The University of Melbourne

SWEN90010: High Integrity Software Engineering

Assignment 1

Due Date: 11:59pm, Sunday 31 March, 2019

This assignment is worth 10% of your total mark.

You will work in pairs for the assignment. Each pair will submit only one solution, produced jointly by both partners. Working in pairs is important since a significant part of this assignment involves carefully understanding a complicated system. You'll need to talk together to understand how it works.

Tip: Perhaps the most difficult part of this assignment is getting your head around the system itself. The questions are not intended to be difficult once you understand how it works, but reading through the online description might be quite challenging.

1 Background

One use of digital signatures is to sign software updates. A device or piece of software can check whether it is receiving a legitimate update from its manufacturer by checking a digital signature on the update it downloads. However, if the signing key or signing mechanism is compromised somehow, this could be a way of introducing apparently-legitimate software onto someone's computer without their knowledge. Perhaps the most famous example is the FBI-Apple case, in which the FBI demanded that Apple sign a software update [1] that would bypass the iPhone's authentication mechanisms and allow the FBI to read the data of a terrorist who had attacked people in San Bernadino, California. Apple refused, arguing that the existence of a properly-signed update of that form had a risk of being redeployed by a bad actor against innocent users.

In December 2018 the Australian government passed a law that effectively makes it illegal for a company in such a situation to refuse a similar request.

In this assignment you'll examine one software manufacturer's approach to this question.

Note: for this assignment, I'm not expressing any particular view on the political controversy (though I've expressed my opinion about the Australian law). I would just like everyone to understand how the technical approaches work.

2 System overview

This assignment asks you to examine Mozilla's binary transparency project described at `https://wiki.mozilla.org/Security/Binary_Transparency`

It uses a system a lot like Certificate Transparency Logs. This allows everyone to check that the update they've downloaded is on the public log of updates.

[1] Strictly speaking, this was a firmware update, not a software update, but the distinction isn't particularly important for this assignment.

3 Questions

1. [**(1 mark)**] An important problem in log proofs is ensuring the secure broadcast of the root hash. Explain the main idea of how Mozilla intends to solve this problem.

   Hint: Look at steps 3 and 4 of "Steps to log." That is:

   3 Obtain a certificate containing that domain name.

   4 Submit the certificate to a Certificate Transparency log.

   They don't really mean that they're obtaining a normal certificate for a real domain or public key.

2. [**(1 mark)**] So now what is the *root of trust* for this system, *i.e.* the thing you have to trust without being able to verify it based on something else?

3. [**(2 marks)**] Explain in your own words how Mozilla builds evidence that an update binary is legitimate. (You can leave out the "ephemeral public key." I'm not sure what that is either.)

4. [**(2 marks)**] Explain in your own words how Firefox verifies that an update binary is legitimate.

5. [**(2 marks)**] Read "steps to verify." Suppose your Firefox browser happened to have a bug in which it failed to do step 4: "Check that that the certificate contains the domain name." Describe how an attacker could use this bug to trick your browser into accepting a fake update. You may write either English or pseudocode.

6. [**(1 mark)**] I could be mistaken, but it seems to me that a cheating certificate authority, if acting in the role of CT log for this process, could trick Firefox into accepting a fake update by simply generating its own (fake) "update" and logging it exactly as Mozilla would. (Please tell me if you see a way this would be prevented.) Could Mozilla or its users detect this? How?

7. [**(1 mark)**] Other than detection (as described in the previous question), can you think of any way of changing the protocol so that a single cheating Certificate Authority couldn't trick a browser into accepting a fake update?

4 Submission

One of your pair should create a pdf or docx file called *your_username*`.pdf` or `.docx`, containing your joint answers to the questions. Submit it via the LMS.

5 Communication Rules

You may discuss the questions freely within your pair, and write up your joint answers together. You may also consult any other materials you find on the Internet (or in the library), as long as you give proper references in your report. You may *not* discuss this with anyone other than your project partner. In particular, cross-pair collaboration is not allowed. However, you may ask or answer any question you like on the LMS discussion board—this is up to you. You may share answers or raise interesting questions if you like, for the benefit of all. This allows ideas to be shared but mitigates the (unfair) advantage of having clever friends.

6 Late submissions

Please submit on time. It's much better to submit a not-quite-finished version on time than a perfect version late. 2 marks will be deducted each day (or part thereof) after the submission deadline. If you have a real reason for needing an extension, please ask permission in advance. I will usually ask to see some form of evidence, *e.g.* a medical certificate.

7 Academic Misconduct

The University misconduct policy applies to this assignment.

The subject staff take plagiarism very seriously. In the past, we have successfully prosecuted several students that have breached the university policy. Often this results in receiving 0 marks for the assessment, and in some cases, has resulted in failure of the subject.

A Code format rules

The layout of code has a strong influence on its readability. Readability is an important characteristic of high integrity software. As such, you are expected to have well-formatted code.

A code formatting style guide is available at `http://en.wikibooks.org/wiki/Ada_Style_Guide/Source_Code_Presentation`. You are free to adopt any guide you wish, or to use your own. However, the following your implementation must adhere to at least the following simple code format rules:

- Every Ada package must contain a comment at the top of the specification file indicating its purpose.
- Every function or procedure must contain a comment at the beginning explaining its behaviour. In particular, any assumptions should be clearly stated.
- Constants and variables must be documented.
- Variable names must be meaningful.
- Significant blocks of code must be commented.

  However, not every statement in a program needs to be commented. Just as you can write too few comments, it is possible to write too many comments.
- Program blocks appearing in if-statements, while-loops, etc. must be indented consistently. They can be indented using tabs or spaces, and can be indented any reasonable number of spaces (three is default for Ada), as long as it is done consistently.
- Lines must be no longer than 80 characters. You can use the Unix command "`wc -L *.ad*`" to check the maximum length line in your Ada source files.