Sandboxing an Android application through system call interposition
Contents

• Ransomware
• Sandboxing
• Android security model
• Trace an application
• One particular approach
Ransomware

- Malicious piece of software that extorts a payment
- Blocks access to the user’s data and blackmails the user to pay a ransom
- Encrypting and non-encrypting variants of ransomware
- Ransomware on mobile phones is not particularly new (research papers from 2015 and earlier)
- The research field in cryptovirology is far broader
- Attacks are no longer theory but have been carried out in practice (and caused real damage)
Sandbox

- Isolate a process from the host machine
- Run unverified or untrusted application without risking harm to the operating system
- Provide a restricted operating system environment
- Various moulding of sandboxes
- Restrict access to system calls
- Rule based access control
Sandbox

- Usable for legitimate software that runs in a risky environment (DNS server, browser)
- Record and analyse the tasks of the unknown application
- Monitoring the activities may yield insights into the unknown application

- When is a set of operations considered legitimate / suspicious / disruptive / ... ?
- Endless possibilities to
  - deceive user
  - cloak intents
  - ...

Privilege separation

- Unique user ID for each application
- Interactions only through interprocess communication
- Group assignment based on the permissions declared in the manifest file

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```
Zygote

- Eukaryotic cell formed by a fertilization event between two gametes
- In Android: System service that is the parent of all Android application processes
  Comparable to init
- User starts new application
- Process class calls the Zygote process
- Zygote creates a copy of itself using fork()
- Zygote returns a new process ID
- The Process class starts the new process through its run() method
- The run() method calls startViaZygote()
- Newly spawned process triggers the loading of the Dalvik virtual machine
- Now there are two Dalvik virtual machines
Trace an application
Root the device

- Modify the kernel and all bets are off
- You do not want do demand the users to root their device
  - Technical knowledge required
  - Easily rip open security holes
  - Sometimes voids the warranty of the device
Trace an application
Share the user ID

- Run in the same user context
- Only possible if the applications
  - are signed with the same developer certificate
  - explicitly specify a common value for the shared UID in their manifest file
Trace an application
Use system call interposition

• The magic of peeking into another program
• Control the execution flow of the traced application

```c
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

• Abbreviation of "process trace"
• One process can control another
• Controller can inspect and manipulate the internal state of its target
  – Used by debuggers and other code-analysis tools
• Here: Usage as a sandbox, run-time environment simulator

• Powerful ability → Attaching is limited to processes that the owner can send signals to (typically only their
  own processes)
  – CAP_SYS_PTRACE capability limitation
  – YAMA Linux Security Module
  – FreeBSD: jails and Mandatory Access Control policies.

• Higher level usage of ptrace: Userspace utility strace
• Projects move on extending strace instead of ptrace
Approximately 60 ptrace tags

- PTRACE_ATTACH
- PTRACE_CONT
- PTRACE_DETACH
- PTRACE_EVENT_CLONE
- PTRACE_EVENT_EXEC
- PTRACE_EVENT_EXIT
- PTRACE_EVENT_FORK
- PTRACE_EVENT_SECCOMP
- PTRACE_EVENT_STOP
- PTRACE_EVENT_VFORK
- PTRACE_EVENT_VFORK_DONE
- PTRACE_GETEVENTMSG
- PTRACE_GETFPREGS
- PTRACE_GETFPXREGS
- PTRACE_GETREGS
- PTRACE_GETREGSET
- PTRACE_GETSIGINFO
- PTRACE_GETSIGMA
- PTRACE_GET_SIGINFO
- PTRACE_INTERRUPT
- PTRACE_KILL
- PTRACE_LISTEN
- PTRACE_O_EXITKILL
- PTRACE_O_MASK
- PTRACE_O_SUSPEND_SECCOMP
- PTRACE_O_TRACECLONE
- PTRACE_O_TRACEEXEC
- PTRACE_O_TRACEEXIT
- PTRACE_O_TRACEFORK
- PTRACE_O_TRACESECCOMP
- PTRACE_PEEKDATA
- PTRACE_PEEKINFO
- PTRACE_PEEKINFO_SHARED
- PTRACE_PEEKTEXT
- PTRACE_PEEKUSER
- PTRACE_PEEKUSR
- PTRACE_POKEDA
- PTRACE_POKETEXT
- PTRACE_POKEUSER
- PTRACE_POKEUSR
- PTRACE_SECCOMP_GET_FILTER
- PTRACE_SEIZE
- PTRACE_SEIZE_DEVEL
- PTRACE_SETFPREGS
- PTRACE_SETFPXREGS
- PTRACE_SETOPTIONS
- PTRACE_SETREGS
- PTRACE_SETREGSET
- PTRACE_SETSIGINFO
- PTRACE_SETSIGMA
- PTRACE_SET_SIGINFO
- PTRACE_SINGLESTEP
- PTRACE_SYSCALL
- PTRACE_SYSEMU
- PTRACE_SYSEMU_SINGLESTEP
- PTRACE_TRACE
- PTRACE_TRACEEX
Some interesting ptrace arguments

- PTRACE_TRACEME
  - Program is conveying its readiness to get traced
- WIFSTOPPED
  - Status variable contains a bit pattern which indicates the fact that the child process has stopped
- PTRACE_CONT
  - Restart the child by invoking ptrace with the request
- PTRACE_GETREGS
  - Request results in the values of the CPU registers used by the stopped child
- PTRACE_SETREGS
  - Change the value of the registers
- PTRACE_PEEKDATA
  - Of the process being traced examine the content localised at the given address
- PTRACE_POKEDATA
  - Alter the contents of a memory location
- PTRACE_SINGLESTEP
  - Restart the stopped process, let it execute a single instruction and then stop it again
- PTRACE_ATTACH
  - Attach to the process specified by its process ID, making it a tracee of the calling process
- PTRACE_SYSCALL
  - Restart the child process (just like PTRACE.CONT) but arrange for it to stop at the next entry to or exit from a system call
Particular approach

- Load and execute the code of the original application in the context of the monitoring application
- Start the application we want to sandbox
- Generate a stub application, that loads the code and monitors its execution through system call interposition
- Patch the parameters of some system calls for e.g. file operations

Limitations:
- Any ptrace-based sandbox limited to register filtering
- No filtering for arguments pointing to memory, e.g. file name provided to the `open()` system call
- Seccomp (secure computing mode) unable to examine memory
Follow-up techniques

- Superseded by access control security policies
- Enforce separation guarantees between applications

- First permissive release of Android 4.3 Jelly Bean (2012)
- Selective root daemon confinement, enforcement on a limited set of crucial domains (installd, netd, vold and zygote) in Android 4.4 KitKat (2013)
- Full confinement, Android Trusted Computing Base protection as well as full enforcement mode in Android 5.0 Lollipop (2014) and higher
Image sources

- Openclipart, licensed under the Creative Commons Zero 1.0 Public Domain License
  URL: http://creativecommons.org/publicdomain/zero/1.0/
  Original file: https://openclipart.org/detail/268848/android-thief

- Wikimedia Commons, licensed as public domain.

- Wikimedia Commons, licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.
  URL: https://creativecommons.org/licenses/by-sa/3.0/deed.en
  Original file: https://commons.wikimedia.org/wiki/File:Android_dance.svg

- Wikimedia Commons, licensed under the Creative Commons Attribution 3.0 Unported license.
  URL: https://creativecommons.org/licenses/by/3.0/deed.en
  Original file: https://commons.wikimedia.org/wiki/File:Android_robot.svg

- Wikimedia Commons, licensed as public domain.
  Original file: https://commons.wikimedia.org/wiki/File:DIN_4844-2_D-P006.svg

- Wikimedia Commons, licensed under the Expat / MIT License.
  URL: https://opensource.org/licenses/mit-license.php