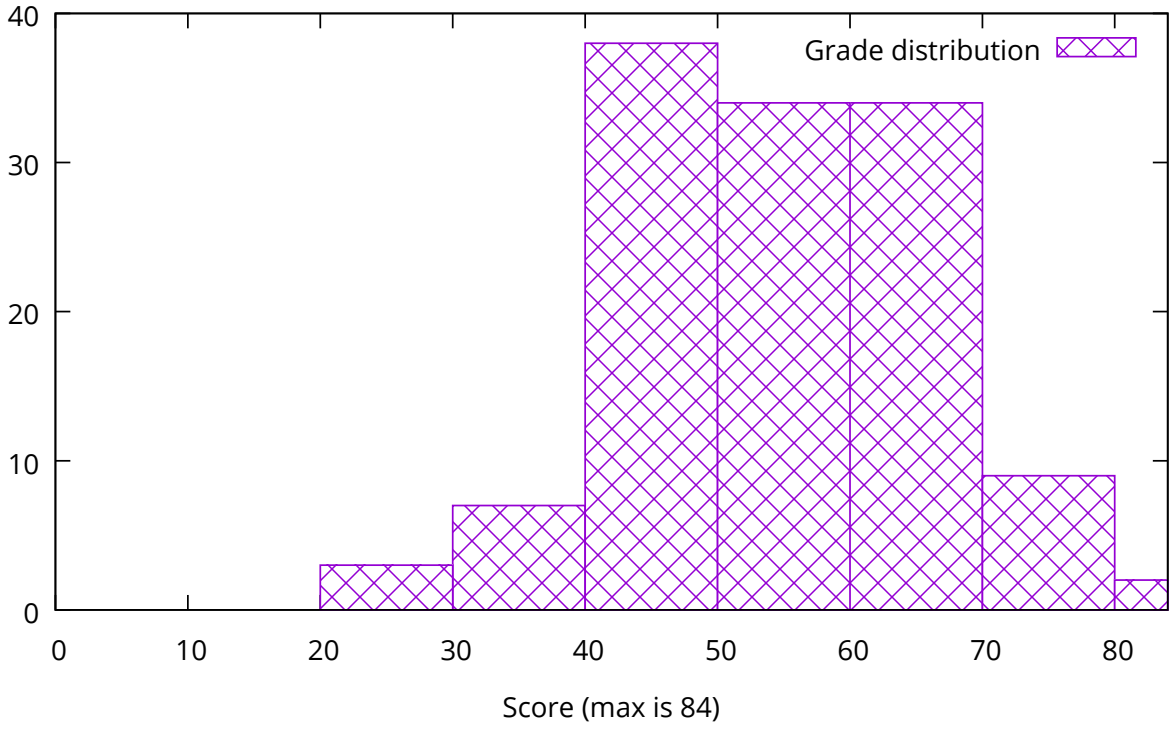
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2015

Quiz I Solutions



Histogram of grade distribution

Mean 55.4, Stddev 11.3

I Lab 1: buffer overflow

Consider the following code similar to `http_serve()` from Lab 1, where an adversary can supply arbitrary input in the string `name`. Throughout this part, assume a 32-bit x86 system like the Lab VM, and assume no compiler optimization or ASLR (Address Space Layout Randomization).

```
void http_serve(int fd, const char *name) {
    void (*handler)(int, const char *) = http_serve_none;
    char pn[1024];
    struct stat st;

    getcwd(pn, sizeof(pn));
    strcat(pn, name);
    if (stat(pn, &st) == 0 && S_ISREG(st.st_mode))
        handler = http_serve_file;
    handler(fd, pn);
}
```

1. [6 points]: The figure on the page after the next page shows the stack layout and register values right after `http_serve()` invokes `handler`, but before any instruction in `handler` is executed. Fill in the first column (Q1) next to the figure with letters "a" to "f" from the following options. Each letter indicates the stack content at the corresponding slot. **Note that not every option will be used, and some options may be used more than once.**

(a) address of `pn`

(b) `fd`

(c) `name`

(d) `handler`

(e) return address for `http_serve`

(f) return address for `handler`

2. [4 points]: Ben Bitdiddle notices that there is a vulnerability in the above code that enables a return-to-libc attack. In particular, the attacker can overflow buffer `pn[]`, and trick the server into calling the library function `unlink("grades.txt")` when `http_serve` returns. Please show how to mount this attack by smashing the relevant stack slots. Mark your solution on the second column in the figure (Q2), using the following notation:

- Write "**unlink**" if the slot should be replaced with `unlink()`'s address.
- Write "**argument**" if the slot should be replaced with the address of the string `"grades.txt"`.
- Write "**preserve**" if when overwritten, the slot must keep its original value.
- Write "**n/a**" if the slot cannot be overwritten.
- Write "**any**" if the slot will be overwritten, but it does not matter what goes in it.

3. [4 points]: Louis Reasoner argues that Ben's exploit will not work if the server uses a stack canary. A stack canary is a value that the compiler pushes on stack at function entry, and pops and checks before return. He figures out a way to delete `grades.txt` even in the presence of a stack canary. Louis' attack overflows `pn[]`, but in a different way than Ben's attack. Please briefly describe how should Louis overwrite the stack in this attack.

Answer: Louis can overwrite the handler pointer on the stack, and replace it with the address of `unlink`. He also has to pass the argument to `unlink` by overwriting `fd` on the stack with the address to the string `"grades.txt"`

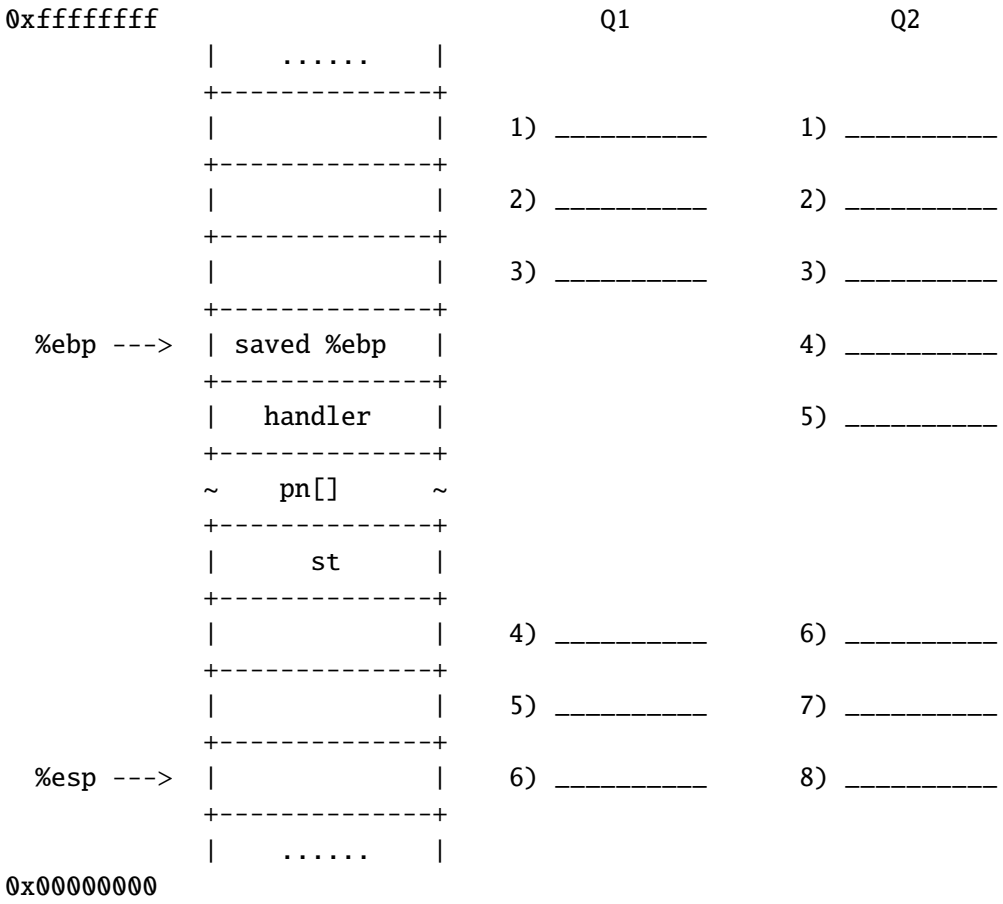
Alternatively, assuming the stack is executable, Louis can also place a shell code that performs `unlink("grades.txt")` in the buffer `pn`, and overwrite `handler` to point to the shell code.

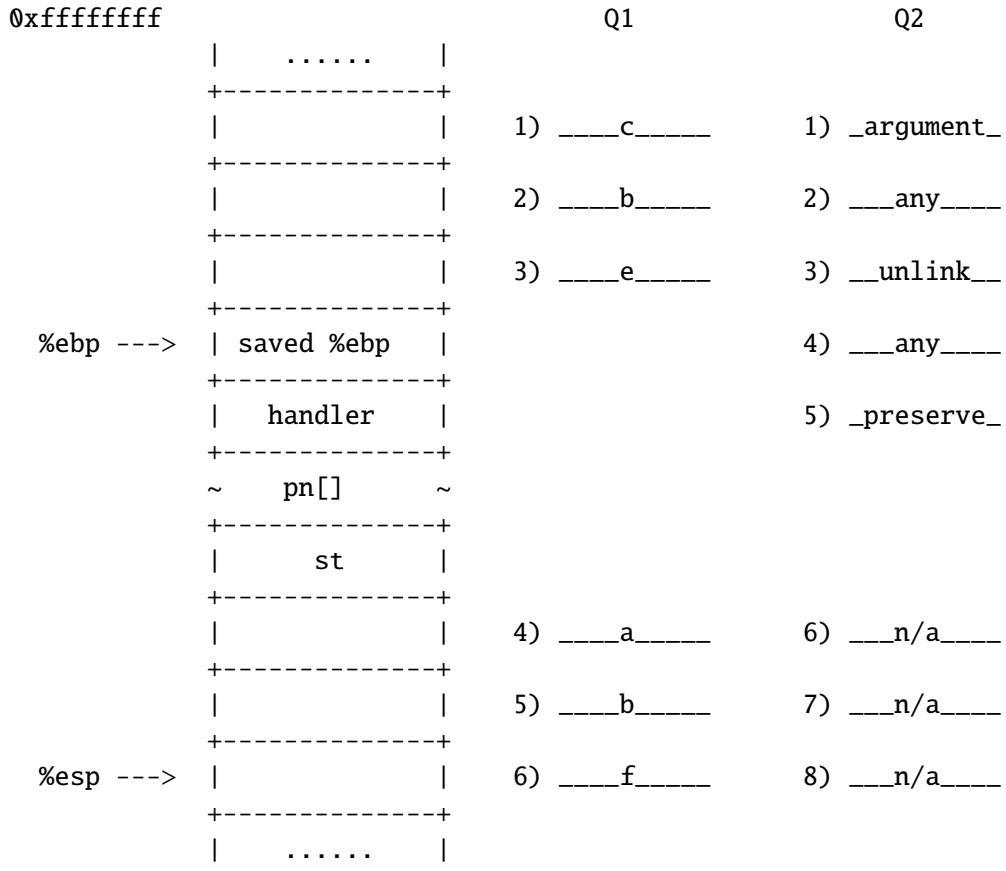
* Options for Q1:

    (a) address of pn   (b) fd   (c) name   (d) handler
    (e) return address for http_serve   (f) return address for handler

* Options for Q2:

    unlink   argument   preserve   n/a   any





Answer: 0x00000000

II User authentication

The paper *The Quest to replace passwords* compares many user authentication mechanisms, ranking them on Usability, Deployability, and Security.

4. [4 points]: Consider a smartphone-based two-factor scheme that uses text messages as the second factor (e.g., Google's two factor login or MIT's Duo). Compare regular passwords to smartphone-based two-factor authentication in terms of usability, deployability, and security. (Briefly explain your answer.)

Answer: Passwords win on usability (you can login when you forgot your phone) and deployability (no smart-phone needed), but weaker security (attacker needs more than the password)

5. [4 points]: Describe an attack that smartphone-based two factor authentication is vulnerable to, but RSA SecurID is less vulnerable to.

Answer: Malware is an issue for smart-phone 2FA but less so for RSA SecurID. Interception of the SMS message with the one-time token.

III OKWS

The Bank of Barton provides a web site for their customers to transfer money to each other. The web site runs on a single server and uses OKWS (see *Building Secure High-Performance Web Services with OKWS*, by Maxwell Krohn).

The bank's original design has two OKWS service processes. The login service handles customer login: it checks the customer's password and issues a session cookie. The banking service handles balance requests and transfers. Both service processes talk to a database proxy via RPC. Each service has its own authentication token which it sends with every RPC so the proxy knows which service is talking to it.

The database proxy accepts the following RPCs:

```
CheckPassword(token, username, password) -> ok, session-cookie
CheckCookie(token, session-cookie) -> ok, username
GetBalance(token, username) -> amount
SetBalance(token, username, amount)
```

The proxy uses the token to ensure that only the login service can use the `CheckPassword` RPC, and that only the banking service can call `GetBalance` and `SetBalance`. The proxy doesn't enforce any other restrictions.

6. [4 points]: Does the separation between login and banking services significantly improve security over a scheme in which they are in the same process? If yes, describe an attack that it prevents; if no, explain why not.

Answer: Yes: a bug in login doesn't immediately help steal money, and a bug in banking doesn't help you steal passwords. However, can also argue that if login is exploited, attacker can sniff usernames and passwords.

The bank is considering changing their design so that each bank branch has a separate OKWS service process, and a separate authentication token. Each customer is associated with a single branch, and the web site handles all of a customer's balance/transfer requests in that branch's service. The web site needs to support bank transfers between customers at different branches.

7. [6 points]: It turns out that the new design won't help security unless the bank also changes the proxy RPC interface and/or the restrictions the proxy enforces. Describe a new proxy RPC interface and/or set of restrictions that would cause the service-per-branch design to significantly improve security, and outline an attack which your change prevents.

Answer: Have a separate token per branch. The database proxy enforces that a branch can only perform RPCs as users belonging to same branch. Replace `SetBalance` with `Transfer` to avoid branches needing to know each-other's tokens. Database proxy must also enforce that transferred amount is not negative. If a branch is compromised, can only give away money.

IV Lab 2: privilege separation

Ben Bitdiddle has been working on a secure email server but is having trouble configuring the proper permissions, uids, gids, and supplementary groups. The mail server is designed as follows:

The system is managed by a daemon called `postld` which is analogous to zookld; it starts other daemons and restarts them if they die. Authentication is managed by the `auth` service as in the lab.

All incoming mail is received by the `smtpd` daemon, on port 25 over TCP. When a client connects to `smtpd`, it can optionally authenticate as a local user. This means that `smtpd` must be able to check user credentials with the `auth` server. When `smtpd` receives a mail message, it handles it in one of two ways depending on the destination address:

A local user: it forwards the message to the `mda` (mail delivery agent) service for local delivery.

A user on a remote system: it forwards the message to the `relayd` service for remote delivery *if and only if the client has authenticated as a local user* (to avoid relaying spam from untrusted users). This is why `smtpd` must be able to authenticate users.

The following processes are involved:

Process	User	Group	Description
`postld`	root	root	the launch daemon
`smtpd`	smtpd	smtpd	receives mail from (remote) clients over smtp and forwards them to the appropriate service if applicable.
`relayd`	relayd	relayd	receives mail from smtpd and relays it to other mail servers.
`mda`	mda	mda	receives mail from smtpd and delivers it to local users' inboxes.
`auth`	auth	auth	handles authentication.

The directory set up is as follows:

```
/
+-- jail/
    +-- authsvc/
    |   +-- sock    auth server socket
    +-- relaydsvc/
    |   +-- sock    relayd server socket
    +-- mda/
    |   +-- sock    mda socket
    +-- creds/      credentials database directory (format unspecified).
    +-- queue/      the relayd's outbound mail queue (format unspecified).
    +-- mail/       user inboxes
        +-- user1/  user1's inbox
        +-- user2/  user2's inbox
```

8. [8 points]: Please fill in the permissions below such that services have the absolute least permissions necessary to function. There may be multiple correct solutions. Note: A user must have read/write permissions to a socket to connect to it.

Directory	user	group	permissions (mode)
+-- jail/	root	root	0755
+-- authsvc/	auth		
| +-- sock	auth		
+-- relaydsvc/	relayd		
| +-- sock	relayd		
+-- mda/	mda		
| +-- sock	mda		
+-- creds/	auth		
+-- queue/	relayd		
+-- mail/	root	root	0755
+-- user1/	user1		
+-- user2/	user2		

Answer: Slightly weaker permissions are allowed, but this is the answer.

Directory	user	group	permissions (mode)
+-- jail/	root	root	0755
+-- authsvc/	auth	smtpd	0710
| +-- sock	auth	smtpd	0660
+-- relaydsvc/	relayd	smtpd	0710
| +-- sock	relayd	smtpd	0660
+-- mda/	mda	smtpd	0710
| +-- sock	mda	smtpd	0660
+-- creds/	auth	auth	0700
+-- queue/	relayd	relayd	0700
+-- mail/	root	root	0755
+-- user1/	user1	mda	0770
+-- user2/	user2	mda	0770

V Capsicum

You have a job at a hedge fund evaluating portfolios: deciding which collections of stocks will likely yield the highest returns. On the Web you find a nifty open-source electronic portfolio optimization program called `epopt`. The program's documentation says that it reads a file containing the amounts of each stock in a portfolio, computes a better portfolio, and then overwrites the file with the new portfolio. The documentation says that the only input it needs is the portfolio file. `epopt` expects the file on the command line, like this:

```
epopt /usr/rtm/my-portfolio
```

The code for `epopt` is millions of lines, and you don't have time to inspect it all carefully. You are thinking of using Capsicum to improve the security of `epopt` (see the paper *Capsicum: practical capabilities for UNIX* by Watson *et al.*).

For each of the following scenarios, indicate whether Capsicum can be used to protect against it. If yes, briefly outline how; if no, briefly explain why not.

9. [3 points]: Can Capsicum prevent `epopt` from sending a copy of the input portfolio file to your competitor over the Internet?

Answer: Yes. Don't give capability for accessing the network.

10. [3 points]: Can Capsicum prevent `epopt` from modifying your portfolio file to contain a lower-quality set of stocks?

Answer: No. Capabilities are for file-descriptor like things.

11. [3 points]: Can Capsicum prevent `epopt` from writing a copy of your portfolio into the file `/tmp/.hidden`?

Answer: Yes. Don't give capability to access tmp directory

12. [3 points]: Can Capsicum prevent an exploit in which you download a portfolio file from somewhere on the web, and give it to `epopt` as input, but the file contains cleverly-crafted contents that trigger a buffer overflow bug in `epopt`?

Answer: No. Capsicum may make the application hard to exploit, but you were not required to say this.

VI Native Client

Native Client runs untrusted assembly code inside a sandbox using software fault isolation. The sandboxed module is located in the address range 0 to 256 Mbyte. Through a trampoline, the sandboxed module can call into a service runtime, which is located above the address 256 Mbyte. The service runtime is trusted and provides functions for I/O etc. Two of the service functions available to the sandboxed module are `mmap(dst, src, len)` and `munmap(addr, len)`. `mmap` allocates `len` bytes of memory and maps the memory into the virtual address range specified by `dst` and `len`, first initializing it by copying `len` bytes starting from `src`. `munmap` allows the module to remove the specified address range from the process' address space. Native client checks that the module doesn't `mmap` memory outside of its sandbox.

13. [6 points]: How can an attacker use `mmap` and `munmap` to break out of a module's sandbox? (Outline an attack in enough detail so that we can decide if it works. Several attacks possible; you have to give only one.)

Answer: Make a module that jumps to some address *v* that that meets the requirements of the verifier (e.g., 64-byte aligned etc.). Store unvalidated instructions into some buffer. Before jumping to *v* ask the service runtime to unmap the region at *v*, and then mmap the buffer with unvalidated instructions to the address *v* inside the inner sandbox. Then, jmp to *v*, which will execute the unvalidated instructions in the buffer. (See Ben Hawkes exploiting native client).

VII Intel SGX

Ben develops a password manager for a bank using SGX's secure attestation features. The bank distributes the password manager to its customers, and they install it on their SGX-enabled computers. When a customer logs into the bank, the bank verifies that Ben's code is running inside the enclave before sending any secret to the enclave.

14. [6 points]: The string library that runs inside the enclave and is used by code inside the enclave turns out to have a buffer overrun exploit. Ben's friend Sam says that SGX is so powerful that exploiting buffer overflows is not possible and that Ben doesn't have to worry about them. Explain why Sam is wrong. (Give a scenario in which an attacker can exploit a buffer overrun and explain why attestation doesn't prevent this attack.)

Answer: If the enclave software takes input from the user at any point (which it is likely to do), for example through a `read` system call, an attacker can supply a very long, malicious input, exploiting the vulnerable code through a buffer overflow. SGX does not do anything to prevent this. Attestation does not prevent the attack, since it only verifies that the initial state of the enclave matches what the remote party expects. If the application contains bugs that makes the application behave differently from what the programmer *intended*, this is not detected by SGX (i.e., SGX does not verify that the given code is correct). The reason Ben should be worried is that the attacker now controls the behavior of the application, but the bank still trusts the application; an attacker could exploit this trust and, for example, leak the user's passwords.

VIII Symbolic Execution

Recall from the EXE paper (*EXE: Automatically Generating Inputs of Death*) that, under certain circumstances, EXE forks execution in order to explore different paths (see the EXE paper's Section 2). For each of the two functions below, how many forks would each cause in total? The argument x is symbolic. Assume that the compiler does not optimize the code, and that EXE has enough time to follow all paths.

```
int
f1(int x)
{
  int z = 0;
  int i;
  for(i = 0; i < 10; i++){
    if(x == i){
      z += 1;
      break;
    }
  }
  return z;
}
```

Note the `break` inside the if statement, which causes the loop to terminate if the if condition is true.

15. [4 points]: How many forks for `f1()` above?

Answer: 10. Each if statement causes a fork; one branch continues the loop (and forks more), the other does not.

```
int
f2(int x[10])
{
  int z = 0;
  int i;
  for(i = 0; i < 10; i++){
    if(x[i] == 1){
      z += 1;
    }
  }
  return z;
}
```

Note that `x` is an array of symbolic integers, and that there is no `break` inside the if statement.

16. [4 points]: How many forks for `f2()` above?

Answer: 1023. First iteration forks once, second forks twice (once for each previous fork), third forks four times, and *i*th forks 2^{i-1} times. Since there are ten iterations, total number of forks is $2^0 + 2^1 + \ldots + 2^9$, which is the same as a number with the first ten bits set, which is 1023. There will be 1024 processes if you count the original.

Suppose you have a server with a secret (in `theSecret`) that it should only reveal under certain circumstances. Your server contains this code that calls `authCheck()` to decide if the client input implies that it's OK to reveal the secret, and (if yes) it calls `emit()` to send the secret to the client.

```
if(authCheck(input))
  emit(theSecret);
```

The `input` variable holds a complete HTTP request (URL, cookies, and other HTTP headers). You are worried that there might be logical bugs in `authCheck()` that cause it to return true in situations where theSecret should not be revealed.

17. [4 points]: Outline how you could use EXE to help you gain confidence that `authCheck()` works correctly.

Answer: Puts some manual asserts in the code and run exe to see if the asserts are triggered. These asserts can either be in `authCheck` itself, or you could place an assert that checks that input is indeed a valid authentication request inside the if branch, and inspect the counter-example produced by EXE.

IX Android security

Consider the Friend application described in the paper *Understanding Android Security*. The FriendTracker application defines the permissions READ_FRIENDS and WRITE_FRIENDS, which are used to limit access to the FriendProvider component in the FriendTracker application.

18. [4 points]: Does the Android security system ensure that no other application than the FriendViewer application can read from the FriendProvider component? (Briefly explain your answer.)

Answer: No. Any application that the user installs can ask for that permission, and the user may give the permission to that application.

X 6.858

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

19. [2 points]: Are there things you'd like to see improved in the second half of the semester?

Answer: 9x Better Q&A in lectures, 8x Reading questions due later, 7x More attack-oriented topics, 7x Show real attacks, 7x More current security research, 7x Better practice Qs, 6x More guest lectures, 6x Grades sooner, 4x Lab 4 is cool!, 4x Lab 1 is too hard (gdb, gcc, etc), 4x Lectures are too similar to reading, 4x More background information, 3x Fewer papers, 3x Labs too easy, 3x More recitations, 3x More OH, 3x Better lab documentation, 3x Cover labs in lecture, 2x 1-190 needs loudspeakers, 2x Labs more based on papers, 2x Video recordings of lectures, 2x Cover papers better during lecture, 2x Better lecture notes,

20. [2 points]: Is there one paper out of the ones we have covered so far in 6.858 that you think we should definitely remove next year? If not, feel free to say that.

Answer: 26x Haven, 16x SGX, 11x Ur/Web, 10x Capsicum, 8x Tangled Web, 8x Password Alternatives, 5x Android, 3x EXE, 2x NaCl, 2x Confused Deputy, 2x Baggy Bounds, 1x OKWS

End of Quiz