

Enhancing Randomization in Loading the Library Image in Android RunTime (ART)

Field of the Invention

This disclosure relates to a method and system for enhancing the randomisation of the address of the library image in Android RunTime (ART). Particularly, this disclosure relates to a method and system for randomising base address of boot image and adding gap in the native codes in an Android platform.

Summary of the Prior Art

Address Space Layout Randomization (ASLR) is an exploitation mitigation technique to defend against Return-Oriented Programming (ROP) attacks. This technique makes it more difficult to locate useful ROP gadgets by randomizing target process's memory layout, especially instruction locations.

ROP is a modern exploitation technique which is normally used by attackers for malicious purposes without code injection. In exploits, ROP attacks chained small code sequences ending with ret/ret-like instructions (called gadgets) in the binary of the target process to perform malicious behaviors. The address of each chosen gadget is assigned to the stack for the previous ret/ret-like instruction to return to. As one of the most effective methods mitigating ROP attacks, ASLR can randomize the absolute addresses of instructions. Therefore, it is more difficult for attackers to locate and chain ROP gadgets.

In Android 5.0 Lollipop, Android runtime (ART) is introduced to replace Dalvik Virtual Machine (DVM) to execute Android applications and some system services. However, ART not only introduces more gadgets to the binary of applications, but also uses a weakened randomization strategy.

The main feature of ART is ahead-of-time (AOT) compilation, which has better application performance than the just-in-time (JIT) mode of DVM. The AOT compilation uses an on-device tool dex2oat to compile DEX files and generate executables in a special ELF (Executable and Linkable Format) format named OAT. This pre-compilation process converts the Java byte code in DEX files to native code in the OAT executables. Since Android applications are mainly developed by Java, the amount of native code tremendously increases in ART mode, especially from the native code of the compiled Java system

libraries. The increased number of binary instructions gives more flexibility to ROP attackers to construct malicious code.

Furthermore, the ASLR used by ART mode is weakened by the pre-compilation of Android Java core libraries. This is because under the ART mode, the boot image containing compiled Java system libraries is only randomized when the system boots for the first time. After that, the boot image will be loaded to the same address in each application's memory, even after the system reboots.

Hence, those skilled in the art are striving to strengthen the randomization scheme of Android ART to protect Android system libraries from ROP based attacks.

Summary of the Invention

The above and other problems are solved and an advance in the state of art is made by systems and methods provided by embodiments in accordance with the disclosure. A first advantage of embodiments of systems and methods in accordance with the disclosure is that the systems and methods may be implemented by either modifying the Android Operating System and by a bootstrap module. Hence, the systems and methods in accordance with the disclosure can be implemented to current platform. A second advantage of embodiments of systems and methods in accordance with the invention is that systems and methods significantly increase the difficulty of ROP attacks on Android ART by introducing more complex randomization strategies to the system. With a more complex randomization strategy, it is harder for attackers to obtain the absolute address of ROP gadgets in Java system libraries and chain them to perform malicious behaviours.

The above advantages are provided by embodiments of a system and method for randomising boot image for an Android RunTime (ART) virtual machine during a system boot.

In accordance with a first aspect of the disclosure, a boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot is provided in the following manner. The method comprises: creating an image space in an ART heap; wherein the image space is used to load a boot image; deleting the boot image in a cache path; relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a

patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space.

In accordance with an embodiment of the disclosure, after the step of relocating the boot image according to a new patch delta, the method further creates a Class Linker according to the ART heap such that when a system class is going to be linked based on the Class linker, an image header will be obtained from the image space in the heap the patched address of boot image will be used to find the system class.

In accordance with an embodiment of the disclosure, the boot image contains a boot.oat file and the step of relocating the boot image according to a new patch delta further comprises selecting gaps randomly in a text section of the boot.oat file, and patching the boot.oat file according to the selected gaps.

In accordance with a second aspect of the disclosure, a boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot is provided in the following manner. The method comprises: creating an image space in an ART heap, wherein the image space is used to load a boot image; checking whether the boot image is cached in a cache path; loading the cached boot image in the image space in response to the boot image is available in the cache path; relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space; creating a Class Linker according to the ART heap; wherein the step of loading the cached boot image in the image space in response to the boot image is available in the cache path comprises: reading an image header from the cached boot image, in which the patch delta for determine loading address is stored; modifying the patch delta randomly to load the boot image in a new address.

In accordance with an embodiment of the disclosure, the method further comprises amending the Class Linker according to the modified patch delta.

In accordance with an embodiment of the disclosure, the boot image contains a boot.oat file and the step of relocating the boot image according to a new patch delta further comprises selecting gaps randomly in a text section of the boot.oat file, and patching the boot.oat file according to the selected gaps.

In accordance with an embodiment of the disclosure, the step of loading the cached boot image in the image space in response to the boot image is available in the cache path

further comprises reading boot.oat from the cached boot image, and adding random gaps in a text section in the boot.oat file and modifying the offsets related to text section accordingly.

In accordance with an embodiment of the disclosure, the step of loading the cached boot image in the image space in response to the boot image is available in the cache path further comprises: reading boot.oat from the cached boot image; and adding random gaps in a text section in the boot.oat file and modifying the offsets related to text section accordingly.

In accordance with a third aspect of the disclosure, a boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot is provided in the following manner. The method comprises creating an image space in an ART heap wherein the image space is used to load a boot image containing a boot.oat file, checking whether the boot image is cached in a cache path, relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space, and wherein the step of relocating the boot image according to a new patch delta further comprises selecting gaps randomly in a text section of the boot.oat file, and patching the boot.oat file according to the selected gaps.

In accordance with a fourth aspect of the disclosure, a boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot is provided in the following manner. The method comprises creating an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file; checking whether the boot image is cached in a cache path; loading the cached boot image in the image space in response to the boot image is available in the cache path; relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space; and wherein the step of loading the cached boot image in the image space in response to the boot image is available in the cache path further comprises: reading boot.oat from the cached boot image; adding random gaps in a text section in the boot.oat file and modifying the offsets related to text section accordingly.

In accordance with a fifth aspect of the disclosure, a device is provided in the following manner. The device comprises a processor, non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an

Android RunTime (ART) virtual machine in a system boot, to: create an image space in an ART heap; wherein the image space is used to load a boot image; delete the boot image in a cache path; relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and load the boot image to the image space.

In accordance with an embodiment of this disclosure, the device further comprises instructions after the instruction to relocate the boot image according to a new patch delta, to: create a Class Linker according to the ART heap such that when a system class is going to be linked based on the Class linker, an image header will be obtained from the image space in the heap the patched address of boot image will be used to find the system class.

In accordance with an embodiment of this disclosure, the boot image contains a boot.oat file and wherein the instruction to relocate the boot image according to a new patch delta further comprises instructions to: select gaps randomly in a text section of the boot.oat file; and patch the boot.oat file according to the selected gaps.

In accordance with a sixth aspect of this disclosure, a device is provided in the following manner. The device comprises a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to: create an image space in an ART heap, wherein the image space is used to load a boot image; check whether the boot image is cached in a cache path; load the cached boot image in the image space in response to the boot image is available in the cache path; relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and load the boot image to the image space; create a Class Linker according to the ART heap; wherein the instruction to load the cached boot image in the image space in response to the boot image is available in the cache path comprises instructions to: read an image header from the cached boot image, in which the patch delta for determine loading address is stored; modify the patch delta randomly to load the boot image in a new address.

In accordance with an embodiment of this disclosure, the device further comprises instructions to amend the Class Linker according to the modified patch delta.

In accordance with an embodiment of this disclosure, the boot image contains a boot.oat file and the instruction to relocate the boot image according to a new patch delta further comprises instructions to: select gaps randomly in a text section of the boot.oat file; and patch the boot.oat file according to the selected gaps.

In accordance with an embodiment of this disclosure, the instruction to load the cached boot image in the image space in response to the boot image is available in the cache path further comprises instructions to: read boot.oat from the cached boot image; add random gaps in a text section in the boot.oat file and modify the offsets related to text section accordingly.

In accordance with a seventh aspect of this disclosure, a device is provided in the following manner. The device comprises a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to: create an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file; check whether the boot image is cached in a cache path; relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and loading the boot image to the image space; and wherein the instruction to relocate the boot image according to a new patch delta further comprises instructions to: select gaps randomly in a text section of the boot.oat file; and patch the boot.oat file according to the selected gaps.

In accordance with an eighth aspect of this disclosure, a device is provided in the following manner. The device comprises a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to: create an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file; check whether the boot image is cached in a cache path; load the cached boot image in the image space in response to the boot image is available in the cache path; relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and load the boot image to the image space; and wherein the instruction to load the cached boot image in the image space in response to the boot image is available in the cache path further comprises instructions to: read boot.oat from the cached boot image; add random

gaps in a text section in the boot.oat file and modify the offsets related to text section accordingly.

In accordance with a ninth aspect of the disclosure, a device running on an Android operating system is provided in the following manner. The device comprises a bootstrap module and a base address randomization module (BARM) wherein the bootstrap module is configured to invoke the BARM upon a system boot, and the BARM is configured to randomise a base address of a boot.oat file in a boot image.

In accordance with an embodiment of the disclosure, the step to randomise the base address of the boot.oat file in the boot image in each system boot comprises the BARM to delete the boot image in a cache path. In accordance with another embodiment of the disclosure, the step to randomise the base address of the boot.oat file in the boot image in each system boot comprises the BARM to modify a patch delta randomly and modify a Class Linker according to the random patch delta for loading boot image.

In accordance with an embodiment of the disclosure, the device further comprises a gap randomisation module (GAM) configured to introduce random gaps in a compiled system libraries in the boot.oat file.

In accordance with an embodiment of the disclosure, the step to introduce random gaps in the compiled system libraries in the boot.oat file comprises the GAM to monitor and detect when the boot image cache is being created, in response to detecting the boot image cache is being created, modify a patchcoat module of the Android operating system to randomly pick gaps and patch boot.oat according to the gaps, modify an offset pointers in the boot.oat file that points to the text section, and modify the boot.art file to update an OAT end pointer.

In accordance with an embodiment of the disclosure, the device further comprises gap randomisation module (GAM) configured to monitor and detect when the boot image cache is being created, in response to detecting the boot image cache is being created, modify a patchcoat module of the Android operating system to randomly pick gaps and patch boot.oat according to the gaps, modify an offset pointers in the boot.oat file that points to the text section, modify the boot.art file to update an OAT end pointer and wherein the bootstrap module is further configured to invoke BARM each time after the first system boot and GAM in the first system boot.

In accordance with an embodiment of the disclosure, the device further comprises a gap randomisation module (GAM) configured to monitor and detect for an OAT file being

loaded into a memory through a specific function of opening the OAT file during initializing an image space, in response to detecting the OAT file is being loaded into the memory, activate a binary rewriting module to reads a boot image from the cache path, add random gaps in the text section and modify the offsets related to text section accordingly, load modified boot image into the memory, and wherein the bootstrap module is further configured to invoke BARM and GAM each time after the first system boot.

In accordance with an embodiment of the disclosure, the device further comprises a gap randomisation module (GAM) configured to monitor and detect when the boot image cache is being created, monitor and detect for an OAT file being loaded into a memory through a specific function of opening the OAT file during initializing an image space, in response to detecting the boot image cache is being created, modify a patchcoat module of the Android operating system to randomly pick gaps and patch boot.oat according to the gaps, modify an offset pointers in the boot.oat file that points to the text section, and modify the boot.art file to update an OAT end pointer, in response to detecting the OAT file is being loaded into the memory, activate a binary rewriting module to reads a boot image from the cache path, add random gaps in the text section and modify the offsets related to text section accordingly, load modified boot image into the memory, and wherein the bootstrap module is further configured to invoke BARM and GAM each time after the first system boot, and wherein the bootstrap module is further configured to invoke GAM to monitor and detect for an OAT file being loaded into a memory through a specific function of opening the OAT file during initializing an image space in the first system boot, invoke BARM each time after the first system boot and invoke GAM to monitor and detect when the boot image cache is being created.

In accordance with a tenth aspect of the disclosure, a device running on an Android operating system is provided in the following manner. The device comprises a bootstrap module and a gap randomisation module (GAM) wherein the bootstrap module is configured to invoke the GAM upon a system boot, and the GAM is configured to introduce random gaps in a compiled system libraries in the boot.oat file.

In accordance with an embodiment of the disclosure, the step to introduce random gaps in a compiled system libraries in the boot.oat file comprises the GAM to monitor and detect when the boot image cache is being created, in response to detecting the boot image cache is being created, modify a patchcoat module of the Android operating system to randomly pick gaps and patch boot.oat according to the gaps, modify an offset pointers in

the boot.oat file that points to the text section, and modify the boot.art file to update an OAT end pointer.

In accordance with an embodiment of the disclosure, the step to introduce random gaps in a compiled system libraries in the boot.oat file comprises the GAM to monitor and detect for an OAT file being loaded into a memory through a specific function of opening the OAT file during initializing an image space, in response to detecting the OAT file is being loaded into the memory, activate a binary rewriting module to reads a boot image from the cache path, add random gaps in the text section and modify the offsets related to text section accordingly, load modified boot image into the memory.

Brief Description of the Drawings

The above advantages and features in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating a structure of a boot image;

Figure 2 illustrating a structure of an original boot.oat file;

Figure 3 illustrating a boot process of Android ART;

Figure 4 illustrating a process of loading boot image into memory;

Figure 5 illustrating a process of implementing the methods in accordance with the disclosure;

Figure 6 illustrating a structure of a modified boot.oat file in accordance with the disclosure; and

Figure 7 illustrating an example of a processing system that represents the processing system of a typical mobile device or device that uses Android platform that execute instructions to perform the processes in accordance with the disclosure.

Detailed Description

This disclosure relates to a method and system for enhancing the randomisation of the address of the library image in ART. Particularly, this disclosure relates to a method and

system for randomising base address of boot image and adding gap in the native codes in an Android platform.

This disclosure presents a new method for enhancing the randomization strategy of Android ART. This method would significantly increase the difficulty of ROP attacks through more complete randomization strategies for Android Java system libraries.

This disclosure proposes a few methods for enhancing randomization of ART and the corresponding implementations to randomize the memory layout of the boot image. Briefly, the first method involves randomising the base address of the boot image each time the system boots. The second method involves adding random gaps inside the boot image. Further details of the strategies would be described below. We would first briefly describe the current randomization strategy of Android ART before we describe the methods as proposed in accordance with this disclosure.

As mentioned above, Android leverages ASLR as one of its secure protection mechanisms. From Android 5.0, ART is introduced to be the execution environment to replace Dalvik Virtual Machine (DVM). ART brings in a large number of useful binaries for ROP attackers, especially from the compiled Java system libraries.

Many Java system libraries are compiled into a boot image, which is stored as non-writable files in a particular system directory. Android system mainly has 14 java system libraries such as `core-libart.jar` (for ART Runtime implementation), `okhttp.jar` (for HTTP protocol communication), `telephony-common.jar` (for phone operation, specifically SMS and MMS messages), and so on. When the system boots for the first time, a cache of the boot image will be created and stored in a cache path. At runtime, the cache of boot image will be loaded into memory and used by the system and the applications.

In other words, the boot image containing compiled Java system libraries is only randomized when the system boots for the first time. For subsequent system reboots, the boot image will be loaded to the same address in each application's memory. Since the native codes are shared between different applications and the boot image remains the same, this allows attackers to predict and utilise the ROP gadgets contained in the boot image.

The boot image 100 contains two files, `boot.art` 110 and `boot.oat` 120, the structure of which are shown in figure 1. The Java libraries of system framework are compiled to native code and stored in `boot.oat` file 120. Each compiled class has a segment of metadata 121d, which contains the pointers to its methods. All the compiled methods are continuously stored

in the text section of the OAT file as native code. The boot.art file 110 will manage the position of boot.oat 120 through the beginning of OAT file and a patch delta. The sum of OAT begin 111 and patch delta 112 will be the absolute address of where the OAT file is loaded into memory at runtime. It also contains pre-initialized classes and objects from the Java core libraries.

The boot.oat file 120 mainly contains a data section 121 and a text section 122. As shown in figure 2, in the data section 121, the first part is the OAT header 121a, which stores information about this OAT file including the count of DEX files 121c. After that contains the Dex file metadata 121b where each DEX file has a corresponding segment of metadata that specifies the offset of the DEX file, the offset of the OAT classes and the offsets for the methods of each class. All of the original DEX files 121c are stored in the data section 121. For each OAT class, an OAT class metadata 121d stores the information about the offset for each method and the offset of the base address of these methods.

The text section 122 continuously stores all the machine code of the compiled methods. Based on the offsets, base address, and patch delta of the boot image, the exact address for each method can be obtained.

As shown in figure 3, Android will initialize memory, load the kernel, and then start the kernel when the system boots shown by reference number 210. After that, it will start and create the ART virtual machine 220 and call Zygote's main function 230. The Zygote's main function 230 would initialize the Zygote process 231, the system server 232, the activity manager 233, and launcher 234. Then, the launcher will listen to click actions 235. If an application's icon is clicked, the corresponding application will be started through forking from the Zygote process 236.

When starting the ART virtual machine 220, the boot image will be initialized and loaded into memory as shown by reference number 221. Particularly, the system first creates an Android RunTime 222, and initialises the ART 223 by creating the ART heap 224. In the process of preparing the ART heap 224, an image space will be created to load the boot image 225. Android will check 226 whether the boot image is cached in the cache path. If the cache is found, the image space will be initialized by loading the cached image 227. Otherwise, the image will be relocated 228 from the one in the system path according to a new patch delta. The image relocating process will firstly remove all the applications' OAT caches, since these OATs refer the address of boot image which is going to be randomized. Then, a new patch delta will be randomly picked, with which the boot image will be patched and written to a path for caching. After that, the image space initialization will be executed

and the boot image will be loaded to the image space. After the ART heap is initialized, the Class Linker will be created and initialized 229 according to the heap. When a system class is going to be linked, the image header will be obtained from the image space in the heap and the patched address of boot image will be used to find the class.

Under ART mode, attackers can search gadgets from all the Android core libraries, DEX file and native libraries from the target application. Hence, the amount of resources for finding gadgets from is increased. Table 1 compares the gadget numbers in Android core library in Android 4.4 and 6.0. The size of boot.oat is more than 50MB which introduces 292,634 extra gadgets.

	Native Libraries			boot.oat file			total
	pop end	blx end	total	pop end	blx end	total	
Android 4.4 (Dalvik)	115,342	104,959	220,283	-	-	-	220,283
Android 6.0 (ART)	56,253	79,151	135,404	102,019	190,615	292,634	428,038

Table 1

To randomize the position of boot image in memory, ART randomly selects a patch delta (i.e. a random offset) when it is created, and then the system use a patchoat function to relocate the boot image during runtime with respect to the patch delta. When the system is booted, Android checks whether the boot image exists in the cache path. If the boot image is found, both boot.art and boot.oat will be reused, as well as the base address of boot.oat. If not, boot.art and boot.oat will be regenerated and, hence, re-randomized. Unless the boot image is not available, boot image will not be regenerated. In other words, boot image will only be generated during the first boot of the system and will not be regenerated after the first time the system boots.

With the above background on the current randomization strategy of Android ART, we shall now turn to the first method. This method involves randomising the base address of boot.oat each time the system boots. There are a few approaches of implementing this method, namely, including regenerating boot image and adding offset to boot.oat at load time.

The first approach involves regenerating the boot image each time the system boots. In this approach, we remove the boot image checking procedure of the cached boot image and always perform an image relocation before image space initialization. In other words, the check process 226 on whether the boot image is cached in the cache path would be bypassed and directed towards relocating image 228 from the one in the system path

according to a new patch delta. Therefore, boot.oat 120 will be generated and the offset of its base address (the patch delta) will be randomized every time the system boots.

To implement this first approach, the image space creating module is modified to bypass the checking process 226 by deleting boot image under the cache path every time the system boots or prior to shutting down of the system. Thus, each time the system boots, a new cache of boot image will be regenerated with a different base address.

The second approach involves randomising the base address of boot.oat 120 when creating and initialising image space 225. Each time when boot image is loaded into memory when initializing the image space, the image header will be read from the cached boot image, in which the patch delta for determine loading address is stored. This second approach will modify the patch delta randomly and dynamically and, thus, load the boot image in a new address. Subsequently the Class Linker needs to be modified according to the new delta of loading boot image.

Figure 4 illustrates the current process of loading boot image into memory. As mentioned above, the sum of OAT begin 111 and patch delta 112 is the absolute address of where the OAT file 120 is loaded into memory at runtime. Hence, to implement the second approach, the base address of the OAT file 120 is changed by randomly generating a patch delta, which is different from the one in the cached boot image. The Class Linker is then modified accordingly. The Class Linker has a function to obtain the required data, which is the patch delta in the image header. Hence, we need to modify the required delta in the Class Linker, whenever it is used to link classes in the boot image.

Figure 5 illustrates a process 500 of implementing the approaches in the first method of randomising the base address of boot.oat each time the system boots. To better differentiate the first and second approaches of the first method against the current operation, the current operation would be shown in dashed lines while the first and second approaches would be shown in solid lines.

As mentioned above, when starting the ART virtual machine 220, the boot image will be initialized and loaded into memory. Process 500 begins with step 505 where the system creates an Android RunTime 222. In step 510, process 500 initialises the ART 223. During initialising of the ART, ART heap 224 is created. Before creating the ART heap, process 500 removes the boot image in the cache path. Thereafter, process 500 continues to create the ART heap in step 520. Step 515 relates to one of the processes of bypassing the checking process 226 by deleting the boot image in the cache path every time the system boots. An

alternative to step 515 is to remove the boot image in the cache path during shutting down of the system.

In the process of preparing the ART heap 224 in step 520, an image space will be created to load the boot image 225 in step 522. In step 525, process 500 will check whether the boot image is cached in the cache path. If the cache boot image is found, the image space will be initialized by loading the cached boot image in step 530. If the cache boot image is not found, the image will be relocated from the one in the system path according to a new patch delta in step 535.

Step 530 is modified to implement the second approach in that the base address of the OAT file 120 is changed by randomly generating a patch delta in step 532, which is different from the one in the cached boot image. The Class Linker is then modified accordingly. The Class Linker has a function to obtain the required data, which is the patch delta in the image header. Hence, we need to modify the required delta in the Class Linker, whenever it is used to link classes in the boot image.

The image relocating process in step 535 will firstly remove all the applications' OAT caches in step 540, since these OATs refer the address of boot image which is going to be randomized. Then, a new patch delta will be randomly picked in step 545, with which the boot image will be patched and written to a path for caching. After that, the image space initialization will be executed and the boot image will be loaded to the image space in step 550.

After the ART heap is initialized, the Class Linker will be created and initialized 229 according to the heap in step 560. When a system class is going to be linked, the image header will be obtained from the image space in the heap and the patched address of boot image will be used to find the class. Process 500 ends after step 560.

The second method involves enhancing the randomization of ART by introducing random gaps in the compiled system libraries. There are a few approaches to introduce random gaps within the boot image. The first approach is to add random gaps when the boot image cache is created. The additional step of adding random gaps follows ART's randomization to the base address of a boot image when the boot image cache is created. As shown in figure 2, the machine code only exists in the text section, so the random gaps will be added among the methods of classes. The structure of boot.oat file will be changed to that in figure 6 after adding random gaps. The randomization will be performed before writing

the OAT file to the cache path which is during step 530 or 532 and 545 in process 500 as shown in figure 5.

The original system only patches according to the new patch delta. Hence, to implement the first approach, the patchoat module of Android system is modified to randomly pick gaps and patch boot.oat according to the gaps in step 545 in process 500. After adding gaps to the text section, the offset pointers in boot.oat is also modified which points to the text section, e.g. the methods offset pointers in DEX file metadata. Adding gaps to text section changes the length of the boot.oat. Hence, the boot.art is also modified to update the OAT end pointer.

The second approach randomises boot.oat by adding random gaps each time after the first system boot. From the process as illustrated in process 500, if the image cache can be found in the check process in step 525, the patch process will not be performed. Hence, the gaps in the text section of boot.oat will not be randomized anymore. In this second approach, we modify the loading process of the boot.oat file in step 530 to perform a binary rewriting process for adding the gaps. The cached boot.oat will be read and modified according to the original structure of it. Then, random gaps will be added among the native code of functions.

We add a binary rewriting module before loading the OAT file into memory. As shown in figure 3, when initializing the image space 227, the OAT file will be loaded into the memory through a specific function of open the OAT file. When this occurs, the binary rewriting module reads a boot image from the cache path. Based on the structure of the boot.oat file, we can add random gaps in the text section and modify the offsets related to text section accordingly. The modified boot image will be loaded into the memory.

The above methods and approaches may be implemented in alone or in various combinations as follows:

1. First approach of method 1 together with first approach of method 2

This involves regenerating the boot image and adding gaps to text section each time the system boots.

2. Second approach of method 1 together with first approach of method 2

This involves randomising the base address of boot.oat 120 when creating and initialising image space 225 each time after the first system boot and randomising boot.oat file by adding random gaps during the first system boot.

3. Second approach of method 1 together with second approach of method 2

This involves randomising the base address of boot.oat 120 when creating and initialising image space 225 and adding gaps to text section each time after the first system boot.

4. Second approach of method 1 together with first and second approaches of method 2

This involves randomising the base address of boot.oat 120 when creating and initialising image space 225 and adding gaps to text section each time after the first system boot, and randomising boot.oat by adding random gaps during the first system boot.

The methods and approaches may also be implemented on the Android system using a bootstrap module to take over the startup of the system. Two further modules may be provided, namely, base address randomization module (BARM) and gap randomisation module (GAM). Essentially, the first method which involves randomising the base address of boot.oat each time the system boots is performed by the BARM while the second method which involves introducing random gaps in the compiled system libraries is performed by the GAM.

During a system boot, the bootstrap module will invoke the BARM and/or the GAM depending on the combination of the methods and approaches that is implemented.

In a first embodiment, when the BARM is invoked by the bootstrap module, the BARM will delete boot image under the cache path. Thus, each time the system boots, a new cache of boot image will be regenerated with a different base address.

In a second embodiment, when the BARM is invoked by the bootstrap module, the BARM will modify the patch delta randomly and dynamically and modify the Class Linker according to the new patch delta for loading boot image.

In a third embodiment, when the GAM is invoked by the bootstrap module, the GAM will introduce gaps when the image is being created in the following manner. In particular, the GAM will monitor and detect when the boot image cache is being created. Upon detecting the boot image cache is being created, the GAM will perform the following instructions:

1. Modify the patchoat module of Android operating system to randomly pick gaps and patch boot.oat according to the gaps.
2. Modify the offset pointers in boot.oat which points to the text section, e.g. the methods offset pointers in DEX file metadata.

3. Modify the boot.art to update the OAT end pointer.

In a fourth embodiment, when the GAM is invoked by the bootstrap module, the GAM will monitor and detect for the OAT file being loaded into the memory through a specific function of opening the OAT file during initializing the image space. Upon detecting the OAT file is being loaded into the memory, the GAM will perform the following instructions:

1. Activate a binary rewriting module to read a boot image from the cache path.
2. Add random gaps in the text section (i.e. among the native code of functions) and modify the offsets related to text section accordingly, based on the structure of the boot.oat file that is read in step 1.
3. Load modified boot image into the memory.

The bootstrap module will invoke the BARM and/or the GAM depending on the combination of the methods and approaches that is implemented. The bootstrap module can invoke the BARM and/or GAM alone or in various combinations as follows:

- Embodiment 1. Invoke BARM to delete boot image under the cache path. This would cause a new cache of boot image to be regenerated with a different base address each time the system boots.
- Embodiment 2. Invoke BARM to modify the patch delta randomly and dynamically and modify the Class Linker according to the new patch delta of loading boot image each time the system boots.
- Embodiment 3. Invoke GAM to introduce gaps when the image is being created each time the system boots.
- Embodiment 4. Invoke GAM to introduce gaps when the OAT file is being loaded into the memory through a specific function of opening the OAT file during initializing the image space
- Embodiment 5. Invoke BARM according to embodiment 1 above and invoke GAM according to embodiment 3 above each time the system boots.
- Embodiment 6. Invoke BARM according to embodiment 2 each time after the first system boot and invoke GAM according to embodiment 3 during the first system boot
- Embodiment 7. Invoke BARM according to embodiment 2 above and invoke GAM according to embodiment 4 above each time after the first system boot
- Embodiment 8. Invoke BARM according to embodiment 2 above and invoke GAM according to embodiment 4 above each time after the first system

boot and invoke GAM according to embodiment 3 above during the first system boot

The methods as disclosed in this disclosure can be used by the Android system developers to create new and more secure Android operating systems. They can secure the Android system by embedding randomization strategies in the original system according to this disclosure. For example, the methods as disclosed in this disclosure can be implemented on Target Android 5.0 + platforms which using ART mode. The randomization may be directed to pre-compiled Java system libraries on Android platform. It is also possible to apply the method as disclosed in this disclosure in other embedded systems that uses the same “image loading” method for system library loading. Particularly, the methods as disclosed in this disclosure may also be implemented on other platforms which use the ahead-of-time (AOT) compilation to Java code.

The method as disclosed in this disclosure significantly increases the difficulty of ROP attacks on Android ART by introducing more complex randomization strategies to the system. With the method as disclosed in this disclosure, it is harder for attackers to obtain the absolute address of ROP gadgets in Java system libraries and chain them to perform malicious behaviours. The implementation of the method as disclosed in this disclosure may be based on system modification or via a bootstrap module to take over the boot process. Thus, there is no need to modify any third party applications.

The method and system described above are applicable to mobile devices or any devices that uses Android platform. Figure 7 illustrates an example of a processing system 700 that represents the processing system of a typical mobile device or device that uses Android platform that execute instructions to perform the processes described above in accordance with embodiments of this disclosure. The system and method described above are executable directly on the processing system 700 or via a virtual machine running on the processing system 700. One skilled in the art will recognize that the exact configuration of each processing system may be different and the exact configuration of the processing system executing processes in accordance with this invention may vary and processing system 700 shown in figure 7 is provided by way of example only.

Processing system 700 includes a processor 710, a radio transceiver 720, an image capturing device 730, a display 740, a keypad 750, a memory 760, a Bluetooth module 770, a Near Field Communication (NFC) module 780, and an I/O device 790.

The radio transceiver 720, image capturing device 730, display 740, keypad 750, memory 760, Bluetooth module 770, NFC module 780, I/O device 790 and any number of other peripheral devices connect to processor 710 to exchange data with processor 710 for use in applications being executed by processor 710.

The radio transceiver 720 is connected to an antenna which is configured to transmit outgoing voice and data signals and receive incoming voice and data signals over a radio communication channel. The radio communication channel can be a digital radio communication channel such as a CDMA, GSM or LTE channels that employs both voice and data messages in a conventional techniques.

The image capturing device 730 is any device capable of capturing still and/or moving images such as complementary metal-oxide semiconductor (CMOS) or charge-coupled sensor (CCD) type cameras. The display 740 receives display data from processor 710 and display images on a screen for a user to see. The display 740 may be a liquid crystal display (LCD) or organic light-emitting diode (OLED) display. The keypad 750 receives user input and transmits the input to processor 710. In some embodiments, the display 740 may be a touch sensitive surface that functions as a keypad to receive user input.

The memory 760 is a device that transmits and receives data to and from processor 710 for storing data to a memory. The Bluetooth module 770 is a module that allows processing unit 710 to establish communication with another similar device based on Bluetooth technology standard. The NFC module 780 is a module that allows processing system 700 to establish radio communication with another similar device by touching them together or by bringing the devices within a close proximity.

Other peripheral devices that may be connected to processor 710 include a Wi-Fi transceiver, a Global Positioning System (GPS), a RFID transceiver, an ultra wideband transceiver and other positioning transceivers.

The processor 710 is a processor, microprocessor, or any combination of processors and microprocessors that execute instructions to perform the processes in accordance with the present disclosure. The processor has the capability to execute various application programs that are stored in the memory 760. These application programs can receive inputs from the user via the display 740 having a touch sensitive surface or directly from a keypad 750. Some application programs stored in the memory 760 that can be performed by the processor 710 are application programs developed for Android or other mobile platforms.

The above is a description of embodiments of a method and system of randomising base address of boot image and adding gap in the native codes. It is foreseeable that those skilled in the art can and will design alternative method and system based on this disclosure that infringe upon this invention as set forth in the following claims.

CLAIMS:

1. A boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot comprising:

- creating an image space in an ART heap; wherein the image space is used to load a boot image;

- deleting the boot image in a cache path;

- relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space.

2. The boot image randomisation method according to claim 1, wherein after the step of relocating the boot image according to a new patch delta, further comprising:

- creating a Class Linker according to the ART heap such that when a system class is going to be linked based on the Class linker, an image header will be obtained from the image space in the heap the patched address of boot image will be used to find the system class.

3. The boot image randomisation method according to claim 1 or 2, wherein the boot image contains a boot.oat file and wherein the step of relocating the boot image according to a new patch delta further comprising:

- selecting gaps randomly in a text section of the boot.oat file; and

- patching the boot.oat file according to the selected gaps.

4. A boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot comprising:

- creating an image space in an ART heap, wherein the image space is used to load a boot image;

- checking whether the boot image is cached in a cache path;

- loading the cached boot image in the image space in response to the boot image is available in the cache path;

- relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space;

creating a Class Linker according to the ART heap; wherein the step of loading the cached boot image in the image space in response to the boot image is available in the cache path comprises:

reading an image header from the cached boot image, in which the patch delta for determine loading address is stored;

modifying the patch delta randomly to load the boot image in a new address.

5. The boot image randomisation method according to claim 4, further comprising:
amending the Class Linker according to the modified patch delta.

6. The boot image randomisation method according to claim 4 or 5, wherein the boot image contains a boot.oat file and wherein the step of relocating the boot image according to a new patch delta further comprises:

selecting gaps randomly in a text section of the boot.oat file; and

patching the boot.oat file according to the selected gaps.

7. The boot image randomisation method according to claim 4 or claim 5, wherein the step of loading the cached boot image in the image space in response to the boot image is available in the cache path further comprises:

reading boot.oat from the cached boot image;

adding random gaps in a text section in the boot.oat file and modifying the offsets related to text section accordingly.

8. A boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot comprising:

creating an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file;

checking whether the boot image is cached in a cache path;

relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space; and

wherein the step of relocating the boot image according to a new patch delta further comprises:

selecting gaps randomly in a text section of the boot.oat file; and

patching the boot.oat file according to the selected gaps.

9. A boot image randomisation method for an Android RunTime (ART) virtual machine during a system boot comprising:

- creating an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file;

- checking whether the boot image is cached in a cache path;

- loading the cached boot image in the image space in response to the boot image is available in the cache path;

- relocating the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, selecting a patch delta randomly, patching and writing the boot image to a path for caching, executing an image space initialization and loading the boot image to the image space; and

- wherein the step of loading the cached boot image in the image space in response to the boot image is available in the cache path further comprises:

- reading boot.oat from the cached boot image;

- adding random gaps in a text section in the boot.oat file and modifying the offsets related to text section accordingly.

10. A device comprising:

- a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to:

- create an image space in an ART heap; wherein the image space is used to load a boot image;

- delete the boot image in a cache path;

- relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and load the boot image to the image space.

11. The device according to claim 10 further comprises instructions after the instruction to relocate the boot image according to a new patch delta, to:

- create a Class Linker according to the ART heap such that when a system class is going to be linked based on the Class linker, an image header will be obtained from the image space in the heap the patched address of boot image will be used to find the system class.

12. The device according to claim 10 or 11, wherein the boot image contains a boot.oat file and wherein the instruction to relocate the boot image according to a new patch delta further comprises instructions to:

- select gaps randomly in a text section of the boot.oat file; and
- patch the boot.oat file according to the selected gaps.

13. A device comprising:

- a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to:

- create an image space in an ART heap, wherein the image space is used to load a boot image;

- check whether the boot image is cached in a cache path;

- load the cached boot image in the image space in response to the boot image is available in the cache path;

- relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and load the boot image to the image space;

- create a Class Linker according to the ART heap; wherein the instruction to load the cached boot image in the image space in response to the boot image is available in the cache path comprises instructions to:

- read an image header from the cached boot image, in which the patch delta for determine loading address is stored;

- modify the patch delta randomly to load the boot image in a new address.

14. The device according to claim 13 further comprising instructions to:

- amend the Class Linker according to the modified patch delta.

15. The device to claim 13 or claim 14, wherein the boot image contains a boot.oat file and wherein the instruction to relocate the boot image according to a new patch delta further comprises instructions to:

- select gaps randomly in a text section of the boot.oat file; and
- patch the boot.oat file according to the selected gaps.

16. The device according to claim 13 or claim 14, wherein the instruction to load the cached boot image in the image space in response to the boot image is available in the cache path further comprises instructions to:

- read boot.oat from the cached boot image;
- add random gaps in a text section in the boot.oat file and modify the offsets related to text section accordingly.

17. A device comprising:

- a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to:

- create an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file;

- check whether the boot image is cached in a cache path;

- relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and loading the boot image to the image space; and

- wherein the instruction to relocate the boot image according to a new patch delta further comprises instructions to:

- select gaps randomly in a text section of the boot.oat file; and

- patch the boot.oat file according to the selected gaps.

18. A device comprising:

- a processor; non-transitory memory; and instructions stored on the non-transitory memory executable by the processor, for an Android RunTime (ART) virtual machine in a system boot, to:

- create an image space in an ART heap, wherein the image space is used to load a boot image containing a boot.oat file;

- check whether the boot image is cached in a cache path;

- load the cached boot image in the image space in response to the boot image is available in the cache path;

- relocate the boot image according to a new patch delta in response to the boot image is not available in the cache path by removing all applications' OAT caches, select a patch delta randomly, patch and write the boot image to a path for caching, execute an image space initialization and load the boot image to the image space; and

wherein the instruction to load the cached boot image in the image space in response to the boot image is available in the cache path further comprises instructions to:

read boot.oat from the cached boot image;

add random gaps in a text section in the boot.oat file and modify the offsets related to text section accordingly.

19. A device running on an Android operating system comprising:

a bootstrap module and a base address randomization module (BARM) wherein the bootstrap module is configured to invoke the BARM upon a system boot, and the BARM is configured to delete a boot image in a cache path and randomise a base address of a boot.oat file in the boot image.

20. The device according to claim 19 wherein the step to randomise the base address of the boot.oat file in the boot image in each system boot comprises the BARM to:

modify a patch delta randomly to load the boot image in a new address.

21. The device according to claim 19 wherein the step to randomise the base address of the boot.oat file in the boot image in each system boot further comprises the BARM to:

modify a Class Linker according to the modified patch delta.

22. The device according to claim 20 further comprising:

a gap randomisation module (GAM) configured to introduce random gaps in a compiled system libraries in the boot.oat file.

23. The device according to claim 22 wherein the step to introduce random gaps in the compiled system libraries in the boot.oat file comprises the GAM to:

monitor and detect when the boot image cache is being created;

in response to detecting the boot image cache is being created,

modify a patchcoat module of the Android operating system to randomly pick gaps and patch boot.oat according to the gaps;

modify an offset pointers in the boot.oat file that points to the text section; and

modify the boot.art file to update an OAT end pointer.

24. The device according to claim 20 further comprising a gap randomisation module (GAM) configured to:

monitor and detect when the boot image cache is being created;
in response to detecting the boot image cache is being created,
modify a patchcoat module of the Android operating system to randomly pick gaps
and patch boot.oat according to the gaps;
modify an offset pointers in the boot.oat file that points to the text section;
modify the boot.art file to update an OAT end pointer and wherein the bootstrap
module is further configured to invoke BARM each time after the first system boot and GAM
in the first system boot.

25. The device according to claim 20 further comprising a gap randomisation module (GAM) configured to:

monitor and detect for an OAT file being loaded into a memory through a specific
function of opening the OAT file during initializing an image space;
in response to detecting the OAT file is being loaded into the memory,
activate a binary rewriting module to reads a boot image from the cache path,
add random gaps in the text section and modify the offsets related to text section
accordingly,
load modified boot image into the memory, and wherein the bootstrap module is
further configured to invoke BARM and GAM each time after the first system boot.

26. The device according to claim 20 further comprising a gap randomisation module (GAM) configured to:

monitor and detect when the boot image cache is being created;
monitor and detect for an OAT file being loaded into a memory through a specific
function of opening the OAT file during initializing an image space;
in response to detecting the boot image cache is being created,
modify a patchcoat module of the Android operating system to randomly pick gaps
and patch boot.oat according to the gaps;
modify an offset pointers in the boot.oat file that points to the text section; and
modify the boot.art file to update an OAT end pointer
in response to detecting the OAT file is being loaded into the memory,
activate a binary rewriting module to reads a boot image from the cache path,
add random gaps in the text section and modify the offsets related to text section
accordingly,
load modified boot image into the memory, and wherein the bootstrap module is
further configured to invoke BARM and GAM each time after the first system boot, and

wherein the bootstrap module is further configured to invoke GAM to monitor and detect for an OAT file being loaded into a memory through a specific function of opening the OAT file during initializing an image space in the first system boot, invoke BARM each time after the first system boot and invoke GAM to monitor and detect when the boot image cache is being created.

27. A device running on an Android operating system comprising:
a bootstrap module and a gap randomisation module (GAM) wherein the bootstrap module is configured to invoke the GAM upon a system boot, and the GAM is configured to introduce random gaps in a compiled system libraries in the boot.oat file.

28. The device according to claim 27 wherein the step to introduce random gaps in a compiled system libraries in the boot.oat file comprises the GAM to:

- monitor and detect when the boot image cache is being created;
- in response to detecting the boot image cache is being created,
- modify a patchoat module of the Android operating system to randomly pick gaps and patch boot.oat according to the gaps;
- modify an offset pointers in the boot.oat file that points to the text section; and
- modify the boot.art file to update an OAT end pointer.

29. The device according to claim 27 wherein the step to introduce random gaps in a compiled system libraries in the boot.oat file comprises the GAM to:

- monitor and detect for an OAT file being loaded into a memory through a specific function of opening the OAT file during initializing an image space;
- in response to detecting the OAT file is being loaded into the memory,
- activate a binary rewriting module to reads a boot image from the cache path,
- add random gaps in the text section and modify the offsets related to text section accordingly;
- load modified boot image into the memory.