UDC 004.94

VIRTUALIZED WEB BASED FARM FOR TESTING AND DEMONSTRATION OF ANDROID APPLICATIONS WITH ADVANCED SECURITY AND PORTABILITY

Kovalenko V.Yu., Kostiuk D.A.

A virtualized testing farm for Android applications developed by the authors is presented, with substitution of default emulation tools of Google with different set of components, using nested virtualization and transparent emulation on the native architecture. Presented solution provides user control via the intranet web access, as far as typical monitoring and development access through ADB. Internal architecture, deployment and scaling principles are discussed.

Introduction

Client-server applications with web interface have gained today the widest usage, and are already replacing classic desktop applications in some areas. List of reasons backing such move include the presence of a web browser on all platforms and architectures, as well as sufficient performance of JavaScript code (client-side web applications are built on) in modern browsers. Universal accessibility (including platforms with a touchscreen interface and other so-called "weak clients," ie. on mobile and portable devices, targeted mainly at the use of cloud services) makes browser to be the most convenient entry point for the end user application / service. This is especially significant when the main computational load is placed on the server and / or other nodes in the network, and the client is used only for management and access to resources (the reverse case, unfortunately, at the moment is not effective as far as JavaScript implementation involves much larger overhead costs compared to traditional languages on server platforms).

One of the tasks that can get significant benefit in case of the client-server implementation based on a private cloud, is mobile applications testing, debugging, or exploration of potential harmful functionality.

Problems of testing code in case of strong fragmentation of target hardware platforms cannot be called new; however, up to date, Android mobile platform is one of the most fragmented. Also it can be admitted as the platform most appreciated by creators of malicious software (including spyware but not limited to it). Despite the fact that Android is based on highly secure GNU/Linux OS, its tendency to rely on the end user’s choice at granting applications any additional privileges they ask for makes a lot of space for abusive activity. Therefore both developer and tester need to supply their systems with the set of emulator images and a variety of operating system versions to run them in turn. In this case, even if the regular tools of Android debugging and activity monitoring would provide a convenient interface to simultaneously run multiple instances of the emulator to test the program, low productivity of the emulation associated with different processor architecture, would cause negative impact on the productivity and efficiency of development. And in case of collaborative development it turns out sometimes that local emulation methods do not fit the workflow at all. Periodically shared access to a copy of the test application is required, as well as a demo access to the intermediate assemblies.

1. The concept of testing farm for Android applications

The obvious solution is to deploy the test farm on the server and access it over the network (in our case – with use of the Web access through an intranet resource). Through the Android open and interchangeable software development tools, the construction of an alternative infrastructure enables benefits such as computation resources saving, centralized automatic control of the emulator, ease of programs demonstration, as well as for a number of tasks, the possibility to replace the regular emulator with a more productive virtual machine, which performs emulation of a native hardware platform that is identical to the host system. In this paper we present a variant of such solution designed for the needs of a particular company, but versatile enough to be the subject of a wider practical interest.

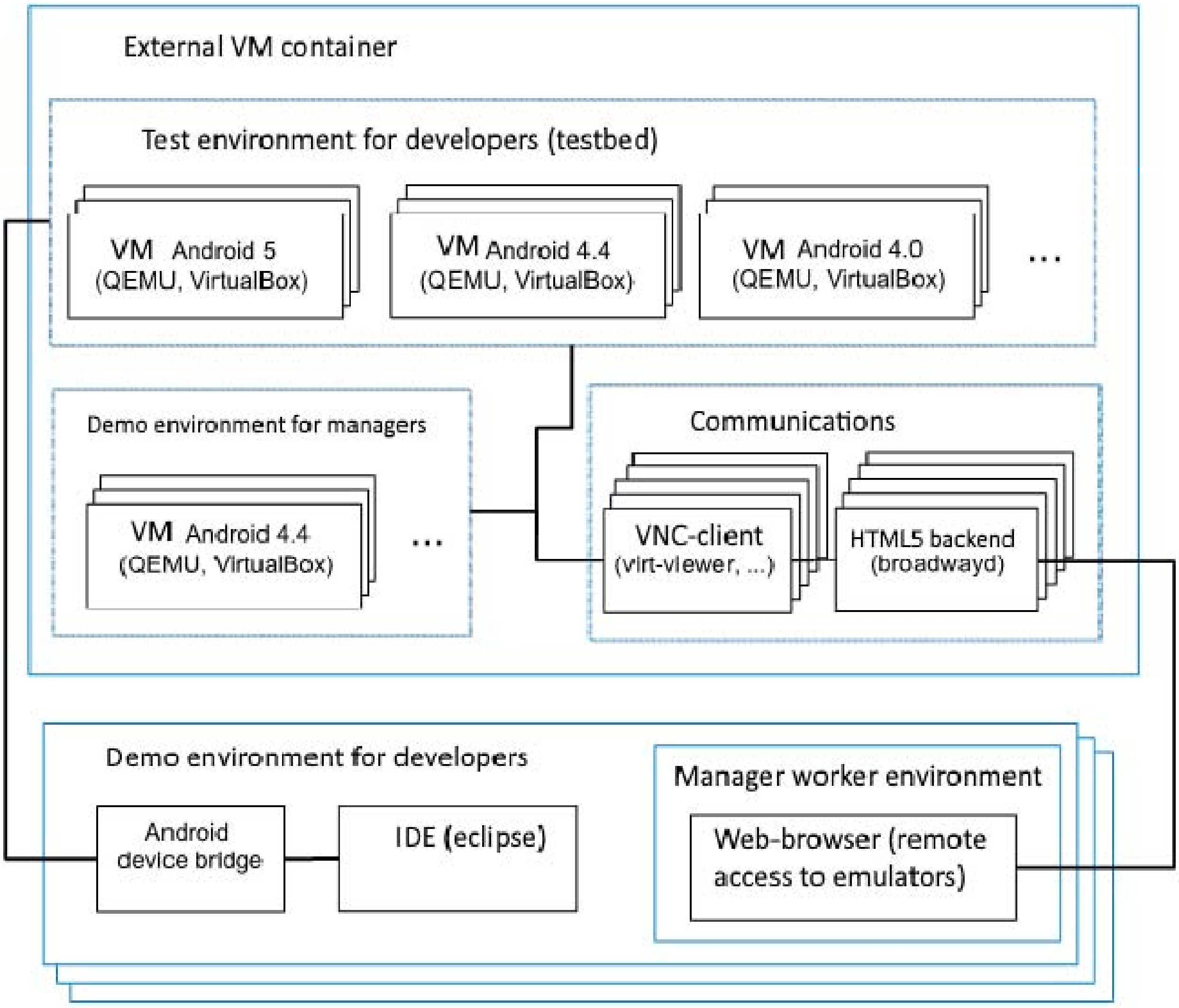
External VM container
Test environment for developers (testbed)
VM Android 5 (QEMU, VirtualBox)
VM Android 4.4 (QEMU, VirtualBox)
VM Android 4.0 (QEMU, VirtualBox)
...
Demo environment for managers
VM Android 4.4 (QEMU, VirtualBox)
...
Communications
VNC-client (virt-viewer, ...)
HTML5 backend (broadwayd)
Demo environment for developers
Android device bridge
IDE (eclipse)
Manager worker environment
Web-browser (remote access to emulators)

Fig. 1. The structure of the test farm for Android apps

2. Farm architecture

The general structure of developed farm is shown in Fig. 1. As in the case of the traditional Android development tools stack, the application is tested on various versions of Android OS running on the emulator. However, for the convenience of the farm deployment and migration, emulators set in this case is located inside of another outer virtual machine (VM), i.e. so-called nested (embedded) virtualization is used.

External VM can be either the VirtualBox virtualization solution, or the VMWare commercial system (choice should be done according to the existing infrastructure of the company to integrate the farm into, as far as combining two virtualization systems in the network may not be feasible due to organizational reasons). The number of sub-VMs (i.e. emulators containing different versions of the Android OS) varies during the operation [2]. QEMU or VirtualBox can be used as an embedded virtual machine in the proposed solution (the choice depends on an external VM, with which hardware-accelerated embedded virtualization should flawlessly work).

As already mentioned, standard Android SDK emulator has problems with productivity, is less suitable for self-deployment and demonstration purposes. For this reason, some manufacturers use Android Build for x86-compatible systems [1], launched in a native virtualization (without different processor architecture simulation). This approach significantly reduces the processing load of the emulation, and retains all the benefits and convenience provided by server virtualization of desktop systems as far as possibilities the automation, cloning and integration into enterprise infrastructures.

3. Images of embedded VM

In addition to web-based access to the emulator, universal for developers, managers, and security staff, the developer interacts with an embedded virtual machine through standard ADB interface (to install applications and perform debugging sessions), as well as a simple web interface that allows cloning one of the enclosed VMs. Security specialist has the same set of capabilities as a developer, but uses additional monitoring tools inside of the external VM container.

The farm contains reference images of a nested VMs with installed Android OS of various versions (Fig. 2). These images are configured and tuned for the fastest possible launch of a new instance. One of the reference images should be cloned to start testing some Android software, and demo software is also loaded on these cloned VM images. ADB is used at deploying and debugging programs executed by the cloned machine, while management and user interaction is carried out through the web interface.

OS Android images in the reference samples are mostly based on the x86 architecture. Tests have shown that in this case the nested virtualization overhead (with a compatible combination of external and nested VM) is as small as a few percent.

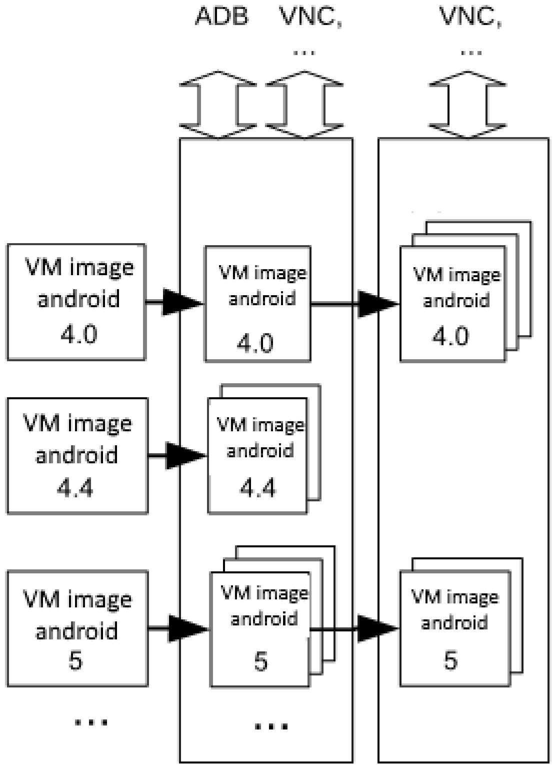

Fig. 2. VM images cloning

Technically image management is provided by a set of rather simple scripts that perform on-demand cloning of master images, as well as start and stop of the VM execution.

4. Web access to the VM

Depending on what it is used in the farm as a sub-VM, user access of developers, security researches, and managers to the emulator can be carried out via either the VNC protocol, the SPICE or RDP (Figure 2.).

People are accessing the desired VM through the web page of the intranet resource. Initially, , we have planned to use remote clients in JavaScript to connect to the emulator via a web environment, which was successfully used by us in other projects. However, in the process of testing the choice was made in favor of the access via the broadway GTK backend – a rendering subsystem in HTML 5 embedded in the latest version of the GTK library. The structure of this web-based access solution, built into the farm is shown in Fig. 3.

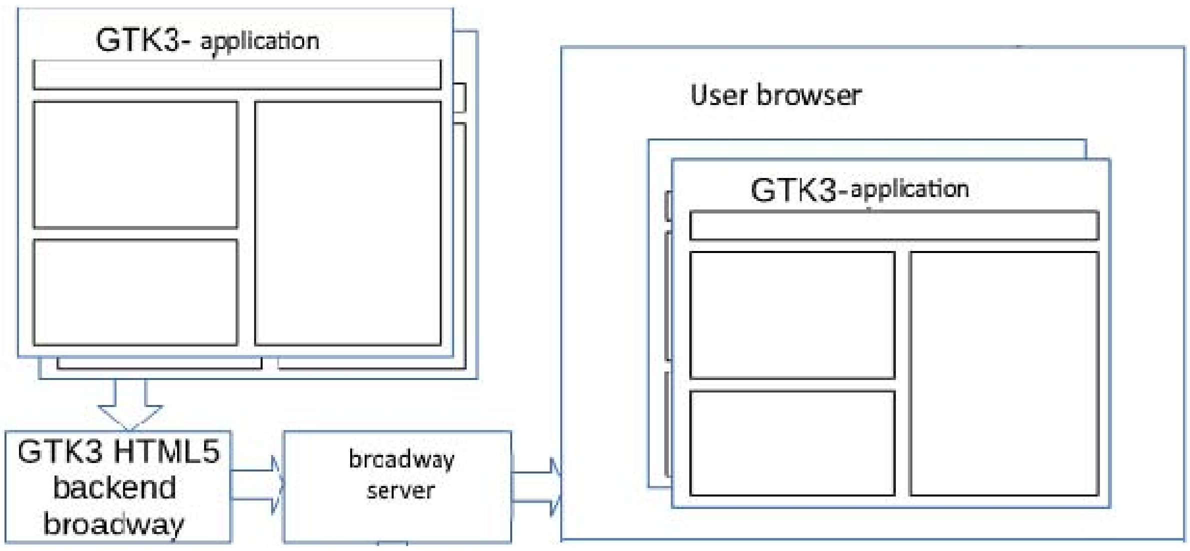

Fig. 3. The broadway operation princilpe

On the client-side access to the emulator is done via the web browser, which communicates with the broadway GTK backend, and broadway draws the window of the chosen GTK3 applications on the HTML 5 canvas object. The graphics program that uses GTK library runs in headless mode, i.e. only for network access. System daemons broadwayd, each of which occupies a single network port should be run for this, and the graphics program should be supplied with an additional set of variables at startup, which are read by the GTK library and thus determine the rendering mode. As a result, the startup code of the broadway system daemon and looks as follows:

```
broadwayd :1 &
GDK_BACKEND=broadway UBUNTU_MENUPROXY= LIBOVERLAY_SCROLLBAR=0 
BROADWAY_DISPLAY=:1 VNCViewer
```

One of VNC or RDP clients written using GTK 3 plays the role of the graphics program running inside of the external VM and connected to the embedded VM (in this example - VNCViewer client is used); broadway technology, in its turn, translates the user session access to the browser outside of the external VM (Fig. 4).

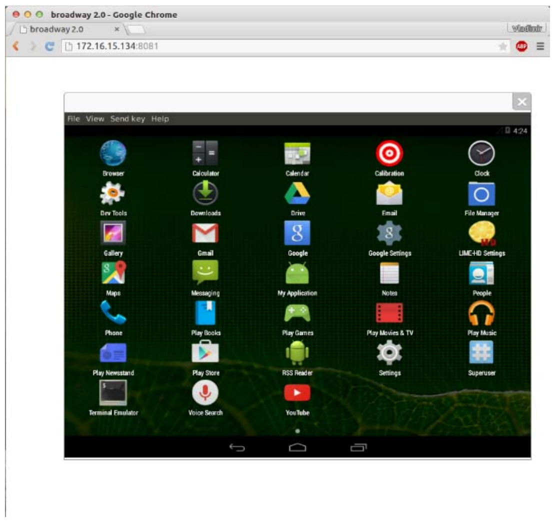

Fig. 4. Android emulator window in web browser

The choice of this web access architecture is influenced by two considerations: the best performance of the code (which will be discussed below), and no need for additional components (broadcastsing a TCP connection to the web sockets) on the user's computer [3].

As for additional components on the client side, which would be necessary when using the VNC-client in JavaScript, embedded in HTML page, their necessity is dictated by the fact that the code in JavaScript, executed by the web browser does not have access to the TCP protocol used for remote access, and therefore requires the application proxy running on the client side and transmitting TCP traffic to a web sockets.

5. Performance testing

Obviously, the reason for better performance of the selected web access solution in comparison with JavaScript VNC client is caused by a highly specialized web server that is a part of broadway, and the VNC-client running inside the external VM (both components are written in C). However, this code is executed on the server, while the use of VNC-client in JavaScript increases the processing load on the client machine. In turn, broadway connection also connects to the client code in JavaScript, executed by a Web browser for the reception, transmission and rendering.

To assess the real broadway impact on the performance of the developed farm, performance testing was separately carried out for the web access (configured on the separate server with Ubuntu 15.04 GNU/Linux OS, with the AMD FX-8320 processor and the RAM capacity of 16 GB). The OS choice is subject to the presence of the components required for broadway operation (GTK library version 3.8 and above, as well as some software using it). While the test the client and server parts of the system were running simultaneously on the server, providing equal impact of the relatively high resource consumption of the selected OS GUI for both sides, so it did not affect the comparison.

The noVNC JavaScript client was selected as an alternative solution of web access for the comparison. Load was assessed by the «load average» parameter, produced with the help of htop tool, which is the most common and generally accepted indicator to assess the server workload.

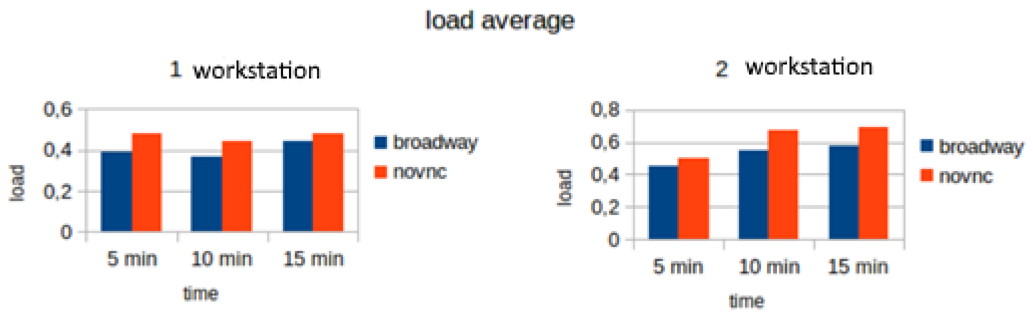

Fig. 5. Performance Comparison of broadway and noVNC

The comparison results are shown in Fig. 5. The analysis shows broadway advantage in terms of load, and this difference increases at growing number of workplaces to essential numbers when the number of virtual desktops reaches tens or hundreds.

Lesser server load of broadway, compared to VNC protocol, can be attributed to several factors: the lack of need for a separate VNC server, absence of operations associated with compression and stream encoding for VNC, more optimal and specialized server code. RAM consumption in the case of broadway allows to save 30-40 MB per virtual workplace.

As for the load on the client machine and the amount of generated traffic, both approaches are demonstrating parity (and both quantities are negligible). However, there broadway advantage is a complete lack of image artifacts that can occur when using VNC.

References

1. Каваленка У.Ю., Касцюк Д.А. Дэманстрацыйна-тэставая ферма праграм для платформы Android з вэб-інтэрфейсам // П‘ята міжнародна науково-практычна конференція FOSS Lviv 2015: Збірник наукових праць / Львів, 23– 26 квітня 2015 р. – С. 42–45.
2. Коваленко В.Ю., Костюк Д.А. Кроссплатформенные виртуальные рабочие места // Информационные технологии и системы 2015 (ИТС 2015): материалы международной научной конференции. Минск, БГУИР, 29.10.2015. – С. 104–105.
3. Kostiuk D., Lutsiuk P., Vlasenko S., Zheludok V. Virtualization-based illustrated reviews of the software history // Открытые технологии: сборник материалов Десятой Международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2014, Гродно, 22 – 24 августа 2014 г. – С. 98 – 101.