Security Model of Firefox OS

Anthony Vérez*, and Guillaume Hugues*
*Télécom SudParis, Institut Mines-Télécom, 91000 Evry, France

June 10, 2013

1. General overview of Firefos OS

Firefox OS (also known as FF OS or B2G for Boot To Gecko) is Mozilla's operating system for smartphones. It aims at creating an open source platform for mobiles, using open web technologies. It is developed around three major software layers, from the lowest to the highest: Gonk, Gecko and Gaia.

Let's go over these three layers in detail.

1.1. Gonk

Gonk is the lower-level interface of Firefox OS. It is, strictly speaking, the operating system. It consists of a firmware, a modified Linux kernel, device drivers and a userspace hardware abstraction layer (HAL). Common userspace libraries (like libusb, BlueZ, ...) that can be found in desktop distributions are directly imported from open-source projects. However, most of libraries and protocols dealing with device specific hardware (GPS, camera, ...) come from the Android project.

1.1.1. Booting. The boot process on Gonk is very similar to the one of any Linux based operating system. After the bootstrap process which is very device dependent, the Linux kernel is launched and execution is handled to it. At the end of this sequence, the init process is launched. At this point only a ramdisk (a virtual hard drive whose memory actually resides in RAM) is mounted on the system and only contains critical utilities, startup scripts and kernel modules. The init process then mounts necessary file systems and launch system services such as:

- netd (which configures network interfaces)
- mediaserver
- rild
- b2g

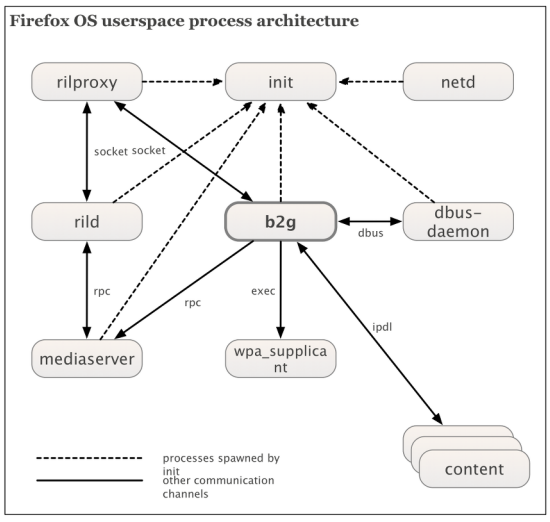

Figure 1. Gonk userspace architecture [1]

1.1.2. mediaserver. This daemon is responsible for playing audio and video files. Depending on their codecs, the media files will be decoded using Linux libraries or proprietary codecs and hardware decoders before being sent to mediaserver.

1.1.3. rild. rild (which stands for Radio Interface Layer Daemon) is the interface to the telephony hardware. It allows clients with sufficient privileges to connect to it through a UNIX-domain socket. In Firefox OS, the b2g process communicates with rild through the rilproxy process because b2g doesn't have the necessary privileges to communicate directly with rild (which requires the client's group to be radio).

1.1.4. b2g. This is the single most important process of Firefox OS. It runs with high privileges and has access to most system resources. This is the process that runs Gecko, the upper layer of Gonk, and that is responsible to spawn low-right process every time an app is launched.

Figure 1 provides an overview of Gonk, its components and how they interact.

1.1.5. supervisor. This is not an implemented feature yet, but is being discussed and might appear in future Firefox OS releases. The idea is to create a process called supervisor which would be responsible for the very few actions that actually need to be executed as root like system update, shutdown or reboot, use of nice to adjust processes priorities, etc. In this implementation, supervisor would run as root:root and b2g would then run as system:system, which would still be very high privileges, but not root.

1.2. Gecko

Gecko is Mozilla's layout engine mostly used in the web browser Firefox. It is run only once on the entire system inside the b2g process, using the libxul.so library and receives rendering requests through IPDL (IPC Protocol Definition Language), a protocol initially developed by Mozilla for Firefox using inter-process communication (IPC) and allowing to pass messages between processes or threads in an organized and secure way [2] [22]. Because Gecko is executed inside a highly privileged process, it's low level implementation differs from the one found on desktop distributions. In Firefox OS Gecko has access to most system resources and hardware, which is something impossible on common desktop OSes

```
gNativeWindow = new
  android::FramebufferNativeWindow();
sGLContext =
  GLContextProvider::CreateWindow();
```

Figure 2. use of FramebufferNativeWindow class

because the process never has enough privileges. The result is an increase in performance since Gecko can communicate directly with hardware devices. For example, since Gecko is the only process using the display, it can safely do it's rendering directly on the hardware video memory. What actually happens is that Gecko uses the C++ FramebufferNativeWindow class (imported from Android) to map an OpenGL context directly on the hardware frame buffers (see figure 2).

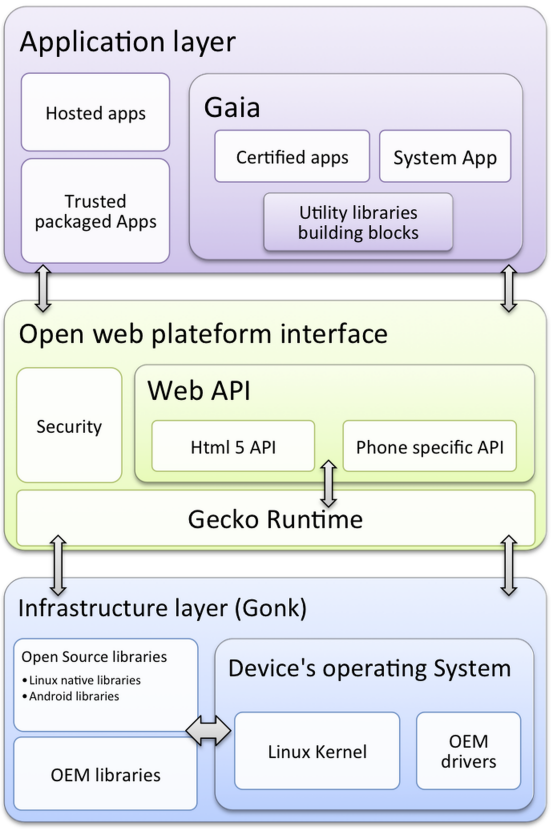

Figure 3. Firefox OS software layers

1.3. Gaia

Gaia is the user interface of Firefox OS. It is written entirely in HTML, CSS and Javascript, and implements all the standard applications of a smartphone: lock screen, home screen with app launcher, telephone dialer, mail client, web browser, marketplace, and many others. It interfaces with the phone underlying operating system using only Open Web APIs. This means that applications - all of them, even native ones - communicate only with Gecko using javascript, and never have direct access to system resources. Because Gaia only use standard web APIs, it can work on environments other than Firefox OS, with degraded functionality though because of the missing mobile phone specific hardware. This is interesting for developers because they can easily test their application using a Firefox plugin that emulates Firefox OS.

```
<window>Gecko Chrome
 |
 +-> <iframe> system app (contains
              the lockscreen)
      |
      +--> <iframe> homescreen app
      |
      +--> <iframe> keyboard
      |
      +--> ... more app iframes are
           created here as apps are
           loaded
```

Figure 4. Window hierarchy of Gaia [3]

1.4. Applications

Just as every other mobile phone operating system, Firefox OS is based on apps. All applications are developed using HTML5 and use the same web APIs. However, Firefox OS distinguishes two kinds of apps: hosted apps and installed - or packaged - apps. Hosted apps are similar to web applications used by desktop web browsers. They are closer to a browser bookmark than an actual application. The only thing stored on the mobile device is the URL to the application plus, optionally, cookies and HTML5 local storage. These applications do not work without Internet connection. Packages apps contain all source code and resources and are installed on the mobile device. They are designed to work offline as standalone autonomous applications. These different kinds of application raise different kinds of security considerations and possible issues which will be discussed further.

1.4.1. App manifest. In Firefox OS, every application is associated with an app manifest. It is a single JSON file containing all the information needed to interact with the application. It is one of the key elements that distinguish a Web App from a website. Among other things the app manifest contains the name, author, icon, locales and description of your application. Most importantly, it will also contain a list of Web APIs (such as geolocation) that your App needs with a quick mandatory description of the intent behind the use of that particular API. This allows users to make informed decisions about apps before installing them. The description field will be used and displayed to the user if the use of the API (such as geolocation) requires user authorization. If a manifest requests permission for forbidden API, launch of the application will fail. Use of an API without a corresponding entry in the manifest will fail as well.

```
{
  "name": "My SSR App",
  "description": "Does nothing.",
  "launch_path": "/",
  "icons": {
    "16": "/img/icon-16.png",
    "48": "/img/icon-48.png",
    "128": "/img/icon-128.png"
  },
  "developer": {
    "name": "Anthony Verez &
             Guillaume Hugues",
    "url": "http://homepage.com"
  },
  "default_locale": "en",
  "installs_allowed_from": [
    "https://marketplace.firefox.com",
    "https://marketplace.example.com"
  ],
```

```
  "locales": {
    "fr": {
      "description": "Ne fait rien.",
      "developer": {
        "url": "http://homepage.fr"
      }
    }
  },
  "orientation": ["portrait"],
  "permissions": {
    "contacts": {
      "description": "Required for id
                      stealing ;)",
      "access": "readwritecreate"
      },
    "geolocation": {
      "description": "Just to know
                      where you are"
    }
  }
}
```

Figure 5. Example of an app manifest

1.5. History

At the time of writing this paper, the latest nighty build of Firefox OS is at version 1.1.0hd

- Jul 2011: Announcement of B2G [4]
- Feb 2012: First partners (Telefónica, Adobe, Qualcomm and Deutsche Telekoms Innovation Labs) at Mobile World Congress 2012 [5]
- Jul 2012: B2G is now known as 'Firefox OS'. TCL Communication Technology (Alcatel) and ZTE will be manufacturing devices running on Firefox OS. major network operators including Deutsche Telekom, Etisalat, Smart, Sprint, Telecom Italia, Telefónica and Telenor sign a partnership with Mozilla [6]
- Nov 2012: Firefox OS Simulator available [7]
- Dec 22, 2012 : Release version 1.0 of Firefox OS [8]
- Jan 15, 2012 : Release version 1.0.1 of Firefox OS [8]
- Feb 2013: Bunch of new partners including major operators and manufacturers [9]. Release of Firefox OS 1.0 [10].
- Mar 29, 2012 : Release version 1.1.1 of Firefox OS [8]
- Apr 2013: Draft to integrate payment system [11]
- May 2013: Release of Firefox OS Simulator 3.0 [12]
- Jun 2013: Foxconn devices will use Firefox OS [13]

1.6. Hardware requirements

Firefox OS will only work on mobile devices that support Android 4.0 because the operating system embeds some - not to say a lot - of Android 4.0 source code. For example it has been successfully tested on a Samsung Galaxy SII [14].

The first devices officially supporting Firefox OS will be Alcatel, ZTE, LG and Huawei devices [15]. The devices currently used by developers are Alcatel One Touch Fire and the ZTE Open (see figure 6 and figure 7).

Alcatel One Touch Fire [16]:

- 3.5-inch display
- 1GHz processor
- 256MB RAM
- 512MB storage
- 3.2-megapixel camera
- WiFi 802.11b/g/n
- Bluetooth 3.0
- GPS
- 1400mAh battery
- microSD slot with 2GB card is included
- Mini-SIM card slot

Figure 6. Alcatel One Touch Fire used by developers

Mozilla also recently (June 2013) announced partnership with 17 operators throughout the globe [9]: Amrica Mvil, China Unicom, Deutsche Telekom, Etisalat, Hutchison Three Group, KDDI, KT, MegaFon, Qtel, SingTel, Smart, Sprint, Telecom Italia Group, Telefnica, Telenor, TMN and VimpelCom.

2. Security Guidelines

Developing an OS always raises many security considerations and issues, and Firefox OS is no exception to this rule. The security risks mostly come

ZTE Open [17]:

- 3.5-inch display
- 800MHz Qualcomm MSM7225A processor
- 256MB RAM
- 512MB storage
- 3.2-megapixel camera
- WiFi 802.11a/b/g/n
- Bluetooth 2.1
- GPS

Figure 7. ZTE Open used by developers

from processes that run with high privileges because if compromised, they could allow an attacker to take control of the entire device.

In order to avoid security holes, Mozilla tries to ensure that the application follows the 10 OWASP Secure Coding Principles [18] [19]:

- Minimize attack surface area
- Establish secure defaults
- Principle of Least privilege
- Principle of Defense in depth
- Fail securely
- Dont trust services
- Separation of duties
- Avoid security by obscurity
- Keep security simple
- Fix security issues correctly

2.1. Security Reviews

Security Reviews are used to identify security-related issues, determine the level of risk associated with those issues, and make informed decisions about risk mitigation or acceptance [20].

Here are some example of security reviews for Firefox OS (coming from [21]) :

Inter-app communication

Explanation: all inputs to the applications need to be validated and sanitized. Other inputs that are sometimes overlooked include the following.

Checklist:

- Validate the origin/content of message events
- Careful with URLs (including location.hash). Although linking from one app:// to another is prevented, some forced browsing scenarios are still possible - for example if you applications periodically sets the location via user input)
- Careful handling any mozbrowser* events from child frames (see Browser API section below)

Client Side Storage

Explanation: The risks of using client-side storage in privileged web apps is no different to in any web content. However given that many privileged applications are designed to function without internet access, it more likely client-side storage will be used for sensitive user data.

Checklist:

- Keep user data to a minimum and avoid storage of private user information where possible.
- Provide users a way to clear sensitive data.
- Consider encryption for particularly sensitive data, prior to storage - being wary that any keys stored on the device can be recovered.

2.2. Outreach

Firefox OS has still a lot of features that require security reviews. As a result Mozilla is actively looking for security testers. They are organizing promotion programs to attract developers and security experts. Even small challenges are organized where developers and testers win points and possibly developer phones (which are phones running under Firefox OS freely given to developers with a great idea and that have the skills to develop an application).

3. Security Implementation

Because of the way Firefox OS is designed, security issues as well as the implementation of security features have bean a great concern for Firefox OS developers.

3.1. User Side

3.1.1. Permissions. Firefox OS has a big list of permissions, figure 12 is just sample of them. For each application requesting a explicit permission, which has to be justified by the app developer, the user can grant of refuse this permission to the app, see figure 8 and figure 9.

However, some permissions are implicit which means that user is to asked if they want to give a permission or not to an app. In the settings, the user can list apps and the permissions granted to them.

For Android for example, a malicious app tends to ask for many if not all permissions which are not always relevant to the use of a legit app.

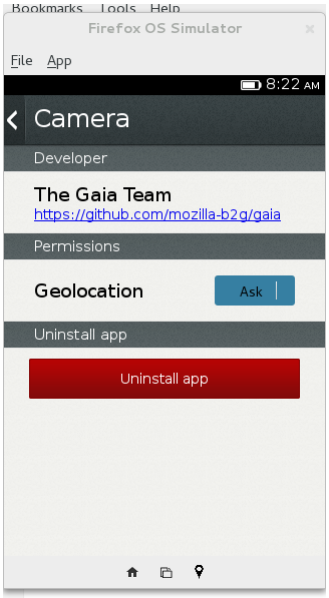

Figure 8. Permissions of the Camera app

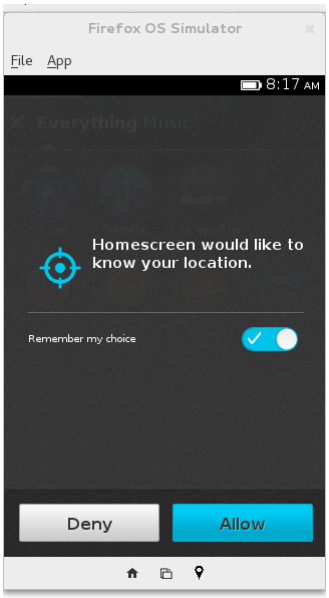

Figure 9. An app requesting the geolocalisation permission

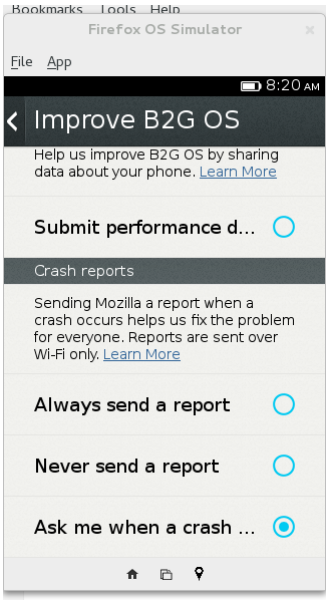

Figure 10. Report bugs setting

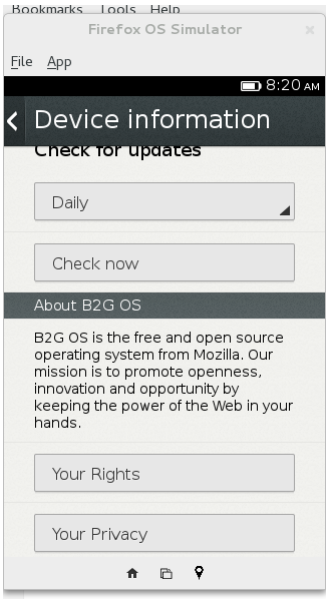

Figure 11. Mozila's privacy policy

3.1.2. Privacy. Mozilla wants to make a difference concerning Privacy. The user can always to send information or not to Mozilla and its partners and decide what is shared. As an example, figure 10 show the report setting and figure 11 shows a link to Mozilla's privacy policy.

Permission name	Hosted App	Installed App	Privilegied App	Certified App
desktop-notification	Explicit (PROMPT ACTION)	Implicit (ALLOW ACTION)	Implicit (ALLOW ACTION)	Implicit (ALLOW ACTION)
tcp-socket	None (DENY ACTION)	None (DENY ACTION)	Implicit (ALLOW ACTION)	Implicit (ALLOW ACTION)
device-storage:music	None (DENY ACTION)	None (DENY ACTION)	Explicit (PROMPT ACTION)	Implicit (ALLOW ACTION)
telephony	None (DENY ACTION)	None (DENY ACTION)	None (DENY ACTION)	Implicit (ALLOW ACTION)
geolocation	Explicit (PROMPT ACTION)	Explicit (PROMPT ACTION)	Explicit (PROMPT ACTION)	Explicit (PROMPT ACTION)

Figure 12. Apps permissions example

3.2. Apps Security

3.2.1. Permissions. In order to avoid malicious applications accessing critical device interfaces of user private information, applications are divided in four kinds for which different permissions apply:

- Hosted apps
- Packaged apps
- Privileged (packaged) apps
- Certified (packaged) apps

Privileged apps are authenticated (signed) applications approved by an app store after a code review or some equivalent risk management process. Certified apps are similar to privileged apps but require additional approval from carrier or OEM. These are the core applications, and their permissions cannot be changed.

Permissions are hierarchical, meaning that certified apps have all the permissions of privileged apps which have all the permissions of Hosted apps, etc. Permissions are used to access some devices functionalities through web APIs (such as geolocation, contacts, alarm, etc). As we have seen before, they need to be explicitly requested in the manifest file. But the application type must also have the permission to use the API. For some API (such as geolocatoin), the application must also explicitly request user approval before using it. For example in figure 12 we see that the API device-storage:music is forbidden for hosted apps and simple packaged apps, requires explicit user approval for privileged apps and is implicitly allowed for certified apps.

3.2.2. Web APIs. WebAPI is a term used to designate Javascript API that allows web applications to access device hardware (like telephony stack, geolocation or vibration hardware). Security is a central aspect of web API development because some APIs are private or reserved for some kinds off apps (privileged or certified).

3.2.3. b2g process. Firefox OS has a core process named b2g. A good comparison would be to think of this process as of the root account on Unix systems: The process has a lot of rights and permissions and is used to access to most hardware devices.

3.2.4. Content process. Each application runs a dedicated process called content process, spawned by b2g process, see figure 13. To mitigate the malicious actions of an app or not have the whole system compromised by an app, the content process has only low privileges. It cannot access to system resources. To communicate with any other process, it needs to go via the b2g process using IPC (Inter-Process Communication). IPC is a popular family of methods used by threads in one or more processes to communicate with one another. This communication is implemented with Mozilla's IPDL (IPC Protocol Definition Language) protocol [22]. This protocol is already used in Firefox by C++ threads to communicate in a secure way. For e.g. tabs and multithreaded plugins in Firefox work on this protocol. Each Web API features at least one IPDL declaration file with the extension .ipdl.

A content process is launched by the b2g process when it processes the *<iframe mozapp>* tag. All content processes belong to a container, a group of processes, similar to the plugin-container used by Firefox. This container is called an *out of process* container because of its separation from the rest of the system. Just before the content process start, all the file descriptors that the content process is not allowed to access are closed.

It is planned to add this architecture to tabs in the Browser app for next releases [23] so that each tab would run in a dedicated content process and maybe even launch the Browser app in a content process, it means that a content process would have to be able to spawn its own content processes.

3.2.5. System updates. Security in software updates are crucial. On the one hand, the system has to be updated with security fixes for known vulnerabilities to be secure and on the other hand the update process must be very secure so that an attacker cannot push his malware through software updates.

Updates are usually created and signed by the device manufacturer (OEM). They can cover only a part of the system or all the software. In both cases, if the changes affect Gonk, then the update is done by a FOTA (Firmware Over The Air) process. It means that the entire firmware is downloaded over a wireless connection from the smartphone. Such an update can also embed changes for other parts of the system (e.g.,

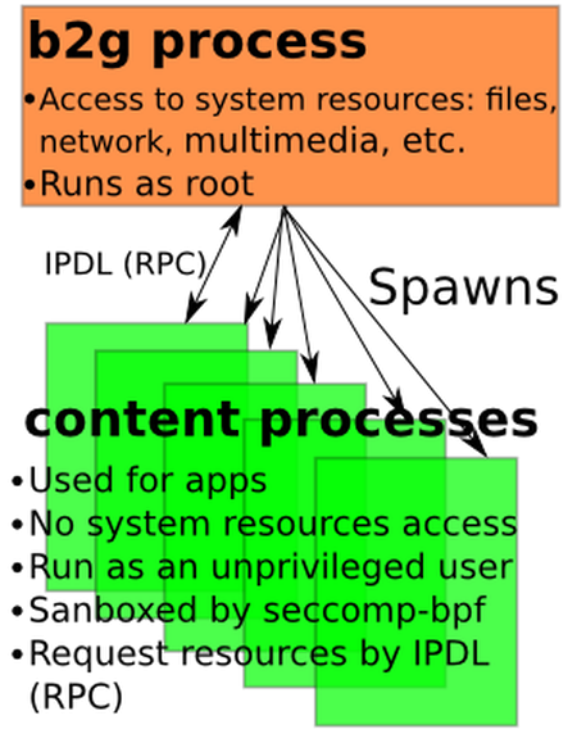

Figure 13. b2g and content processes

Gaia, Settings, etc.). A strong cryptographic verification is made before installing to prevent attackers to impersonate the OEM or Mozilla to install their own firmware update in order to have a full and permanent control over the device. If the update doesn't include changes for Gonk, it can be installed with the Mozilla System Update Utility. In this case, the process is the same as the one used by Firefox, including the MAR (Mozilla ARchive) format.

An update service runs on the device and periodically connect to the Internet to check for updates. If an update for the system is available, a notification for the user shows up asking to confirm the installation. Before starting to download the package, the system checks that the space requirements are met and that no web apps are currently running. The update is fetched using SSL with a trusted certificate, saved in a secure system folder, whenever possible and stored in a secure location to prevent attackers from replacing the update by their software if they already have some control over the device. Between the download and the installation of the update, the system makes some verifications:

- Update origin, the location from where the update was downloaded must match the protocol, domain and port of the system update settings
- file integrity, strong checksums algorithms are used to verify that the downloaded file is the one we were expecting
- code signature, a certificate authenticate the issuer of the update and it must be checked against a trusted root certificate

The manufacturer (OEM) can also provide its own update service.

```
.
|-- icon-128.png
|-- index.html
|-- manifest.webapp
|-- META-INF
    |-- A.RSA
    |-- A.SF
    |-- ids.json
    |-- MANIFEST.MF

1 directory, 7 files
```

Figure 14. Hierarchy of an sample signed app

3.2.6. App signing. To add integrity protection to the apps, Firefox OS will require downloadable apps to be packaged in a zip file signed by the store from which it has been downloaded. Mozilla will run its marketplace for apps but also encourages developers to run their own app stores which raises concerns from security experts because on other platforms most malicious apps come from third party app repositories. Mozilla reuses security libraries of Firefox for Firefox OS' cryptography.

The signature files are packed into the zip, in the META-INF folder, see figure 14. Inside this folder, the MANIFEST.MF file displayed in figure 15 stores the SHA-1 hash of each file. The standard used for signing the app is PKCS #7.

As of the time of writing this article, there are patches to add signing mechanisms for updates [24] but they are not yet merged in the main codebase.

3.2.7. App validation. An app validation process will be used to ensure that the apps in Mozilla's marketplace are legit. Testers review the manifest and try submitted applications for a few minutes [25]. Their task is to give feedback and constructive bug report to the developer if the app doesn't work and decide to reject it, at least temporarily, until the app is fixed and the Marketplace conditions are respected. As for security restrictions, the app manifest needs to have

```
Manifest-Version: 1.0

Name: icon-128.png
SHA1-Digest: oYaPCEWEfFz3nB80N9+
PyHh8bRY=

Name: index.html
SHA1-Digest: mMPI7v18vD5EIg1wga8
lGvt6JE4=

Name: manifest.webapp
SHA1-Digest: UHd/+C+eY2FPgYLfGlF
L4cN+taU=

Name: META-INF/ids.json
SHA1-Digest: IcEKUdr8XMXWn1DnJhV
N8pv7luU=
```

Figure 15. MANIFEST.MF

the same origin (scheme, hostname and port number) as the app. The Content-Type header of the manifest have to be application/x-web-app-manifest+json. The reviewer will check that an app doesn't abuse redirections or iframes, which is a common practice for web malware. The developer needs to describe the usage of each requested privilege. Finally, the app will have to include some sort of privacy policy. In addition to the manual review, automatic tests will be run submitted apps, especially for the javascript code.

An app has to be considered as major attack vector on a mobile platform. It can be created or modified (e.g. bundled with a malware) by an attacker. To lower the risks having of an app compromised, Mozilla reviews the apps sent to its marketplace (but developers can send their app to other marketplaces). Firefox OS also features an app signing mechanism to verify the integrity of an app before installing it. In case an app tries to compromise the system, it shouldn't be able to access to resources of other apps and should be confined to the permissions the user has given to the app. Apps security has been thought over and designed from the very start of the project. Although the technologies used to create apps in Firefox OS bring new security challenges to the mobile platforms, they are common on the web and Mozilla seems to use its security experience in this area to design their apps security features. That being said, a lot of security features are yet in development and are very likely not to be available when the first devices using Firefox OS will be on the market.

3.3. Security Architecture

3.3.1. Sandboxing. Sandboxing is defined as the action of "confining a helper application to a restricted environment, within which it has free reign." [26]

Sandboxing usage in Firefox OS aims at mitigating risks and protecting users' data, the platform and mobile phone.

inter-process communications. First, this concept is used for an app run-time execution. A workspace is dedicated for each app; it can use only the Web APIs and data it is supposed to. It also has resources specifically associated such as cookies, databases, offline storage, etc. Resources are not accessible directly but through Web APIs. These restrictions are implemented with a file descriptors whitelist.
Gecko needs to have high privileges to be able to access to hardware features of the mobile device. An app runs in a child process of the b2g (Gecko) process and isolated in the sandbox described above. The new process has its own memory space and cannot elevate its privileges to have permissions of the b2g process. The latter is responsible for checking if the child process has the necessary permission to perform the attempted actions. The app process can only communicate with the b2g process and not directly with other processes (including apps). With this architecture, the app process requests resources using the IPDL protocol to the b2g process that will check the security policy of the app including type of app and permissions specified in the app manifest before performing the action on the behalf of the content process and pass the result back.

Calls from the content process using the IPDL protocol must be sanitized by the b2g process because the input given by a content process cannot be trusted.

Seccomp. Seccomp is a sandboxing mechanism for the Linux kernel. It was first released in Linux kernel 2.6.12 in March 8, 2005 [27]. It can change the state of a process so that this process cannot make any system calls (function to use a feature of the kernel from a process in the user-space), except exit(), sigreturn(), read() and write() to already-open file descriptors.

An evolution of seccomp, Seccomp-bpf (or seccomp mode filter) was released by Google with Linux kernel 3.5 [28]. Google Chrome 20 and above sandboxes Adobe Flash Player with seccomp-bpf [30] and as of Chrome 23, renderers processes are also sandboxed [29].

BPF (Berkeley Packet Filter) filters are used with Seccomp-bpf, these filters also exist in tcpdump for

```
#include <stdio.h>
#include <stdlib.h>
char tab[3] = {'A', 'B', 'C'};
char val = 'Z';

int main() {
  printf("Before overflow=%c\n", val);
  tab[3] = 'D';
  printf("After overflow=%c\n", val);
  return EXIT_SUCCESS;
}
```

Figure 16. buffer_overflow.c

```
$ gcc buffer.c -o buffer
$ ./buffer
Before overflow=Z
After overflow=D
```

Figure 17. Buffer Overflow execution

example. It provides seccomp for a process the ability to flexibly define allowed syscalls and the action to take if the security policy is violated.

At the time of writing this paper, Guillaume Destuynder has developped patch implementing seccomp-bpf in gonk [31]. He had to backport Seccomp-bpf to Linux Kernel 3.0 [32] because Firefox OS currently runs on an Android's kernel based on Linux kernel 3.0.

Of course, this sandboxing has an impact on performance, it is believed to increase the response time up to 1% when a system call is made in the process managed by seccomp-bpf.

3.3.2. Address Space Layout Randomization (ASLR). A buffer overflow consists in overwrite data in memory next to a buffer by exceeding the limit of this buffer. Figure 16 and figure 3.3.2 provide an example of a simple buffer overflow where the *val* variable is stored in memory just after the *tab* table. *val* value is overwritten by a buffer overflow of *tab*.

ASLR is a exploitation mitigation mechanism designed to randomize memory space layouts used by a program so that they are different for between two execution of a program. Without that feature, an attacker could predict addresses in the memory (e.g. stack or heap) to exploit a buffer overflow overwriting for example the return address of a function stored in the stack by another address of his choosing, so maybe let him run some malicious code.

As shown in the example in figure 18 on a Linux Kernel with an amd64 architecture, the heap, the dynamic linker and the stack addresses are randomized because they vary from an execution to another.

Patches are available and working on the current version of Gonk and Gecko but it's basically a backport of Android 4.1's ASLR feature [33] using the PIE flag. Mozilla seems to have decided to wait the migration porting Gonk from Android 4.0 to Android 4.2 rather than merging the current patches into the main codebase.

3.3.3. Build flags hardening. It's a good security practice to review the security flags available for a compiler, usually to make it more difficult to exploit memory corruption vulnerabilities. Bionic is the standard C library used for Firefox OS (it's also used in Android). Mozilla's engineers have created patches to use hardened gcc and bionic flags [34].

RELRO (RELocation Read-Only) is another flag used for exploit mitigation. A Linux binary (ELF) using shared libraries has a look-up table called Global Offset Table (GOT) which is used to resolve functions available in these libraries. When such a function is called, the call is first pointing to the Procedure Linkage Table (PLT), stored in the .plt section of the executable, see (1) in figure 20. This section has instructions pointing to the GOT (here 0x400450), stored in the .got.plt section (2). Without going into further details (which are available at [36] and [37]), without RELRO, the GOT and the PLT use mechanisms making them weak against memory corruption: the PLT is a known offset from the .text segment, the memory is allocated at a known address and is writable. So we just need to create an exploit to write some data at the desired location, see figure 19. We can see that this exploit works without RELRO but is blocked if the flag is used figure 20. There are also *partial RELRO* flags where only non-PLT GOT is read-only but this is much less secure than the flags we have just seen with a *full RELRO* mode. Our demonstration provides the same result without RELRO and with partial RELRO. To provide some comparison, on a current Arch Linux x86_64 system, systemd and chromium run in full RELRO modes, most of the programs run in partial RELRO mode and ruby and dropbox don't use any RELRO flag.

-D_FORTIFY_SOURCE is another flag considered for gcc. This feature detects a subset of buffer overflows and tries to automatically specify the buffer size when the program is compiled.

Finally, the gcc Stack-Smashing Protector (SSP)

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 fe:02 138177            /usr/bin/cat
0060a000-0060b000 r--p 0000a000 fe:02 138177            /usr/bin/cat
0060b000-0060c000 rw-p 0000b000 fe:02 138177            /usr/bin/cat
012b1000-012d2000 rw-p 00000000 00:00 0                 [heap]
7f144b0fa000-7f144b29d000 r-xp 00000000 fe:02 134369    /usr/lib/libc-2.17.so
7f144b29d000-7f144b49d000 ---p 001a3000 fe:02 134369    /usr/lib/libc-2.17.so
7f144b49d000-7f144b4a1000 r--p 001a3000 fe:02 134369    /usr/lib/libc-2.17.so
7f144b4a1000-7f144b4a3000 rw-p 001a7000 fe:02 134369    /usr/lib/libc-2.17.so
7f144b4a3000-7f144b4a7000 rw-p 00000000 00:00 0
7f144b4a7000-7f144b4c8000 r-xp 00000000 fe:02 134376    /usr/lib/ld-2.17.so
7f144b697000-7f144b69a000 rw-p 00000000 00:00 0
7f144b6c8000-7f144b6c9000 r--p 00021000 fe:02 134376    /usr/lib/ld-2.17.so
7f144b6c9000-7f144b6ca000 rw-p 00022000 fe:02 134376    /usr/lib/ld-2.17.so
7f144b6ca000-7f144b6cb000 rw-p 00000000 00:00 0
7fff9c2e1000-7fff9c302000 rw-p 00000000 00:00 0         [stack]
[...]
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 fe:02 138177            /usr/bin/cat
0060a000-0060b000 r--p 0000a000 fe:02 138177            /usr/bin/cat
0060b000-0060c000 rw-p 0000b000 fe:02 138177            /usr/bin/cat
0141d000-0143e000 rw-p 00000000 00:00 0                 [heap]
7fb4ed9fe000-7fb4edba1000 r-xp 00000000 fe:02 134369    /usr/lib/libc-2.17.so
7fb4edba1000-7fb4edda1000 ---p 001a3000 fe:02 134369    /usr/lib/libc-2.17.so
7fb4edda1000-7fb4edda5000 r--p 001a3000 fe:02 134369    /usr/lib/libc-2.17.so
7fb4edda5000-7fb4edda7000 rw-p 001a7000 fe:02 134369    /usr/lib/libc-2.17.so
7fb4edda7000-7fb4eddab000 rw-p 00000000 00:00 0
7fb4eddab000-7fb4eddcc000 r-xp 00000000 fe:02 134376    /usr/lib/ld-2.17.so
7fb4edf9b000-7fb4edf9e000 rw-p 00000000 00:00 0
7fb4edfcc000-7fb4edfcd000 r--p 00021000 fe:02 134376    /usr/lib/ld-2.17.so
7fb4edfcd000-7fb4edfce000 rw-p 00022000 fe:02 134376    /usr/lib/ld-2.17.so
7fb4edfce000-7fb4edfcf000 rw-p 00000000 00:00 0
7fff0a408000-7fff0a429000 rw-p 00000000 00:00 0         [stack]
[...]
```

Figure 18. ASLR on Linux with an amd64 architecture

```
#include <stdio.h>

int main(int argc, char** argv) {
  size_t *p = (size_t *)
    strtol(argv[1], NULL, 16);
  p[0] = (size_t)
    strtol(argv[2], NULL, 16);
  printf("RELRO: %p\n", p);
  return 0;
}
```

Figure 19. relro.c

also known as ProPolice can be activated. It provides, among other security features, a mechanism called canaries. They are indicators just between local variables and the return address of a function where a buffer may be overflowed. If an attacker overflows a buffer, the canary will be erased. Before calling the return address, a check on the canary is done and if it was altered, the program will end immediately before calling the return address that may redirect to malicious code.

These patches need more performance tests to make sure it doesn't have a performance impact that is too important for the users. To provide some context, Ubuntu, a popular Linux distribution, provides ASLR. This distribution also uses the immediate binding, -

```
$ # No RELNO
$ gcc relro.c -o relro_code
(1) $ objdump -M intel -d
relro | grep printf@plt
4005b4: [...] call 400410 <printf@plt>
(2) $ objdump -M intel -d
relro | grep 400410: -A 2
400410: [...] jmp [...] # 600988
 <_GLOBAL_OFFSET_TABLE_+0x18>
400416: [...] push 0x0
40041b: [...] jmp 400400 <_init+0x20>
$ gdb -q ./relro
(gdb) x 0x600988
0x600988 <printf@got.plt>:
0x00400416
(gdb) r 0x600988 0x424242
Program received signal SIGSEGV,
  Segmentation fault.
0x0000000000424242 in ?? ()
$ # With (full) RELRO
$ gcc -Wl,-z,relro,-z,now relro.c -o relro
(1) $ objdump -M intel -d
relro | grep printf@plt
4005f4: [...] call 400450 <printf@plt>
(2) $ objdump -M intel -d
relro | grep 400450: -A 2
400450: [...] jmp [...] # 600fd8
 <_GLOBAL_OFFSET_TABLE_+0x18>
400456: [...] push 0x0
40045b: [...] jmp 400440 <_init+0x28>
$ gdb -q ./relro
(gdb) x 0x600fd8
0x600fd8 <printf@got.plt>:
0x00400456
(gdb) r 0x600fd8 0x424242
Program received signal SIGSEGV,
  Segmentation fault.
0x00000000004005e0 in main ()
```

Figure 20. RELRO demonstration

D_FORTIFY_SOURCE=2 and (partial) RELRO flags for all their packages.

3.3.4. File system. Firefox OS features a file system hardening policy designed to prevent information leaks, privilege escalation and execution of native code. The main principle of the policy is to give read-write rights only to areas with user content. This file system hardening is based on Android.

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, const char* argv[])
{
  execv("/usr/bin/id", argv)
  return 0;
}
```

Figure 21. setuid_demo.c

```
$ su foo
$ gcc setuid_demo.c -o setuid_demo
$ chmod +s setuid_demo
$ ./setuid_demo
uid=45067(foo) gid=449(mygroup)
groups=449(mygroup)
$ su bar
$ ./setuid_demo
uid=45068(bar) gid=449(mygroup)
euid=45067(foo) groups=449(mygroup)
```

Figure 22. commands for a setuid demo

Unix permissions. Firefox OS uses the classic Linux filesystem permission model which specifies which user,group or others has access to a file to read,write or execute it. The system ensures that the user under which web application are executed shouldn't be able to write to any file in our system. For new files, especially those created by content processes, we want to make the sure the default permissions follow this policy. Firefox OS make use of umask for this.

The flags setuid (set user ID upon execution) and setgid (set group ID upon execution) are also an attack vector. When an executable file has this flag it is executed with the permissions of the owner of the file. For example, if a executable file is created with the root user (so that the owner of this file is root) and has the setuid flag it has unlimited permissions, see figure 21 and figure 22. the setgid flag give the same behavior but instead of working on users, it applies to groups.

Mount points. A mount point is a physical location in a partition that is used by a filesystem. Each mount point has:

- a path describing where it is accessible in the filesystem
- a file system type describing how data is stored,

retrieved or updated
- options usually giving features to specify data access mechanism for performance but also security features

The mount points chosen for Firefox OS are described in table 1 [38]. We can see that writable mount points have the nosuid option which does not allow set-user-identifier or set-group-identifier bits to take effect in these mount points. These mount points also have the noexec option so that binaries cannot be executed. So for example even if an attacker manages to copy an malicious binary in the filesystem, they won't be able to execute it. More precautions were taken to mount SD cards, uid=1000 specifies that only the root user has access the files on the card. For this mount point, default permissions for files and directory are set using respectively the fmask and dmask options, only the owner of a file or directory can write in it.

Since Firefox OS is an operating system, it needs security hardening features to make it more difficult for an attacker to exploit a security bug. Firefox OS uses technologies that have been implemented for a long time by other OS like ALSR or file system hardening. An effort has been made on sandboxing although no groundbreaking features have been created, it often consists in backporting security features of Android or porting Firefox's features. Some interesting security patches are available but not merged, maybe because the project is currently more interested by performance than advanced exploit mitigation techniques.

3.4. Security of Competitors' Products

3.4.1. Android. Android is the most used mobile operating system in Q1 2013. Popularity brings many attackers. The main security layers of Android [39] are very similar to those of Firefox OS: using the Linux kernel for OS security, each application runs in a sandbox, secure communication between processes, application signing and permissions. The application sandbox creates a user and a process for each app and benefits from the user resources (such as processes and files) protection of the Linux kernel. Whereas only apps are sandboxed in Firefox OS, so are libraries in Android, using the same mechanisms. The file system hardening of Android is the same as b2g's. As for memory hardening Android seems a little more advanced than Firefox OS, since Android 1.5 ProPolice is used, NX (to prevent code execution on the stack and heap) is available since Android 2.3, ASLR has been implemented in Android 4.0 and improved in Android 4.1. RELRO appeared in Android 4.1, FORTIFY_SOURCE has been activated since Android 4.2.

Full disk encryption using AES128 is available for Android 3.0+, this feature is discussed for next b2g versions [40].

Security vendors (Avast, Lookout, AVG, ...) provide additional security such as antivirus, privacy settings, etc.

A negative aspect of Android's security is that on most devices, you cannot upgrade Android's version due to hardware limitation, so you don't have any security improvement.

On the kernel level and for external dependencies Firefox OS mostly use Android's codebase. As a consequence, the security level of these layers are the same on the two platforms.

3.4.2. iOS. iOS has a lot of security features [41] such as signed firmware files for the boot chain, code signing, Data Execute Prevention (DEP), ASLR, Stack Canaries, File system hardening and Sandboxing.

Despite all these security features, shortly after a new iOS version is out, a jailbreak, a program to gain full control of its device, is released. Malware is limited on this platform compared to Android, most of them are related to memory corruption. It is thought to be thanks to Apple Store's restrictions. Reviewers take more time to do their job, verify publishers' real-world identities and don't hesitate to refuse apps or ban attackers who submit malicious apps. iOS also have a reduced attack surface because it doesn't include a lot of external software: No flash, no Java, limited PDF viewer, no shell.

3.4.3. Blackberry. RIM has made security one if its main marketing arguments. Blackberry smartphones are known to make heavy use of cryptography using the AES algorithm. Blackberry runs in a QNX kernel [42] which give a process manager to each process making sure it doesn't try to write or access a memory address it's not supposed to. Blackberry 10 include external dependencies such as Flash or WebKit. A particularity of Blackberry is that RIM sells Blackberry Enterprise Server, a software solution for companies used to centrally configure and control a fleet of blackberry smartphones. This solution is a great tool to ensure security compliance of all the smartphones in the company and push configurations that will greatly reduce the attack surface. In May 2013, after a rigorous security review, Blackberry 10 has been the first mobile platform approved by the U.S. Department of Defense for future agency use [43].

Compared to the 3 most popular mobile operating systems, Firefox OS appears close to Android (but behind), from which it takes many security ideas although this system is definitely the most secure. Currently Android, iOS and Blackberry have more advanced security features but all these systems share similar security guidelines. Security has now become a major argument for selling smartphones to companies. iOS and Blackberry currently have better reputation on this topic.

4. Conclusion

It appears that Firefox OS has suffered some delay but most of the basic and most crucial security features are ready. From a general point of view, Firefox OS currently provides far less features for users than other mobile operation systems. The tools to develop great applications are however already here. An interesting point is that because Firefox OS has been created when some people have seen the consequences of poor security on smartphones, Mozilla has adopted from the start a security aware architecture. Maybe it'll make developing interesting security features easier in the future. What may be the most important in the security model of Firefox OS is that it's designed for a new kind of applications, which are developed using web technologies (HTML5, javascript) and may magnify the impact of web attacks since web apps while be running not only a browser but also directly in an OS. We were not able to find an independent security research paper about Firefox OS yet but it'll probably be a hot topic in next security conferences.

Acknowledgment

The authors would like to thank Joaquin Garcia-Alfaro for tutoring the redaction of this paper.

References

[1] B2G/Architecture on Mozilla's Wiki https://wiki.mozilla.org/B2G/Architecture as of 06-02-2013

[2] Firefox OS architecture on Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Architecture as of 06-02-2013

[3] Gaia overview on Google Drive https://docs.google.com/document/d/1Q6VZAN4GI_3zUe1oILrsAlTR6ODUnF2xjpHq5LrNAdk as of 06-02-2013

[4] Announcing Boot to Gecko (B2G) Booting to the Web on July 27, 2011 by Robert Nyman. https://hacks.mozilla.org/2011/07/announcing-boot-to-gecko-b2g-booting-to-the-web/ as of 05-05-2013

[5] Telefónica and Mozilla pioneer first Open Web Devices on 27 February 2012. http://saladeprensa.telefonica.com/documentos/nprensa/OpenWebDevice_Eng_DEF.pdf as of 05-05-2013

Mount point	File system	Options
/	rootfs	read-only
/dev	tmpfs	read-write, nosuid, noexec, mode=0755
/dev/pts	ptsfs	read-write, nosuid, noexec, mode=0600
/proc	proc	read-write, nosuid, nodev, noexec
/sys	sysfs	read-write, nosuid, nodev, noexec
/cache	yaffs2 or ext4	read-write, nosuid, nodev, noexec
/efs	yaffs2 or ext4	read-write, nosuid, nodev, noexec
/system	ext4	read-only, nodev
/data	ext4	read-write, nosuid, nodev, noexec
/mnt/sdcard	ext4 or vfat	read-write, nosuid, nodev, noexec, uid=1000, fmask=0702, dmask=0702
/acct	cgroup	read-write, nosuid, nodev, noexec
/dev/cpuctl	cgroup	read-write, nosuid, nodev, noexec

Table 1. Filesystem Mounts

[6] Mozilla Gains Global Support For a Firefox Mobile OS, 07-02-2012 https://blog.mozilla.org/blog/2012/07/02/firefox-mobile-os/ as of 05-07-2013

[7] Announcing the prototype Firefox OS Simulator on November 15, 2012 by Kevin Dangoor https://hacks.mozilla.org/2012/11/announcing-the-prototype-firefox-os-simulator/ as of 05-08-2013

[8] Release Management/B2G Landing on Mozilla's Wiki https://wiki.mozilla.org/Release_Management/B2G_Landing as of 06-09-2013

[9] Mozilla Announces Global Expansion for Firefox OS, 02-24-2013 https://blog.mozilla.org/press/2013/02/firefox-os-expansion/ as of 05-08-2013

[10] B2G FTP mirror http://ftp.mozilla.org/pub/mozilla.org/b2g/manifests/ as of 05-14-2013

[11] Introducing navigator.mozPay() For Web Payments on April 4, 2013 by Kumar McMillan https://hacks.mozilla.org/2013/04/introducing-navigator-mozpay-for-web-payments/ as of 05-18-2013

[12] Firefox OS Simulator 3.0 released on May 2, 2013 by Robert Nyman [Editor] and Myk Melez https://hacks.mozilla.org/2013/05/firefox-os-simulator-3-0-released/ as of 05-18-2013

[13] Foxconn Adopts Firefox OS, 06-03-2013. https://blog.mozilla.org/press/2013/06/foxconn-adopts-firefox-os/ as of 05-20-2013

[14] Mozilla B2G on Galaxy SII review on YouTube, 29 February 2013 http://www.youtube.com/watch?v=TaujwbpbLk0 as of 06-09-2013

[15] Mozilla reveals Firefox smartphone launch partners on BBC News, 24 February 2013 http://www.bbc.co.uk/news/technology-21522713 as of 06-09-2013

[16] Alcatel One Touch Fire joins the Firefox OS cuddle party, we go hands-on (video) By Richard Lai, Feb 24th, 2013 http://www.engadget.com/2013/02/24/alcatel-one-touch-fire/ as of 06-09-2013

[17] ZTE Open, the company's first Firefox OS phone, gets a spec sheet at MWC By Richard Lai, Feb 23rd, 2013 http://www.engadget.com/2013/02/23/zte-open-firefox-os-mwc-leak/ as of 06-09-2013

[18] Firefox Security Guidelines on Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Security/Firefox_Security_Guidelines as of 06-09-2013

[19] Secure Coding Principles on OWASP https://www.owasp.org/index.php/Secure_Coding_Principles as of 06-09-2013

[20] Security Review, University of California, Santa Cruz http://its.ucsc.edu/itsm/securityrev.html as of 06-09-2013

[21] Developing secure Firefox OS applications on Google docs https://docs.google.com/document/d/1DLs1jhTMxN5fh2PSb_O7FDaSadjjAW-MlK1xCBRWGmM as of 06-09-2013

[22] IPC Protocol Definition Language (IPDL) on Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/IPDL as of 06-03-2013

[23] Bug 761935 - (nested-processes) Tracking: Support nested content processes https://bugzilla.mozilla.org/show_bug.cgi?id=761935 as of 06-07-2013

[24] Bug 797477 - Enable loading certificates and MAR verification in updater code for B2G https://bugzilla.mozilla.org/show_bug.cgi?id=797477 as of 06-06-2013

[25] Marketplace review criteria on Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Web/Apps/Publishing/Marketplace_review_criteria as of 06-07-2013

[26] A Secure Environment for Untrusted Helper Applications (Confining the Wily Hacker), Ian Goldberg et al., Proceedings of the Sixth USENIX UNIX Security Symposium, San Jose, CA, July 1996. http://static.usenix.org/publications/library/proceedings/sec96/full_papers/goldberg/goldberg.pdf as of 05-25-2013

[27] [PATCH] seccomp: secure computing support by Andrea Arcangeli on 2005-03-08 01:54:43 http://git.kernel.org/cgit/linux/kernel/git/tglx/history.git/commit/?id=d949d0ec9c601f2b148bed3cdb5f87c052968554 as of 05-26-2013

[28] Using simple seccomp filters http://outflux.net/teach-seccomp/ as of 06-03-2013

[29] A safer playground for your Linux and Chrome OS renderers on Monday, November 19, 2012 http://blog.chromium.org/2012/11/a-safer-playground-for-your-linux-and.html as of 06-01-2013

[30] Chrome 20 on Linux and Flash sandboxing on Wednesday, July 4, 2012 http://scarybeastsecurity.blogspot.de/2012/07/chrome-20-on-linux-and-flash-sandboxing.html as of 06-02-2013

[31] Bug 790923 - (b2g-seccomp) Content process sandboxing via seccomp filter, patch provided by Guillaume Destuynder https://bugzilla.mozilla.org/show_bug.cgi?id=790923 as of 05-26-2013

[32] Seccomp-bpf for Samsung Crespo's Android kernel on Github (source code) https://github.com/gdestuynder/android_kernel_samsung_crespo as of 06-02-2013

[33] Bug 777948 - (ASLR-b2g) Consider implementing address space layout randomization (ASLR) for B2G https://bugzilla.mozilla.org/show_bug.cgi?id=777948 as of 05-27-2013

[34] Bug 620058 - use hardened gcc build flags https://bugzilla.mozilla.org/show_bug.cgi?id=620058 as of 06-02-2013

[35] Security Features on Ubuntu Wiki https://wiki.ubuntu.com/Security/Features as of 06-06-2013

[36] RELRO: RELocation Read-Only from NYU Poly ISIS Lab on June 1, 2011 by Julian Cohen https://isisblogs.poly.edu/2011/06/01/relro-relocation-read-only/ as of 06-01-2013

[37] RELRO - A (not so well known) Memory Corruption Mitigation Technique by Tobias Klein on February 21, 2009 http://tk-blog.blogspot.fr/2009/02/relro-not-so-well-known-memory.html as of 06-01-2013

[38] System security of Firefox OS on Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Security/System_security as of 06-05-2013

[39] Android Security Overview https://source.android.com/tech/security/ as of 06-08-2013

[40] Bug 777917 - Consider implementing Full Disk Encryption (FDE) for B2G https://bugzilla.mozilla.org/show_bug.cgi?id=777917 as of 06-08-2013

[41] iOS Security, October 2012 http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf as of 06-09-2013

[42] Blackberry Z10 Reasearch Primer, Dissecting Blackberry 10 – An initial analysis v1.0, by Alexander Antukh, SEC Consult Vulnerability Lab, Vienna, 05/2013 https://www.sec-consult.com/fxdata/seccons/prod/downloads/sec_consult_vulnerability_lab_blackberry_z10_initial_analysis_v10.pdf as of 06-09-2013

[43] BlackBerry 10 wins Pentagon's security approval http://news.cnet.com/8301-1035_3-57582683-94/blackberry-10-wins-pentagons-security-approval/ as of 06-09-2013