Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2010

Quiz I

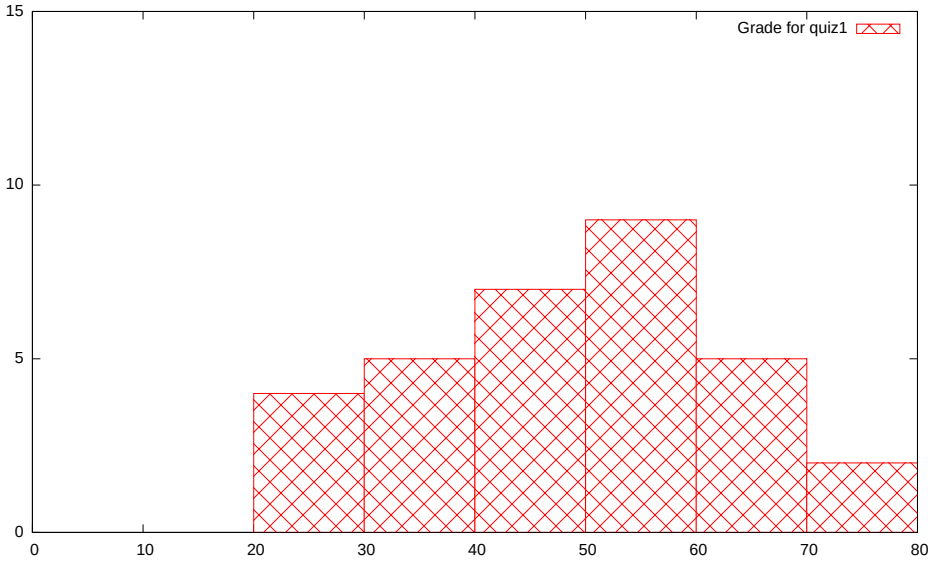
All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to finish this quiz.

Write your name on this cover sheet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Please do not write in the boxes below.



Mean 49, Median 51, Std. dev. 14

I Baggy bounds checking

Suppose that you use Baggy bounds checking to run the following C code, where X and Y are constant values. Assume that *slot_size* is 16 bytes, as in the paper.

```
1  char *p = malloc(40);
2  char *q = p + X;
3  char *r = q + Y;
4  *r = '\0';
```

For the following values of X and Y, indicate which line number will cause Baggy checking to abort, or NONE if the program will finish executing without aborting.

Answer: Recall that Baggy bounds checking rounds up the allocation size to the nearest power of 2 (in this case, 64 for pointer p), and can track out-of-bounds pointers that are up *slot_size/2* out of bounds.

1. [2 points]: $X = 45, Y = 0$
Answer: NONE: the access is within the 64 bytes limit.

2. [2 points]: $X = 60, Y = 0$
Answer: NONE: the access is within the 64 bytes limit.

3. [2 points]: $X = 50, Y = -20$
Answer: NONE: the access is within the 64 bytes limit.

4. [2 points]: $X = 70, Y = -20$
Answer: NONE: q goes out of bounds, but by less than *slot_size*/2, so r is in bounds again.

5. [2 points]: $X = 80, Y = -20$
Answer: Line 2: since q goes out of bounds by more than *slot_size*/2, Baggy aborts the pointer arithmetic. (We also accepted the answer of Line 4, due to some confusion.)

6. [2 points]: $X = -5, Y = 4$
Answer: Line 4: the reference is 1 byte before the start of the object, i.e. out of bounds.

7. [2 points]: $X = -5, Y = 60$
Answer: NONE: within the 64-byte object bounds.

8. [2 points]: $X = -10, Y = 20$
Answer: Line 2: since q goes out of bounds by more than *slot_size*/2, in the negative direction. (We also accepted the answer of Line 4, due to some confusion.)

II Control hijacking

Consider the following C code:

```
struct foo {
    char buf[40];
    void (*f2) (struct foo *);
};

void
f(void)
{
    void (*f1) (struct foo *);
    struct foo x;

    /* .. initialize f1 and x.f2 in some way .. */

    gets(x.buf);
    if (f1)   f1(&x);
    if (x.f2) x.f2(&x);
}
```

There are three possible code pointers that may be overwritten by the buffer overflow vulnerability: $f1$, $x.f2$, and the function's return address on the stack. Assume that the compiler typically places the return address, $f1$, and x in that order, from high to low address, on the stack, and that the stack grows down.

9. [5 points]: Which of the three code pointers can be overwritten by an adversary if the code is executed as part of an XFI module?

Answer: $x.f2$ can be overwritten, because it lives at a higher address than $x.buf$ on the allocation stack. $f1$ and the return address live on the scoped stack, which cannot be written to via pointers.

10. [5 points]: What code could the adversary cause to be executed, if any, if the above code is executed as part of an XFI module?

Answer: Any function inside the XFI module that is the target of indirect jumps (i.e., has an XFI label), and any stubs for allowed external functions (which also have XFI labels). The adversary cannot jump to arbitrary functions inside the XFI module that are not the targets of indirect jumps (and thus do not have an XFI label).

11. [5 points]: What code could the adversary cause to be executed, if any, if the above code is executed under control-flow enforcement from lab 2 (no XFI)?

Answer: Any code that was jumped to during the training run at the calls to $f1$ or $x.f2$, or any call sites of this function f during the training run.

III OS protection

Ben Bitdiddle is running a web site using OKWS, with one machine running the OKWS server, and a separate machine running the database and the database proxy.

12. [12 points]: The database machine is maintained by another administrator, and Ben cannot change the 20-byte authentication tokens that are used to authenticate each service to the database proxy. This makes Ben worried that, if an adversary steals a token through a compromised or malicious service, Ben will not be able to prevent the adversary from accessing the database at a later time.

Propose a change to the OKWS design that would avoid giving tokens to each service, while providing the same guarantees in terms of what database operations each service can perform, without making any changes to the database machine.

Answer 1: Implement a second proxy on the OKWS machine that keeps the real database tokens, accepts queries from services (authenticating the service using UIDs or another token), and forwards the queries to the real database server / proxy.

Answer 2: Establish connections to the database proxies in the launcher, send the 20-byte token from the launcher, and then pass the (now authenticated) file descriptors to the services.

13. [5 points]: Ben is considering running a large number of services under OKWS, and is worried he might run out of UIDs. To this end, Ben considers changing OKWS to use the same UID for several services, but to isolate them from each other by placing them in separate `chroot` directories (instead of the current OKWS design, which uses different UIDs but the same `chroot` directory). Explain, specifically, how an adversary that compromises one service can gain additional privileges under Ben's design that he or she cannot gain under the original OKWS design.

Answer: The compromised service could use `kill` or `ptrace` to interfere with or take over other services running under the same UID.

IV Capabilities and C

Ben Bitdiddle is worried that a plugin in his web browser could be compromised, and decides to apply some ideas from the "Security Architectures for Java" paper to sandboxing the plugin's C code using XFI.

Ben decides to use the capability model (§3.2 from the Java paper), and writes a function `safe_open` as follows:

```
int
safe_open(const char *pathname, int flags, mode_t mode)
{
    char buf[1024];
    snprintf(buf, sizeof(buf), "/safe-dir/%s", pathname);
    return open(buf, flags, mode);
}
```

which is intended to mirror Figure 2 from the Java paper. To allow his plugin's XFI module to access to files in `/safe-dir`, Ben allows the XFI module to call the `safe_open` function, as well as the standard `read`, `write`, and `close` functions (which directly invoke the corresponding system calls).

14. [10 points]: Can a malicious XFI module access files (i.e., read or write) outside of `/safe-dir`? As in the Java paper, let's ignore symbolic links and ".." components in the path name. Explain how or argue why not.

Answer: No. There are two possible attacks. First, a malicious XFI module could guess legitimate integer file descriptor numbers of other open files in the browser process (e.g., cookie files or cache files), and invoke `read` or `write` on them. Second, a malicious XFI module could write arbitrary data D to address A by first writing D to a file in `/safe-dir`, and then invoking `read` on that file, passing A as the buffer argument to `read`. This will write D to memory location A (since `read` is outside of XFI), and allow the attacker to gain control of the entire process.

V Browser security

15. [6 points]: In pages of a site which has enabled ForceHTTPS, `<SCRIPT SRC=...>` tags that load code from an `http://.../` URL are redirected to an `https://.../` URL. Explain what could go wrong if this rewriting was not performed.

Answer: An active attacker could replace the Javascript code in the HTTP response with arbitrary malicious code that could modify the containing HTTPS page or steal any of the data in that page, or the cookie for the HTTPS page's origin.

Ben Bitdiddle runs a web site that frequently adds and removes files, which leads to customers complaining that old links often return a 404 File not found error. Ben decides to fix this problem by adding a link to his site's search page, and modifies how his web server responds to requests for missing files, as follows (in Python syntax):

```
def missing_file(reqpath):
    print "HTTP/1.0 200 OK"
    print "Content-Type: text/html"
    print ""
    print "We are sorry, but the server could not locate file", reqpath
    print "Try using the <A HREF=/search>search function</A>."
```

16. [10 points]: Explain how an adversary may be able to exploit Ben's helpful error message to compromise the security of Ben's web application.

Answer: An adversary could construct a link such as:

`http://ben.com/<SCRIPT>alert(5);</SCRIPT>`,

containing arbitrary Javascript code, and trick legitimate users into visiting that link (e.g., by purchasing ads on some popular site). Ben's server would echo the request path back verbatim, including the Javascript code, causing the victim's browser to execute the resulting Javascript as part of Ben's page, giving the attacker's Javascript code access to the victim's cookies for Ben's site.

VI 6.858

We'd like to hear your opinions about 6.858, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

17. [2 points]: How could we make the ideas in the course easier to understand?

18. [2 points]: What is the best aspect of 6.858?

19. [2 points]: What is the worst aspect of 6.858?

End of Quiz