ITIS 6200/8200 Principles of Information Security and Privacy

Final Exam

If the questions ask for explanations, please **briefly** explain the **key reasons**.

**Notation**: here are some of techniques and notations that may be useful.

* AES: Enc(K, M) and Dec(K, C) where K, M, and C denote the key, plaintext, and ciphertext
* HMAC: HMAC(K, V) where K, V are the inputs to the HMAC function
* Hash: Hash(M) where M is the input to the hash function
* RSA: Ekey(M), and Dkey(C). The key here could be pub\_[name], pri\_[name] with the corresponding names. For example, Epub\_alice(M).
* c1 || c2: String concatenation of c1 and c2.

**Question 1**. **Cryptography Fundamentals (15 points)**

Q 1.1: Fill the following table. The first row shows security properties, and first column shows cryptographic techniques. Put a mark in a cell if its corresponding technique is designed for the corresponding security property.

|  |  |  |
| --- | --- | --- |
|  | Confidentiality | Integrity |
| AES-CBC |  |  |
| MAC |  |  |
| RSA encryption |  |  |
| RSA signature |  |  |

Q 1.2: Fill the following table. The first row shows cryptographic techniques, and the first column shows a few mechanisms/building blocks for these techniques. Put a mark in a cell if its corresponding technique uses the corresponding mechanism in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | AES-CBC | MAC | RSA Encryption | RSA Signature |
| Hash |  |  |  |  |
| PRNG |  |  |  |  |
| Modulo operation |  |  |  |  |
| XOR |  |  |  |  |

**Question 2**. **Encryption (15 points)**

Alice wants to send a large size message M to Bob. Their public keys are known to each other.

Q 2.1: What is the main advantage of using symmetric key encryption like AES? What is the main advantage of using Asymmetric key encryption like RSA? (be brief in your answer)

Q 2.2: Alice wants to use AES to encrypt M. To do that, Alice first needs to share a symmetric key K with Bob. How can Alice share K with Bob with both confidentiality and integrity? Write down the formula for sending K.

Q 2.3: What can go wrong (with the scheme from Q 2.2) if a powerful attacker Mallory can intercept messages passing between Alice and Bob?

**Question 3**. **Confidentiality and Integrity (15 points)**

Alice and Bob want to communicate with confidentiality and integrity. They share a symmetric key K and know each other’s public key.

Q 3.1: Can Bob get the message M? Enc(K1, M || MAC(K2, M))

Q 3.2: Design a scheme that provides both confidentiality and integrity.

Q 3.3: Assume AEC-CBC, but the same IV is reused for encryption. Does the scheme provide IND-CPA? If not. Design two inputs that can break INC-CPA.

**Question 3. Confidentiality & Integrity**

Alice is sending message M to Bob in the following way:

Ciphertext c = c1 || c2 where c1 = Enc(K, m) and c2 = Hash(c1)

Here, Enc(K,m) is the secure encryption scheme AES-CBC, and Hash(m) is the cryptographic hash function SHA-256.

Q 4.1: Does this scheme provide confidentiality? E.g., can an eavesdropper Eve learn about the contents of the message?

Q 4.2: Does this scheme provide integrity? E.g., can Mallory tamper with message without being detected?

Q 4.3: Can you design an approach for sending the message so it provides both integrity and confidentiality?

Bob has a public-private key pair (pub\_Bob, priv\_Bob). Alice needs to send some information to Bob. She wants to make sure that when Bob opens the message, he can verify that this is from Alice but not anyone else.

Q 5.1: If Alice sends out the message as: [ Alice || Epub\_Bob(message) ] to Bob. That is, she sends out her name in clear text, followed by ciphertext of the message encrypted with Bob’s public key. Can powerful attacker Mallory impersonate Alice and send out a packet in Alice’s name? How can she do it? Assume that Mallory also has the public key of Bob.

Q 5.2: If Alice sends out [ Epub\_Bob(Alice || message) ], where Alice puts her name in the encryption. Can Mallory still impersonate Alice?

Q 5.3: Design a way that Bob can ensure the message comes from Alice.

To use RSA signatures on messages, we first create a RSA key pair: (N, e) is the RSA public key and d is the RSA private key, where N is the RSA modulus. For standard RSA signatures, we typically set e to a small prime value such as 5.

Q 6.1: For RSA signatures, we often sign the hash of a message, rather than the message directly.

Why is that?

Q 6.2: Assume that we **skip using a hash function**, and sign the messages directly. That means, if Alice wants to send a signed message to Bob, she will send (M, S) to Bob where S = Md mod N is computed using her private signing key d. With such a scheme, how does Bob verify this message come from Alice? What formula does Bob need?

Q 6.3: Mallory learns that Alice and Bob are using the simplified signature scheme that without using hash functions. Can Mallory find a (M, S) pair such that S will be a valid signature on M?

Assume that Mallory knows Alice’s public key N and e, but not Alice’s private key d.

**Question 4**. **Hash (15 points)**

Alice and Bob want to communicate with confidentiality and integrity. They share a symmetric key K and know each other’s public key.

Q 4.1: What if

Q 4.2: What if

Q 4.3: What if

Alice wants to store a file in an untrusted file system that the adversary can read and modify. Assume that Alice does not know if hashes or files have been modified in the untrusted datastore.

Alice’s computer stores the files in the following way: for every file F, the computer will calculate the hash value of the file hash(F) and store it after the file, i.e., F || hash(F). Every time when Alice login, the machine will automatically hash all the files and compare the results to the stored hash values. In this way, if by accident the hard drive is mis-functioning and flips a few bits in a file, Alice can immediately detect it since the hash value will be different. Now an attacker hacks into Alice’s machine and he tries to change several files. The attacker also knows the hash function that the computer uses.

Q 3.1: Describe what the attacker needs to do so that the next time Alice login, the machine will not detect the changes.

Q 3.2: How do we improve the mechanism to prevent / detect such changes?

**Question 1.b**

* **You can use:**
  + **An IND-CPA encryption scheme (e.g. AES-CBC): Enc(*K*, *M*) and Dec(*K*, *M*)**
  + **An unforgeable MAC scheme (e.g. HMAC): MAC(*K*, *M*)**
* **First attempt: Alice sends Enc(*K*1, *M*) and MAC(*K*2, *M*)**
  + **Integrity? Yes, attacker can’t tamper with the MAC**
  + **Confidentiality? No, the MAC is not IND-CPA secure**
* **Idea: Let’s compute the MAC on the *ciphertext* instead of the plaintext:  
  Enc(*K*1, *M*) and MAC(k2, Enc(*K*1, *M*))**
  + **Integrity? Yes, attacker can’t tamper with the MAC**
  + **Confidentiality? Yes, the MAC might leak info about the ciphertext, but that’s okay**
* **Idea: Let’s encrypt the MAC too: Enc(*K*1, *M* || MAC(*K*2, *M*))**
  + **Integrity? Yes, attacker can’t tamper with the MAC**
  + **Confidentiality? Yes, everything is encrypted**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Confidentiality** | **Integrity** | **Authenticity** | **Availability** |
| **AES** |  |  |  |  |
| **MAC** |  |  | | |
| **RSA encryption** |  |  | | |
| **RSA signature** |  |  | | |
| **Hash function** |  |  | | |
| Diffie-Hellman Key Exchange |  |  | | |
| PRNG |  |  | | |
| Certificate |  |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Confidentiality** | **Integrity** | **Availability** |
| **MITM** |  |  |  |
| **XSS** |  |  | |
| **CSRF** |  |  | |
| Distributed Denial of Service (DDoS) |  |  | |
| Denial of Service |  |  | |
| SQL Injection Attack |  |  | |
| Buffer Overflow Attack |  |  | |
| TCP spoofing attack |  |  | |
| SYN flooding |  |  | |
| ARP spoofing attack |  |  | |

* CSRF
* CSRF token and Referer
* HTML sanitization
* Content Security Policy (CSP)
* reflected XSS
* SQL injection
* ARP spoofing
* TCP data injection
* TCP spoofing
* DoS / DDoS
* SYN flooding
* SYN cookies
* Firewall
* Packet Filters
* Path Traversal Attack
* Intrusion Detection
* Buffer Overflow

**Question 5. Access Control**

**5.a ACL of a HR system SIS in UNCC**

**5.b how a Trojan Horse attack works by compromising Chao’s computer**

**5.c How a MAC can help with that?**

**Question 6. Web Security (20 points)**

A website uses a Go handler is processing HTTP requests to URL **https://vulnerable.com/hello** that takes an argument of name. The website is vulnerable to XSS attacks.

**func handleSayHello(w http.ResponseWriter, r \*http.Request) {**

**name := r.URL.Query()["name"][0]**

**fmt.Fprintf(w, "<html><body>Hello %s!</body></html>", name)**

**}**

Q 4.1: Design a GET request that can launch an attack to *Zmazon.com* to reset password to 123. *Zmazon.com* provides a function for resetting password: http://Zmazon.com/resetPassword?password=abc.

(7 points)

Ans: https://vulnerable.com/hello?name=<script>fetch(“https:// bank.com/transfer?amount=1000&to=Mallory”)</script>

Q 4.2: Assume EvanBook does not check user inputs. If Alice opens Eve’s post, which of these cookies would be attached?

Q 4.3: To log into EvanBook, you must go through authentication on login.evanbook.com, and set a cookie to keep track of your authenticated status.

The session token cookie should be secure against network attackers, and should get sent to as many pages on evanbook.com as possible.

Q 4.3: Suppose *bank.com* decides to defend against CSRF attacks with a cookie-based CSRF token, as follows:

1. When a user logs in, *bank.com* sets a cookie csrf\_token with domain attribute of *bank.com*.
2. When the user sends a POST request, the value of the csrf\_token is embedded as one of the form fields.
3. On receiving a POST request, *bank.com* checks that the value of the csrf\_token cookie matches the one in the form.

If the chat forum has domain *evil.com*, can the CSRF attack above succeed? If the chat forum has domain *evil.bank.com*, can the CSRF attack succeed?

**Question 7. Network Security (20 points)**



***TCP handshake***

EvanBot realizes that the server is sending messages to the student, but the student only responds with ACKs and never sends any messages after the initial handshake. They design a Half TCP protocol which provides TCP’s properties for communications from the server to the student, but not for communications from the student to the server. This is accomplished using a modified version of the standard three step handshake pictured below.

Q 7.1: What information do we needed in this 3-way handshake?

Q 7.2: What attack can be easier?

* TCP data injection
* TCP spoofing
* DoS / DDoS
* SYN flooding
* SYN cookies

Q 7.3: What guarantees are still provide?

Q 7.2: Consider a on-path attacker Eve who can observe the traffic but cannot modify it. Can Eve hijack the TCP connection between the Client and the Server? What can she do?

Q 7.3: Consider a off-path attacker David who cannot observe and modify the traffic. Can David do anything malicious to the connection? If so, what can he do?

**Question 8. Memory Vulnerability (20 points)**

Consider the following vulnerable C code. Assume you are on a little-endian 32-bit x86 system and no memory safety defenses are enabled.

A screen shot of a computer program

Description automatically generated

Q 5.1: Assume that execution has reached line 6. Fill in the following stack diagram. Assume that each row represents 4 bytes. Note that arrays are filled from lower addresses to higher addresses and are zero-indexed.

***Stack***

|  |
| --- |
|  |
|  |
| RIP of **foo** |
| SFP of **foo** |
|  |
|  |
|  |
|  |

Q 5.2: Assume that the address of the RIP of **foo** is 0x12345678.

Construct an input to gets that would cause the program to execute malicious shellcode. You may reference SHELLCODE as a 16-byte shellcode.

* AES: Enc(K, M) and Dec(K, C) where K, M, and C denote the key, plaintext, and ciphertext
* HMAC: HMAC(K, V) where K, V are the inputs to the HMAC function
* Hash: Hash(M) where M is the input to the hash function
* RSA public-key encryption: Enc(pub-Bob, M), and Dec(priv-Bob, C).
* RSA signature: Enc(priv-Alice, M) and Dec(priv-Alice, C)
* Diffie-Hellman Key Exchange
* Certificate and Certificate Authority
* Access Control (different models)
* CSRF
* CSRF token and Referer
* HTML sanitization
* Content Security Policy (CSP)
* reflected XSS
* SQL injection
* ARP spoofing
* TCP data injection
* TCP spoofing
* DoS / DDoS
* SYN flooding
* SYN cookies
* Firewall
* Packet Filters
* Path Traversal Attack
* Intrusion Detection
* Buffer Overflow

**BACKUP**

|  |  |
| --- | --- |
|  | **Man-In-The-Middle Attack** |
| **AES-CBC** |  |
| **MAC** |  |
| **RSA encryption** |  |
| **RSA signature** |  |
| **Diffie-Hellman Key Exchange** |  |