Precise and Comprehensive Provenance Tracking for Android Devices

Michael Gordon
Aarno Labs
mgordon@aarno-labs.com

Jordan Eikenberry
Aarno Labs
jeikenberry@aarno-labs.com

Anthony Eden
Aarno Labs
aeden@aarno-labs.com

Jeff Perkins
MIT/CSAIL
jhp@csail.mit.edu

Martin Rinard
MIT/CSAIL
rinard@csail.mit.edu

***Abstract*—Detailed information about the paths that data take through a system is invaluable for understanding sources and behaviors of complex exfiltration malware. We present a new system, ClearScope, that tracks, at the level of individual bytes, the complete paths that data follow through Android systems. These paths include the original source where data entered the device (such as sensors or network connections), files in which the data was temporarily stored, applications that the data traversed during its time in the device, and sinks through which the data left the device.**

The ClearScope system design enables this unprecedented level of provenance tracking detail by 1) structuring the provenance representation as references, via *provenance tags*, to *provenance events* that record the movement of data between system components and into or out of the device and 2) adopting a split design in which provenance events are streamed to a remote server for storage, with only the minimal information required to generate the tagged stream of events retained on the device. ClearScope also includes compiler optimizations that enable efficient provenance tracking within applications by eliminating unnecessary provenance tracking computations and adopting and efficient aggregate provenance representation for arrays when all array elements have the same provenance.

Experience using ClearScope to analyze the notorious Adups FOTA malware highlights the significant benefits that this level of comprehensive detail can bring. Performance experiments with the Caffeine Mark benchmarks show that the overall ClearScope provenance tracking overhead on this benchmark suite is 14%.

I. Introduction

Understanding the flow of information through a device can be critical for finding and understanding information and privacy leaks. A standard approach is to instrument the software running on the device to tag data with information about its source [1], [2], [3]. The information can then be propagated through the device and read at specified points to enforce privacy policies.

We present a new system, ClearScope, for precise and comprehensive provenance tracking of information that flows through Android devices. In contrast to previous systems, ClearScope tracks the complete path that data takes through the device, from its initial entry into the device through to its exit point, including applications, files, binders, and pipes that the data traverses along this path. ClearScope can also track up to 2^{32} combinations of information sources and intermediate information traversal points. Previous systems, in contrast, can track only a small fixed number of information sources (typically between 1 to 32 sources). And the information that ClearScope delivers has unprecedented precision, including the time of data traversal events, the precise location in the application where data traversal events take place, and the initial source or sources of relevant data at the level of individual bytes.

ClearScope includes several implementation techniques that enable this level of information to be productively collected from a running Android device. First, its system architecture includes a remote server that maintains the majority of the detailed information. This system design effectively partitions the maintained provenance information between the device and the server, maintaining the majority of the detailed information on the server and only the minimal amount of information required for efficient operation on the device. With this design, the device streams collected provenance information to the server as it executes. The device itself maintains only the tables that it needs to generate the stream of provenance events. The server retains the full provenance tracking information, including all information required to create a provenance web that captures the movement of data through the device.

ClearScope also includes several program optimizations. These include optimizations that maintain a single provenance tag for an array of values if all values in the array have the same provenance (without these optimizations the device does not even boot) and optimizations that remove provenance propagation calculations for values that do not escape the application. Together, these optimizations can reduce the provenance tracking overhead from a factor of two or more to 14% (as measured in the standard Caffeine Mark benchmark set).

We have used our implemented ClearScope system to analyze the notorious Adups FOTA malware [4] shipped with over 700 million Android devices. This malware implements a persistent, hidden information exfiltration algorithm that exfiltrates SMS messages, histories, call logs, and contacts to an external Chinese web site, with both 24 and 72 hour

exflitration cycles. Understanding this malware took Kryptowire months of analysis effort [5]. With ClearScope, we were able to analyze the exact flow, pinpoint the source of the information leak, and characterize the behavior of the malware with several hours examining the provenance logs.

This paper makes the following contributions:

- **Provenance Tracking System:** It presents ClearScope, a new provenance tracking system for Android devices. Unlike previous systems, ClearScope records the complete path that data takes as it traverses the system, including data entry and exit points, and application, file, pipe, binder, and socket traversals. The recorded provenance information includes detail such as times when provenance actions occur and provenance information that the level of individual bytes of data.
- **Design:** ClearScope collects an unprecedented amount of information about the flow of data through the device. It is infeasible to maintain this information only the device itself — the amount of information would exceed the storage capacity of typicaly Android devices. ClearScope therefore adopts a new design that streams information off to a remote server, maintaining only the information required to efficiently generate the stream locally. This novel split design is one of the key prerequisites to the effective collection and maintenance of this level of provenance information.
- **Optimizations:** ClearScope implements several optimizations that enable it to operate with acceptably low overhead (14% on Caffeine Mark benchmarks). These optimizations include using a single provenance tag to represent the provenance for all array elements when the array elements all have the same provenance and eliminating provenance calculations for provenance that does not escape the application.
- **Results:** We have used our implemented ClearScope system to analyze the Adups FOTA malware as well as 35 top Android applications from the Google Play Store. These results highlight the effectiveness of ClearScope in collecting detailed and comprehensive provenance information for these applications.

Accurate provenance information is critical for understanding device behavior and how information flows through the device. This information can be particularly critical for understanding persistent and stealthy information exfiltration malware. In comparison with previous taint tracking systems, ClearScope provides comprehensive provenance tracking that it unprecedented in the quality and detail of the information that it can provide.

II. Implementation

We next present the ClearScope implementation, including the representation of provenance throughout the system, when provenance events are generated, the overall system design, and the different ClearScope optimizations.

A. Provenance Events

ClearScope instruments the Android system and the DEX executables to emit provenance events at program points where data enters or exits the device, is stored or retrieved from files on the device, or enters or exits Android applications. These provenance events are then streamed off to a remote server, which maintains the streamed provenance information. Each event has the following fields:

- **Flow:** Tells whether the event is a source event (when data enters an application), a sink event (when information leaves an application), or other event. Examples of other events include file events (such as file creation, deletion, or open), binder events (such as open or close), and pipe events (such as open or close).
- **Event Type:** Information about the type of the event. Many events are triggered by system calls; this field typically records the system call that triggered the event.
- **Application Information:** The application id, thread id, and program point (summarized as debugging information that identifies the specific point in the program where the event occurred) for the event.
- **Time:** The time when the event occurred.
- **Event Data:** Provenance data for the event. This data typically includes the provenance tags for each byte of transferred data (run-length encoded for events that transfer multiple bytes). It can also optionally include all of the transferred data.

B. Provenance Tags

ClearScope maintains a 32 bit provenance tag for every byte of primitive data (characters, integers, floating point numbers, booleans, etc.) accessed by Android applications, stored in the local file system, or transferred between Android applications. For data in Android applications, these tags are stored in shadow fields that ClearScope adds to the Java data structures for this purpose. For data stored in file systems, each file has a shadow file that stores this provenance information. For data between Android applications, we have modified the communication mechanism (such as Binder) to include additional metadata that carries these provenance tags.

Each 32 bit tag indexes data structures that maintain detailed provenance information about the tagged data. Conceptually, these data structures maintain information about the last provenance event for the data, with the data structures linked together to enable the reconstruction of the complete provenance web of events for each byte of primitive data in the system. This provenance web captures the complete path through the device for that byte. We next detail the information that the provenance tags index.

Provenance Sets: Some values are derived from multiple pieces of data. For example, a value may be computed by adding values read from a file to values read from the GPS on the device. Provenance sets record the sets of provenance tags that capture this value derivation information.

Previous Provenance Tag: This tag links provenance data structures together to enable the server to reconstruct the

complete provenance web for each byte of information. The nodes in this web are the provenance events that record the movement and computation of data through the system. The edges record relationships between these events. For example, if an application reads data from a file, the provenance tags for the data inside the application will index a data structure that stores information about the corresponding file read event. The previous provenance tag in this data structure will index a data structure that stores information about the file write event that wrote the data into the file. The previous provenance tag for this file write data structure will, in turn, index a data structure that stores the provenance information for the provenance event that injected the data into the application that wrote the file. In this way the previous provenance tags enable the reconstruction of the complete provenance provenance web that captures the detailed flow of information through the device.

File Provenance: This data structure maintains information about provenance events on files. There are several cases:

- **File Write:** Each file has a shadow file that stores the provenance information for the data in that file. Each of the tags in this shadow file references provenance information that summarize the file write events that stored the data in that file. The recorded information includes the application that wrote the data and the statement in the application that wrote the data. The previous tag enables ClearScope to trace the provenance of the data back through the application that write the data.
- **File Read:** Data that was obtained by reading a file has a provenance tag that indexes a data structure that records information about the file read events that generated the data. The recorded information includes the file, the offset within the file for the data, and the time of the read. The previous tag indexes the corresponding file write data structures that summarize the events that wrote the data into the file.
- **File Open, Close, Delete:** The indexed data structure records information about the file open, close, or delete operation. This information includes the application and statement within the application that performed the operation.

Binder Provenance: Android uses the Binder mechanism [6] to communicate information between Android software components. ClearScope maintains detailed provenance information for information communicated via the Binder, including byte-level provenance for all communicated data. Supporting this detailed provenance information required extensive changes to the Binder implementation to support passing additional provenance information through the Binder interface.

Android applications also access Android services via the Binder. To support these services, we developed 81 provenance types to identify the specific service that generated each byte of data. Examples include the camera, the microphone, the GPS, and a wide variety of sensors. So, for example, if an application reads data from the camera, the provenance tags for the camera data inside the application will index data structures that identify the data as coming from the camera along with metadata such as the time when the data was read from the camera.

Binder performs file descriptor translation across binder calls — a file descriptor in one application can be transferred via the Binder to another application, which can then use the translated file descriptor to read the referenced file. ClearScope augments the Binder implementation to appropriately configure the file descriptors for the shadow file in the application receiving the information from the Binder.

Network Provenance: Network provenance data structures record information about provenance events for the network. The recorded information includes the IP address and port and the time of the network read or write. It is also optionally possible to record the transmitted or received information. ClearScope also records provenance events that open or close network connections.

Pipe Provenance: Pipe provenance data structures record information about provenance events involving pipes. The recorded information includes the two communicating applications and the time of the communication. ClearScope also records provenance events that open or close pipes.

C. System Design

By streaming much of the provenance information off the device to a remote server, ClearScope avoids the need to accumulate all of the provenance information on the device. This design decision is critical to enabling ClearScope to function on Android devices such as smartphones, which typically have limited storage capacity in comparison with a remote server.

The decision to structure the provenance system as events referenced by provenance tags enables this productive division of responsibility between the device and the server. With this design, the events, which contain the vast majority of the information, are stored on the server and available for analysis. The device stores the shadow files for the file system on the device and per-application provenance tag mappings that store just enough data to enable the device to memoize provenance lookups and generate the stream of provenance events. This approach enables ClearScope to deliver unprecedented levels of provenance detail, including the construction of a complete byte-level provenance web, while still operating on devices with limited resources.

With this system design, each application has its own provenance table, stored in application memory in the (modified) Dalvik runtime. This table enables the application to perform the required quick memoized provenance tag lookups. Provenance tags are unique across applications and allocated to applications in blocks by a tag system service built for this purpose.

D. Provenance Propagation

We next present an overview of the provenance propagation algorithm in ClearScope. We start with the basic algorithm,

then discuss several optimizations: aggregate array provenance, loop specialization, method specialization, and dead provenance elimination.

Basic Provenance Propagation: The basic provenance propagation algorithm instruments the DEX code to appropriately propagate the provenance across individual computing instructions within the Android application. The instrumentation augments each primitive Java value with a shadow provenance tag. Provenance information for composite values such as Java objects are comprised of the union of the provenance information for the primitive values contained in the object.

The ClearScope compiler instruments the Android DEX code to include additional instructions that propagate the provenance tags. For each load or store instruction, the compiler adds a corresponding load or store that propagates the provenance tags to the corresponding shadow fields. For compute instructions (such as instructions that add two values), the ClearScope compiler inserts a provenance join operation. This operation takes the provenance tags for the operands of the compute instruction and returns a new provenance tag for the join of the two operand provenance tags. This returned join value typically indexes a provenance set containing a list of the two operand provenance tags.

The ClearScope instrumentation memoizes calls to the provenance join operation. If the two operand provenance tags have been previously joined, the instrumentation simply returns the provenance tag from the previous join operation. This memoization improves performance and eliminates the excessive creation of new operand tags that would otherwise take place.

The instrumentation also augments procedure calls with shadow parameters to hold the provenance information for any primitive parameters. There is a single global object that holds the provenance information for the return value. Binder and pipe calls are also augmented to pass provenance information in addtion to the values. This provenance information is maintained at the level of the individual bytes of transferred data.

The DEX instrumentation can be added either offline or on the device. ClearScope implements a mechanism that intercepts the call to the DEX compiler, adds the instrumentation, then proceeds on to invoke the DEX compiler on the instrumented DEX code.

Array Aggregation Optimization: Many arrays store data with homogeneous provenance information, i.e., all array elements have the same provenance tags. ClearScope optimizes for this common case by storing a single provenance tag for all array elements when these elements have the same tag. This optimization substantially reduces the ClearScope memory footprint and makes the difference between a feasible and infeasible system — without this optimization the device will not boot.

Because of this optimization, the ClearScope DEX instrumentation has to check several cases on each array access (in the absence of the loop specialization optimization described below). Each array can be in one of two states: *aggregated* (in which there is a single provenance tag for all array elements) or *expanded* (in which there is a shadow array that holds the provenance information, with each element of the shadow array holding the provenance for the corresponding array element). Array reads check array the state to determine whether to fetch the provenance tag from the aggregate tag or the shadow array. Array writes check the array state along with the provenance tag for the written value to determine if the instrumentation should 1) leave the aggregate provenance tag intact (if the array is in the aggregate state and the written array element has the same state as the array's aggregate state), 2) write a shadow array element (if the array is in expanded state), or 3) expand the array and write a shadow array element (if the array is in aggregate state and the provenance tag for the written element does not match the aggregate tag).

Loop Specialization: The loop specialization optimization is designed to work with the array aggregation optimization. This optimization adds a loop header to loops that access arrays. The loop header checks for common optimizable cases, then jumps to specialized code generated for each such case. The most common optimizable case occurs when the provenance can be precomputed in the loop header for all accesses in the loop. To apply this optimization, the ClearScope compiler:

- **Array Extraction:** The ClearScope compiler analyzes the loop body to find all arrays accessed in the body.
- **Aggregate Checks:** For each extracted array, the ClearScope compiler checks to see if the array is in aggregate state. If so, it retrieves the provenance tags for each array.
- **Write Checks:** For each array written in the loop, the ClearScope compiler checks that all of the writes will write values into the array whose provenance information matches the aggregate provenance tag.

If the aggregate and write checks succeed, the loop will not change the provenance information and the ClearScope compiler generates specialized loop code that completely omits the provenance tracking code.

The ClearScope compiler also implements more sophisticated checks that, for example, check that the loop writes every element of an array and that all written elements have the same provenance. In this case the generated code inserts a single provenance assignment operation into the header that sets the provenance tag to the new value and again generates specialized code that completely omits the provenance tracking code.

Method Specialization: In some cases, depending on the calling context, ClearScope can detect that method calls wil leave the provenance information unchanged. In such cases ClearScope generates and invokes a specialized version of the method that omits provenance tracking instrumentation.

Dead Provenance Information Elimination: ClearScope does not need to maintain provenance information for computed values that do not escape the application in which they are located. Such values often occur, for example, in conditionals or loop bounds. ClearScope implements a program

analysis that detects such values and eliminates all provenance instrumentation for these values.

III. Performance Analysis

We use the CaffeineMark benchmarks [7] to measure the performance overhead that ClearScope imposes. We ran all of the benchmarks on a Samsung Nexus 6 running Android 5. The table below presents the resulting Caffeine Mark performance scores. We compare the scores without instrumentation (Original score) and with instrumentation (Instrumented score). The results show that the overhead ranges from negligible to 42%, with the overall Caffeine Mark score of approximately 14%.

Test	Original Score	Instrumented Score	Overhead
Sieve	39076	44332	-13.45%
Loop	58831	55424	5.79%
Logic	81698	92731	-13.50%
String	27504	20245	26.39%
double	28608	16394	42.69%
Method	34240	26139	23.66%
Overall	41434	35426	14.50%

With this overhead, the instrumented Android systems remain responsive and the ClearScope overhead is typically not noticeable for most interactive tasks.

IV. Adups FOTA: Forensic Case Study

This section discusses our analysis of Adups FOTA, a pre-installed firmware Android application that, at the time of discovery, included undocumented gathering and exfiltration of sensitive information. Ostensibly, the Adups application and service is a user-behavior monitoring and analytics solution distributed by Shanghai Adups Technology Company. OEM device manufactures often install these types of analytics and data-harvesting services to derive added value from their devices by accumulating (and analyzing and/or selling) data on their users. The company claims an installed base of over 700 million devices as of 2017 [8]. In the US, Adups software was distributed as pre-installed system applications on Android devices marketed by BLU and sold at leading retailers including Amazon.

In November of 2016, the computer security company Kryptowire released an analysis that claimed that Adups FOTA harvested and exfiltrated personally-identifiable information (PII) including device IMEI, SMS message history with message bodies, call logs, contact database information, installed and uninstalled applications, and application execution time and order [9]. Kryptowire noted that there are two distinct exfiltration cycles, a 24-hour cycle and a 72-hour cycle, both of which encrypt PII and send data to servers in China. The version of the application they analyzed is persistent and system-privileged as it comes pre-installed on a device. It is difficult to uninstall, and has the ability to be updated without user intervention.

Considerable manual analysis was required for Kryptowire's report on Adups FOTA, upwards of multiple analyst-months (based on personal communications with Kryptowire employees). Firstly, the discovery of the threat was purportedly due a "happenstance" series of events [10]. Furthermore, Kryptowire was able to extract the private key for which data was encrypted and using this key, they were able to inspect the network traffic of Adups FOTA for values that signified PII. Without the key, which may have been exposed due to careless or incorrect cryptography implementation, it is possible that the analysis of Adups could not have been performed via analysis of communication at all. Their analysts also performed manual analysis of decompiled source code to verify their findings guided by communication analysis, a difficult and time-consuming process, made more difficult by byte-code obfuscation.

Kryptowire extracted the APKs for the version of Adups FOTA which they analyzed and sent the packages to us for analysis with ClearScope. We instrumented the APKs with our static instrumentation system, and installed them on a Nexus 6 device running a stock Android Open Source Project (AOSP) version 6.0.1 release 74. We had to sign the system operation application (see below) using the system key prior to installation. We sporadically used the device for 4 days, including making calls, sending and receiving SMS messages, installing / uninstalling apps, and running applications (including the stock AOSP browser and email applications). We then analyzed our provenance event stream from the device.

The version of Adups FOTA analyzed included 2 APKs, identified by their package names: com.adups.fota, com.adups.fota.sysoper. The former is installed as a normal 3rd-party application with many privileges, its code obfuscated, and includes 3,580 classes and 25,806 methods. The latter is installed as the system user (essentially giving it root privileges), its code obfuscated, and is comprised of 775 classes and 5,326 methods.

In the remainder of this section, we present findings from our analysis of Adups FOTA. We were able to elicit and verify all of the behaviors reported by Kryptowire, except we did not see an update of the application. The salient different is that our analysis was performed in 4 hours of a single analyst's time. Our analysis employed tools that summarize, for each sink provenance type, the sensitive sources that flow into the sink. So within minutes our analyst was able to see that, for example, SMS message data was exfiltrated via network communication to particular IP addresses. We did not look at decompiled code, and our sinks report values and tags prior to encryption.

The four days of data for the Adups FOTA capture comprises 27 million provenance events (sources, sinks, non-provenance events, provenance tag definitions, etc.). In uncompressed human-readable ASCII form this is approximately 4GB, and includes all primitive values passed to and returned from sources, sinks, and non-provenance events, and run-length encoded provenance tags on the argument and return values.

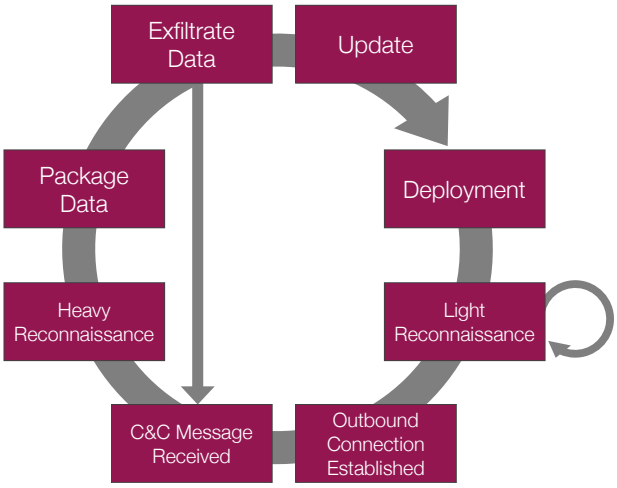

Fig. 1. Adups Advanced Persistent Threat Lifecycle.

```
POST /dm/pushInterface.do HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 483
Host: push5.adups.com
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp/2.7.5

mid=201612230195154YU223&module=register&appv=V3.3.0&model=AOSP%20on%20Shamu&project=unknownoem_unknownproduct_en_other&channel=unknownoem_unknownproduct&product=fota5&imei=990005302945978&imsi=310260517339036&wifimac=5c%3A51%3A88%3A48%3A74%3A1d&operator=310260&sn=89012605157733390367&sim=14136298485&sdklevel=22&sdkversion=5.1.1&apn=WIFI&language=en_US&resolution=1440*2392&oem=motorola&buildnumber=aosp_shamu-userdebug%205.1.1%20LMY48B%20eng.jeikenberry.20161229.134919%20test-keysiso-8859-15, iso-8859-2, iso-8859-3, iso-8859-4, iso-8859-5, iso-8859-6, iso-8859-7, iso-8859-8, iso-8859-9, jis_x0201, jis_x0212-1990, koi8-r, koi8-u, shift_jis, tis-620, us-ascii, utf-16, utf-16be, utf-16le, utf-32, …
```

Fig. 2. Adups 24-hour exfiltration HTTP post.

Figure 1 provides an overview of the life-cycle of what we interpret as an advanced persistent threat. The light reconnaissance takes place on a 24 hour cycle, after which it opens an outbound connection and sends the retrieved information to the Adups server. The 72 hour reconnaissance cycle takes place when the device receives the command and control message from the Adups server. The ensuing heavy reconnaissance retrieves SMS messages and other data, packages the data, then sends it off to the Adups server. We next discuss this process in more detail.

A. 24-Hour Exfiltration Cycle

Every 24 hours, Adups sends a message to a server that includes PII. The message includes the device IMEI and IMSI (both of which by are considered sensitive), and device hardware and software information. In Figure 2, we show an annotated example of the HTTPS post for this exfiltration cycle. Distinct colors of the text denote distinct tags on the character of the post (black characters are program constant data). This data appears in a `ssl_write` sink call that is included in Google's Conscrypt secure socket library, and the data is sent to push.adups.com at IP 118.193.187.35 port 443. Here we can see that ClearScopeis providing character-level provenance that associates with this simple encoding scheme, i.e., fields of the HTTP post.

In Figure 3 we provide summarized provenance derivations for three of the tags used in the post of Figure 2. The

Fig. 3. Three provenance examples from Adups 24-hour exfiltration.

SSL_write(…) to bigdata.adups.com/118.193.254.27:443

```
211@1872300 4@1872334 2230@1872300 4@1872376 255@1872300
 4@1872414 163@1872300 4@1872438 845@1872300 4@1873047
 65@1872300 4@1873052 1124@1872300 4@1873057 8@1872300
  16@1873059 4@1873060 65@1873059 4@1873061 56@1873059
   4@1873062 64@1873059 4@1873063 58@1873059 4@1873064
  62@1873059 4@1873065 61@1873059 4@1873066 67@1873059
```

Fig. 4. Beginning of run-length encoded provenance tag stream for Adups's 72-hour exfiltration communication. Communication is compressed prior to exfiltration, so ASCII representation of data is not helpful.

figure presents that the provenance on the IMEI data represents data returned from the `com.android.internal.telephony.ITelephony.getDeviceId()` RPC call on the telephony service (via Binder). Also, we can see the call that retrieves the IMSI. Finally, we show that data that looks like character encoding schemes was originally retrieved from an Adups server, and read via the Conscrypt library.

B. 72 Hour Exfiltration Cycle

We next discuss the 72 hour exfiltration cycle. This cycle starts with the reception of a command and control packet from bigdata.adups.com/118.193.254.27:443. When the device receives this packet it reads the SMS and contacts databases and writes the information to `analytics.db`. It then reads the data back from `analytics.db` and writes the data to intermediate JSON files. It zips the files, deletes them, the sends the zipped files to `bigdata.addups.com:443`.

Figure 4 presents the start of the run-length encoded provenance tag stream for the 72 hour exfiltration cycle. We capture the data before SSL encryption and that we maintain accurate provenance information even through the compression algorithm code.

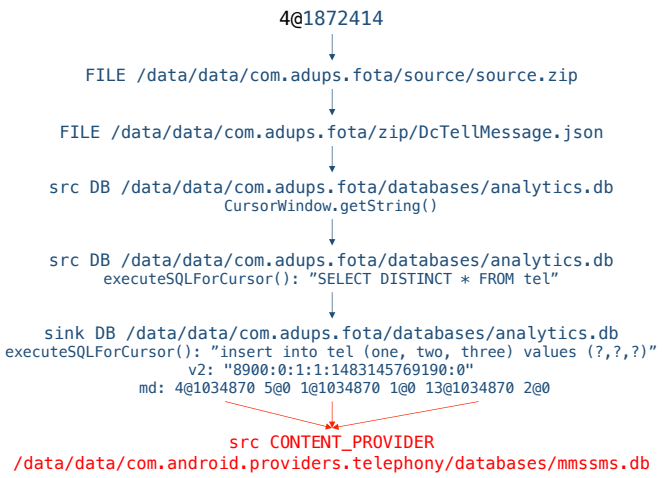

Fig. 5. Example of one provenance tag derivation from 72-hour exfiltration cycle.

Fig. 6. Timeline of reads of sensitive information relative to network send operation for 72-hour exfiltration.

Figure 5 presents the provenance web for 4 bytes of transmitted data with provenance tag 18723414. This web traces the data back starting from the `source.zip` file containing the zipped JSON data. The zipped JSON data came from the `DcTellMessage.json` file, then from the `analytics.db` via a call to `CursorWindow.getString()` and `executeSQLForCursor()`. The provenance web eventually traces the transmitted data back to the SMS database containing the SMS data (red text in Figure 5), clearly indicating the exfiltration of that data.

Figure 6 presents information that shows the relative timing of various events involved in the exfiltration. This timing information shows that these events are spread over several days up to the actual exfiltration. All of these events are opportunities to observe the impending exfiltration.

C. Discussion

As the Adups case study indicates, the detailed provenance information can provide insight into the flow of data through the device that can immediately highlight the operation of the information exfiltration malware. The generated provenance web can immediately surfaces the sequence of events that caused the exfiltration, in this case reducing the time required to understand the exfiltration from months to hours. ClearScope makes the information immediately apparent and can deliver detailed information available via no other mechanism or system.

V. Related Work

There have been numerous taint systems that have tracked from one to thirty-two binary properties by instrumenting binaries [11], [12], [13], [14], virtual machines [1], [2], and source/byte code [3], [15], [16], [17], [18], [19].

Unlike these systems, ClearScope tracks detailed provenance information rather than a short list of binary properties. Significant information included in ClearScope's provenance information that cannot be provided by such systems includes: detailed program point information (class function, line number); date and time the data enters/exits the program; pathname for data read from a file; and previous processes that have manipulated the data. The split device/server ClearScope design, which is unique in this field, is critical to enabling this level of provenance tracking detail.

A. Android

TaintDroid [1] tracks 32 bits of taint information with a modified Android VM at the level of primitives, strings and arrays (conflating the taint over array elements). TaintDroid is designed to track information leaks specifically and sets the 32 bits to track different information sources. For efficiency, TaintDroid encodes each information source in a different bit and can therefore track at most 32 information sources.

TaintART [2] applies an approach similar to TaintDroid but applies it to newer versions of Android that use the Android RunTime (ART) environment. ART uses ahead-of-time compilation for Android applications. TaintART modifies the ART compiler to track taint. Like TaintDroid it conflates string and array elements for efficiency.

Unlike TaintDroid and TaintART, ClearScope tracks detailed provenance information for each value in the application. Instead of tracking only 32 information sources, ClearScope uses its 32 bit tag to index provenance information data structures, enabling it to support many more information sources with much more precision. And the ClearScope design, which stores the majority of the provenance information on a remote server rather than locally, enables ClearScope to maintain detailed information on the complete path that each byte of data took through the device. And ClearScope is precise for all primitive values, including characters within strings and elements within arrays.

Aurasium [20] applies a security policy to APKs. These policies monitor for possible security and privacy violations such as attempts to retrieve a user's sensitive information, send SMS covertly to premium numbers, or access malicious IP addresses. Aurasium does not track provenance and cannot apply policies that depend on the source of the data.

Unlike Aurasium, ClearScope tracks detailed provenance information for each value in the application allowing it to support more fine grained policies (such as data written from a particular file should never be sent to the network).

Dagger [21] tracks application behavior at runtime including system calls, Android binder transations and application process details. It builds a data provenance graph of the interactions between the application and the phone system. It is intended to be able to dynamically identify patterns that indicate malware in the application. Dagger does not track data within the program and is thus unable to identify the source of application data at system calls.

Unlike Dagger, ClearScope tracks precise provenance information for each value in the application. This allows ClearScope to provide specific provenance information for application data in system calls. ClearScope also provides the information required to track the complete path of each byte of data through the system, including the precise source and path of each byte of data that appears at system calls.

B. Non-Android systems

Ninja [22] is a malware analysis framework that uses ARM hardware debug features to enhance its transparency to malware. It is not intended to be deployed on user devices but can be used to analyze captured malware. It does not track taint or provenance information and is instead focused on system calls, call stacks and other debugging features.

Unlike Ninja, ClearScope can track detailed provenance information on user devices. Its performance is sufficient to allow it to be used at runtime. Runtime analysis provides complete transparency since malware that hides in the presence of analysis presents no danger.

[23] instruments programs using LLVM to track programs at the level of function calls. It can build a graph that relates callers to callees and capture the arguments of the function calls. Users can specify the function calls of interest. It does not track data so it is unable to identify the source of data at system calls.

Unlike [23], ClearScope tracks precise provenance information for each value in the application allowing it to provide much more detailed and pertinent information. It is focused on data flow rather than on function calls.

VI. Conclusion

Detailed provenance tracking provides information that can be critical to the rapid understanding of information and privacy leaks. To date, however, the overhead and complexity of obtaining such information has hampered the development of systems that can deliver this information. ClearScope, with its combination of split device/server design and effective compiler optimizations, enables, for the first time, the ability to collect the information required to build a complete, byte-level provenance web that tracks the complete path each byte follows through the system. Experience using ClearScope on the Adups FOTA malware highlights the benefits that this information can deliver in this context; performance results highlight the performance benefits that its compiler optimizations can deliver.

VII. Acknowledgements

This research was supported by DARPA (Grant FA8650-15-C-7564)

References

[1] W. Enck, P. Gilbert, B. Chun, and L. Cox, "TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones," in *OSDI*, 2010.

[2] M. Sun, T. Wei, and J. C. Lui, "Taintart: A practical multi-level information-flow tracking system for android runtime," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 331–342. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978343

[3] J. Bell and G. Kaiser, "Phosphor: Illuminating dynamic data flow in commodity jvms," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '14. New York, NY, USA: ACM, 2014, pp. 83–101. [Online]. Available: http://doi.acm.org/10.1145/2660193.2660212

[4] K. LLC, "Kryptowire discovers mobile phone firmware that transmitted personally identifiable information (pii) without user consent or disclosure," https://www.kryptowire.com/adups_security_analysis.html, 2016.

[5] A. Stavrou, personal communication.

[6] Android, "Using binder ipi," https://source.android.com/devices/architecture/hidl/binder-ipc, 2017.

[7] P. S. Corporation, "Caffeinemark 3.0," http://www.benchmarkhq.ru/cm30/, 2016.

[8] ADUPS. (2016) adups fota. [Online]. Available: http://www.adups.com/index.php

[9] Kryptowire. (2017) Kryptowire discovers mobile phone firmware that transmitted personally identifiable information (pii) without user consent or disclosure. [Online]. Available: https://www.kryptowire.com/adups_security_analysis.html

[10] M. Apuzzo and M. S. Schmidt, "Secret back door in some u.s. phones sent data to china, analysts say," *New York Times*, November 2016.

[11] F. Qin, C. Wang, Z. Li, H. Kim, Y. Zhou, and Y. Wu, "Lift: A low-overhead practical information flow tracking system for detecting security attacks," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, Dec 2006, pp. 135–148.

[12] P. Saxena, R. Sekar, and V. Puranik, "Efficient fine-grained binary instrumentationwith applications to taint-tracking," in *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '08. New York, NY, USA: ACM, 2008, pp. 74–83. [Online]. Available: http://doi.acm.org/10.1145/1356058.1356069

[13] Q. Zhao, D. Bruening, and S. Amarasinghe, "Efficient memory shadowing for 64-bit architectures," in *Proceedings of the 2010 International Symposium on Memory Management*, ser. ISMM '10. New York, NY, USA: ACM, 2010, pp. 93–102.

[14] E. Bosman, A. Slowinska, and H. Bos, "Minemu: The world's fastest taint tracker," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6961 LNCS, 2011, pp. 1–20.

[15] W. G. J. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter sql injection attacks," pp. 175–185, 2006.

[16] E. Chin and D. Wagner, "Efficient character-level taint tracking for Java," in *Proceedings of the 2009 ACM Workshop on Secure Web Services*, 2009.

[17] W. Xu, S. Bhatkar, and R. Sekar, "Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley, CA, USA: USENIX Association, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267336.1267345

[18] R. Mui and P. Frankl, "Preventing web application injections with complementary character coding," in *Proceedings of the 16th European Conference on Research in Computer Security*, ser. ESORICS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 80–99. [Online]. Available: http://dl.acm.org/citation.cfm?id=2041225.2041232

[19] S. Son, K. S. McKinley, and V. Shmatikov, "Diglossia: detecting code injection attacks with precision and efficiency," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1181–1192. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516696

[20] R. Xu, H. Saïdi, R. Anderson, and H. Saıdi, "Aurasium: Practical Policy Enforcement for Android Applications," *Proceedings of the 21st USENIX conference ...*, p. 27, 2012. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final60.pdf{\%}5Cnhttp://dl.acm.org/citation.cfm?id=2362793.2362820

[21] C. Yang, G. Yang, A. Gehani, V. Yegneswaran, D. Tariq, and G. Gu, "Using provenance patterns to vet sensitive behaviors in android apps," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 164, 2015, pp. 58–77.

[22] J. Paupore, E. Fernandes, A. Prakash, S. Roy, and X. Ou, "Practical Always-On Taint Tracking on Mobile Devices."

[23] D. Tariq, M. Ali, and A. Gehani, "Towards Automated Collection of Application-Level Data Provenance," *TaPP'12 Proceedings of the 4th USENIX conference on Theory and Practice of Provenance*, pp. 16–16, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342875.2342891{\&}coll=DL{\&}dl=GUIDE{\&}CFID=206221576{\&}CFTOKEN=57724911