Advanced Topics in Operating Systems Security Vulnerabilities and Cyber Defenses
Assignment 2

By Idan Mosseri

Part1:

The Android OS uses a mechanism called the Application sandbox to enforce security. This mechanism is a Linux, kernel-level Application Sandbox. The Application sandbox provides Android with several key security features, including:

- **A user-based permissions model**: In this mechanism, The Android platform takes advantage of the Linux user-based protection as a means of identifying and isolating application resources. The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process. This approach is different from other operating systems (including the traditional Linux configuration), where multiple applications run with the same user permissions.
- **Process isolation:** Applications get a dedicated part of the filesystem in which they can write private data, including databases and raw files. The Application Sandbox Prevents user A from reading user B's files. Ensures that user A does not exhaust user B's memory. Ensures that user A does not exhaust user B's CPU resources. Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, Bluetooth).
- **Extensible mechanism for secure IPC:** apart from the standard Linux IPC mechanisms Android also provides new IPC mechanisms:
    - Binder: A lightweight capability-based remote procedure call mechanism designed for high performance when performing in-process and cross-process calls.
    - Services: Services can provide interfaces directly accessible using binder.
    - Intents: An Intent is a simple message object that represents an "intention" to do something. For example, if your application wants to display a web page, it expresses its "Intent" to view the URL by creating an Intent instance and handing it off to the system. The system locates some other piece of code (in this case, the Browser) that knows how to handle that Intent, and runs it. Intents can also be used to broadcast interesting events (such as a notification) system-wide.
    - ContentProviders: A ContentProvider is a data storehouse that provides access to data on the device; the classic example is the ContentProvider that is used to access the user's list of contacts. An application can access data that other applications have exposed via a ContentProvider, and an application can also define its own ContentProviders to expose data of its own.
- **The ability to remove unnecessary and potentially insecure parts of the kernel**

this security model also extends to operating system applications. All software above the kernel, such as operating system libraries, application framework, application runtime, and all applications, run within the Application Sandbox. Thus, both ART and native applications run within the same security environment, contained within the Application Sandbox. this allows no restrictions on how an application can be written that are required to enforce security; in this respect, native code is just as secure as interpreted code. this security model also extends to operating system applications.

In contrast to what we learned in class where sandboxes like chromes sandbox are maintained within the application and need the explicitly written and supported by the app developer. Here the broker process is the kernel itself.

In Android, personal information cost sensitive APIs and Sensitive Data Input Devices are secured by using protected APIs. When trying to access a resource via a protected API the requesting users (process) privileges are checked and access is allowed only if the user has sufficient rights. Applications that choose to share information can use Android OS permission checks to protect the data from third-party applications. Upon installing any application, the user in prompted with a dialog showing all the permissions this app requires. The user in turn must allow these permissions in order for the app to be installed.

The above information is taken from:

https://source.android.com/security/overview/kernel-security

https://source.android.com/security/overview/app-security

<u>Part 2:</u>

Assumptions:

- All securable objects have a better than null security descriptor.
- The computer is not already compromised.
- Third party software dose not weaken the security of the system.
- After initialization, the target process does not require any access to folders outside its allowed folder. Any request to do so will fail.
- If the target wishes to write to stdout it should flush stdout after a call to printf (or any other function that does not automatically flush stdout) in order for the message to be received at the brokers side and be printed.

Architecture:

- The Broker process is named mysandbox it is responsible for spawning the target process with all the restrictions specified below, and receive and input from the user and forward it to the targets stdin and any output from the target and forward it to the its out stdout.
- The Broker first parses the command line and extracts the targets executable path and the path of the folder it will be allowed to access.
- The Broker then creates the allowed folder assigning it a security descriptor with a null DACL and an SACL with only the only the untrusted integrity level. This folder will later be assigned to the target as its current directory allowing the target to access it without searching any absolute path which will be denied.
- Next, The Broker creates new windows station and desktop for the target to run in thus preventing it from interacting directly with the UI and other processes via windows messages.
- Next, The Broker creates an enclosing job object for the target process. The job object prevents giving the target process the following limits:
    - Prevents Writing and reading from clipboard.
    - Prevents using user handles not associated with the job.
    - Gives the target its own global atoms table preventing it from using attacks related to the global atom table.
    - Prevents the target from changing the display settings.
    - Prevents the target from changing system parameters using SystemParametersInfo.
    - Prevents the target from creating and switching desktops using CreateDesktop and SwitchDesktop.
    - Prevents the target from calling ExitWindows or ExitWindowsEx.

  The job also forces the target process to die after receiving an unhandled exception and when the job object is closed.

- The Broker next creates the IPC that will be used to communicate with the target process. The IPC consists of two pipes one which will replace the targets stdout and another to replace its stdout and error. To both the broker will be listening/writing on the other end. The IPC users two threads one for listening to incoming messages from the target and displaying them on the brokers stdout and the second for writing any input from the console to the targets stdin.
- The Broker now creates two Access Tokens for the target process to use.

- The first is the lockdown token which will be the targets primary token this is a highly restricted token which has all groups in deny only except for the logon SID and integrity level. The token also has the restricted SID and logon SID as restricted SIDs used for the double check. This tokens integrity level will later be changed to untrusted prior to it executing any code. Using this token, the target process is effectively denied any access to securable objects except for those that have a null DACL, and any handles given to the target process from the broker.
- The second token is the initial token this token has higher access rights with low integrity level, all groups in deny only except logon SID, everyone, users and domain related SIDs
  This token also has the restricted, everyone, users and logon SIDs as a restricted SIDs.
  This token is used by the target process as an impersonation token during the process startup allowing it to load any DLLs and resources it needs prior to running any code.
- After creating these two token the broker creates the target process suspended with the lockdown token, inside the new desktop, with the allowed folder as its current directory and with only the IPCs target side read and write handles inherited.
- Next, the Broker Assigns the target process to the job object and sets the targets impersonation token to the initial token.
- Next, the Broker replaces the first two bytes of the target process's code with a two byte jump backwards effectively backing enter an infinite loop when it is started.
- After changing the targets entry point the broker resumes the target process giving it time to load any necessary resources with the initial token (note that the targets code is not being executed at this point because of the injected jump).
- The Broker then suspends the Target process once more, resets its token to the lockdown token sets its integrity level to untrusted and rewrites the original first to byte to the targets memory.
- Finally, the Broker Resumes the target for the last time. Now the target is running its own code inside the sandbox.

Usage:

From the command line write:

"mysandbox.exe -torun <TargetPath> -folder <AllowedFolderPath>"

Potential problems:

- According to the following blog https://blogs.msdn.microsoft.com/david_leblanc/2007/07/31/practical-windows-sandboxing-part-3/ there is a way for the target process to breakout of its stating and desktop though it did not work when I tried. Furthermore, even if the target dose brake out of is desktop and switches to the default one, its untrusted integrity level will prevent it from communicating with other process via windows messages. The above blog suggests fix to this problem using a random SID given to the target and adding a deny ACE in the default desktops DACL.
- AS of this implementation the target process is not allowed to use any recourse other than handles given to it at startup and files inside the given allowed folder. The IPC mechanism could be further expanded to receive formatted requests from the target and allow or deny these

requests according to a given policy. Depending on its implementation this mechanism might compromise the sandboxes integrity by exploiting errors in the parsing mechanism.

- Any external software executing code from the allowed folder may be risking the system.
- Securable objects that have null security descriptor, if the computer is already compromised, and third-party software that weakens the systems security may also compromise the sandbox.

Sources:

- https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md
- https://blogs.msdn.microsoft.com/david_leblanc/2007/07/27/practical-windows-sandboxing-part-1/
- https://blogs.msdn.microsoft.com/david_leblanc/2007/07/30/practical-windows-sandboxing-part-2/
- https://blogs.msdn.microsoft.com/david_leblanc/2007/07/31/practical-windows-sandboxing-part-3/
- https://wiki.mozilla.org/Security/Sandboxz
- https://msdn.microsoft.com/EN-US/library/windows/desktop/ms682499(v=vs.85).aspx
- https://blogs.msdn.microsoft.com/oldnewthing/20111216-00/?p=8873
- https://stackoverflow.com/questions/44027935/windows-process-with-untrusted-integrity-level
- http://s7ephen.github.io/SandKit/