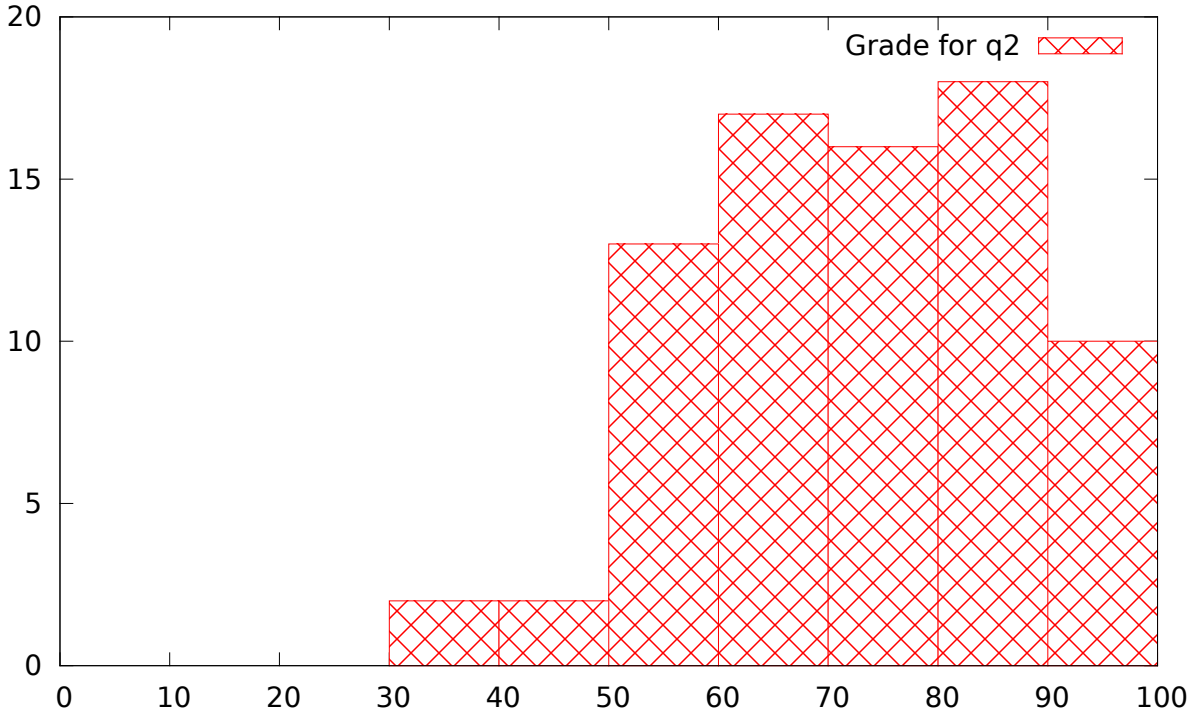
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2012

Quiz II Solutions



Histogram of grade distribution

Mean: 73.2. Stddev: 14.5.

I RSA timing attacks / Tor

1. [9 points]: Answer the following questions based on the paper "Remote Timing Attacks are Practical" by David Brumley and Dan Boneh.

(Circle True or False for each choice.)

A. True / False Removing access to a high-precision clock on the server running Apache would prevent the attack described in Brumley's paper from working.

Answer: False. The server clock is irrelevant; the client clock is what David Brumley's attack relies on for making precise timing measurements.

B. True / False Avoiding the use of the Chinese Remainder Theorem and the associated optimizations (i.e., performing computations mod p and mod q, instead of mod n) in OpenSSL would prevent the attack described in Brumley's paper from working.

Answer: True. David Brumley's attack relies on finding p and q, but these would not be exposed if the server does not perform RSA decryption mod p and mod q separately.

C. True / False David Brumley's attack, as described in the paper, would work almost as well if the neighborhood of n values was chosen as $g, g-1, g-2, \ldots, g-n$ (as opposed to $g, g+1, g+2, \ldots, g+n$ as in the paper).

Answer: True. With a very small probability, the value q being guessed will lie between $g-n$ and g, but this probability is negligibly small.

2. [8 points]:

Alyssa wants to learn the identity of a hidden service running on Tor. She plans to set up a malicious Tor OR, set up a rendezvous point on that malicious Tor OR, and send this rendezvous point's address to the introduction point of the hidden service. Then, when the hidden service connects to the malicious rendezvous point, the malicious Tor OR will record where the connection is coming from.

Will Alyssa's plan work? Why or why not?

Answer: Will not work. A new Tor circuit is constructed between the hidden service and the rendezvous point. Assuming the right precautions are taken by the hidden service (e.g., building the circuit through a sufficient number of randomly-chosen nodes, and re-building the circuit after some suitably short period of time), the rendezvous point will not be able to learn the IP address of the hidden service.

II OAuth

After reading the "Empirical Analysis of OAuth" paper, Ben Bitdiddle wants to revise the OAuth protocol to avoid some of the pitfalls of using OAuth in a real system. For each of the following proposed changes, indicate whether the change would make the protocol more secure, less secure, or the same in terms of security, by writing **MORE**, **LESS**, or **SAME**, respectively. If the change affects the security of the protocol (or makes some pitfalls more likely), give a specific attack that is possible with the change, or that is prevented by the change. The changes are not cumulative between questions.

3. [8 points]: Ben removes **r** from steps 2 and 3 of the server-flow protocol as shown in Figure 1 of the "Empirical Analysis" paper. Now, instead of matching the supplied **r** against a list of allowed URL patterns for **i**, the IdP server keeps track of the redirect URL to use for each RP (identified by parameter **i**).

Answer: More secure. An adversary cannot choose an arbitrary URL matching the RP's registered patterns (which might inadvertently redirect to other servers); instead, only one redirect URL is possible.

4. [8 points]: Ben combines steps 1, 3, and 5 to improve performance: the user enters their credentials, and the RP's Javascript code sends the credentials along with the Authz request to the IdP server.

Answer: Less secure. The RP site gets the user's credentials directly, and can misuse them in many ways (e.g., logging into the user's account at IdP and gaining all of the user's privileges).

III BitLocker

5. [10 points]: Suppose that an adversary steals a laptop protected with BitLocker in TPM mode, and wants to gain access to the data on the BitLocker-encrypted partition. The adversary discovers a buffer overflow in the BIOS code that can be exploited to execute arbitrary code by a specially-crafted USB drive plugged in during boot. How can the adversary gain access to the encrypted data? Be as specific as possible: what operations should the adversary's injected code perform, both on the main CPU and on the TPM?

Answer: After exploiting the buffer overflow and starting to run arbitrary code, the adversary should "measure" the BIOS, bootloader, and kernel as if it were booting correctly. Now the adversary should run `TPM_extend` with hashes of the legitimate code. Now the TPM's PCR registers are in the right state. Instead of booting the legitimate code, the adversary should read the sealed key from disk, unseal it with the help of the TPM (which will work because the PCR register is in the right state), and decrypt the disk using this key.

IV TrInc

Alyssa P. Hacker is building FiveSquare, a peer-to-peer application in which friends share their locations with each other (not to be confused with FourSquare, which uses a central server). Alyssa's FiveSquare application runs on mobile phones, and works by directly connecting to a friend's mobile phone over TCP.

6. [18 points]:

Alyssa is worried that users will modify the FiveSquare application to report different locations to different friends. Supposing every mobile phone had a TrInc trinket, how could Alyssa use the trinket to ensure FiveSquare users cannot pretend to be in two different locations at the same time? Assume that every FiveSquare user corresponds to a fixed ⟨trinket-id, counter-id⟩ tuple.

Specifically, what should the counter value represent? How should a user update their location, and in particular, what arguments should they pass to `Attest(counter-id, new-counter-value, message-hash)`? How should a user check a friend's location, and in particular, what arguments should the friend pass to `Attest(..)`?

Answer: Pick some minimum time between updates, D = 10 minutes.

Use the counter as a timestamp. To report a location, let T be the current time. To not leak the time of the last update, first advance the counter to T - D and discard the attestation. The location update is

`location | Attest(counter-id, T, hash(location))`

To accept a location update, `location | <I,i,c,c',h>_K`, the following must be true:

– attestation must be a valid attestation. signature is good, key is good, etc.

– hash(location) == h

– `|c - current_time| < minimum_clock_skew`

– `c - c' >= D`

D is needed to prevent an attacker from incrementing the counter by 1ns and sending a new update. 2 * minimum_clock_skew should be less than D.

To verify whether a location update is fresh, a user can send a nonce to the friend and ask the friend to run `Attest(counter-id, T, nonce)`

(Another solution could be to use the counter as an actual counter and include the chain of previous attestations to verify the timestamps don't rewind. But that leaks hashes of all previous locations. To avoid that, attest to `hash(location | random)`. Still leaks when previous updates were.)

V Android

You are implementing the Koi Phone Agent for Android, based on the papers from lecture. You implement interaction between the application and the Koi phone agent using intents. Refer to Figure 1 in the Koi paper in the following questions.

7. [8 points]: What Android permissions does the Koi phone agent have to request access to in its manifest? Where are these permissions defined (i.e., whether these permission strings are dangerous, etc)?

Answer: Location (GPS), network (INTERNET). Defined by Android platform.

8. [8 points]: What permissions does the application have to request access to in its manifest? Where are these permissions defined (i.e., whether these permission strings are dangerous, etc)?

Answer: Permission to talk to the Koi agent. Defined by the Koi agent.

9. [8 points]: How should the Koi agent, together with the phone's user, ensure that the only applications that can access to the Koi agent are the ones that the user trusts?

Answer: The Koi agent should mark the permission for talking to the Koi agent as dangerous, and the user should grant this permission only to applications that he trusts.

VI Zoobar

Alyssa is reviewing Ben's lab 6 code for sandboxing Javascript, and observes that she can invoke arbitrary code using the following Javascript code (which goes through Ben's sandbox rewriter):

```
var code = 'alert(5)';  // or any other code that will escape the sandbox
var a = function() { return '__proto__'; };
var b = -5;
var c = { toString: function() {
            b++;
            if (b > 0) { return 'constructor'; } else { return 'eval'; }
          }
        };
var d = a[c];
var e = d(code);
e();
```

10. [4 points]: What did Ben miss in his sandbox implementation?

Answer: His `bracket_check` function doesn't (reliably!) stringify the key once before doing checks. So `c` ends up returning a safe attribute ("eval") for each check and then an unsafe one when actually accessing.

11. [8 points]: Propose a fix for Ben's problem; be as specific as possible (i.e., code is best).

Answer:

```
var old_bracket_check = bracket_check;
bracket_check = function(s) {
return old_bracket_check(String(s));
}
```

VII 6.858

We'd like to hear your opinions about 6.858. Any answer, except no answer, will receive full credit.

12. [2 points]: What was your favorite paper, which we should definitely keep in future years?

Answer: Tor (x23). Timing attacks (x14). Click trajectories (x11). Android (x9). BitLocker (x9). TrInc (x6). OAuth (x5). Browser security handbook (x3). Kerberos (x3). Koi (x3). Native Client (x2). OWASP-top-10 (x2). Baggy / buffer overflows (x2). OKWS. Password replacement. iSEC guest lecture. FBJS. Backtracker. Capsicum.

13. [2 points]: What was your least favorite paper, which we should drop in future years?

Answer: Koi (not a real system) (x15). OAuth (bad paper) (x13). TrInc (solves irrelevant problem, not a real system) (x12). Backtracker (x7). Click trajectories (boring paper, interesting topic) (x6). Static analysis (too formal/confusing) (x5). Capsicum (redundant with OKWS) (x4). Tor (x3). BitLocker (unclear paper, good topic) (x3). Timing attacks (crypto-heavy) (x3). Android (bad paper, good topic) (x2). Password replacement (although lecture discussion was good) (x2). Kerberos (too broken). OKWS. Javascript subsets. ForceHTTPS. Browser security handbook. Native Client.

14. [2 points]: What topic did 6.858 not cover, but you think would be a good fit in future years?

Answer: Network / Internet security (sniffing, DNS, DoS, firewalls, packet inspection) (x9). Crypto basics (x7). Social engineering / phishing / physical security (lock-picking) (x7). More Android, iOS security (e.g., lab) (x6). More timing / side channel (e.g., lab) (x5). Real-world attacks, recent incidents (x5). OS/kernel exploits/security (x3). BitCoin, payment systems (x3). Hardware security (x2). More exploiting in labs (x2). Botnets (x2). Labs that simulate real-world security attacks (x2). Wireless security (802.11, bluetooth, RFID, NFC) (x2). SQL injection in a lab (x2). Homomorphic encryption (x2). Honeypots (x2). Static analysis: Singularity (x2). Anti-cheating. UI security. Decentralized authentication schemes. Practical escaping. Non-Linux stuff. More general topics, less research papers. Game console security. Virtualization. Background material for papers. Fuzzing, real-world penetration testing tools. Real-world SSL attacks (e.g., from guest lecture). ASLR and similar techniques. Research being done by course staff. Information leaks. CryptDB. Java security. Sybil attacks. How to protect your own laptop / phone / etc? Database security. Antivirus. Resin.

End of Quiz