# Preserving Dates and Timestamps for Incident Handling in Android Smartphones

Robin Verma, Jayaprakash Govindaraj, Gaurav Gupta



Chapter 14

# PRESERVING DATES AND TIMESTAMPS FOR INCIDENT HANDLING IN ANDROID SMARTPHONES

Robin Verma, Jayaprakash Govindaraj and Gaurav Gupta

**Abstract** The "bring your own device" (BYOD) policy is rapidly being adopted by enterprises around the world. Enterprises save time and money when they allow employees to bring their own electronic devices to the workplace; employees find it convenient and efficient to use a single device for professional and personal use. However, securing the personal and professional data in the devices is a huge challenge for employers and employees. Dates and timestamps constitute important evidence when devices have been compromised or used for illegal activities. This paper focuses on the malicious tampering of dates and timestamps in Android smartphones. The proposed reactive approach gathers kernel-generated timestamps of events and stores them in a secure location outside an Android smartphone. In the case of a security incident, the stored timestamps can assist in an offline digital forensic investigation. To our knowledge, this is the first attempt to preserve authentic Android event timestamps in order to detect potential malicious actions, including anti-forensic measures.

**Keywords:** Android smartphones, dates and timestamps, preservation

## 1. Introduction

The growth and adoption of smartphones have gathered pace during the past five years. According to a Gartner report [12], during the second quarter of 2013, the sales of smartphones surpassed the sales of feature phones and accounts, corresponding to 51.8% of market share, which amounted to approximately 225 million units. In the same report, Gartner notes that Android, an open source mobile operating system, leads the smartphone operating system market with a 79.0% share. According to the International Data Corporation [16], Android dominated the

smartphone operating system market during 2013, with around 75.3% of the share, and will continue to dominate until 2017.

In a May 2013 survey, the Pew Research Center [27] noted that 61% of American cell phone owners have smartphones, out of which 28% own Android devices. Most people prefer to have one device for both personal and professional needs, so instead of taking a new device from their employers, they bring their personal devices to the workplace. This has given rise to the "bring your own device" (BYOD) trend that is catching on around the world [19]. According to another Gartner report [29], 50% of the companies are expected to allow their employees to carry their own devices by 2017, and the majority of these devices will be smartphones and tablets.

In an enterprise environment, the BYOD policy helps employers save on the cost of devices and services, and enables employees to conveniently manage their personal and professional data on a single device. However, BYOD also has opened up new windows of opportunities for fraudsters to access, modify, edit and steal information, and then cover their tracks by tampering with data on the devices. Indeed, personal mobile devices are one of the weakest links for gaining unauthorized access to enterprise networks and carrying out malicious actions.

In a BYOD environment, the incentive for stealing information from a phone is high because both personal and corporate data are stored on the same device [19]. Access control restrictions imposed on files can protect against unauthorized access by someone other than the owner of the device; however, access control fails if the device owner decides to compromise security or adopts a cavalier attitude regarding device security. To hide their tracks, malicious individuals could attempt to tamper with metadata, including modification, access, change and/or creation dates and timestamps (MAC DTSs) to match the MAC DTSs that existed before the malicious access.

MAC DTSs of digital data on a smartphone constitute fundamental evidence in digital forensic investigations. Thus, establishing the authenticity of the recovered MAC DTSs is of prime importance. Malicious tampering of MAC DTSs mainly involves modifying dates and timestamps and/or file contents. Commercial tools, including the Cellebrite Universal Forensic Extraction Device (UFED) system and FTK Mobile Phone Examiner, can recover mobile device data; however, most of them prove to be inadequate when attempting to establish the authenticity of MAC DTSs.

This paper demonstrates that tampering with MAC DTSs on Android smartphones is possible using anti-forensic techniques. A mechanism is described for preserving dates and timestamps for incident handling

involving Android smartphones. System-generated MAC DTSs are collected with the aid of a loadable kernel module (LKM) [14], which hooks system calls to capture the values. The dates and timestamps, along with location details, are stored as event logs in a secure location outside the smartphone (e.g., a local enterprise server or the cloud). In the event of a security incident, the stored data and timestamps can assist in an offline digital forensic investigation.

## 2. Related Work

Weil [28] has shown that dynamic date and timestamp analysis can determine the actual times of events on a personal computer (especially when they are not available or tampered with) by utilizing an external source of data-time data. Carrier and Spafford [7] have suggested that dynamic timeline analysis is essential to reconstructing digital events after a security incident and identifying the suspects. To implement these ideas, a customized forensic tool named Zeitline was developed by Buchholz and Falk [6] that creates timelines from various sources and presents the details via a graphical interface; however, the tool has certain limitations in handling clock differences and clock drifts. An improved tool, Cyber-Forensic Time Lab (CFTL), was created by Olsson and Boldt [20]. A different approach to address the same problem is described by Marrington, *et al.* [18], who trace computer activity timelines by utilizing the "causality" of events in computer systems. Their tool extracts the MAC timestamps from a hard drive and correlates events according to their causality. However, the problem of MAC timestamp tampering was not been addressed in their implementation, which leads to certain irregularities in the results.

Barik, *et al.* [4] were the first to propose the logging of MAC DTSs for filesystems. Das, *et al.* [8] and Ansari, *et al.* [2] have enhanced this approach for use in filesystem intrusion detection. All three works focus on Linux systems with conventional hard disk drives as storage media. In contrast, this paper describes an Android solution that has considerable differences in terms of functionality and implementation.

Grover [11] has proposed an enterprise monitoring application, "Droid-Watch," that continuously collects data from Android phones. The collected data sets are uploaded to an enterprise server for assisting security personnel in monitoring, auditing and responding to incidents and conducting forensic investigations. However, the approach is not secure against root attacks and application uninstallation attacks.

Android phone anti-forensics is also an important related area of research. Distefano, *et al.* [9] have devised an anti-forensic technique that

uses the private folder of an installed Android application to hide counterfeit evidence; they propose a safe uninstallation process to hide evidence of tampering. Albano, *et al.* [1] have demonstrated that the timestamps of tampered files can be restored without raising suspicion and without leaving any traces in the filesystem. However, their method is only able to change the contents of the storage media; data that has been uploaded to a location external to device is untouched.

Linux security modules (LSMs) were created as a framework to support security functionality [26]. Security modules that have been implemented include Security Enhanced Linux (SELinux), AppArmor, Smack and TOMOYO Linux. Shabtai, *et al.* [24] have implemented SELinux on an Android platform to perform strict access control. The steps involved compiling the kernel with SELinux support, followed by designing an Android-specific security policy. The standard startup processes and supporting scripts were modified to load the new policy at startup. In the final step, the Android disk image (popularly called ROM) was created to be flushed on the phone.

In 2012, the National Security Agency specified the SEAndroid standard [25] for implementing SELinux on Android devices; the SELinux-enabled Android has been successfully tested against most contemporary exploits that seek to gain root access. Shabtai, *et al.* [24] and Smalley [25] require that the disk image of an Android phone or tablet be flushed with their own customized ROM, which is impractical in a BYOD environment. Our solution, on the other hand, focuses on preserving the MAC DTSs without reflushing, which keeps the phone data intact and is more suited to a BYOD environment. Additionally, the user's personal data residing on the device is preserved, enabling regular phone use to continue without any impact after the LKM installation.

## 3. MAC DTS Tampering

Our research initially focused on developing and testing an attack tool that could tamper with files and their timestamps without leaving forensic evidence. Following this, a tool was developed that could detect such tampering. We selected the HTC Wildfire and Samsung Galaxy S2 GT-I9100 Android phones as prototypes for the research. The two phones were chosen because they represent two ends of the smartphone segment in terms of processing power, storage capacity, battery power and cost. HTC Wildfire is a low-end smartphone with a 0.60 GHz CPU, single core processor, 512 MB RAM and 1230 mAh battery capacity. The Samsung Galaxy S2 is a high-end smartphone with a 1.20 GHz CPU, dual core processor, 1 GB RAM and 1,650 mAh battery capacity.

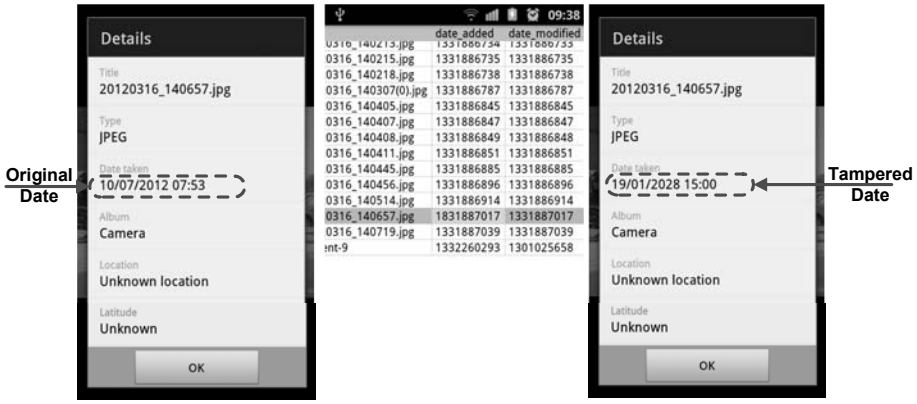


*Figure 1.* Tampering with the date of a camera image file.

The Android filesystem stores data and the corresponding metadata (including MAC DTS values) in SQLite databases. The data includes phone calls, short message service (SMS) messages, multimedia messaging service (MMS) messages, photographs/video captured by the camera, audio recorded by the microphone and other types of files (e.g., Word, Excel, PDF and text). User applications access the data through various APIs and interfaces controlled by the operating system. Applications need explicit permissions to use the APIs and interfaces; the permissions are generally granted to them at the time of their installation. The Android filesystem ensures that users have no direct access to the SQLite databases by storing them in the restricted internal memory of the phone.

Although Android is claimed to be secure by design, it is possible to bypass the Android security mechanisms (see, e.g., [5, 17]). In fact, several timestamp tampering applications are available. For example, one application named SQLite Editor (available at `goo.gl/ppxAXo`) can be used to overwrite the timestamps of received SMS messages. We also built a custom application that can tamper with the timestamps of calls, SMS messages, camera images and video files on Android phones. Figure 1 shows screenshots of a camera image file before and after tampering with the date.

## 3.1 Attack Methodology and Anti-Forensics

Anti-forensic actions seek to destroy or tamper with digital evidence [9, 13]. Specific actions include destroying evidence, hiding evidence, altering evidence and counterfeiting evidence. Our attack methodology

involved creating a MAC DTS tampering application that performs the following actions:

- **Eliminating/Altering Evidence:** Evidence is not created or evidence is altered to thwart a forensic investigation. The MAC DTS tampering application extracts data entries from the SQLite database via APIs into a text file, and the data in the text file is changed. For example, the Type column value of a call entry in the *calls* table in file `contacts2.db` is changed to "5," for which the legitimate values are "1" for an incoming call, "2" for an outgoing call and "3" for a missed call. After the change is made, the application writes the text back to the database. A forensic tool that extracts database entries would ignore the edited entry when extracting call records. Several other similar operations can be performed on other potential evidence in a SQLite database.

- **Counterfeiting Evidence:** Fake evidence is implanted to thwart a forensic investigation. For example, the tampering application can overwrite entries in a SQLite database. Also, it can overwrite the logcat buffer that stores the debugging logs.

- **Destroying Evidence:** Evidence is deleted to thwart a forensic investigation. For example, the tampering application can delete the contents of a SQLite database and the logcat buffer.

Note that data retrieval is possible after these anti-forensic actions are performed. However, the retrieval is both costly and time-consuming [10, 22].

**Attack Scenario 1.** The first scenario assumes that an individual left his Android phone unattended for five to ten minutes, during which time an attacker gained physical access to the phone. The following steps are involved in the attack scenario:

- The phone is connected to a laptop that has Android SDK (open source software development kit) installed.

- The MAC DTS tampering application is installed on the phone via the laptop. The application can be installed even if the phone is locked. However, like all contemporary mobile forensic tools, the USB debugging mode must be enabled.

- The contents of camera image database are extracted to the internal memory folder of the application, following which the file is pulled to the laptop.

- The file metadata and MAC DTSs are modified.
- The updated file is pushed back to the phone and the phone database is updated.
- The logcat entries in the phone are cleared.

The process requires the execution of a small set of simple commands (that could easily be combined into a script). The entire process takes about three minutes (five iterations), ample time to carry out the attack in the real world. Note that physical acquisition from the phone could reveal the deleted data, but this would be expensive and time consuming [22].

**Attack Scenario 2.** The second scenario assumes that a company employee intends to steal a confidential document. Since the time at which he accessed the document would be suspicious, he accesses the file via his Android phone and modifies the MAC DTS values to hide his tracks. The following steps are involved in the attack scenario:

- The MAC DTS tampering application is installed on the phone.
- The confidential file is accessed and stolen.
- The MAC DTS values are changed to their values before the access.
- The tampering application is uninstalled and the logcat entries are cleared.

## 4. Implementation

Our solution for preserving dates and timestamps engages a custom LKM that hooks the `sys_open()` system call and captures the kernel-level timestamp values of selected files. The timestamps are then uploaded to a secure local server.

Figure 2 shows a schematic diagram of the implementation architecture. Details about the LKM algorithm and the procedures for compiling the LKM and loading the LKM on a phone are provided below. After the LKM is loaded, it reads the MAC DTS values from the phone. The captured values are written to a temporary log file that is stored in the internal memory of the phone. The log file is uploaded to a local server when network connectivity is available.

### 4.1 LKM Algorithm

The LKM algorithm shown in Figure 3 incorporates four functions: `root_start`, `hacked_open`, `hacked_unlink` and `root_stop`. Function

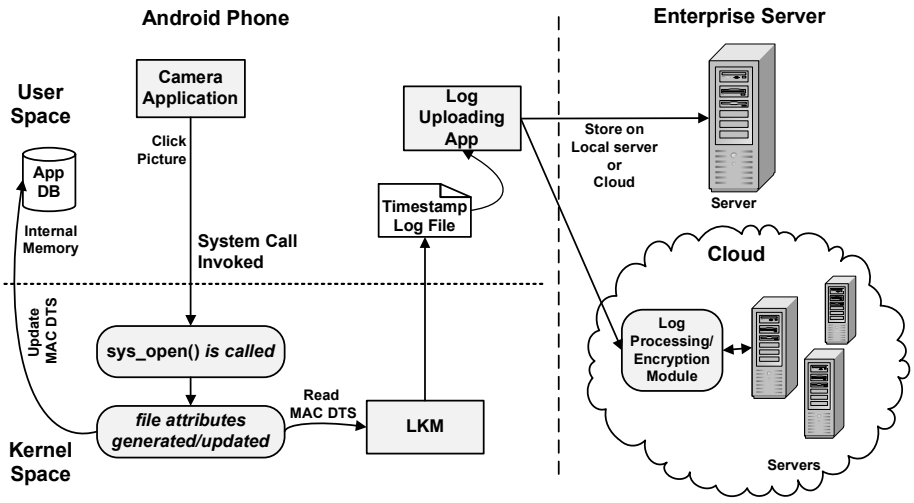


*Figure 2.* Preserving dates and timestamps.

`root_start` initializes the log file pointer, sets the `sys_open()` call address to point to the `hacked_open()` call and sets the `sys_unlink()` call address to point to the `hacked_unlink()` call. Function `hacked_open` takes the file path and file attributes as input parameters. If the file path name matches the selected file or folder in the filter array, then the date and timestamp details are written to a log file; following this, control is passed back to the original `sys_open()` call. Function `hacked_unlink` takes the file path name as an input parameter. If the file path matches the log file path, then the delete operation is denied; otherwise, control is passed back to the original `sys_unlink()` call. Function `root_stop` function closes the log file pointer, restores the address back to the original `sys_unlink()` call routine and restores the address back to the original `sys_open()` call routine.

## 4.2 LKM Compilation

The LKM must be compiled against the kernel source code on which it has to be loaded. The compilation instructions are posted on the Android Open Source Project website (`goo.gl/TQVVV`). The first step is to set up the kernel environment on a computer. Following this, it is necessary to acquire the original kernel source code of the Android version installed on the device. For the two scenarios described above, we downloaded the Android 2.1 Eclair kernel source from HTC [15] and the Android 2.3 Gingerbread kernel source from Samsung [23].

```
Function root_start
  Output: File Pointer to the record.log file
    {Log File Pointer =  File Open ("/data/data/record.log")
     Original Sys Open Pointer = sys_call_table[__NR_open]
     Original Sys Unlink Pointer = sys_call_table[__NR_unlink]
     sys_call_table[__NR_open] = Our Sys Open call hacked_open
     sys_call_table[__NR_unlink] = Our Sys Open call hacked_unlink
    }

Function hacked_open
  Input: File Path Name, Flags
  Output: File Pointer to the record.log file
    {Filter  = File/Folder path for enabling timestamp logging
     Path Name = Current working directory
     If (Filter = Path Name)
         Log Path Name, Date and Time and File Operation Type ID
     Else
         Return to Original Sys Open Operation
     EndIf
    }

Function hacked_unlink
  Input: File Path Name
    {If (Path Name = Log File Path)
         Return -1    /* Deny log file deletion. */
     Else
         Return to Original Sys Unlink Operation
     EndIf
    }

Function root_stop
  Original Sys Open Pointer, Log File Pointer
    {Restore to Original System Call
     sys_call_table[__NR_open] = Original Sys Open Pointer
     sys_call_table[__NR_unlink] = Original Sys Unlink Pointer
     Close Log File Pointer
    }
```

*Figure 3.* Algorithm for logging MAC DTS values using the LKM.

The next step is to locate the physical address of the system call table on each phone in order to hook system calls. A mandatory condition for loading the LKM successfully on the phones is that the "vermagic string" (see Table 1 for details) of the kernels and LKMs should match [21].

*Table 1.* Smartphones used in the experiment.

| Phone | Android Version | Vermagic String | sys_call_table Address |
|---|---|---|---|
| HTC Wildfire | 2.1 Eclair | 2.6.29-4266b2e1 | `0xc002bfa4` |
| Samsung Galaxy S2 GT-I9100 | 2.3 Gingerbread (XWKL1) | 2.6.35.7-I9100XWKL1-CL809037 | `0xc0028fe4` |

## 4.3 Loading the LKM

Root permissions (administrative privileges) are required to load the LKM on an Android phone. Generally, temporary rooting is a good option because it involves the least number of changes to the filesystem, which is preferred in digital forensics. We decided to apply temporary rooting on the Samsung Galaxy S2; however, we permanently rooted the HTC wildfire to explore how our solution performs on permanently rooted devices. (As a matter of fact, our solution works seamlessly on temporary and permanently rooted devices.) We followed the rooting instructions listed on the XDA Developers mobile software development community portal (`www.xda-developers.com`).

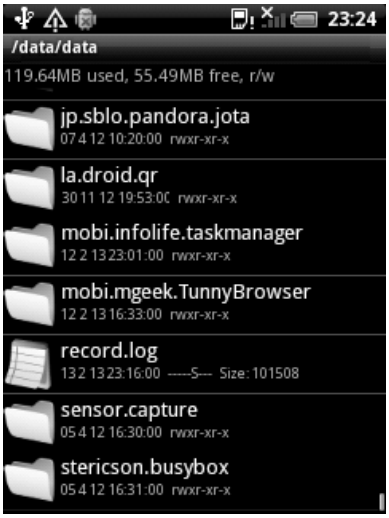


*Figure 4.* Storing the `record.log` file in internal memory.

After the phone was rooted, the LKM was loaded. The LKM monitors the events occurring on a selected set of files and directories. It reads the MAC DTS values for these files and writes them to a log file (`recordfile.log`) on the phone (Figure 4).

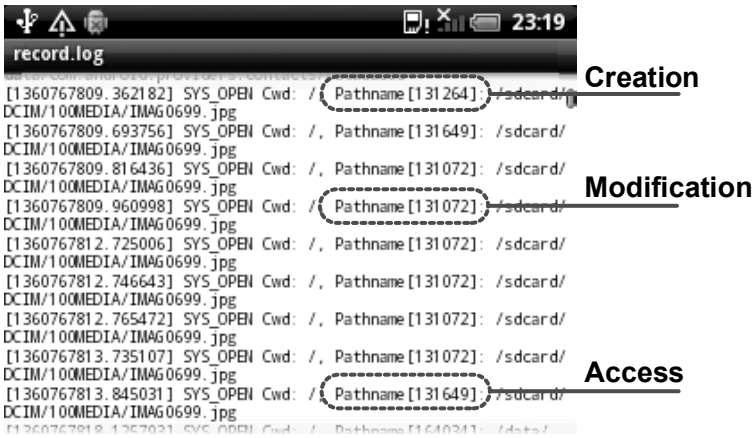


*Figure 5.* Snapshot of the `record.log` file for camera image `IMAG0699.jpg`.

## 4.4 Hiding Log File Deletions

The log file is stored in the internal memory of the Android smartphone, which is not accessible directly by applications; thus, the file is secure from deletion attempts. In the event that an attacker manually locates the log file, an attempt to delete it is denied by the LKM module, which bypasses the call to `sys_unlink` (`hacked_unlink()` call); this ensures the security of the log file. The overall approach can be categorized as "anti-anti-forensics," in that, by not allowing deletions, a possibly destructive anti-forensic attempt is foiled.

## 4.5 Uploading the Log File

The log file generated by the LKM is subsequently uploaded at regular intervals, depending on the availability of network connectivity, to a local server with the help of our uploader application. We used the WAMP software stack on our local server, which runs Apache, PHP and MySQL. The system can easily be extended to use the cloud as external storage for the uploaded logs. The MAC DTS entries for the files (Figure 5) can be identified on the basis of unique flag values associated with operations such as file creation, modification and deletion.

The LKM can read MAC DTS values for individual files such as image files, audio files, text files and SQLite databases. However, it cannot capture timestamps for individual database transactions. In the case of SMS messages and calls, records are added to their respective databases. All images, audio, video and other common files are updated directly on the storage device and the entries are added to the corresponding

databases. For this reason, along with uploading the log file, the uploader application also extracts and uploads specific details (e.g., called numbers, call status and call/SMS message times) from the call-records folder and SMS database to track the entries that have been updated in the databases.

The uploader application also writes the cell tower information and the GPS location of the phone to the log file before uploading it. Adding location information to the log file enhances the credibility of the logged MAC DTS values because it can reveal the location where the tampering may have taken place. Privacy issues arising from storing the log file on a third-party cloud server can be addressed by creating a middleware component in the cloud that encrypts the log file before storing it on the cloud server.

## 4.6 Incident Handling

When tampering is suspected, the MAC DTS values of suspected files on the phone must be cross-checked with the corresponding MAC DTS log file values stored on the local server or cloud. If the values match, no tampering is indicated; otherwise, the discrepancy should be investigated.

In some cases, tampering can be detected even when the solution is not deployed on a phone, or if there is some discrepancy that was present on the phone before the solution was operational. For example, every SQLite database on a device has a primary key (named `_id`), which increases in sequential order whenever a new record is inserted into the database. If a call is made or an SMS message is sent to an existing number, then a new record is written to the database. Camera image, video and audio files also have distinct records in their respective databases and, when they are edited, new entries with new `_id` values are written to the databases. If the timestamp values of one record with a lower `_id` value are after those of another record with a higher `_id` value (i.e., it was inserted later), then it is certain that timestamp values were tampered with. These results can then be forwarded to a digital forensic professional for further investigation.

## 4.7 Anti-Forensics

In addition to the anti-forensic techniques discussed above, we are considering an additional category, namely, forensic tools that are operating on a device [3]. When attempting to destroy evidence, the first thing that an attacker might do is to delete the evidence file itself. Our implementation cannot recover deleted data, but it can tell the time

*Table 2.* Performance benchmark parameters.

| Test Parameter | Operations |
|---|---|
| RAM R/W Operations | MB per 10 seconds for integer array copying and adding in RAM |
| CPU Integer | Million operations per 10 seconds |
| CPU Float | Million operations per 10 seconds |
| 2D Graphics | Performance evaluation for 2D animation |
| 3D Graphics | Performance evaluation for 3D animation |
| Database I/O | Performance of SQLite queries such as INSERT, SELECT and UPDATE |
| SD Card W Operations | MB of data written to SD card per 10 seconds |
| SD Card R Operations | MB of data read from SD card per 10 seconds |

when a file was deleted. If the attacker tries to tamper with the metadata of an evidentiary file, the implementation can detect the attempt and provide the original values from the stored log. If the attacker somehow identifies the location of the log file and attempts to delete it, the deletion would not be supported by the operating system.

Our solution is resistant to anti-forensic techniques that seek to hide evidence. This is because log redirection is not possible when the LKM is running. In the event that an attacker attempts to alter or counterfeit the evidence (e.g., using the technique described in [1]), the LKM would be unloaded and the logging would stop. This would result in log files not being uploaded to the local server or cloud. To address this, a provision could be made at the local server or in the cloud that, if no update is received over a certain period of time, an alarm is raised that could be used to isolate the device until further examination. The detection of running forensic tools is not directly applicable to our solution because only activities that occur after its installation can be captured. If an enterprise deploys our solution on an Android device before the device enters its ecosystem, then the activities that occurred earlier are assumed to be of no concern to the enterprise.

## 5. Impact on Performance

To examine the impact of installing the LKM on a phone, we ran 30 iterations of the System Test benchmarking tool (see `goo.gl/0f67R`) on the HTC Wildfire smartphone with and without the LKM. We selected this tool specifically because it provides results for various system peripherals at the same time. The system components considered in the benchmarking are listed in Table 2. The graph in Figure 6 summarizes the results.

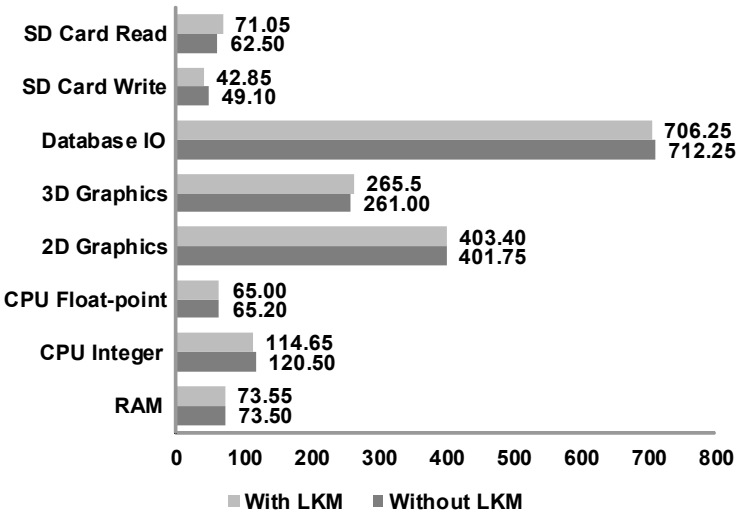


*Figure 6.* Performance benchmarking results for the HTC Wildfire smartphone.

Our experiments demonstrated that the solution does not impact the performance of the HTC Wildfire smartphone to a significant degree. Similar results were observed for the Samsung Galaxy S2 smartphone.

## 6. Conclusions

Android smartphones used in a BYOD environment are vulnerable to confidentiality and integrity attacks, especially those involving tampering with potential evidentiary data stored on the phones. The solution described in this paper helps detect malicious data tampering by storing authentic copies of the MAC DTS values of selected files and directories on a local server or in the cloud for later verification and validation. The solution is also applicable to Android tablets. Indeed, the concept of storing kernel-generated authentic timestamps at secure external locations for the purpose of identifying malicious activities can be implemented for a variety of mobile device operating systems.

Our future research will focus on developing an anomaly-based intrusion detection system for Android devices. MAC DTS logs can be used to construct models of expected behavior, enabling malicious activities to be detected and blocked in real time. Another topic of future research is to create a customized kernel using SE Linux as part of an integrated solution.

## References

[1] P. Albano, A. Castiglione, G. Cattaneo and A. De Santis, A novel anti-forensics technique for the Android OS, *Proceedings of the International Conference on Broadband and Wireless Computing, Communications and Applications*, pp. 380–385, 2011.

[2] M. Ansari, A. Chattopadhayay and S. Das, A kernel level VFS logger for building efficient filesystem intrusion detection systems, *Proceedings of the Second International Conference on Computer and Network Technology*, pp. 273–279, 2010.

[3] S. Azadegan, W. Yu, H. Liu, M. Sistani and S. Acharya, Novel anti-forensics approaches for smartphones, *Proceedings of the Forty-Fifth Hawaii International Conference on System Science*, pp. 5424–5431, 2012.

[4] M. Barik, G. Gupta, S. Sinha, A. Mishra and C. Mazumdar, An efficient technique for enhancing the forensic capabilities of the Ext2 filesystem, *Digital Investigation*, vol. 4(S), pp. S55–S61, 2007.

[5] M. Becher, F. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck and C. Wolf, Mobile security catching up? Revealing the nuts and bolts of the security of mobile devices, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 96–111, 2011.

[6] F. Buchholz and C. Falk, Design and implementation of Zeitline: A forensic timeline editor, *Proceedings of the Fifth Digital Forensic Research Workshop*, 2005.

[7] B. Carrier and E. Spafford, An event-based digital forensic investigation framework, *Proceedings of the Fourth Digital Forensic Research Workshop*, 2004.

[8] S. Das, A. Chattopadhayay, D. Kalyani and M. Saha, Filesystem intrusion detection by preserving MAC DTS: A loadable kernel module based approach for Linux kernel 2.6.x, *Proceedings of the Fifth Annual Workshop on Cyber Security and Information Intelligence Research*, art. 57, 2009.

[9] A. Distefano, G. Me and F. Pace, Android anti-forensics through a local paradigm, *Digital Investigation*, vol. 7(S), pp. S83–S94, 2010.

[10] E. Gal and S. Toledo, Algorithms and data structures for flash memories, *ACM Computing Surveys*, vol. 37(2), pp. 138–163, 2005.

[11] J. Grover, Android forensics: Automated data collection and reporting from a mobile device, *Digital Investigation*, vol. 10(S), pp. S12–S20, 2013.

[12] A. Gupta, C. Milanesi, R. Cozza and C. Lu, Market Share Analysis: Mobile Phones, Worldwide, 2Q13, Gartner, Stamford, Connecticut, August 13, 2013.

[13] R. Harris, Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem, *Digital Investigation*, vol. 3(S), pp. S44–S49, 2006.

[14] B. Henderson, Linux Loadable Kernel Module HOWTO (`tldp.org/HOWTO/Module-HOWTO`), September 24, 1006.

[15] HTC Corporation, The HTC Developer Center, Taoyuan, Taiwan (`www.htcdev.com/devcenter`), 2013.

[16] International Data Corporation, Worldwide mobile phone market forecast to grow 7.3% in 2013 driven by 1 billion smartphone shipments, according to IDC, Press Release, Framingham, Massachusetts, September 4, 2013.

[17] M. La Polla, F. Martinelli and D. Sgandurra, A survey on security for mobile devices, *IEEE Communications Surveys and Tutorials*, vol. 15(1), pp. 446–471, 2013.

[18] A. Marrington, I. Baggili, G. Mohay and A. Clark, CAT Detect (Computer Activity Timeline Detection): A tool for detecting inconsistency in computer activity timelines, *Digital Investigation*, vol. 8(S), pp. S52–S61, 2011.

[19] K. Miller, J. Voas and G. Hurlburt, BYOD: Security and privacy considerations, *IT Professional*, vol. 14(5), pp. 53–55, 2012.

[20] J. Olsson and M. Boldt, Computer forensic timeline visualization tool, *Digital Investigation*, vol. 6(S), pp. S78–S87, 2009.

[21] C. Papathanasiou and N. Percoco, This is not the droid you're looking for..., presented at *DEF CON 18*, 2010.

[22] J. Reardon, S. Capkun and D. Basin, Data node encrypted filesystem: Efficient secure deletion for flash memory, *Proceedings of the Twenty-First USENIX Security Symposium*, 2012.

[23] Samsung, Samsung Open Source Release Center, Suwon, South Korea (`opensource.samsung.com`), 2013.

[24] A. Shabtai, Y. Fledel and Y. Elovici, Securing Android-powered mobile devices using SELinux, *IEEE Security and Privacy*, vol. 8(3), pp. 36–44, 2010.

[25] S. Smalley, The case for SE Android, presented at the *Linux Security Summit*, 2011.

[26] S. Smalley, T. Fraser and C. Vance, *Linux Security Modules: General Security Hooks for Linux*, NAI Labs, Santa Clara, California (`tali.admingilde.org/linux-docbook/lsm.pdf`), 2001.

[27] A. Smith, Smartphone Ownership 2013, Pew Research Center, Washington, DC, June 5, 2013.

[28] M. Weil, Dynamic time and date stamp analysis, *International Journal of Digital Evidence*, vol. 1(2), 2002.

[29] D. Willis, Bring Your Own Device: The Facts and the Future, Gartner, Stamford, Connecticut, April 11, 2013.