November 16, 2018

Context-aware debug channel authentication for device testing

N/A

Context-aware debug channel authentication for device testing

ABSTRACT

Testing of a consumer device often includes coupling the device to a host computer and running test suites from the host computer. For authentication purposes, the device typically includes an extra image, e.g., a replica of the software to be tested on the device, which includes a pre-loaded debug bridge key. However, for testing of the complete software stack, e.g., including hardware-dependent portions, a new image cannot flash (overwrite) the existing image on the device, since the device is to be tested with as-shipped software. As a result, automated testing is not possible without a manual device bring-up step.

This disclosure provides several options to authenticate a debug-channel, e.g., by customizing builds, by conditionally preloading the debug bridge key, by preloading based on an elapsed time since the last build, by enabling or disabling the debug bridge key based on location of the device, by creating a new partition on the device that stores the debug bridge key, etc.

KEYWORDS

- compatibility test
- compliance test
- device test
- debug bridge key
- real time clock
- RTC
- debug authentication

BACKGROUND

A debug bridge is a communications interface that allows a developer to test a consumer device coupled to a host computer. A debug bridge often uses public-key encryption to enable the device-under-test and the host computer to authenticate each other.

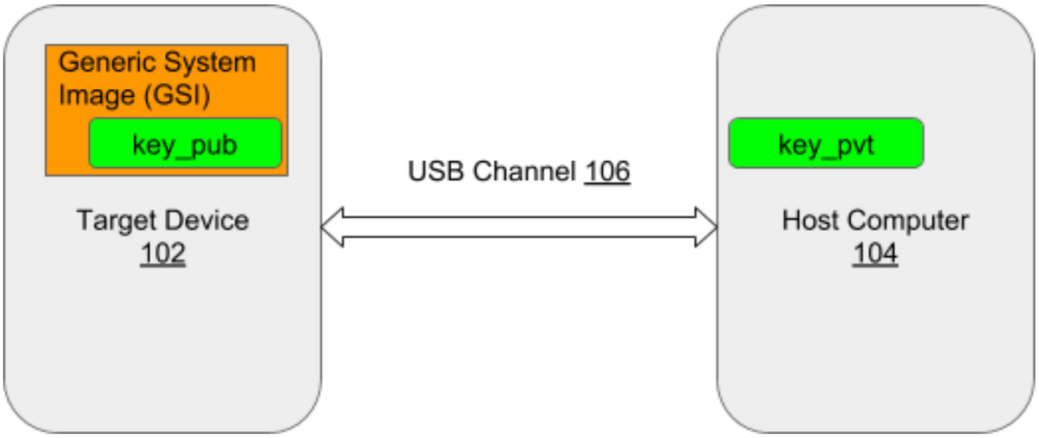

Fig. 1: Typical setup for testing consumer devices

Fig. 1 illustrates a typical test setup used for testing consumer devices. Target device (102) that is to be tested is coupled to a host computer (104) that runs test suites that exercise various modes of the test device. The coupling medium can be, e.g., a USB channel (106). For authentication purposes, the device typically includes an image, e.g., a replica of the software to be tested on the device, which includes a pre-loaded debug bridge key. The host computer maintains a private key for use during automatic authentication when the host computer and the device are coupled.

However, for testing of the complete software stack, e.g., including hardware-dependent portions, a new image cannot flash (overwrite) the existing image on the device, since the device is to be tested with as-shipped software. As a result, automated testing is not possible without a manual device bring-up step. Continuous integration is impractical for test devices that use as-shipped software. The manual intervention steps result in relatively high quality-assurance costs.

DESCRIPTION

For testing software branches of devices that do not use as-shipped software, e.g., those that use development branches, the software to be tested is customized, e.g., by pre-loading debug bridge keys into the software build. For release branches of software, e.g., as-shipped software builds, customization is done by creating a separate build target that is preloaded with a debug bridge key and built with the original user build targets.

If the device-under-test includes the debug key bridge, the preloading can be conditional based on software package type. For example, if the device image is signed, then the debug key bridge is not loaded at runtime; if device image is not signed, then the debug key bridge is loaded at runtime.

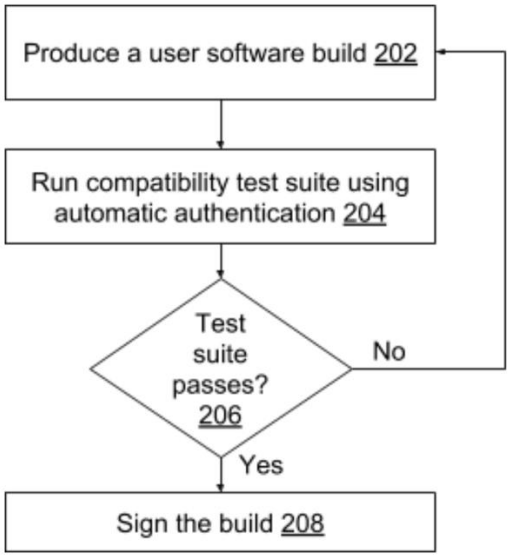

Fig. 2: Release procedure

The release process is as shown in Fig. 2. A user-targeted software build is built (202). Compatibility tests, e.g., a set of compatibility tests in a suite, are performed on this build using automatic authentication (204). If the tests pass, (206), the build is signed (208). At this point, there is no need to preload the debug bridge key, with no impact to users. If one or more of the

tests fail, the release process restarts with a new build. As long as the release process is locked down, over-the-air transmission of the build to the end-users cleans up the keys as part of release signing. The over-the-air tool is hardened to ensure that the keys are removed.

For added security, the debug bridge key can be preloaded if the time elapsed since build-production is less than a threshold. A real-time clock (RTC) can be used to check the time since a build was produced. For example, it generally takes more than one week from the time a build is ready to the point the build is transmitted over-the-air to end-users. Thus, if the threshold is set to one week, testing becomes automated, while ensuring that no user can take advantage of the preloaded debug bridge keys. A drawback of having such extra protection is that tests cannot be automated after a certain time interval, e.g., where demand is low but not zero.

Similarly, if users provide permission, other factors such as device location, e.g., subnet of an allocated IP address, GPS coordinates, etc., can be used to check whether a device is in a test environment and accordingly enable/disable the debug bridge keys. Alternatively, a new partition, e.g., an original design manufacturer (ODM) partition, can be created that includes the debug bridge key. By design, such partition is mounted to a specific location such that a debug bridge key daemon automatically loads the key if present in the new (partitioned) file system location.

CONCLUSION

This disclosure provides several options to authenticate a debug-channel, e.g., by customizing builds, by conditionally preloading the debug bridge key, by preloading based on an elapsed time since the last build, by enabling or disabling the debug bridge key based on location of the device, by creating a new partition on the device that stores the debug bridge key, etc.

REFERENCES

[1] Zhonglei, Wang. "Systems and methods for device compatibility testing and reporting." U.S. Patent Application 15/247,357, filed Aug. 25, 2016.