Test Plan

Skullduggery

Greg Baker - Robert Koeninger

Advisor: Prof. Wilsey

Feb 22, 2010

Revised: May 25, 2010

Test Plan Overview

The testing will take place in three environments; The first functional tests will take place between two emulated instances of Android phones. This will be performed using the Android Debug Bridge to test the functionality of the two devices. The other will be a performance-based test using a physical Android smart phone and a Java app built to run natively on the host machine. This test will perform a more realistic simulation of a call between the two phones. The final functional tests are functional tests of various abilities of Android and Java capabilities, to ensure that the phone is performing what we think it is.

In the first test, two virtual phones will be launched using the Skullduggery application. A call will be made from one phone to the other, during which the Skullduggery application will run, and ensure that various stages of communication are taking place, as well as the correctness of each step. The purpose of this is to verify the correctness of the functioning program before the functional test. This will be termed the “emulated” call.

The second test is a speed and capability based test. Since, during development, the emulated instances had proved insufficiently fast on accessible machines, a mock Skullduggery application was written that would communicate with the phone, and prove that the app worked sufficiently fast that communication could take place between two people. This will be termed the “simulated” call.

The third test set is a basic functionality test. These tests helped us model various aspects of the communication, and are primarily minor and experimental, but essential to our understanding and the progress of the project.

Various debug data will be written during these tests; During the emulated phone call instance, debug data will be accessible using the Android Debug Bridge and Eclipse environment to enable us to look at various stages of communication between the two phones. During the simulated call, data from the phone will be logged to a raw audio file, which can be reconstructed afterwards.

Test Case Description

1. AES Encryption/Decryption - Validate AES (unit, black-box)
    1.1. Validate AES library against standard Java Library
    1.2. Encrypt several different strings with a known valid AES library.
    1.3. Example inputs:
        1.3.1. '\..\
        1.3.2. 'abcd....'
        1.3.3. 'abcd...\n'
        1.3.4. 'ab\...\n'
    1.4. Outputs should match between our Android and Java AES implementations
        1.4.1. Decrypted strings should match plaintext
    1.5. Abnormal test cases should be spotted with null input, eol inputs.

2. Playing of voice data (integration, white-box)
    2.1. Ensure that we're able to play voice data on the speaker
    2.2. Play arbitrary sound data
    2.3. Example inputs:
        2.3.1. raw data from music sample
    2.4. Output of the test should be what we wanted to output (tone, waves, etc)
    2.5. Abnormal test cases include static, no sound, errors, crashes, etc.

3. Capturing of voice packets (integration, white-box)
    3.1. Ensure that we're catching the voice data
    3.2. Capture voice packets, play them back immediately
    3.3. Example inputs:
        3.3.1. Raw data from music sample
    3.4. Outputs should be the matching voice phrase
    3.5. Abnormal test cases include mangling of voice, no sound, loud sound

4. Encryption of voice packets (integration, white-box)

    4.1. Ensure that we're encrypting the voice data

    4.2. Capture voice packets, encrypt voice packets, output encrypted, raw voice

    4.3. Example inputs:

        4.3.1. "This is my encrypted voice"

    4.4. Outputs should be encrypted voice followed by raw voice

    4.5. Abnormal test cases include no mangling of voice, no sound, loud sound

5. Decryption of voice packets (integration, white-box)

    5.1. Ensure that we're encrypting the voice data

    5.2. Capture voice packets, encrypt voice packets, output encrypted, raw voice

    5.3. Example inputs:

        5.3.1. "This is my decrypted voice"

    5.4. Outputs should be encrypted voice followed by decrypted voice

    5.5. Abnormal test cases include mangling of voice, no sound, loud sound

6. Test that key is sent (integration, black-box)

    6.1. Ensure that the key we're using to encrypt the voice data is used in a dialed call

    6.2. Capture data packets, check for the key

    6.3. Example Inputs:

        6.3.1. AES Key (), Public key ()

    6.4. Output should encrypt the AES key with the private key, decrypt-able with public key.

    6.5. Abnormal test cases include sending raw AES key, all-bytes, invalid encryption, etc.

7. Test that key is received (integration, black-box)

    7.1. Ensure that the key we're using to encrypt the voice data is received in a received call

    7.2. Examine the state of the phone while receiving a call

    7.3. Example inputs:

7.3.1. Sent data over line

7.3.1.1. Should include encrypted AES

7.3.1.2. Should include RSA Public key

7.4. Output should be the correct AES key sent

7.5. Abnormal test cases include receiving the wrong AES key, using all-bytes or non-initialized bytes, etc.

8. Test that key is used (integration, white-box)

8.1. Ensure that the key we receive is used correctly.

8.2. Examine the state of the phone while receiving a call

8.3. Example inputs:

8.3.1. Sent data over line (as

8.3.2. Encrypted voice

8.4. Output should be unencrypted voice

8.5. Abnormal test cases include not-receiving the correct key, etc.

9. Test public-key generation (unit, black-box)

9.1. Ensure that public-key generation works

9.2. Generate a public key for use in transmitting session-keys

9.3. Example inputs:

9.3.1. Key generation "entropy"

9.4. Output should be a valid public/private key pair.

9.5. Abnormal test cases include a key pair which does not encrypt, decrypt.

10. Test public-key usage (integration, white-box)

10.1. Ensure that the key we're using to encrypt the AES key is used

10.2. Capture data packets, check for Public key, encrypted AES key

10.3. Example Inputs:

10.3.1. AES Key (), Public key ()

10.4. Outputs include:

10.4.1. Receiver should send public key

10.4.2. Caller should send AES key, encrypted with public key

10.5. Abnormal test cases include sending raw AES key, all-bytes, invalid encryption, encryption with incorrect key, etc.

11. Test a normal Skullduggery call. (integration, black box)

11.1. Ensure that the call is unencrypted

11.2. Hold an extended phone call (> minutes)

11.3. Example inputs:

11.3.1. Sent data over line (as

11.4. Output should include:

11.4.1. Unencrypted voice

11.4.2. Warning on encryption-enabled phone.

11.5. Abnormal test cases do not include dropped packets, cut-out voice, abrupt call-end, etc.

12. Test a normal Skullduggery call. (performance, black box)

12.1. Ensure that the call data can be heard

12.2. Hold an extended phone call (> 1 minute)

12.3. Example inputs:

12.3.1. Sent data over line (as 11)

12.4. Output should include:

12.4.1. Unencrypted voice

12.5. Abnormal test cases involve a sizable lag in processing, cut-out voice, garbled voice

13. Test voice compression algorithms (performance, black box)

13.1. Test voice compression algorithms for speed

13.2. Take a sample of known length, compress it

13.3. Example inputs:

13.3.1. Sample music clip

13.3.2. Speex algorithm

13.4. Output should include: Compressed audio, timing of algorithms

13.5. Abnormal test cases include non-compressed or poorly compressed or inflated data, slow algorithm.

Test Case Matrix

Test ID	Normal Abnormal Boundary	Black box White box	Functional Performance	Unit Integration
1	Normal	Black Box	Functional	Unit
2	Normal	White Box	Functional	Integration
3	Normal	White Box	Functional	Integration
4	Normal	White Box	Functional	Integration
5	Normal	White Box	Functional	Integration
6	Normal	White Box	Functional	Integration
7	Normal	Black Box	Functional	Integration
8	Normal	Black Box	Functional	Integration
9	Normal	White Box	Functional	Integration
10	Normal	Black Box	Functional	Unit
11	Normal	White Box	Functional	Integration
12	Normal	Black Box	Functional	Integration
13	Normal	Black Box	Performance	Integration
14	Normal	Black Box	Performance	Unit