Robomaster S1 – Sandbox Escape

Took some time, but now we have finally root access to the Robomaster S1. The following instructions are for Windows 10 users, but it should also work for any other OS.

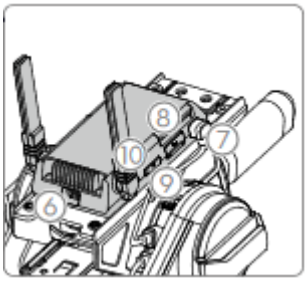
What you need

- Android SDK Platform-Tools (https://developer.android.com/studio/releases/platform-tools)
- Micro USB Cable
- Latest Robomaster S1 App & Firmware

Step by step instructions

1) Unzip the Android SDK Platform-Tools. Any directory will do for now. Remember the path, we will need it later.

2) Use the Intelligent Controller Micro USB Port (8) and connect the S1 to your computer:



Please be aware that no other Android device should be connected via USB!

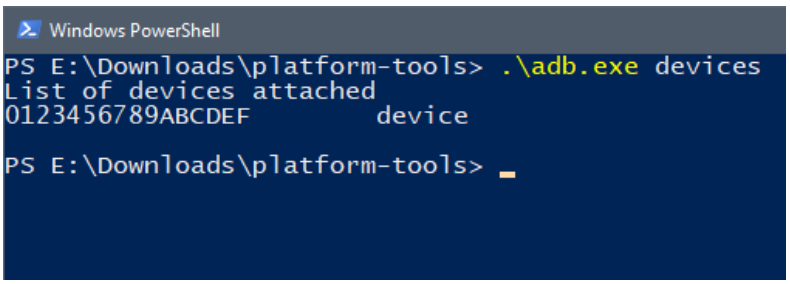
3) Start the Robomaster S1 application. Go to the Lab, create a new Python application and paste the following code:

```
def root_me(module):
    __import__=rm_log.__dict__['__builtins__']['__import__']
    return __import__(module,globals(),locals(),[],0)

builtins=root_me('builtins')

subprocess=root_me('subprocess')
proc=subprocess.Popen('/system/bin/adb_en.sh',shell=True,executable='
/system/bin/sh',stdout=subprocess.PIPE,stderr=subprocess.PIPE)
```

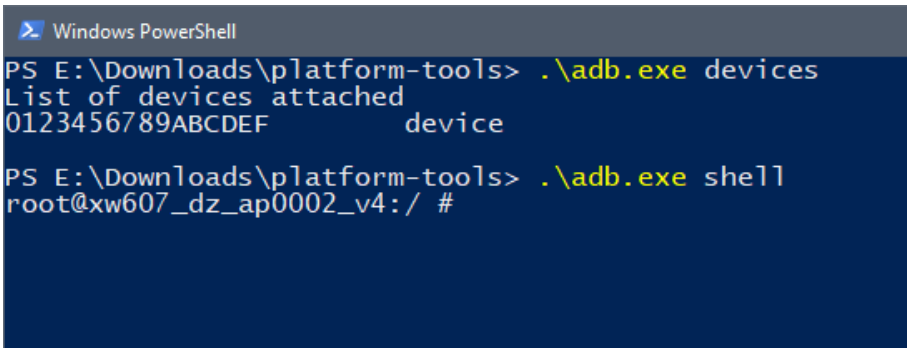
4) Run the Code within the S1 Lab. If you followed the steps correctly there should be no compilation errors. The Console will show: `Execution Complete`

5) Don't close the S1 Application! Open an Explorer window and go to the directory which holds the earlier extracted Android Platform Tools. Open a PowerShell in this directory (Shift + Right-Click)

6) Run the ADP command as follows: `.\adb.exe devices`. You should see something like this:



7) Execute: `.\adb.exe shell`
Ladies and Gentlemen, welcome to the Matrix:



Explanation

In *Python*, the module namespace is based on a dictionary/hash table implementation. Using the import statement we are able to import other modules. Specifically calling the built-in function [`__import__`] for module import.

In the *CPython* implementation the built-in functions are located in the built-in namespace and are accessible from the [`__builtins__`] property of the module namespace. By default, this property is associated with the [`__dict__`] property of the standard library [`builtins`] module.

Sandbox escape

DJI modified the [`__builtins__`] attribute & the [`__import__`] function, removed the built-in functions and set a whitelist for module import. However, the module in the whitelist (whose namespace is not modified) is still associated with the [`builtins`] module. Safe module names are:

```
'event_client'
'rm_ctrl'
'rm_define'
'rm_block_description'
'rm_log'
'tools'
'time'
'math'
'random'
'threading'
'traceback'
'tracemalloc'
```

[`rm_log`] is obviously the responsible module for log reading and writing - and has at least read and write permissions to the log directory. We can use this module as a springboard to access the Robomaster S1.

Analyzing the system directory shows:

- That the system is Android 4.4, the firmware is protected by selinux and is read-only
- The SD card directory is writeable
- The Python subsystem starts up and runs with root privileges
- The firmware determines whether the debug mode can be entered via `/proc/cmdline` using a whitelist

But, but, but … :) [`adb_en.sh`] can directly enable the adb service and open the serial port:

```
subprocess=root_me('subprocess')

proc=subprocess.Popen('/system/bin/adb_en.sh',shell=True,executable='/system/bin/sh',stdout=subprocess.PIPE,stderr=subprocess.PIPE)
```

At this point we can use the Python standard library for the Robomaster S1 and access the underlying hardware. Nice :)