BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016

#58

DYNAMIC SECURITY POLICY ENFORCEMENT ON ANDROID PLATFORM

Matúš Vančo – xvanco02@stud.fit.vutbr.cz

This poster presents the system for dynamic enforcement of access rights. Each suspicious application can be repackaged by this system, so that the access to selected private data is restricted for the outer world. The system is designed and developed, utilizing the possibilities of one of the examined frameworks – Aurasium framework. The system adds an innovative approach of tracking the information flows from the privacy-sensitive sources using tainting mechanism without need of administrator rights. There has been designed file-level and data-level taint propagation and policy enforcement based on Android binder.

System Calls Monitoring

Communication between components is maintained using *Reference Monitor*, which is part of Android OS Middleware. It ensures communication between applications separated on the OS level, permission label mediation and mandatory access control. ACL in Android is statically defined and user-authorized during installation and does not provide ability to further modificate the subset of manifested requested permissions as it is possible in the Windows Phone OS. Reference Monitor utilizes *Inter-component Communication* similar to IPC in Unix-based systems. Android utilizes this communication in several different forms accoring to involved components which is shown in figure 2. However, all forms are using the same IPC principle based on system call *ioctl()*. In this project, there are also monitored system calls *open()*, *close()*, *read()* and *write()* which enables communication with file system.

Data Tainting

The tainting is based on the principle used in *TaintDroid* architecture. In TaintDroid, there are monitored instructions on the level of virtual machines. TaintDroid uses *Virtual Taint Map* (VTM), which mirrors the address space, but does not contain the content of memory. It represents the division of memory into protected and public part (fig. 1). Before tainting process, the tainted files are marked in VTM. Then, every copying of memory invokes copying of blocks in VTM. In this project, there has been designed and integrated two granularities of taint propagation – file-level tainting and data-level tainting. The message-level tainting (between components) principle from TaintDroid is used only for final policy enforcement (restriction), because Aurasium intercepts only single applications and does not have possibility to monitor the unhardened ones.

Fig. 1: Tainting Principle

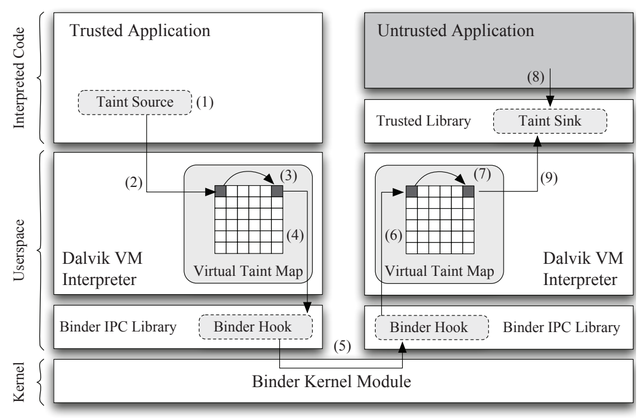


Fig. 2: Forms of Component Interaction

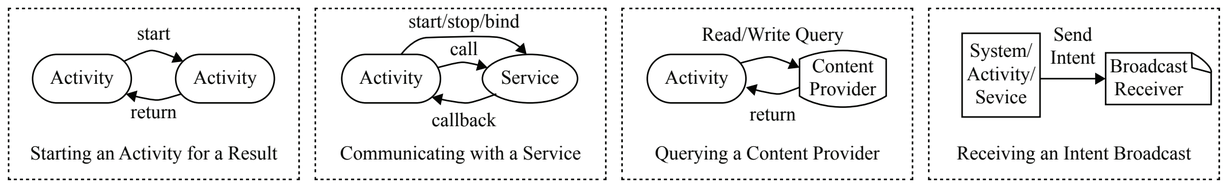


Fig. 3: Analysis of Interception

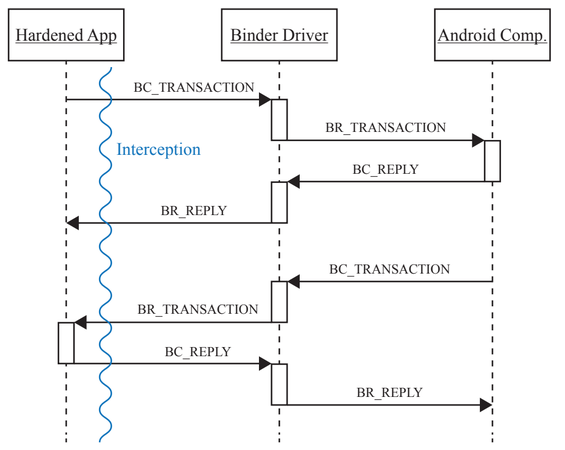
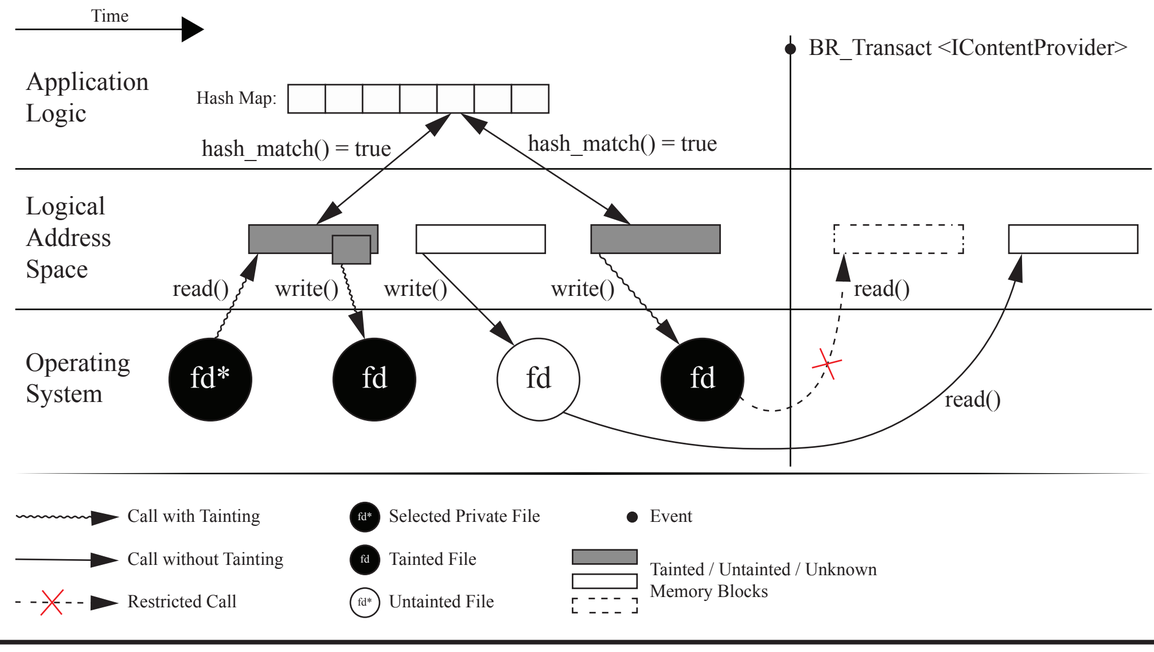
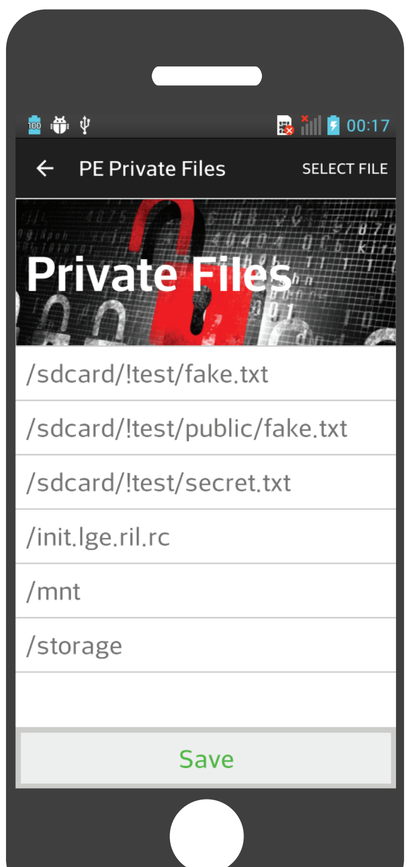


Fig. 4: Design of Architecture





System for Private Data Protection

File-level tainting between memory and the OS's file system can be performed in full scope, because Aurasium can fully intercept this communication using system calls *open()*, *close()*, *write()* and *read()*. Fuction *open()* is used for obtaining the opening mode. This is used for taint adaptation. If the untainted memory is written to tainted file in append mode, the files remains tainted, but if it is written in read mode, the file becomes untainted. The *system* calls are used for tainting the memory blocks as well as new files. The data in memory read from tainted file are marked similarily and the files, which are read from tainted memory blocks becomes tainted too. However, data in memory are also directly propagated.

Since the *Aurasium* can intercept only specific places (system calls) and not instruction itself, it is impossible to implement full-scope memory-level tainting as is introduced by TaintDroid. This is replaced by the newly designed data-level tainting concept (fig. 4). Data-level tainting uses the storage of file contents or hashes in order to track the private files. If the selected file (upper screenshot) is copied to another location and shared like in lower screenshot, the system prevents the leakage. The project is designed to secure the user-selected files or folders as a entity, which are intended to be invariable like images, pictures or videos. Usability is a big advantage, since the application does not require administrator rights.