Computer Science 4271
Spring 2022
Midterm exam 2
April 19th, 2022
Time Limit: 75 minutes, 9:45am-11:00am

- Before starting the exam, you can fill out your name and other information of this page, but don't open the exam until you are directed to start. Don't put any of your answers on this page.
- This exam contains 7 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing.
- You may use any textbooks, notes, or printouts you wish during the exam, but you may not use any electronic devices: no calculators, smart phones, laptops, etc.
- You may ask clarifying questions of the instructor or TAs, but no communication with other students is allowed during the exam.
- Please read all questions carefully before answering them. Remember that we can only grade what you write on the exam, so it's in your interest to show your work and explain your thinking.
- By signing below you certify that you agree to follow the rules of the exam, and that the answers on this exam are your own work only.

The exam will end promptly at 11:00am. Good luck!

Your name (print): _____

Your UMN email/X.500: _____`@umn.edu`

Number of rows ahead of you: _____ Number of seats to your left: _____

Sign and date: _____

Question	Points	Score
1	20	
2	22	
3	28	
4	30	
Total:	100	

1. (20 points) Matching definitions and concepts. Fill in each blank with the letter of the corresponding answer. Each answer is used exactly once.

   (a) ____ the structure used in DES

   (b) ____ given y, compute x such that $g^x = y$

   (c) ____ accessing unintended file system locations with `..`

   (d) ____ a mode of operation that can be used like a stream cipher

   (e) ____ an algorithm that would be insecure if factoring were easy

   (f) ____ the standard name for UID 0

   (g) ____ a virtual machine implemented underneath a normal kernel

   (h) ____ another name for JavaScript injection

   (i) ____ any pair x, y with $x \neq y$ such that $h(x) = h(y)$

   (j) ____ Unix system call for changing file permissions

   (k) ____ a security policy that forbids information disclosure

   (l) ____ subject for access control in web browsers

   (m) ____ given y, compute an x such that $h(x) = y$

   (n) ____ Unix system call for filesystem isolation

   (o) ____ idealized functionality equivalent to a random function

   (p) ____ a logical statement that is always true

   (q) ____ for an object, records allowed subjects and operations

   (r) ____ reading plaintext data from a network

   (s) ____ used to implement sessions for web applications

   (t) ____ truly random bits used as a keystream

   A. ACL B. `chmod` C. `chroot` D. confidentiality E. cookie F. directory traversal G. discrete log H. Feistel cipher I. free collision J. hypervisor K. OFB L. one-time pad M. origin N. preimage attack O. random oracle P. `root` Q. RSA R. sniffing S. tautology T. XSS

2. (22 points) Another kind of injection attacks.

   Technologie pour Bateaux Rouen SARL (Technology for Boats) is a small business in northern France that specializes in control and management software for autonomous watercraft in the logistics industry, and they have brought you in as a security consultant. Their best-selling product has a web interface written in PHP providing commands to back-end software written using a proprietary interpreted programming language named Boating Oxide. You need to break the bad news to them that the way they have implemented this architecture is vulnerable to code injection, by building example attacks.

   Boating Oxide's name (also Oxide or BoatOx for short) is reminiscent of Rust, but the languages are not very similar. Oxide supports integer literals, arithmetic and logical operations, and control flow using similar syntax as C, and statements are terminated with a semicolon (`;`), but much of the other syntax is different. Comments start with a `#` and continue to the end of the line, as in Perl and the Unix shell. Strings are enclosed with the character sequences `<<` at the start and `>>` at the end (modeled on the quotation marks « and » used in French), and can be compared with the operator `==`. As in PHP, Perl, and the Unix shell, variable names can appear inside strings, where their values are substituted. However unlike PHP, Perl, and the Unix shell, which use a dollar sign (`$`) before variable names, Oxide uses a Euro symbol `€`. For instance in Oxide if `€name` contains `Bob`, then `<<Hello, €name>>` is the same as `<<Hello, Bob>>`, just like how `"Hello, $name"` works in PHP.

   The PHP command `oxide` takes a string as an argument, and evaluates it as a Boating Oxide command. The Oxide command `self_destruct()` would cause a boat to blow itself up with explosives, and should only be used in a severe emergency.

   (a) The following line of PHP appears to initialize an Oxide string variable based on a similarly named PHP variable:

   ```
   oxide("€location_ox = <<$location_php>>;");
   ```

   The code works correctly for many values of the PHP variable `$location_php`, but if the variable is supplied from an untrusted web interface with no sanitization, other Oxide code could be executed. For instance, give a value for `$location_php` that would cause the boat to self destruct:

   (b) The following PHP code tries to create an Oxide loop to print the destination of the package named in the PHP variable `$pkg`:

   ```
   oxide("for (€p in packages()) { if (€p == <<$pkg>>) print(get_dest(€p));}");
   ```

   As in the previous part, suppose that `$pkg` is supplied from an untrusted web interface with no sanitization. Give a value for `$pkg` that would cause the destinations of *all* packages to be printed:

3. (28 points) ECB cryptanalysis.

   The following 5 24-byte messages, shown in hexadecimal, have been encrypted with an unknown block cipher:

   (a) `989caa4c9a84aa9aaabe9a3859aa9aa8beaa849ccb3895f8`
   (b) `a09fae3d094f9fae3d09279fae3d09225ba03d099f5b9f3d`
   (c) `42257890e89d628298690021db15fc557a41c226ee92f103`
   (d) `98aa9b9a95f2ceaa98aa849a4cceaa98aa9bcba8493839be`
   (e) `4c9ac69ce7e7e74c9ac69ce7e7e74c9ac69ce7e7e727c69c`

   Luckily for you as a cryptanalyst, you know that the block cipher made some design choices that are bad for its security: the block size is only 1 byte (8 bits), the cipher is used in ECB mode, and the same key was used for all the messages. Also, through the work of your colleagues in human intelligence, your agency has been able to determine that the messages correspond to the following 5 plaintexts, in some order:

   1. `ABCDEFGHIJKLMNOPQRSTUVWX`
   2. `8271>5271>4271>3081>2021`
   3. `It was a dark and stormy`
   4. `wait...wait...wait...4it`
   5. `I came, I saw, I conqr'd`

   Due to an unexpected power outage, you've been tasked with determining which message is which, just using pencil and paper. A colleague who has done this kind of cryptanalysis before suggests that you proceed by trying to figure out the encryptions of certain common byte values in the messages. To make it easier to see what's going on, we've copied the plaintexts, their correspond hex values, and the ciphertexts into a table to line them up. We've used the symbol ␣ to represent a space, which would otherwise be invisible. For your convenience, the table is shown together with the spaces for your answers on the next page. Remember, you don't have to do parts (a)–(e) before part (f): you may find it helpful to go back and forth.

1.	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
2.	8	2	7	1	>	5	2	7	1	>	4	2	7	1	>	3	0	8	1	>	2	0	2	1
3.	I	t	␣	w	a	s	␣	a	␣	d	a	r	k	␣	a	n	d	␣	s	t	o	r	m	y
4.	w	a	i	t	.	.	.	w	a	i	t	.	.	.	w	a	i	t	.	.	.	4	i	t
5.	I	␣	c	a	m	e	,	␣	I	␣	s	a	w	,	␣	I	␣	c	o	n	q	r	’	d
1H.	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f	50	51	52	53	54	55	56	57	58
2H.	38	32	37	31	3e	35	32	37	31	3e	34	32	37	31	3e	33	30	38	31	3e	32	30	32	31
3H.	49	74	20	77	61	73	20	61	20	64	61	72	6b	20	61	6e	64	20	73	74	6f	72	6d	79
4H.	77	61	69	74	2e	2e	2e	77	61	69	74	2e	2e	2e	77	61	69	74	2e	2e	2e	34	69	74
5H.	49	20	63	61	6d	65	2c	20	49	20	73	61	77	2c	20	49	20	63	6f	6e	71	72	27	64
(a)	98	9c	aa	4c	9a	84	aa	9a	aa	be	9a	38	59	aa	9a	a8	be	aa	84	9c	cb	38	95	f8
(b)	a0	9f	ae	3d	09	4f	9f	ae	3d	09	27	9f	ae	3d	09	22	5b	a0	3d	09	9f	5b	9f	3d
(c)	42	25	78	90	e8	9d	62	82	98	69	00	21	db	15	fc	55	7a	41	c2	26	ee	92	f1	03
(d)	98	aa	9b	9a	95	f2	ce	aa	98	aa	84	9a	4c	ce	aa	98	aa	9b	cb	a8	49	38	39	be
(e)	4c	9a	c6	9c	e7	e7	e7	4c	9a	c6	9c	e7	e7	e7	4c	9a	c6	9c	e7	e7	e7	27	c6	9c

Fill in the plaintext number corresponding to each ciphertext:

(a) ____

(b) ____

(c) ____

(d) ____

(e) ____

(f) Fill in common byte values:

Plaintext char	Plaintext hex	Ciphertext hex
I	0x49	
␣	0x20	
.	0x2e	
1	0x31	

4. (30 points) A race condition attack.

   The following C pseudo-code attempts to write a line of text to the end of a log file stored in the `/tmp` directory. However, you may be able to see that is has TOCTTOU/race condition problems.

```
char *log_fname = "/tmp/logfile";
void write_log_msg(char *msg) {
    /* point A */
    if (!file_exists(log_fname))
        create_log_file(log_fname);
    /* point B */
    if (!is_regular_file(log_fname))
        print_error_and_exit();
    /* point C */
    FILE *fh = fopen(log_fname, "a");
    if (!fh) print_error_and_exit();
    fputs(msg, fh);
    fclose(fh);
}
```

   Suppose that the program containing this code is running with superuser privileges, and your goal as an attacker is to cause it to write the message to the system configuration file `/etc/passwd` instead of to the normal log file. Assume that before this function starts executing, no file named `/tmp/logfile` exists. But you as the attacker are able to run other programs, with write access to `/tmp`, at the same time this code is running. In particular, to achieve your attack, you will try to get certain operations to occur in between the vulnerable program's operations, namely at the points marked point A, point B, and/or point C.

   In the parts below, describe what racing attacker actions should take places at each point for a successful attack (the next page has more explanation of the code). You may not need to use all three points.

   (a) At point A:

   (b) At point B:

   (c) At point C:

Assume that the function `file_exists` returns true if a file exists with a given pathname, and false otherwise. Assume that the function `is_regular_file` returns true if its pathname argument is a regular file with only one link, and false if it is a directory, a symbolic link, a hard link, or does not exist. Assume that the function `create_log_file` creates a file with the given name and appropriate permissions for a log file. Assume that the function `print_error_and_exit` prints an error message and then causes the program to exit.

The functions `fopen`, `fputs`, and `fclose` are from the standard library. The function `fopen` opens a file, and a second argument of `"a"` causes it to write to the end of the file (append). It returns a null pointer if the open fails. The functions `fputs` and `fclose` write a string to a file and close the file respectively. We have omitted error handling for `fputs` and `fclose` because it is not relevant to the attack.