Fuzzy Fastboot

Fuzzy Fastboot (FF) is a standalone automated conformance and penetration tester for
validating device-side fastboot protocol implementations.
The tool is completely generic, and uses a simple extensible XML
configuration file to auto-generate device-specific tests for any device.
Any Android device that uses the fastboot protocol should have fuzzy fastboot run on it prior to
release to find implementation bugs, make sure it conforms to the fastboot spec,
and that it safely handles malicious inputs.

Background

The fastboot protocol provides an easy way to manage low level
aspects of the device directly from bootloader. However, with great power comes
great responsibility. An improper or insecure fastboot implementation can
open the possibility for critical security exploits on the bootloader via fastboot
commands. Furthermore, an untrustworthy or insecure bootloader means nothing that is
either directly or indirectly bootstrapped by the bootloader can be trusted (including Android).
By checking a bootloader's conformance to the fastboot spec, as well as make sure
nefarious/malformed input is properly and gracefully handled, easy exploits of a
device's bootloaders can be mitigated.

Additionally, since the fastboot tool itself must support a myriad of fastboot
implementations, it is important to make sure an implementation is conforming to
avoid potential incompatibilities with the fastboot command line tool itself.
Thus, Fuzzy Fastboot also checks for proper conformance to the spec.

Overview

Fuzzy Fastboot is written in C++ and uses Google Test
for the underlying test framework. This means that Fuzzy Fastboot supports all of
gtest's command line flags and options.

Additionally, by using gtest it makes it extremely easy to add additional C++ based
tests to Fuzzy Fastboot. However, in most cases the optional device specific
XML configuration file that is provided to Fuzzy Fastboot supports the necessary
features and hooks for testing device specific commands/features
without touching the underlying C++.

Generic Tests

Without a provided device XML configuration, Fuzzy Fastboot can only perform
some basic tests that are generic to any fastboot device. These generic tests are
divided into several test suite categories:

1. **USBFunctionality** - Test USB communication
2. **Conformance** - Test the device properly handles well-formed fastboot commands
3. **UnlockPermissions** - Test commands only allowed in the unlocked state work
4. **UnlockPermissions** - Test commands only not allowed in the locked state are rejected
5. **Fuzz** - Test malicious and/or ill-formed commands are properly and gracefully handled

XML Generated Tests

With a provided XML device configuration, Fuzzy Fastboot will be able to generate
many more additional tests cases.

The device config XML has five element pairs all inside a root level `<config>`:

`<getvar>` Element

Inside the `<getvar></getvar>` element pairs, one should list all the device's getvar
variables, with an associated ECMAScript regex you wish the returned variable to match on.
Each tested variable should appear in a `<var key="key" assert="regex"/>` format.
For example:

```
<getvar>
  <var key="product" assert="superphone2000"/>
```

```
    <var key="secure" assert="no|yes"/>
    <var key="battery-voltage" assert="[34][[:digit:]]{3}"/>
    <!-- etc...  -->
</getvar>
```

`<partitions>` Element

Inside the `<partitions></partitions>` element pairs, one should list all the device's partitions. Each device partition has should be put inside a `<part/>` element.
The `<part/>` element supports the following attributes:

Attribute	Value	Purpose	Default
value	Partition name	The name of the partition	Required
slots	"yes" or "no"	Is this partition is slotted	"no"
test	"yes" or "no"	Is Fuzzy Fastboot is allowed to generate tests that overwrite this partition	Required
hashable	"yes" or "no"	Is this partition hashable with the hash command specified in `<checksum>`	"yes"
parsed	"yes" or "no"	Does the bootloader parse this partition, such as look for a header, look for magic, etc…	"no"

For example:

```
<!-- All the device partitions should be listed here -->
<partitions>
  <part value="boot" slots="yes" test="yes" hashable="yes" parsed="yes"/>
  <part value="modem" slots="yes" test="yes" hashable="yes"/>
  <part value="userdata" slots="no" test="yes" hashable="no"/>
  <!-- etc...  -->
</partitions>
```

`<packed>` Element

Most devices have pseudo partitions, such as a `bootloader` partition, that in reality is composed of several real partitions.
When one of these pseudo partitions is flashed, the bootloader will internally expand the image into the individual images for each underlying partition. These pseudo partitions should be listed inside a `<part></part>` element pair. Each element `<part>` has a mandatory attribute `value`, which lists the name of this pseudo partition, and a `slots` attribute, which can be yes or no if this pseudo partition is slotted.
Additionally, inside the `<part></part>` element pair, one should list all the real partition that make up this pseudo partition inside of `<child>PART_NAME</child>` element pairs.
An example is should below:

```
<!-- All the device packed partitions should be listed here -->
<packed>
    <part value="bootloader" slots="yes">
        <!-- We list the real partitions it is composed of -->
        <child>foo1</child>
        <child>foo2</child>
        <child>bar3</child>
        <!-- We list tests, expect defaults to 'okay' -->
        <test packed="bootloader.img" unpacked="unpacked"/>
        <test packed="bootloader_garbage.img" expect="fail"/>
    </part>
</packed>
```

You might notice there are additional `<test/>` elements as well contained inside of a `<part></part>` pair. This is because Fuzzy Fastboot allows (and recommends) one to specify valid and invalid test packed images for flashing this particular pseudo partition.
Additionally, one should specify a folder with all the partitions' images that the packed image unpacks to. If your device supports hashing partitions, this will allow Fuzzy Fastboot to validate the images are unpacking correctly, and the correct slots are being flashed.

Each `<test/>` element has the following supported attributes:

Attribute	Value	Purpose	Default

packed	The name of the packed test image	The image uploaded to the device. It is searched for in dir if --search_path=dir	Required
unpacked	The name of the directory containing the unpacked version of packed	Searched for in dir if --search_path=dir. This folder should have the all the images that packed unpacks to. The name of each of the images should be the name of the real partition it is flashed to.	Required if expect != "fail"
expect	"okay" or "fail"	If uploading a invalid or garbage image the bootloader should reject use "fail" otherwise "okay"	"okay"

`<oem>` Element

Vendors can extend the fastboot protocol with oem commands. This allows vendors to support device/vendor specific features/commands over the fastboot protocol. Fuzzy Fastboot allows testing these oem commands as well.

Oem commands are specefied in `<command></command>` element pairs. Each command element supports the following attributes:

Attribute	Value	Purpose	Default
value	The oem command name	Ex: if value="foo", the oem command will start with "oem foo"	Required
permissions	"none" or "unlocked"	Whether the bootloader must be "unlocked" to perform command	"none"

An example is should below:

```
<oem>
  <command value="self_destruct" permissions="unlocked">
    <!-- This will test that "oem self_destruct now" returns 'okay' -->
    <test value="now" expect="okay"/>
    <!-- This will test that "oem self_destruct yesterday" returns 'fail' -->
    <test value="yesterday" expect="fail" />
  </command>

  <command value="foobar" permissions="unlocked">
    <!-- FF will first stage test_image.img before running 'oem foobar use_staged' -->
    <test value="use_staged" expect="okay" input="test_image.img" />
    <!-- FF will run 'oem foobar send_response', upload data from device, then run the validator script -->
    <test value="send_response" expect="fail" validate="python validator.py"/>
  </command>
<oem/>
```

Again you will notice that one can, and should, specify tests to run with `<test/>` elements.
The test elements support the following attributes:

Attribute	Value	Purpose	Default
value	The oem command argument	Ex: if value="bar", and the oem command name was "foo", the full command will be "oem foo bar"	Empty String (no argument)
expect	"okay" or "fail"	Whether the bootloader should accept or reject this command	"okay"
input	A image filename	Some oem commands require staging files before the command is executed	Empty String (no argument)
validate	A program/script to run to validate the response	Some oem commands will stage data that can be downloaded afterwards and should be validated to be correct. Fuzzy Fastboot will launch the validation program with the first arg the oem command executed, the second arg the path to the downloaded image. Ex: "python validate.py'. If the program has a non-zero return code, the validation is marked as failed and anything from the launched programs stderr is logged in the test failure.	Empty String (no argument)
assert	A Regular expression	In the "okay" or "fail" response, Fuzzy Fastboot will assert the response matches this regular expression.	Empty String (no argument)
output	The name of the saved file	This is the name of the saved output file passed to the validation script. It is saved in whatever location is specified by the --output_path argument	out.img

`<checksum/>` Element

If the bootloader supports hashing partitions (implementing this is strongly recommended), Fuzzy Fastboot can use it to do a bunch more testing. Make sure this hash is a cryptographically secure hash, as a non-secure one might reveal secrets about the partitions' contents.

The checksum element as no children and only two required attributes:

- **value** - The command that hashes a partition. Note that the name of the partition will be appended to the end of the command. For example, if `value="oem hash"`, hashing the partition `bar` would be issued with `oem hash bar`.
- **parser** - How the hash is returned is up to the vendor's implementation. It could be part of the `OKAY` response, or be encoded in `INFO` responses. Thus, the parser attribute is used to specify a program/script that will extract the hash. The first argument to the program will be the be the response from `OKAY`, the second argument will be all the `INFO` responses joined by newlines. The extracted hash should be sent back to Fuzzy Fastboot as a string written to stderr, and a return code of 0 to signal the parsing was successful. In the case of failure, return a non-zero return code, an optionally an associated error message written to stderr.

Full Example XML Configuration

Here is a basic example configuration. This can also be found in the 'example' folder as well as the associated python scripts 'checksum_parser.py' (used to extract partition hash), and 'validator.py' (used to validate an oem command that returns data).

```
<?xml version="1.0"?>
<config>
<!-- All the device getvar variables should be listed here -->
<getvar>
    <var key="product" assert="superphone2000"/>
    <var key="secure" assert="no|yes"/>
</getvar>

<!-- All the device partitions should be listed here -->
<partitions>
    <part value="boot" slots="yes" test="yes" hashable="yes" parsed="yes"/>
    <part value="modem" slots="yes" test="yes" hashable="yes"/>
    <part value="userdata" slots="no" test="yes" hashable="no"/>

    <!-- Bootloader partitions -->
    <part value="foo1" slots="yes" test="no" hashable="yes"/>
    <part value="foo2" slots="yes" test="no" hashable="yes"/>
    <part value="bar3" slots="yes" test="no" hashable="yes"/>
</partitions>

<!-- All the device packed partitions should be listed here -->
<packed>
    <part value="bootloader" slots="yes">
        <!-- We list the real partitions it is composed of -->
        <child>foo1</child>
        <child>foo2</child>
        <child>bar3</child>
        <!-- We list tests, expect defaults to 'okay' -->
        <test packed="bootloader.img" unpacked="unpacked"/>
        <test packed="bootloader_garbage.img" expect="fail"/>
    </part>
</packed>

<!-- All the oem commands should be listed here -->
<oem>
    <!-- The 'oem self_destruct' command requires an unlocked bootloader -->
    <command value="self_destruct" permissions="unlocked">
        <!-- This will test that "oem self_destruct now" returns 'okay' -->
        <test value="now" expect="okay"/>
        <test value="yesterday" expect="fail" />
    </command>

    <!-- Test a fictional 'oem get' command -->
    <command value="get" permissions="none">
        <test value="batch_id" expect="okay" assert="[[:digit:]]+"/>
        <test value="device_color" expect="okay" assert="green|blue"/>
        <test value="build_num" expect="okay" assert="[\w\-.]+"/>
        <test value="garbage" expect="fail" assert="Invalid var '[\w ]+'"/>
    </command>

    <!-- Some oem commands might require staging or downloading data, or both -->
    <command value="foobar" permissions="unlocked">
        <!-- FF will first stage test_image.img before running 'oem foobar use_staged' -->
        <test value="use_staged" expect="okay" input="test_image.img" />
        <!-- FF will run 'oem foobar send_response', upload data from device, then run the validator script -->
        <test value="send_response" expect="fail" validate="python validator.py"/>
    </command>
</oem>

<!-- If there is a custom oem checksum command to hash partitions, add it here -->
<checksum value="oem sha1sum"/>
</config>
```

Running Fuzzy Fastboot

Fuzzy Fastboot is built with the fastboot tool itself. It will appear in `out/host/linux-x86/testcases/fuzzy_fastboot/x86_64`.

Command Line Arguments

- **--config=**: Specify the name of the configuration XML file. If omitted, only the generic tests will be available.
- **--search_path=**: Specify the path where Fuzzy Fastboot will look for files referenced in the XML. This includes all the test images and the referenced programs/scripts. This is also where the --config is searched for. If this argument is omitted it defaults to the current directory.
- **--output_path**: Some oem tests can download an image to the host for validation. This is the location where that image is stored. This deafults to '/tmp'.
- **--serial_port**: Many devices have a UART or serial log, that reports logging information. Fuzzy Fastboot can include this logging information in the backtraces it generates. This can make debugging far easier. If your device has this, it can be specified with the path to the tty device. Ex: "/dev/ttyUSB0".
- **--gtest_***: Any valid gtest argument (they all start with 'gtest_')
- **-h**: Print gtest's help message

Using Fuzzy Fastboot on my Device

All Fuzzy Fastboot tests should pass on your device. No test should be able to crash the bootloader. Invalid input MUST be handled gracefully. Using "asserts" or panicking on invalid or malformed input is not an acceptable way to handle these tests, as ungraceful forced termination of the bootloader can expose vulnerabilities and leave the device in a bad state.

The following is the recommended workflow for using Fuzzy Fastboot on a new device:

Step 1: Pass the generic Conformance tests

Begin with just the generic tests (i.e. no XML file). In particular, make sure all the conformance tests are passing before you move on. All other tests require that the basic generic conformance tests all pass for them to be valid. The conformance tests can be run with `./fuzzy_fastboot --gtests_filter=Conformance.*`.

Understanding and Fixing Failed Tests

Whenever a test fails, it will print out to the console the reason for failure and the lines and file where the error happened. At the end of each failure block, there will usually be a message that Fuzzy Fastboot reports to gtest explaining what went wrong. An example is shown below:

```
Expected equality of these values:
  resp
    Which is: "no"
  unlock ? "yes" : "no"
    Which is: "yes"
getvar:unlocked response was not 'no' or 'yes': no
system/core/fastboot/fuzzy_fastboot/fixtures.cpp:227: Failure
Expected: SetLockState(UNLOCKED) doesn't generate new fatal failures in the current thread.
  Actual: it does.
[THERE WILL BE A MESSAGE HERE EXPLAINING WHY IT FAILED]
```

In most cases this message at the bottom is all that is needed to figure out why it failed. If this is not enough information, below this, gtest will also print out a human readable backtrace of the underlying fastboot commands leading up the failure in this test. Here is an example:

```
<<<<<<<< TRACE BEGIN >>>>>>>>>
[WRITE 0ms](15 bytes): "getvar:unlocked"
[READ 20ms](6 bytes): "OKAYno"
<<<<<<<< TRACE END >>>>>>>>>
```

One can easily see the reason for the failure was the test expected the device to be unlocked.

If it is still unclear why the failure is happening, the last thing to do is look at what line number and file is generating the error. Gtest will always print this out. You can then manually look through Fuzzy Fastboot's test source code, and figure out what went wrong.

Step 2: Pass all the other generic tests

Run all the other generic tests (still no XML file). A list of all of them can be
printed out with: "./fuzzy_fastboot --gtest_list_tests". As before, "--gtest_filter"
can be used to select certain tests to run, once you figure out which ones failed.

One particular set of tests to watch out for are the ones that involve USB resets.
USB resets effectively unplug and replug the device in software. Fuzzy Fastboot,
expects USB resets to cancel whatever transaction is currently going on.
This is also how Fuzzy Fastboot attempts to recover from errors when the device is
unresponsive.

Step 3: Create a device XML configuration

Without a device specific configuration file, Fuzzy Fastboot will have poor test
coverage of your device. The vast majority of tests are auto-generated via the XML
configuration file. Use the guide above to generate a configuration for your device.
Make sure to be as thorough as possible, and list everything in the configuration
that can be tested. Finally, make sure that the packed pseudo partitions and
oem commands all have provided test cases. Be sure to test both the positive case
(i.e. with valid input), as well as the opposite. Make sure the failure tests
have good coverage by thinking about all the ways invalid and malicious inputs
could be formed. These means creating images with malformed headers, illegal chars,
and other evil inputs.

Now run fuzzy_fastboot with the supplied configuration file. If you do "--gtest_list_tests",
you should see a ton more tests that were autogenerated by Fuzzy Fastboot.
As before, run these tests till everything passes. Again, use "--gtest_filter"
to select specific tests to run once you know what fail,
as running the whole things with a large configuration can take over 30 minutes.
See the gtest documentation, for nifty tricks and command line options.

Step 4: Figure out what Fuzzy Fastboot can't/isn't testing

While Fuzzy Fastboot with a XML configuration file, should provide good test coverage.
Think about what device specific things are not being tested, and test them manually.
In particular, things that if not handled carefully could create security exploits.
Don't be lazy here, as you already put in the time to get this far.

Step 5: Celebrate

You're done :). Now you can be more confident that your implementation is sound, and
have piece of mind knowing you are protecting the users' security and data by
running these tests. Don't get too complacent. If the bootloader's source code
is modified in a way that could introduce bugs or security issues. Make sure to
test again. You might have to add to your existing configuration file.

Limitations and Warnings

- Currently this only works on Linux (even if it builds on Mac)
- Passing these tests does not mean there are not bugs/security issues. For example, a buffer overrun might not always trigger a crash or have any noticeable side effects.
- **Be extremely careful of the Fuzzy Fastboot tests you are running. Know exactly what the tests do you are about to run before you run them. It is very possible to brick a device with many of these tests.**

Fuzzy Fastboot Missing Features TODO's

The following are missing features that should eventually be added

- *Sparse Image Tests*: Currently there are no tests that tests sparse images. Both well-formed and malicious images need to be tested.
- *Saved Test Log*: Fuzzy Fastboot should be able to create a failure log for every failing test and save it to a file. This file should include the test information, the reason it failed, and the fastboot command trace (with the serial console logs). Currently it just prints it to the console at the end of every test.
- *Host Side Hashing*: One should be able to provide the hashing algorithm to the Fuzzy Fastboot, so it can be checked to agree with what the device is reporting.

Author

Aaron Wisner - awisner@google.com