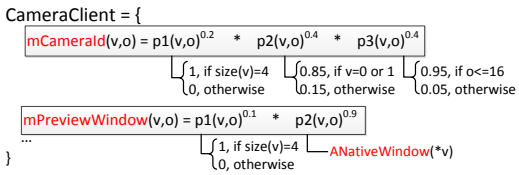


Figure 4: Illustration of a partial CameraClient signature (an unordered set of field constraints).

3.2 Vendor-Generic Signature Derivation

VCR operates on only an input memory snapshot and assumes no source code availability. Thus VCR must adapt signatures for any necessary data structures dynamically. To this end, VCR comes packaged with a set of *Vendor-Generic Signatures*. Vendor-Generic Signatures are data structure signatures which contain invariants on the structure's fields but *do not have set locations (offsets in the structure) for those fields*. Specifically, we preprocessed the AOSP "gold standard" version of each data structure D_i which VCR must recover. For each field f_j within the "gold standard" D_i, a field constraint (described below) is built.

Field Constraints We define 4 *primitive constraints* to describe each field: 1) A Type/Size constraint defines the field's type definition (e.g., floating point, pointer, etc.) and in-memory byte size. Since VCR operates on binary data these constraints are essentially sanity-checks on the discovered memory locations. 2) Value Range constraints are value invariants specific to field f_j. 3) Field Offset constraints define where f_j is *likely* to be in D_i. For some fields (e.g., inherited from a superclass) we know the byte offset in D_i for certain, but for most fields we cannot know where the vendor's modifications moved them. 4) For pointer fields, Pointer Target constraints define a set of *other primitive constraints* on the pointer's target. Specifically, our confidence in f_j being a pointer to a data structure depends on the validity of the target data structure.

Therefore, based on the AOSP definition of f_j in D_i, we automatically build a probability match function $p_i(v, o)$ for each primitive constraint. $p_i(v, o)$ defines the probability that a value v at byte offset o in a discovered data structure matches that constraint. We can then define a *field constraint* for f_j as:

$$f_j(v, o) = p_1(v, o)^{w_1} \times p_2(v, o)^{w_2} \times ... \times p_n(v, o)^{w_n} \quad (1)$$

where p_i is the i^{th} primitive constraint for field f_j, and w_i is a corresponding weight to adjust for stronger constraints (the sum of all weights must be 1).

Therefore, the signature of the structure D_i is an *unordered set of field constraints*. The set is unordered because VCR cannot know the offsets of those fields in a vendor customized data structure *a priori*. During memory image scanning, VCR will order the field constraints to adapt to the vendor customizations (described in the next section).

Figure 4 shows part of a CameraClient signature. Notice that each field constraint includes a number of primitive constraints — for instance, the mCameraId field has constraints on its type and size (a 32-bit integer), value range (between 0 and 1 with high probability), and offset (with high probability in the top 16 bytes of the data structure).

Not all data structures which VCR recovers are vendor customizable. For these we rely on existing points-to and value invariant signature generation techniques to build a "hard signature." When a signature contains a pointer to

one of these structures, we set that pointer field's Pointer Target constraint value to 1.0 (i.e., pointing to a valid hard signature provides full confidence in that pointer field).

Field Dependence We notice that not all fields in a data structure are independent. For simplicity, we only consider dependence based on two (or more) fields' *location in a data structure*: (1) Non-pointer fields *of the same type* tend to be clustered (e.g., floating point width and height fields) and (2) Fields accessed consecutively in a C++ class's member function are likely to be defined next to one another.

For these two cases, a scaling factor (α) is applied to increase the match probability of the dependent fields when a signature matching maps them consecutively. Simply put, if VCR locates these fields next to each other in a potential signature match then we can be more confident in that match — compared to matching those fields in separate locations. Fields dependent by (1) above are given $\alpha = 0.2$ scaling factor (i.e., matching such fields consecutively increases the probability of the entire signature matching by a factor of 0.2). Conversely, we assume stronger correlation for fields dependent by (2) and thus set $\alpha = 0.8$.

Based on the signatures generated before, we update each field constraint of any dependent fields to account for the scaling factor. Here we use the function $dep(c_a, c_b)$ to denote that the field constraints c_a and c_b are dependent. Consider two matches for those field constraints a_i and a_j where each a_n is the n$^{\text{th}}$ field in a potentially matching data structure instance. We define $P_{match}(c_a, a_i)$ as follows (where $c \rightarrow a$ denotes "c matches to a"):

$$P(c_a \rightarrow a_i | \forall c_b : dep(c_a, c_b) \wedge c_b \rightarrow a_j) = P_{match}(c_a, a_i)$$

$$\text{where } P_{match}(c_a, a_i) = \begin{cases} (c_a(a_i))^{\alpha} & \text{if } i = j-1 \text{ or } j+1 \\ c_a(a_i) & \text{otherwise} \end{cases} \quad (2)$$

Essentially, Equation 2 applies the scaling factor to c_a if c_a and c_b are dependent and c_b has previously mapped to a neighbor of a_i. Otherwise, the formula simplifies to the field constraint probability defined in Equation 1.

3.3 Memory Image Scanning

To use VCR to recover photographic evidence, investigators need only input a context-free memory image. VCR then employs a two-pass scanning algorithm. In the first pass, VCR marks all memory locations which match hard signatures (i.e., not vendor customizable and do not require probabilistic inference) — we refer to these as "hard matched" data structures. During the second pass, VCR uses probabilistic inference with our previously generated Vendor-Generic Signatures to construct *Vendor-Specific Signatures* to identify and recover true data structure instances.

Starting from the hard-matched data structures, VCR backward propagates confidence to all potential matches to Vendor-Generic Signatures. To calculate a match for a signature S, VCR first converts a candidate memory region (the region we want to map to S) into a set A of tuples:

$$A = \{(v_0, o_0), (v_1, o_1), \ldots (v_n, o_n)\} \quad (3)$$

where v_i is the i$^{\text{th}}$ value at offset o_i in the candidate memory region A. To match Vendor-Generic Signature fields, VCR may combine adjacent tuples to satisfy the field's type/size constraint. If no such match can be made, then the type/size constraint will yield a 0 probability. Later, we will use a_i to denote (v_i, o_i).

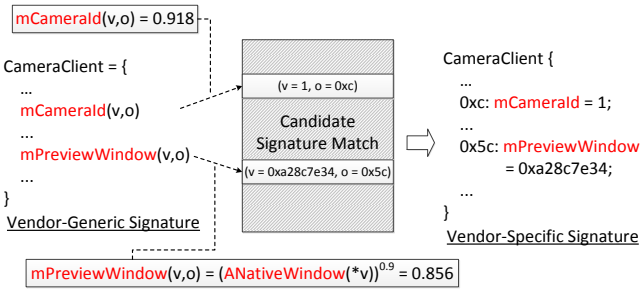


Figure 5: Matching a candidate CameraClient instance via field constraint computation.

VCR then creates a permutation of the vendor-generic signature by computing the best fit mapping $S \rightarrow A$, using the following greedy algorithm: For each randomly chosen field constraint c_i, VCR matches a binary tuple a_j (from the remaining unmatched tuples in A) which maximizes c_i's match probability (i.e., $P_{match}(c_i, a_j)$). This repeats for each c_i until all field constraints have been matched. Yielding the final match probability equation for a signature S to a candidate structure A:

$$S(A) = P_{match}(c_0, a_0) \times P_{match}(c_1, a_1) \times \ldots P_{match}(c_n, a_n) \quad (4)$$

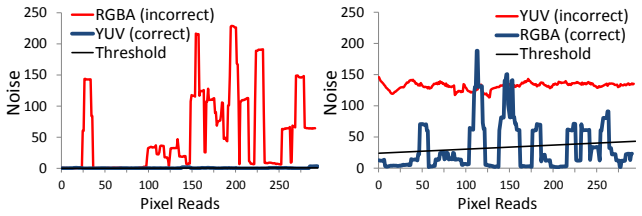
where the subscripts indicate order of matching (not order in the signature). Computing Equation 4 for a single $S \rightarrow A$ mapping yields VCR's confidence in that particular permutation of S for that candidate A.

Figure 5 shows an example of matching the CameraClient signature's field constraints to a candidate memory location. Computing each field constraint for the best matching a_i yields one permutation of the CameraClient signature's fields for that candidate memory location.

Computing the match probability of Pointer Target constraints requires knowing the probability of the pointer target's match. Because matching is done via backward propagation, VCR only computes a signature's match probability once all of its Pointer Target constraints have been computed. Recall that if the target is a hard-match then this confidence is 1.0. Thus, VCR requires recoverable data structure networks to have hard-signatures at the leaves (which can be ensured automatically during signature generation).

VCR repeats the above greedy algorithm and Equation 4 computation independently for all candidate memory regions for a single signature. Note that the first greedy matching may not result in the correct mapping of c_i to a_i (resulting in different $S \rightarrow A$ mappings for different candidate memory locations). We observe that: for a final mapping of c_i constraints to be correct, it must be *constant across all discovered instances of that data structure*. Thus only a single mapping of $S \rightarrow A$ can be correct for all candidate memory locations (i.e., A sets) for that signature.

VCR iteratively repeats the above process for all candidate memory locations (choosing the $S \rightarrow A$ mapping which maximizes the probability of a match), until an optimal mapping is found across all candidates for a single signature. This final signature which VCR selects is referred to as a Vendor-Specific Signature (i.e., the Vendor-Generic Signature with set field locations). VCR will recover all candidate data structures which match the Vendor-Specific Signatures to a certain threshold. In our evaluation, we use a threshold of 0.75 since most candidates polarize with invalid candidates near 0.3 and valid matches near 0.8.



(a) YUV Buffer (b) RGBA Buffer

Figure 6: Reading (a) YUV and (b) RGBA buffers with different decoding algorithms. Correct decoding algorithms will minimize the area under the noise curve. In fact, we can hardly see the YUV curve in (a) because its noise is always close to 0.

3.4 Rendering Evidence

Once VCR has recovered the structures containing photographic evidence, the data must be reconstructed into human-understandable images. This is essential as the raw contents of these photographic buffers would be unintelligible to forensic investigators.

Photographic data may be in any format that the app requests (e.g., NV21, ARGB, etc.). We observe, however, that apps (i.e., image buffer consumers) must have access to decoding logic for any format supported by the AOSP. Based on this, VCR automatically reuses the existing AOSP image decoding logic, but which decoding algorithm to use cannot be known *a priori*. VCR must determine (specifically avoiding burdening human users) which decoding is appropriate for each recovered image.

Image buffers, specifically photographs, are highly *periodic*. That is, the data values follow regular periods across the image's strides, height, and width. Based on this, image processing techniques often compute the *spacial locality* of the image's pixel values when performing image analysis [14]. VCR builds on this idea to *validate image decoding* by enforcing a periodic constraint on the image data as it is read by the decoding algorithm.

VCR instruments each decoding algorithm to verify that the values read from the input buffer follow a *periodic data constraint* (i.e., the spacial locality between each pixel value should be small and form a smooth curve). VCR attempts to decode each recovered image buffer with each available algorithm. During image decoding, VCR computes the Euclidean distance between the pixel values read from the input buffer [14] which we refer to as the decoding "noise."

Ideally, decoding an image buffer with the correct decoding algorithm should produce very little noise (i.e., the pixel values closely follow the periodic data constraint). In practice, images may contain local, sharp changes of color or brightness (which is particularly obvious when encoded in YUV format), so VCR computes a moving average of the noise values as a *noise threshold*. The algorithm which produces the smallest noise threshold is marked and the output of that decoding is presented to investigators as evidence.

For empirical comparison, Figure 6 shows graphs of two image buffers being decoded by the correct and incorrect algorithms. For simplicity, we only compare two algorithms, but VCR considers all 19 decoding algorithms used by the AOSP camera framework.

Figure 6(a) plots the noise values for a buffer encoded in YUV format being decoded using the RGBA algorithm (red curve) and YUV algorithm (blue curve). We can see that decoding the YUV buffer with an RGBA decoder produces a large amount of noise, whereas decoding with the YUV (i.e., correct) algorithm produces so little noise that the blue curve is hardly visible. We also plot the noise threshold from the YUV decoding, but this too is always near 0.

In Figure 6(b), we plot an RGBA buffer being decoded as YUV (red curve) and RGBA (blue curve). In this case, we see that the correct decoding (RGBA) contains some noise but the incorrect decoding (YUV) induces 3 to 4 times more noise. Further, the RGBA noise threshold (plotted in black) is again always near 0.

Finally, all recovered buffers of a single type (i.e., preview frames, photographs, or video frames) must use the same encoding — because an app only specifies this encoding once. Thus as a final sanity check, VCR ensures that the chosen decoding algorithm minimizes the decoding noise across all image buffers. Because of the large noise disparity between correct and incorrect algorithms, in our evaluation VCR was able to identify the correct decoding algorithm in all of our test cases.

4. EVALUATION

Our evaluation tested a variety of different Android devices as "devices under investigation." We performed evaluations with two new commercially available Android smartphones: an LG G3 and a Samsung Galaxy S4. Both smartphones run two different, highly vendor-customized versions of the AOSP. Further, we set up unmodified Android emulators running AOSP versions 4.3, 4.4.2, and 5.0. These are the most recent major versions of Android and represent nearly half of all the Android market-share [16]. In total, this allows us to stage "crimes" involving 5 vastly different Android devices to evaluate VCR's effectiveness and generality.

We first installed each app on our test devices and interacted with its camera features (i.e., taking photos, videos, and simply watching the preview screen). We then closed the app and used `gdb` to capture a memory snapshot from the mediaserver process. To attain ground truth, we manually instrumented the mediaserver to log allocations and deallocations of data structures containing photographic evidence. This log was later processed to measure false positives (FP) and false negatives (FN).

We used VCR to analyze the previously captured memory images and recorded the output photographic evidence. Despite the variety of different and customized AOSP versions tested, all evaluation was conducted using VCR with *the same set of Vendor-Generic Signatures* (generated from Google's AOSP 4.4.2 repository) which VCR automatically adapted to each input memory image. In a real-world law enforcement scenario, in-the-field investigators obtain images of a device's volatile RAM and non-volatile storage, and the collected memory images are later analyzed using VCR by forensic lab staff. Also note that VCR is a lightweight, efficient tool and could even be operated at the scene of a crime from an investigator's laptop. In all of our tests, VCR produced fully-rendered results from an input memory image in under 5 minutes (except for two specially noted cases at the end of this section).

4.1 App-Agnostic Evidence Recovery

This section presents the results of applying VCR to memory images containing photographic evidence generated by the following seven apps on our two smartphone devices.

Device	App	Evidence	Live Instances	w/ Image Data	Recovered	FP	FN
LG G3	Instagram	Preview	32	11	11	0	0
		Photo	1	1	1	0	0
		Video	20	20	20	0	0
	Facebook	Preview	32	11	11	0	0
		Photo	1	1	1	0	0
		Video	20	20	20	0	0
	Chase Banking	Preview	32	2	2	0	0
		Photo	1	1	1	0	0
	Skype	Preview	32	9	9	0	0
		Video	9	9	9	0	0
	LG Default Camera	Preview	32	10	10	0	0
		Photo	1	1	1	0	0
		Video	20	20	20	0	0
	Google Camera	Preview	32	11	11	0	0
		Photo	1	1	1	0	0
		Video	20	20	20	0	0
Samsung Galaxy S4	Instagram	Preview	32	7	7	0	0
		Photo	1	1	1	0	0
		Video	16	8	8	0	0
	Facebook	Preview	32	7	7	0	0
		Photo	1	1	1	0	0
		Video	16	8	8	0	0
	Chase Banking	Preview	32	8	8	0	0
		Photo	1	1	1	0	0
	Skype	Preview	32	7	7	0	0
		Video	9	8	8	0	0
	S4 Default Camera	Preview	32	7	7	0	0
		Photo	1	1	1	0	0
		Video	16	8	8	0	0
	Google Camera	Preview	32	7	7	0	0
		Photo	1	1	1	0	0
		Video	16	8	8	0	0

Table 1: Results from recovering photographic evidence from apps on commodity Android smartphones.

Five of the apps have features for taking individual photographs, videos, and displaying preview frames: the two smartphones' pre-installed camera apps, Google's Google Camera app, the Facebook app, and Instagram app. Each of these apps accesses and uses the camera device in different and forensically interesting ways. We also investigated evidence from the Skype app, which employs only video capture and preview functionalities. We also analyzed the Chase Bank app's check image and upload feature. In the next sections we will highlight some of these apps as case studies.

Table 1 shows a summary of our evaluation results. Column 1 shows the device on which the evaluation was performed. Columns 2 and 3 show the app's name and which types of photographic evidence it can generate, respectively. The number of "live" frames (i.e., frames which were allocated and not yet freed) in the memory image is shown in Column 4. Column 5 shows the subset of those image frames which the camera HAL had filled when the memory image was captured[3]. Column 6 shows the number of images (i.e., photograph, video frames, or preview frames) which VCR recovered and rendered. Columns 7 and 8 show false positives (image frames which VCR wrongly reported) and false negatives (image frames which VCR missed).

From Table 1, we can make a number of key observations. First, VCR is highly effective at recovering and rendering photographic evidence left behind by a variety of Android apps. This confirms that VCR's Vendor-Generic Signatures ensure that the recovery mechanism is highly accurate. Table 1 shows that these constraints are indeed strong enough to effectively prune all invalid data and attest to the accuracy of any recovered evidence — resulting in VCR producing no false positive or false negative results. In total, VCR recovered 245 pieces of photographic evidence in these test cases.

Table 1 shows that of the 32 total test cases, all 12 cases left behind several preview frames. These results range from a high of 11 preview frames in the LG G3's Google Camera, Facebook, and Instagram cases to only 2 frames in the LG G3's Chase Bank test case. Interestingly, the average "preview frames recovered per app" appears to be phone dependent: 7.17 for the Samsung Galaxy S4 and 9 for the LG G3 (or even 10.4 if we ignore the outlier: the Chase Bank app). This implies some connection between phone hardware or vendor customizations versus the amount of potential evidence. Since both phones have relatively equally powerful hardware, we reason that the latter is more influential. Again, these preview frames are generated by the apps *automatically* when the user *only opens the app's photographic features*.

Also shown in Table 1 is that video frames are far more prevalent than any other form of photographic evidence. This is intuitive given that video frames are often sampled at higher rates than preview frames. Our evaluation shows that on average each app left 12.9 video frames. Again, the LG G3 provides more evidence with an average of 17.8 video frames per app versus the Samsung at 8 video frames on average. Intuitively, Skype leaves fewer frames than the other apps in our tests (9 frames on the LG G3 and 8 on the Samsung Galaxy S4) likely because of the high throughput design of Skype's video-call feature. Also note that the

[3]The information in Columns 4, 5, and 6 was obtained via manual instrumentation only for the purpose of evaluation. VCR does not have access to such runtime information and operates on only the input static memory image.

recovered video frames are the result of explicitly recording video with the tested apps, unlike the preview frames which are generated without any explicit user command to record.

Finally, Table 1 shows that only one photograph per application is available in the memory images. Manual investigation revealed that the Android framework prefers to reuse buffers as quickly as possible, so despite taking several photos during our testing only a single photograph is left buffered — always accompanied by a number of preview frames.

4.1.1 Case Study 1: Camera Apps

Camera apps are standard Android apps which only provide a front-end user interface to the camera back-end (the mediaserver and camera HAL). A newly purchased Android device will come with a pre-installed camera app, but the user may install a new camera app and select one to use as the default. To illustrate the generality of VCR, we evaluate both pre-installed camera apps from our test phones as well as the third-party Google Camera app. The results of the LG G3 Default Camera and Google Camera tests are shown in Rows 5 and 6 of Table 1, and the Samsung Galaxy S4 Default Camera and Google Camera tests are shown in Rows 11 and 12.

Table 1 shows that in each of the camera app tests VCR is able to accurately recover and render all photographic evidence. For the LG G3 Default Camera case, we see that VCR recovered 10 preview frames, 1 photo, and 20 video frames, and similarly for the Google Camera test VCR recovered 11 preview frames, 1 photo, and 20 video frames. Again we observe fewer recoverable frames in the Samsung cases: 7 preview frames, 1 photo, and 8 video frames for both the Default Camera and Google Camera evaluations.

The default camera app is important because other apps may rely on it for photographic operations. When choosing test cases, we intentionally included the Facebook app as an example of this (shown in Rows 2 and 8 of Table 1). The Facebook app allows users to capture and post videos and photos on-the-fly (i.e., without leaving the Facebook app). To implement this, the Facebook app requests the default camera app to take a photo or video and then return the resulting image. Thus when the Facebook app user requests to capture a photo or video, the default camera app opens, manages the image capture, and makes the resulting image available to the Facebook app.

The fact that the Facebook app (and others like it) employ the default camera to handle photography, leads to a forensically interesting observation: photographic evidence from such apps will likely use similar formatting and sizing parameters to conform with the default camera pass-through interface. In the Facebook app case studies from Table 1, we see that VCR is able to render 11 preview and 20 video frames plus 1 photograph for the LG G3 test and 7 preview and 8 video frames plus 1 photograph for the Samsung S4 case.

It is important to note that among all of our test cases only the Facebook app is an example of requesting photography through the default camera. Although default camera pass-through is common, we intentionally focused the majority of our evaluation on test cases which implement their own photography features. This directly shows VCR's generality with regards to the evidentiary apps' implementation.

4.1.2 Case Study 2: Skype

In this case study, we highlight the Skype app because image frames collected by Skype are *never present on non-volatile stores* — Skype immediately encodes, packages, and transmits the image frames over the internet. Thus the only visual artifacts of a Skype video-call will be the frames left in the device's memory. Such frames provide vital evidence in a digital investigation — as we will show with a scenario based on the Usenix Security 2014 invited talk "Battling Human Trafficking with Big Data." [23]

Imagine, for the sake of example, that a human-trafficking suspect is using Skype video calls from a smartphone to show victims to potential clients. While the criminal may be careful not to show his or her identity, the video frames of the Skype call clearly link the smartphone user to the victims of the crime. Further, this criminal may try deleting (or obfuscating) Skype's call history, but even after the criminal has ended the Skype calls and finished trying to hide the evidence, the last snippets of video are still recoverable in the device's memory. Later, when law enforcement agents arrest the suspect, investigators will not find any evidence on the smartphone's non-volatile storage. Applying VCR to the smartphone's memory will reveal the last video frames of the Skype call showing one or more victims of this crime, and providing vital evidence to investigators which would otherwise be inaccessible.

For this case study, we set up a simplified crime reenactment by having one of the authors walk slowly through a Skype video call's field of view. We then used VCR to recover the remaining video frames frozen in the device's memory image. The LG G3 device was used in this trial, and the results of analyzing the device's memory image are shown in Row 4 of Table 1.

Figure 7 shows some of the recovered video frames and gives a clear example of the importance of VCR-recovered photographic evidence to an investigation. The 6 frames shown in Figure 7 are a subset of the 9 video frames in total which VCR recovered. These frames reveal a person walking through the Skype call's field of view, and we can easily see how this provides substantial evidence to investigators about the human-trafficking victims in our crime scenario above.

In addition to recovering the video frames shown in Figure 7, VCR also recovered 9 preview frames still buffered in the memory image (as shown in Table 1). Visual inspection of all 18 images recovered for this test case revealed that 4 of the 9 preview frames were identical (to the investigator's eye) to 4 of the recovered video frames. Thus yielding 14 total unique images to be used as evidence. This case study shows the importance of VCR recovered photographic evidence to aiding a digital investigation.

4.2 Analysis Across Android Frameworks

Given that many versions of the AOSP are being widely used today [16], VCR must be effective for a majority of devices that investigators may face. In this section, we evaluate VCR's effectiveness against memory images taken from the three most recent, widely used versions of the AOSP.

To perform this evaluation, we set up unmodified Android emulators running AOSP versions 4.3, 4.4.2, and 5.0. As before, we used VCR to analyze memory images after interacting with each of the tested applications. For this evaluation, we selected three of the apps to use in each of the three

Figure 7: Sample video frames recovered from the Skype case study. This is an example of how multiple recovered frames can capture evidence of time and direction for the suspect shown here.

Device	App	Evidence	Live Instances	w/ Image Data	Recovered	FP	FN
Android 4.3	Facebook	Preview	32	3	3	0	0
		Photo	1	1	1	0	0
		Video	31	31	31	0	0
	Skype	Preview	32	3	3	0	0
		Video	1	1	1	0	0
	Default Camera	Preview	32	5	5	0	0
		Photo	1	1	1	0	0
		Video	202	202	202	0	0
Android 4.4.2	Facebook	Preview	32	3	3	0	0
		Photo	1	1	1	0	0
		Video	16	16	16	0	0
	Skype	Preview	32	3	3	0	0
		Video	1	1	1	0	0
	Default Camera	Preview	32	3	3	0	0
		Photo	1	1	1	0	0
		Video	24	24	24	0	0
Android 5.0	Facebook	Preview	32	3	3	0	0
		Photo	1	1	1	0	0
		Video	19	19	19	0	0
	Skype	Preview	32	3	3	0	0
		Video	1	1	1	0	0
	Default Camera	Preview	32	3	3	0	0
		Photo	1	1	1	0	0
		Video	297	297	297	0	0

Table 2: Results from recovering photographic evidence from current and future Android versions.

emulators: Facebook, Skype, and each emulator's default camera app.

Table 2 presents the results which VCR rendered from the different emulators' memory images. Column 1 shows the version of Android that the emulator is running. Columns 2 and 3 show the app and types of photographic evidence evaluated respectively. Like in Section 4.1, the number of "live" image frames in each memory image is show in Column 4, and Column 5 shows the subset of these which contained image data. Column 6 shows the number of images which VCR recovered and rendered. Finally, Columns 7 and 8 report the false positives and false negatives.

Table 2 shows that VCR is highly effective at recovering and rendering photographic evidence produced on the most widely used Android versions. We observe that the emulated camera device used in the Android emulator does not produce frames at a high rate similar to our test smartphone devices. This leads to (as shown in Table 2) fewer frames being available in the memory images. On average, the tests in Table 2 produce only 5.8 frames (with the exception of the outliers: the 4.3 and 5.0 emulators' default cameras).

Additionally, the preview frame buffer is rarely filled above 3 frames. This results in VCR recovering only those 3 preview frames for all three apps on all three emulators, except for the Android 4.3 Default Camera test in which VCR recovered all 5 preview frames. Again, VCR is able to recover and render all instances of photographic data in the evaluated memory images without any false positive or false negative results — as Table 2 shows, 627 pieces of photographic evidence in total for these test cases.

Notably, Table 2 contains two exceptional cases. The default cameras for Android 3.4 and Android 5.0 report very large numbers of video frames. We performed manual inspection of the results and found that all output images were valid (i.e., from distinct buffers filled individually by the camera HAL). Further investigation revealed that there existed a bottleneck when saving those video frames to the emulator's storage. Admittedly, this is likely an emulator configuration error, but the resulting backup of frames further demonstrates the effectiveness of VCR's recovery and rendering — though run-times for these two cases were nearly 30 minutes.

4.3 Recovering Temporal Evidence

As shown in Table 1, numerous preview frames and/or video frames can be recovered for a single app — represent-

ing a *time-lapse* of what the camera was viewing. Here, we analyze how a set of preview or video frames can give investigators temporal evidence of the incident under investigation.

To measure the time captured by a set of recovered frames, we reran the two camera app test cases on the two smartphones. Time lapses were measured using the camera apps to record video of a stopwatch for a period of 1 minute, and the phones were rebooted between each test. Note that the "stopwatch" used here was actually a stopwatch app on the first author's smartphone. While this measurement may seem "low-tech," our results in Table 3 show that the time-lapse captured by the recovered sets of frames is long enough to make an empirical measurement very accurate.

After recording for 30 seconds, we captured a memory image from the device, and VCR was used to recover all available preview and video frames from the memory images. The output image frames were grouped into three sets: Preview frames, Video frames, and a Union set containing all visually unique frames from both the preview and video sets (which would be recoverable for any app which captures video). We then manually measured the difference between the earliest frame and the latest frame in each set. Figure 8 shows an example of some recovered stopwatch preview frames.

Device	App	Evidence	Frames	Time-Lapse
LG G3	LG Default Camera	Preview	11	1.3s
		Video	20	0.6s
		Union	22	1.4s
	Google Camera	Preview	11	0.9s
		Video	20	0.4s
		Union	25	0.9s
Samsung Galaxy S4	S4 Default Camera	Preview	7	0.5s
		Video	8	0.3s
		Union	10	0.5s
	Google Camera	Preview	7	0.4s
		Video	8	0.3s
		Union	11	0.5s

Table 3: Time-Lapse Evaluation.

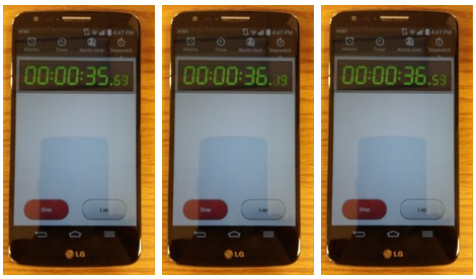


Figure 8: Recovered preview frames used to measure temporal evidence. For this experiment, we recorded another smartphone's stopwatch app and used VCR to recover the preview and video frames — yielding empirical measurements of the temporal evidence captured in VCR recovered evidence.

Table 3 presents the time measurements captured within the sets of recovered preview and video frames. Columns 1 and 2 show the tested device and app. Column 3 names the type of set being measured: Preview, Video, or the Union set. Column 4 shows the number of frames in the set, and the measured time difference is shown in Column 5.

From the times in Table 3 we can make several observations: First, the windows of time captured by the recovered frames are large enough to provide substantial evidence to an investigation — we already empirically saw this in the evidence recovered for the "crimes" in Figures 1 and 7. To best analyze the results show in Table 3, consider the first row as meaning: The set of preview frames from the LG G3's Default Camera captures 1.3 seconds of time divided over 11 images. From this, we see that a majority of the results yield over a half second of time-lapse.

This may seem like a small amount of time, but considering how quickly many crimes can occur and how powerful this evidence can be (such as an image of a car involved in a shooting or a human-trafficking victim) this provides a significant amount of evidence to investigators. Specifically, the example sequences of images shown in Figures 1 and 7 both represent a time-lapse of less than 1 second. Table 3 shows that of the 12 measurements, the LG G3 provides much longer time windows with the average being 0.92 seconds per test. The Samsung provides an average of 0.42 seconds per test.

A second observation we make from Table 3 is that preview frames capture longer time windows in fewer frames but video frames provide many more images. Preview frame sets on the LG G3 average more than double the time window of video frame sets (i.e., 0.4 seconds versus 0.9 seconds and 0.6 seconds versus 1.3 seconds). However, the video frame sets in the LG G3 test contain 20 frames compared to only 11 frames in the preview sets. The Samsung results show a similar pattern but the differences between sets are much closer (e.g., 8 frames over 0.3 seconds versus 7 frames over 0.5 seconds). As a consequence, we can observe that the time delta between images is much shorter between video frames than between preview frames.

Finally, Table 3 shows that when video and preview frames are available then (not surprisingly) considering the union of those sets yields the best results. In practice, nearly any app which generates video frames will also generate preview frames. Table 3 shows that the video frames will mostly be enclosed by the larger time delta captured by the preview frames. For example, consider the LG G3's Google Camera Union test: the investigator can now see 0.9 seconds of time captured in 25 images — leading to roughly a 0.036 second time delta between each image. Using such analysis, investigators can gain a wealth of evidence from only the frames being recovered by VCR.

4.4 Privacy Concerns

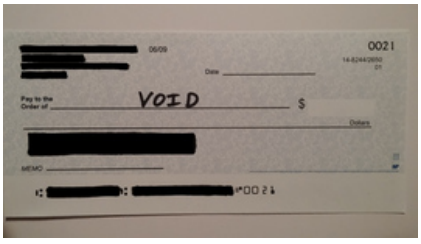


Figure 9: Recovered check image left behind in a memory image. This case study gives an example of the potentially sensitive user information which VCR (or worse, malware) can *generically* recover from the mediaserver's memory.

Finally, this section highlights a potential privacy concern which VCR reveals. VCR exploits the centralized design of the Android framework to access app-agnostic photographic evidence. It should be noted however that the same properties which make the mediaserver beneficial for digital forensics also make it a target for attack.

There has been extensive prior work on exploiting vulnerabilities in the Android framework to glean information about a smartphone's owner [12, 19]. Following that line of work, we can envision a malware which aims to steal confidential information and remain as stealthy as possible. Unfortunately, the mediaserver is a great target for such malware for a few reasons: 1) As we will show, the mediaserver handles very sensitive data regarding the device's owner, 2) As we have shown, it is beneficial to utilize the mediaserver's centralized design to capture photographic evidence from all apps generically, and 3) The mediaserver runs as a background service in a dedicated process (which makes for a great hiding spot for malware).

To underscore the potential danger of malware gaining access to a device's intermediate service processes (like the mediaserver), we have included the Chase Bank app in our previous evaluations. The Chase Bank app, like many other financial institutions' apps, includes a check image and upload feature. When the device's owner has a check to deposit, they simply take a picture of the check and upload the image to Chase from within the app. The image is never saved to non-volatile storage and handled securely once the Chase app has received the image. Unfortunately, the image is buffered in the mediaserver long before it is returned to the Chase app and may remain buffered for long after.

To highlight this point, Figure 9 shows one of the check images that VCR recovered during our previous evaluations. Further, Table 1 shows that VCR was effective at recovering and rendering all forms of photographic evidence from the Chase Bank app test cases (12 images in total). The real danger here is that by employing the same techniques as VCR, malware can also have access to *any image taken by any app on the smartphone* — in the same way that VCR operates independent of which app generated the photograph. Moreover, if malware has access to the image buffers in the mediaserver at the right time, it may even *alter the check image before the Chase app receives it*. In light of this, we hope to emphasize the importance of Android's intermediate service processes as a security critical component and the need for security mechanisms to prevent malware from tampering with these services.

5. RELATED WORK

Smartphone memory acquisition tools, such as LiME [3] which captures memory via a kernel module and TrustDump [32] which leverages ARM's TrustZone, have only recently become widely available.

Due to the relatively recent interest in Android memory forensics, few works have focused specifically on the topic. Originally, Thing et al. [35] investigated recovering Android in-memory message-based communications. Sylve et al. [33], followed by Saltaformaggio [27], ported existing Linux memory analysis tools to recover Android kernel data. Later, Macht [24] recovered raw Dalvik-JVM control structures. Dalvik Inspector [2] built on that to recover Java objects from app memory dumps. Most recently, GUITAR [28] recovered app GUIs from Android memory images. Hilgers et al. [17] proposed using memory analysis on cold-booted Android phones. Apostolopoulos et al. [5] recovered login credentials from memory images of certain apps. VCR follows the trend of these works as it provides a new Android memory forensics capability (i.e., generically recovering photographic evidence), which these existing efforts can hardly provide.

Originally, memory forensics works focused on in-memory value-invariants [6, 7, 15, 26, 30, 34], where data structures are identified via brute-force scanning for constant or expected values. DEC0DE [36] enhances such signatures to recover formatted textual evidence from smartphones. VCR and DEC0DE share complimentary goals: both recovering different categories of smartphone evidence. Increasingly, memory forensics tools are employing pointer traversal to locate data structures [8, 11, 25, 37]. In particular, SigGraph [21] builds maps of structures in a memory image via brute force scanning. VCR also employs value-invariants and pointer traversal in its signatures, but focuses on both recovering and rendering photographic evidence, which is a step beyond only locating data structure instances.

Later, DSCRETE [29] used binary analysis to identify data structure rendering logic within the application which defines that structure. However, no rendering logic exists within the mediaserver (which only fetches photographic data from the camera device). VCR is designed based on our knowledge of the photographic evidence buffers, and operates independently of any app's implementation (avoiding DSCRETE's application-specific binary analysis step).

Other efforts aimed to derive data structure signatures from applications via binary analysis [18, 22, 31] or unsupervised learning [13]. Such tools are essential when the subject data structures are entirely unknown, but luckily, we can rely on the "gold standard" AOSP definitions to build VCR's vendor-generic signatures. More closely related to VCR, DIMSUM [20] uses probabilistic inference to locate known data structures in un-mapped memory. However, DIMSUM requires input data structure definitions to be correct. In contrast, we assume that the AOSP definitions are not correct and employ probabilistic inference to derive signatures for vendor customizations.

6. CONCLUSION

In this paper, we have presented VCR, a memory forensics tool which recovers and renders photographic evidence from memory images. VCR contributes novel memory forensics techniques to recover key data structures in the face of vendor customizations. Our evaluation shows that VCR is highly effective at recovering and rendering photographic evidence regardless of the app which generates it. Further, our tests with different versions of the Android framework show VCR to be robust across the most popular Android versions in use today. Finally, we make several key observations about the importance of VCR rendered photographic evidence and the temporal evidence which they provide to investigations.

7. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and suggestions. We also thank Dr. Golden G. Richard III for his valuable input on the legal and technical aspects of memory forensics. This work was supported in part by NSF under Award 1409668.

8. REFERENCES

[1] Riley v. California. 134 S. Ct. 2473, (2014).

[2] 504ENSICS Labs. Dalvik Inspector (DI) Alpha. `http://www.504ensics.com/tools/dalvik-inspector-di-alpha`, 2013.

[3] 504ENSICS Labs. LiME Linux Memory Extractor. `https://github.com/504ensicsLabs/LiME`, 2013.

[4] F. Adelstein. Live forensics: diagnosing your system without killing it first. *Communications of the ACM*, 49(2), 2006.

[5] D. Apostolopoulos, G. Marinakis, C. Ntantogian, and C. Xenakis. Discovering authentication credentials in volatile memory of android mobile devices. In *Collaborative, Trusted and Privacy-Aware e/m-Services*. 2013.

[6] C. Betz. Memparser forensics tool. `http://www.dfrws.org/2005/challenge/memparser.shtml`, 2005.

[7] C. Bugcheck. Grepexec: Grepping executive objects from pool memory. In *Proc. Digital Forensic Research Workshop*, 2006.

[8] M. Carbone, W. Cui, L. Lu, W. Lee, M. Peinado, and X. Jiang. Mapping kernel objects to enable systematic integrity checking. In *Proc. CCS*, 2009.

[9] B. D. Carrier. Risks of live digital forensic analysis. *Communications of the ACM*, 49(2), 2006.

[10] B. D. Carrier and J. Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1, 2004.

[11] A. Case, A. Cristina, L. Marziale, G. G. Richard, and V. Roussev. FACE: Automated digital evidence discovery and correlation. *Digital Investigation*, 5, 2008.

[12] Q. A. Chen, Z. Qian, and Z. M. Mao. Peeking into your app without actually seeing it: UI state inference and novel android attacks. In *Proc. USENIX Security*, 2014.

[13] A. Cozzie, F. Stratton, H. Xue, and S. T. King. Digging for data structures. In *Proc. Symposium on Operating Systems Design and Implementation*, 2008.

[14] P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3), 1980.

[15] B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin. Robust signatures for kernel data structures. In *Proc. CCS*, 2009.

[16] Google, Inc. Android dashboards - platform versions. `https://developer.android.com/about/dashboards/index.html`, 2015.

[17] C. Hilgers, H. Macht, T. Muller, and M. Spreitzenbarth. Post-mortem memory analysis of cold-booted android devices. In *Proc. IT Security Incident Management & IT Forensics (IMF)*, 2014.

[18] J. Lee, T. Avgerinos, and D. Brumley. TIE: Principled reverse engineering of types in binary programs. In *Proc. NDSS*, 2011.

[19] C.-C. Lin, H. Li, X. Zhou, and X. Wang. Screenmilker: How to milk your android screen for secrets. In *Proc. NDSS*, 2014.

[20] Z. Lin, J. Rhee, C. Wu, X. Zhang, and D. Xu. DIMSUM: Discovering semantic data of interest from un-mappable memory with confidence. In *Proc. NDSS*, 2012.

[21] Z. Lin, J. Rhee, X. Zhang, D. Xu, and X. Jiang. SigGraph: Brute force scanning of kernel data structure instances using graph-based signatures. In *Proc. NDSS*, 2011.

[22] Z. Lin, X. Zhang, and D. Xu. Automatic reverse engineering of data structures from binary execution. In *Proc. NDSS*, 2010.

[23] R. R. Lopez. Battling Human Trafficking with Big Data. Invited talk, USENIX Security Symposium, 2014.

[24] H. Macht. Live memory forensics on android with volatility. *Friedrich-Alexander University Erlangen-Nuremberg*, 2013.

[25] P. Movall, W. Nelson, and S. Wetzstein. Linux physical memory analysis. In *Proc. USENIX Annual Technical Conference, FREENIX Track*, 2005.

[26] N. L. Petroni Jr, A. Walters, T. Fraser, and W. A. Arbaugh. FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digital Investigation*, 3, 2006.

[27] B. Saltaformaggio. Forensic carving of wireless network information from the android linux kernel. *University of New Orleans*, 2012.

[28] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. GUITAR: Piecing together android app GUIs from memory images. In *Proc. CCS*, 2015.

[29] B. Saltaformaggio, Z. Gu, X. Zhang, and D. Xu. DSCRETE: Automatic rendering of forensic information from memory images via application logic reuse. In *Proc. USENIX Security*, 2014.

[30] A. Schuster. Searching for processes and threads in microsoft windows memory dumps. *Digital Investigation*, 3, 2006.

[31] A. Slowinska, T. Stancescu, and H. Bos. Howard: A dynamic excavator for reverse engineering data structures. In *Proc. NDSS*, 2011.

[32] H. Sun, K. Sun, Y. Wang, J. Jing, and S. Jajodia. Trustdump: Reliable memory acquisition on smartphones. In *Proc. European Symposium on Research in Computer Security*. 2014.

[33] J. Sylve, A. Case, L. Marziale, and G. G. Richard. Acquisition and analysis of volatile memory from android devices. *Digital Investigation*, 8, 2012.

[34] The Volatility Framework. `https://www.volatilesystems.com/default/volatility`.

[35] V. L. Thing, K.-Y. Ng, and E.-C. Chang. Live memory forensics of mobile phones. *Digital Investigation*, 7, 2010.

[36] R. Walls, B. N. Levine, and E. G. Learned-Miller. Forensic triage for mobile phones with DEC0DE. In *Proc. USENIX Security*, 2011.

[37] J. Zeng, Y. Fu, K. A. Miller, Z. Lin, X. Zhang, and D. Xu. Obfuscation resilient binary code reuse through trace-oriented programming. In *Proc. CCS*, 2013.