Paranoid Android: Versatile Protection For Smartphones

Authors: Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, Herbert Bos

Conference: Annual Computer Security Applications Conference 2010

Presented by: Jevin Sweval

Problems in Smartphone Security

- Smartphones are a new, emerging target for attacks
  - Market penetration is increasing
    - 15 million smartphones sold in the US during Q2 2010
  - Users store sensitive info on their devices (payment methods, business secrets, etc.)
  - Throughout the day, a user is carrying a potential "always-on" listening bug and location tracker
- Current security checks are unsuitable for smartphones due to resource constraints
  - ClamAV on Android takes 30 minutes to run and drains 2% of the battery

Who Cares?

- Average users may not currently care about security on their smartphone
  - There haven't been wide scale attacks **yet**
- There are still many security-minded markets
  - Government
  - Financial institutions
  - Health care
  - Military
- These markets *do* care about security and are actively seeking solutions
  - President Obama had to fight to get a specially secured BlackBerry when he became President
  - Microsoft Exchange can remotely wipe a lost/compromised phone
  - Enterprise users often cripple smartphone functionality (disable WiFi, SMS, browser, etc.) in attempts to increase security

Paranoid Android: "Cloud" Based Security

- Paranoid Android is a security infrastructure for Android smartphones
- All activity on a user device is recorded and replayed on a remote, virtual Android device
- The virtual Android device is not running on resource constrained hardware and can implement as many security checks as desired

Recording

- Computer program execution is deterministic if the nondeterministic inputs and events are known
- Paranoid Android records and transmits these nondeterministic inputs to the remote virtual Android device
- Sources of nondeterministic inputs and events
  - System calls (mostly interaction with hardware)
  - Process signals (SIGALRM, SIGINT, etc)
  - Concurrency and IPC
- Only userspace applications are traced – the kernel is assumed to be secure
  - Kernel exploits usually originate from userspace and could theoretically be detected and blocked

Nondeterminstic System Calls

- Most of an application's inputs come from system calls
  - read()/write() on a file or socket
  - Time/location data is accessed through a kernel device
- All system calls into the kernel and their corresponding results are traced and and transferred to the replayer

Nondetermistic Signals

- Serious errors (SEGFAULT, floating-point errors) are delivered synchronously when the offending instruction is executed
  - These do not need to be traced because they are deterministic
- Asynchronous signals (timer expire, SIGKILL, etc) are non-deterministic and have **no guarantees** for delivery
  - The lack of guarantees lets Paranoid Android defer their delivery until a system call is performed, keeping the actual device and replicant in sync

Concurrency and IPC

- Most IPC mechanisms are performed through syscalls
  - The syscall tracer can deterministically record and replay these types of IPC
- Shared memory and memory mapped files can be accessed entirely within the userspace
  - Solution is to serialize access to shared state
  - CREW (concurrent-read/exclusive-write) protocol relies on hardware MMU to control access to shared state
  - A deterministic task scheduler will also result in serialized access
    - This was chosen for PA's implementation because of its good performance on uniproccessor architectures

Synchronization

- Paranoid Android must upload the trace data to the replica
- Loose Synchronization
  - Data is transmitted to the replica only when the device is awake and connected to the internet
    - Battery life is preserved
  - This is the most likely time of attack (user is interacting with internet services)
- Extremely Loose Synchronization
  - Synchronize only when charging
    - Eliminated battery drain from transmitting trace data
  - Prevents timely notification of security compromises
- Errors
  - If a device fails to synchronize within a certain time period or if tampering of the trace data is detected, the device is treated as compromised

Secure Trace Storage

- The trace data is stored on the device until the synchronization strategy determines it is time to transmit it to the replica
- Malicious software could try to erase its tracks within the traces before they're transmitted
- Solution: Hash the traces before storing
  - STORE(message + HMAC(key, message)
  - key = HASH(key)
    - The key is continuously rolled prevent tampering of old records if a key is leaked

Replication

- The trace data captured on the user device is sent to the remote replication over the WWAN
  - Bandwidth used during heavy load: 2 KB/s and while idle: 64B/s
  - The device uses a PA proxy to receive network data
    - The proxy can clone data destined for the device to the replicant so the device does not have to retransmit the received data
- The replicated device uses an Android emulator to replay the trace sent by the user device

Security Methods

- Antivirus software
  - Authors use ClamAV to run periodic scans on the replicant file system
  - With system call traces, the AV could be run *on-access*
  - Multiple AV suites could be ran, if desired
- Dynamic analysis within the Android emulator
  - The authors implemented dynamic taint analysis that tracks untrusted (tainted) data throughout execution
  - Prevents tainted data from being executed or used as an operand to a CALL instruction
  - Dynamic taint analysis can detect some zero-day attacks

User Notification and Recovery

- If the replica detects an attack, it must notify the user of the device and prevent further security compromises
  - This can be difficult if the compromised device is maliciously firewalled from the PA replica
  - The carrier could shutdown service to the device (or remotely wipe it) upon request from the replica
- Since the replica is a mirror of the device, recovery before the point of attack is simple

Device Generated Data Handling

- Data that is generated locally on the device cannot be proxied and required transmission to the replica
- Transfers via Bluetooth are usually small and retransmission to the replica may not be a problem
- Large photos or video recorded by the device may require the user to disconnect from PA to prevent battery drain
  - Users who share media via Flickr, Youtube, etc may already be effectively proxying their data

Implementation Details

- The tracer was implemented as a userspace program that uses ptrace() to hook syscalls from the traced process
  - Ptrace is slow but easier to implement than a kernel space tracer
- Deterministic scheduling
  - The scheduler was modified to never run threads that share memory concurrently

Results

- Trace Data Rates
  - Booting, web browsing, navigation, and audio playback used about 1.5-2.5 KB/s
  - An idle device and making a phone call used about 100 B/s
- Device Overhead
  - CPU overhead from tracing was about 15%
    - 65% of the overhead was due to ptrace – would be eliminated by kernel space tracing
  - Battery consumption increase ranged from negligible (idle, calling) to 30% (heavy use)

Results

- The replica server's performance varies with the average device CPU utilization
  - Running on a extra-large EC2 instance with an average device CPU utilization of 25%, 100 replicas were able to run without delay
    - 30 at 100% utilization
  - A dual core laptop can run 30 replicas at 25% utilization

Criticisms

- Little discussion was given to why smartphone-aware security frameworks can't work on mobile devices
- The authors did not present a threat model as motivation for selecting their implemented security methods