CS 155: Spring 2017
June 8, 2017

CS 155 Final Exam

This exam is open book and open notes. You may use course notes and documents that you have stored on a laptop, but you may NOT use the network connection on your laptop in any way, especially not to search the web or communicate with a friend. **You have 2.5 hours.**

Print your name legibly and sign and abide by the honor code written below. All of the intended answers may be written in the space provided. You may use the back of a page for scratch work. If you use the back side of a page to write part of your answer, be sure to mark your answer clearly.

The following is a statement of the Stanford University Honor Code:

A. *The Honor Code is an undertaking of the students, individually and collectively:*
   (1) *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*
   (2) *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*

B. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*

C. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

I acknowledge and accept the Honor Code.

____
(Signature)

_______________________ _______________________
(SUNet ID) *(Print your name,* legibly!*)*

☐ **GRADUATING?**

Prob	# 1	# 2	# 3	# 4	# 5	# 6	Total
Score							
Max	10	20	10	20	15	20	95

1. (*10 points*) .... **True or False**

For each question, please write `T` or `F` in the space provided. No explanation needed.

______ (a) Ingress filtering at the ISP, if implemented universally, is a way to prevent a DoS attack using packets with a spoofed source IP address.

______ (b) The CSP directive `upgrade-insecure-requests` tells the browser to use HTTPS for all in-site resource requests, regardless of the protocol in the URL.

______ (c) DDoS attacks often use covert channels.

______ (d) Protecting an application using ASLR requires recompiling that application (even if the executable is position independent).

______ (e) Fuzzing tools like `afl-fuzz` are guaranteed to find at least one vulnerability.

______ (f) Checking the `Referer` header is a robust defense against CSRF attacks.

______ (g) Subresource integrity (SRI) is not needed if the data is transferred over HTTPS.

______ (h) Stateful firewalls are a good way to protect a high speed link between two large data centers.

______ (i) On iOS, copy-and-paste is a safe way for a user to copy a password from a password manager app to another app. Recall that the copy-and-paste buffer is readable by all installed applications.

______ (j) Two containers running on the same system are far better isolated from each other than two VMs running on the same system.

2. (*20 points*) .... Questions From All Over With a Short Answer

(a) (*4 points*) Consider a Web site `xyz.com` that implements a phone dialer. When the user enters a phone number to call, the browser opens a new window containing the following Javascript that defines a `postMessage` event listener:

```
function receiveMessage(event) {
    // event.data is a phone number from sender
    initiatePhoneCallTo(event.data);
}
window.addEventListener("message", receiveMessage, false);
```

The parent page then sends a `postMessage` to this window to initiate the call. This activates the `receiveMessage` function which makes the call. Explain how an evil web site can cause a visitor to initiate phone calls to arbitrary phone numbers. Assume the visitor is logged in to her `xyz.com` account, but does not have `xyz.com` open in a window.

(b) (*4 points*) Continuing with the previous question, if the function `receiveMessage` started with the following line:

```
if (event.origin !== "https://www.xyz.com")   return;
```

Would this eliminate the problem you identified in part (**2**a)? Recall that `event.origin` is the true origin that initiated the `postMessage` call.

(c) (*4 points*) When running a setuid root program under GDB, the program does not run with root privileges. Consider a patch to the Linux kernel that makes it so that if a user debugs a setuid root program with GDB, the program being debugged runs with root privileges. Is this safe? If so, explain why. If not, describe an attack that is made possible by this change.

(d) (*4 points*) In the DDoS lecture we discussed the Github attack, where a popular site was used to mount a DDoS on Github by injecting certain Javascript into every response from the popular site. Suppose Github were protected by an anti-DDoS service like Google Project Shield. How could Google have blocked the DDoS requests? One approach is to only allow incoming requests from existing Github customers. Suggest one way that Google could implement this.

(e) (*4 points*) A common approach to software update is as follows: the software vendor digitally signs the update file using a secret signing key and posts the update along with the signature on a public server. Every client periodically checks the public server. When an update is found, the client downloads the update file, checks the signature, and installs the update if the signature is valid. Explain why this can be insecure. Hint: consider a client running version n. Version $n + 1$ has a known vulnerability which is fixed in version $n + 2$. Version $n + 2$ is available for download on the update server.

3. (*10 points*) .... Jump-Oriented Programming

Elizabeth is attacking a buggy application. She has found a vulnerability that allows her to control the values of the registers ecx, edx, and eip, and also allows her to control the contents of memory locations 0x9000 to 0x9014. She wants to use return-oriented programming, but discovers that the application was compiled without any ret instructions! Nonetheless, by analyzing the application, she learns that the application has the following code fragments (gadgets) in memory:

```
0x3000: add edx, 4      ; edx = edx + 4
        jmp [edx]       ; jump to *edx

0x4000: add edx, 4      ; edx = edx + 4
        mov eax, [edx]  ; eax = *edx
        jmp ecx         ; jump to ecx

0x5000: mov ebx, eax    ; ebx = eax
        jmp ecx         ; jump to ecx

0x6000: mov [eax], ebx  ; *eax = ebx
        ...             ; don't worry about what happens after this
```

Show how Elizabeth can set the values of the registers and memory so that the vulnerable application writes the value 0x2222 to memory address 0x8888.

ecx	
edx	
eip	0x4000

0x9000	
0x9004	
0x9008	
0x900c	
0x9010	
0x9014	

Recall that eip is the instruction pointer. It holds the address of the next instruction to execute. ecx and edx are general purpose registers.

4. (*20 points*) .... Control hijacking

(a) (*5 points*) In class we explained how ProPolice, an enhancement to stack canaries, re-orders local variables on the stack so that pointers are always allocated before string buffers in a stack frame (i.e. local pointers are allocated at a lower address in memory). Give example code that is vulnerable when the basic stack canaries architecture is used (when local variables are allocated in the same order as in the code), but is not vulnerable if this re-ordering is done. Explain why your code satisfies these properties.

(b) (*4 points*) Intel recently introduced a new instruction called `endbranch`. The instruction does nothing (a NOP). However, whenever the processor executes an indirect jump (e.g., `jmp ecx` as in Question 3) or an indirect call, the immediate next instruction in the instruction stream must be an `endbranch`. If not, the processor signals a protection exception that terminates the process. What exploitation technique is `endbranch` intended to prevent and how? Make sure to explain where you would place `endbranch` instructions in the code and why you would place them there.

(c) (*2 points*) In homework #1 we discussed the shadow stack, a technique used to protect return addresses on the stack. Intel recently introduced hardware support for this. The shadow stack is stored in memory pages marked by a new "shadow stack" attribute. The stack is addressed by a new register called the shadow stack pointer (SSP). When the processor executes a `call` instruction it pushes the return address onto the regular stack (as usual) and also onto the shadow stack (new). When `ret` is executed, the processor compares the return address on the regular stack with the value stored at the top of the shadow stack. If the values differ, the processor signals a protection exception. Memory pages that are marked as shadow stack cannot be read or written by standard data instructions like `mov` (which is used to read and write to memory). Only `call` and `ret` instructions read and write to the shadow stack. Note that only return addresses are written to the shadow stack; arguments are not.

Why is it important that `mov` instructions cannot write to the shadow stack?

(d) (*5 points*) Give example code that is vulnerable to a buffer overflow when the full stack canaries technique is implemented (including re-ordering as in part (a)), but is not vulnerable to return address smashing when a shadow stack is used (without canaries).

(e) (*4 points*) If Intel's shadow stack is used, is there any value in also using stack canaries? Justify your answer.

5. (*15 points*) .... Timing attacks on HTTPS

Recall that HTTPS does not hide the length of an HTTP request or response. A number of attacks, such as CRIME and BREACH, show that the lengths of HTTPS responses from the server can completely expose the contents of these responses. Clearly a network attacker can measure the response length by simply observing network traffic. However, this requires monitoring the server's or client's network. In this question we show how a remote Web attacker can measure the server's response length.

Suppose Alice is logged into her bank. Alice then visits `evil.com` who sends back the following Javascript:

```
fetch('https://bank.com/account').then(
    function(response) {  T1 = performance.now();   });
```

This causes Alice's browser to issue a GET request for `https://bank.com/account`. Her browser establishes a TCP connection to the bank, sets up an HTTPS session, sends the GET request (including Alice's cookies), and waits for the response. The function on the second line is called when the *first byte* of the response is received. `T1` records the time accurate to the millisecond.

(a) (*2 points*) Can `evil.com` read the contents of the response? Justify your answer.

(b) (*2 points*) Additional Javascript from `evil.com` can record the time `T2` when the *last byte* of the response is received. Let `Delta = T2 - T1`. Suppose the maximum packet size (called MSS) is 1460 bytes, and the server sends ten packets in sequence before stopping to wait for ten `ACK` packets from the client. Once the server receives the ten `ACK` packets it sends another ten packets, waits again, and so on. Explain how `evil.com` can tell if the response size is more than 10×1460 bytes, or less. You may assume that the `bank.com` site has *public* objects, such as images, of different sizes. (To keep things simple, we are ignoring the bytes generated by the TCP and TLS handshakes).

(c) (*4 points*) Next, suppose that the GET request for account data takes an argument, so that the request looks like:

```
https://bank.com/account?requestID=RID
```

where `RID` is an arbitrary string provided by `evil.com`. The response from `bank.com` contains the given `RID`, but the response is otherwise unaffected by the value of `RID`. Explain how `evil.com` can use this reflected argument to determine the *exact* response size from `bank.com`.
As mentioned earlier, your answer enables a remote `evil.com` site to use the BREACH or CRIME attacks to then completely decrypt the server's response.

(d) (*3 points*) How would you prevent a Web attacker from learning the exact response length using part (c) if you were designing `bank.com`?

(e) (*4 points*) Looking more generally at the correct use of HTTPS, suppose that a page loaded over HTTP loads a login iframe as

```
<iframe src="https://site.com/login"> </iframe>
```

Can an active network attacker steal the password entered into the login frame? Justify your answer.

6. (*20 points*) .... Android fragment injection

Android apps are composed of application components of different types, including activities. An activity defines a single UI such as a browsing window or a preferences screen. Activities can contain fragments, which are implemented by the `android.app.Fragment` class and provide a part of a UI.

A fragment (or full activity) can be exported by setting an export tag `android:exported="true"` in the app manifest section for this fragment. This allows the fragment to be invoked by any other app. An app invokes a fragment by sending an intent. An intent may include extra data fields (called Intent 'extras'), passed inside a bundle (implemented by the `android.os.Bundle` class). It is easy for an app to create a bundle and set its data fields before sending an intent.

A PreferenceActivity provides a way for apps to show preferences to the user. In certain versions of Android, any app that extends the `PreferenceActivity` class using an exported activity is vulnerable to a fragment injection attack. The vulnerability affects many apps, including Settings, Gmail, Google Now, DropBox and Evernote. The Settings app is shipped on every Android device to allow users to change user settings such as the device locking password. Assume throughout this problem that we are in a vulnerable version of Android.

An intent sent to a PreferenceActivity may contain two extra data fields that are especially interesting: `PreferenceActivity.EXTRA_SHOW_FRAGMENT` and `PreferenceActivity.EXTRA_SHOW_FRAGMENT_ARGUMENTS`.
When the PreferenceActivity receives an intent with these data fields, it dynamically loads the `EXTRA_SHOW_FRAGMENT` and passes it the `EXTRA_SHOW_FRAGMENT_ARGUMENTS`. In other words, an exported fragment present in the system may be loaded by a malicious app and passed any arguments. The loaded fragment will run in the context of the vulnerable app, will have the same privileges and have access to its private data.

(a) (*3 points*) Suppose app `A` exports a PreferenceActivity as described in the problem statement and app `B` passes `A` an `EXTRA_SHOW_FRAGMENT`. Which app's permissions, `A` or `B`, govern the behavior of the dynamically loaded fragment? Explain your reasoning succinctly.

(b) (*6 points*) The main `Settings` activity of the Settings app extends `PreferenceActivity`. One fragment of the `Settings` activity is `ChooseLockPassword$ChooseLockPasswordFragment`. This fragment is designed to be loaded under the `ChooseLockPassword class` which is not exported. Therefore, the authors of `ChooseLockPasswordFragment` did not expect to have to check for malicious input. This fragment has an argument `confirm_credentials` that can be set to true or false when called to change the device pin code. When this flag is `true`, the caller must supply the current pin code in order to change it.

(i) (*3 points*) How can a malicious app load `ChooseLockPasswordFragment` and provide arbitrary input to it?

(ii) (*3 points*) How would you design an app to change the user's pin code to 123456, using the method you described in part (a)?

(c) (*2 points*) Assuming dynamic loading is implemented using the Java Reflection API, how might an information-flow tool determine that there is a potential vulnerability in Settings (*e.g.,* untrusted input source flows to sensitive sink)?

(d) (*3 points*) Fragment injection was addressed in KitKat, which added a method `PreferenceActivity.isValidFragment` that is called before a fragment is dynamically instantiated by PreferenceActivity. What should be the default implementation of this method?