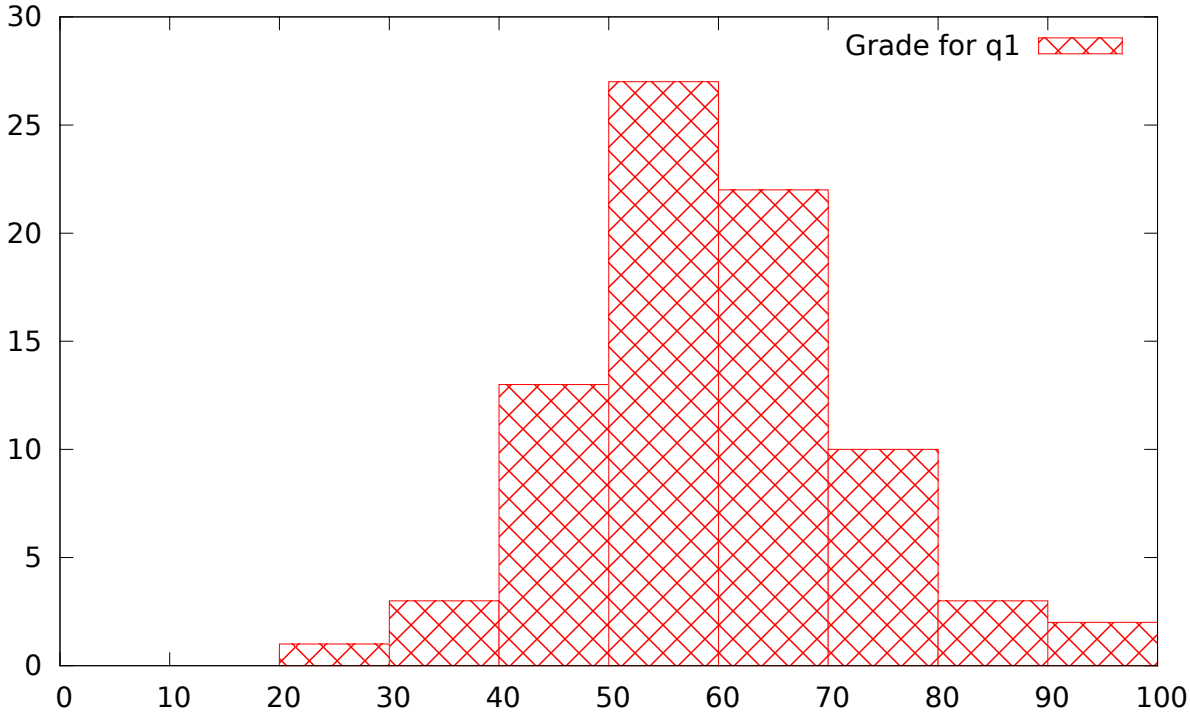
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2012

Quiz I Solutions



Histogram of grade distribution

I Buffer overflows

Consider the following C program, where an adversary can supply arbitrary input on `stdin`. Assume no compiler optimizations, and assume a 32-bit system. In this example, `fgets()` never writes past the end of the 256-byte `buf`, and always makes it NULL-terminated.

```
int main() {
  char buf[256];
  fgets(buf, 256, stdin);
  foo(buf);
  printf("Hello world.\n");
}
```

1. [12 points]: Suppose the `foo` function is as follows:

```
void foo(char *buf) {
  char tmp[200];  // assume compiler places "tmp" on the stack

  // copy from buf to tmp
  int i = 0;      // assume compiler places "i" in a register
  while (buf[i] != 0) {
    tmp[i] = buf[i];
    i++;
  }
}
```

Which of the following are true? Assume there is an unmapped page both above and below the stack in the process's virtual memory.

(Circle True or False for each choice.)

A. True / False An adversary can trick the program to delete files on a system where the stack grows down.

Answer: True. The adversary can overwrite `foo`'s return address.

B. True / False An adversary can trick the program to delete files on a system where the stack grows up.

Answer: False. If the stack grows up, then no other state is placed above `tmp` on the stack, so even if the adversary overflows `tmp`, it will not affect the program's execution.

C. True / False An adversary can trick the program to delete files on a system using Baggy bounds checking with `slot_size=16`. (Stack grows down.)

Answer: False. Baggy will prevent memory writes to `tmp` from overflowing to the return address or any other stack variable.

D. True / False An adversary can prevent the program from printing "Hello world" on a system using Baggy bounds checking with `slot_size=16`. (Stack grows down.)

Answer: False. The argument `buf` points to a string that is at most 255 bytes long, since `fgets` NULL-terminates the buffer. Baggy enforces a power-of-2 allocation bound for `tmp`, which ends up being 256 bytes.

E. True / False An adversary can trick the program to delete files on a system using terminator stack canaries for return addresses. (Stack grows down.)

Answer: False. A terminator stack canary includes a NULL byte, and if the adversary overwrites the return address on the stack, the canary value will necessarily not contain any NULL bytes (since otherwise the `while` loop would have exited).

F. True / False An adversary can prevent the program from printing "Hello world" on a system using terminator stack canaries for return addresses. (Stack grows down.)

Answer: True. The adversary could simply overwrite the canary value, which will terminate the program as `foo` returns.

2. [8 points]: Suppose the `foo` function is as follows:

```
struct request {
  void (*f)(void);  // function pointer
  char path[240];
};

void foo(char *buf) {
  struct request r;
  r.f = /* some legitimate function */;
  strcpy(r.path, buf);
  r.f();
}
```

Which of the following are true?

(Circle True or False for each choice.)

A. True / False An adversary can trick the program to delete files on a system where the stack grows down.

Answer: True. The adversary can overwrite `foo`'s return address on the stack.

B. True / False An adversary can trick the program to delete files on a system where the stack grows up.

Answer: True. The adversary can overwrite `strcpy`'s return address on the stack.

C. True / False An adversary can trick the program to delete files on a system using Baggy bounds checking with `slot_size=16`. Assume `strcpy` is compiled with Baggy. (Stack grows down.)

Answer: False. Baggy bounds checking will prevent `strcpy` from going past `r`'s allocation bounds, and `r.f` is before `r.path` in `r`'s memory layout.

D. True / False An adversary can prevent the program from printing "Hello world" on a system using Baggy bounds checking with `slot_size=16`. Assume `strcpy` is compiled with Baggy. (Stack grows down.)

Answer: True. If the adversary supplies 255 bytes of input, then `strcpy` will write past `r`'s allocation bounds of 256 bytes, and Baggy will terminate the program.

II OS sandboxing

Ben Bitdiddle is modifying OKWS to use Capsicum. To start each service, Ben's `okld` forks, opens the service executable binary, then calls `cap_enter()` to enter capability mode in that process, and finally executes the service binary. Each service gets file descriptors only for sockets connected to `okd`, and for TCP connections to the relevant database proxies.

3. [6 points]: Which of the following changes are safe now that the services are running under Capsicum, assuming the kernel implements Capsicum perfectly and has no other bugs?

(Circle True or False for each choice.)

A. True / False It is safe to run all services with the same UID/GID.

Answer: True.

B. True / False It is safe to run services without `chroot`.

Answer: True.

C. True / False It is safe to also give each service an open file descriptor for a per-service directory `/cores/servicename`.

Answer: True.

Ben also considers replacing the `oklogd` component with a single log file, and giving each service a file descriptor to write to the log file.

4. [5 points]: What should `okld` do to ensure one service cannot read or overwrite log entries from another service? Be as specific as possible.

Answer: `okld` should call:

```
lc_limitfd(logfd, CAP_WRITE);
```

To ensure that the service cannot seek, truncate, or read the log file.

5. [5 points]: What advantages could an `oklogd`-based design have over giving each service a file descriptor to the log file?

Answer: `oklogd` can enforce structure on the log file, such as adding a timestamp to each record, ensuring each record is separated from other records by a newline, ensuring multiple records are not interleaved, etc.

III Network protocols

Ben Bitdiddle is designing a file server that clients connect to over the network, and is considering using either Kerberos (as described in the paper) or SSL/TLS (without client certificates, where users authenticate using passwords) for protecting a client's network connection to the file server. For this question, assume users choose hard-to-guess passwords.

6. [6 points]: Would Ben's system remain secure if an adversary learns the server's private key, but that adversary controls only a single machine (on the adversary's own home network), and does not collude with anyone else? Discuss both for Kerberos and for SSL/TLS.

Answer: With Kerberos, no: the adversary can impersonate any user to this server, by constructing any ticket using the server's private key.

With SSL, yes: the server can impersonate the server to another client, but no clients will connect to the adversary's fake server.

7. [6 points]: Suppose an adversary learns the server's private key as above, and the adversary also controls some network routers. Ben learns of this before the adversary has a chance to take any action. How can Ben prevent the adversary from mounting attacks that take advantage of the server's private key (e.g., not a denial-of-service attack), and when will the system be secure? Discuss both for Kerberos and for SSL/TLS.

Answer: With Kerberos, Ben should change the server's private key. The system will be secure from that point forward. If the adversary was recording network traffic from before the attack, Ben should also make sure he changes the server's private key over a secure network link, because `kpasswd` does not provide forward secrecy. The adversary may be able to decrypt network traffic to the file server before the key is changed.

With SSL, Ben should obtain a new SSL certificate for the server, with a new secret key, but the adversary can continue to impersonate Ben's file server until the compromised certificate expires. The system will only be secure once the certificate expires, or once all clients learn of the certificate being revoked.

IV Static analysis

Would Yichen Xie's PHP static analysis tool for SQL injection bugs, as described in the paper, flag a potential error/warning in the following short but complete PHP applications?

8. [10 points]:

A. True / False The tool would report a potential error/warning in the following code:

```
function q($s) {
  return mysql_query($s);
}

$x = $_GET['id'];
q("SELECT .. $x");
```

Answer: True. The summary for `q()` indicates that the argument must be sanitized on entry, and the main function does not sanitize the argument.

B. True / False The tool would report a potential error/warning in the following code:

```
function my_validate() {
  return isnumeric($_GET['id']);
}

$x = $_GET['id'];
if (my_validate()) {
  mysql_query("SELECT .. $x");
}
```

Answer: False. The summary for `my_validate()` indicates that `$_GET[id]` is sanitized if the return value is true.

9. [10 points]:

A. True / False The tool would report a potential error/warning in the following code:

```
mysql_query("SELECT .. $n");
```

Answer: True. The tool reports warnings when any variable must be sanitized on entry into the main function, and the variable is not known to be easily controlled by the user, such as `$_GET` and `$_POST`.

B. True / False The tool would report a potential error/warning in the following code:

```
function check_arg($n) {
  $v = $_GET[$n];
  return isnumeric($v);
}

$x = $_GET['id'];
if (check_arg('id')) {
  mysql_query("SELECT .. $x");
}
```

Answer: True. The summary for `check_arg()` indicates that `$_GET[`$\perp$`]` is sanitized if the return value is true, but the call to `mysql_query()` requires `$_GET[id]` to be sanitized.

V Runtime instrumentation

10. [10 points]:

Consider the following Javascript code:

```
function foo(x, y) {
  return x + y;
}

var a = 2;
eval("foo(a, a)");

var p_c = {
  k: 5,
  f: function() { return a + this.k; }
};
var kk = 'k';
p_c[kk] = 6;
p_c.f();
```

Based on the description in the paper by Sergio Maffeis et al, and based on lecture 9, what will be the FBJS rewritten version of this code, assuming the application-specific prefix is `p_`?

Answer: FBJS adds a `p_` prefix to every variable name, and wraps `this` and variable array indexes in `$FBJS.ref()` and `$FBJS.idx()` respectively.

```
function p_foo(p_x, p_y) {
  return p_x + p_y;
}

var p_a = 2;
p_eval("foo(a, a)");

var p_p_c = {
  k: 5,
  f: function() { return p_a + $FBJS.ref(this).k; }
};
var p_kk = 'k';
p_p_c[$FBJS.idx(p_kk)] = 6;
p_p_c.f();
```

VI Browser security

Ben Bitdiddle is taking 6.858. Once he's done with his lab at 4:55pm, he rushes to submit it by going to `https://taesoo.scripts.mit.edu/submit/handin.py/student`, selecting his `lab`*N*`-handin.tar.gz` file, and clicking "Submit". The 6.858 submission web site also allows a student to download a copy of their past submission.

For your reference, when the user logs in, the submission web site stores a cookie in the user's browser to keep track of their user name. To prevent a user from constructing their own cookie, or arbitrarily changing the value of an existing cookie, the server includes a signature/MAC of the cookie's value in the cookie, and checks the signature when a new request comes in. Finally, users can log out by clicking on the "Logout" link, `https://taesoo.scripts.mit.edu/submit/handin.py/logout`, which clears the cookie.

Alyssa P. Hacker, an enterprising 6.858 student, doesn't want to do lab 5, and wants to get a copy of Ben's upcoming lab 5 submission instead. Alyssa has her own web site at `https://alyssa.scripts.mit.edu/`, and can convince Ben to visit that site at any point.

11. [16 points]: How can Alyssa get a copy of Ben's lab 5 submission?

Alyssa's attack should rely only on the Same-Origin Policy. Assume there are no bugs in any software, Ben's (and Taesoo's) password is unguessable, the cookie signature scheme is secure, etc.

Answer: Alyssa's web site should force Ben's browser to log out from the 6.858 submission web site, by inserting the following tag:

```
<IMG SRC="https://taesoo.scripts.mit.edu/submit/handin.py/logout">
```

and then set a cookie for `domain=scripts.mit.edu` containing Alyssa's own cookie. When Ben visits the submission web site to upload his lab 5, he will actually end up uploading it under Alyssa's username, allowing Alyssa to then download it at 4:56pm.

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]: This year we started using Piazza for questions and feedback. Did you find it useful, and how could it be improved?

Answer: Generally good; UI not so great; appreciate anonymity. Would be good if answers show up quicker. Bypass email preferences for important announcements. In-person office hours are also important. Submit paper questions via Piazza. Signal-to-noise ratio too low. More TA/professor participation other than David. Ask people not to be anonymous. Separate login is annoying. RSS feed.

13. [2 points]: What aspects of the labs were most time-consuming? How can we make them less tedious?

Answer: Include more debugging tools, especially for the PyPy sandbox. Explain where errors / debug output goes. Explanation of provided lab code; explain what parts to focus on. Start discussions of papers. Avoid asking for the same thing multiple times in the lab; 2nd part of lab 2 was repetitive. Speed up the VM / run Python fewer times. More office hours. Better / more fine-grained / faster make check. Review/recitation session to provide background knowledge for a lab.

14. [2 points]: Are there other things you'd like to see improved in the second half of the semester?

Answer: Past exploits. More time on labs. More late days. More summaries of papers / what to focus on / background info. More attacks. More recent papers. Explicit lectures on lab mechanics. More design freedom in labs / more challenging design choices; less fill-in-the-blank style. Some papers are too technical. Fewer ways to turn things in (submit via make; submit via text file; email question; Piazza). Balance first and second parts of labs. Novelty lecture on lockpicking. Shorter quiz. Weekly review of lecture (not labs) material – recitation? Relate labs to lectures, teach more hands-on stuff. Labs where you solve some problem rather than get the details right? Explain what corner cases matter for labs. Lecture should focus on application of paper's ideas and short review of paper content. More info on final projects. Scrap the answers.txt stuff, just code.

End of Quiz