Security Cheatsheet #1

SimonXie2004.github.io
UC Berkeley CS161 Fall 24

1. Security Principles.

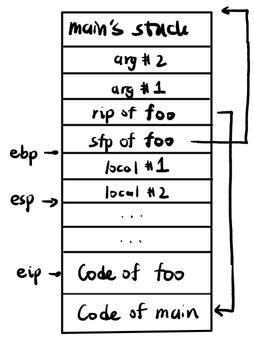
- Know your threat model • Consider human factors. • Security is economics. • Detect if you can't prevent • Defense in depth
- Least privilege • Use fail-safe defaults • Seperation of responsibility • Ensure complete mediation ⇒ unbypassable
- Shannon's maxim: (enemy knows whole system) • Design security from start

2. x86 ASM, Call Stack

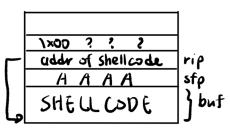
- Endianess: Rep of 0x0A0B0C0D {0D0C0B0A / 0A0B0C0D mem addr ++
- Inst: op, src, dst; $constant; %register (eax~edx, esi, edi)
- Stack frame: [esp, ebp)  ← ebp, ← esp; eip = PC + 4
- Stack layout: mem↑ [local 1 / [array] / local 3] ↓ sequence; Inside arr/struct: mem↑ [arr[1] / arr[0]] ↑ seq
- Calling convention

main {
1. push args inversely
2. push old eip → } push %eip } call foo
3. update eip → } jmp foo

foo {
4. push old ebp (sfp). → push %ebp
5. update ebp → mov %esp %ebp
6. update esp → sub $... %esp
7. execute function
8. update esp → mov %ebp %esp
9. restore sfp → pop %ebp
10. restore rip → pop %eip ⇒ ret
11. remove args from stack



3. Buffer overflow



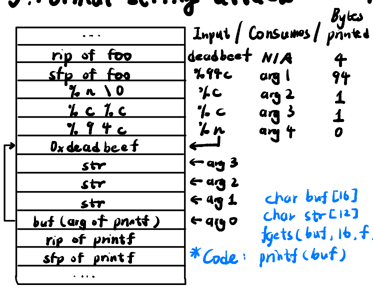
3. exploiting gets(buf).

4. Integer overflow attack

```
void func (int len, ...)
  char buf [64]
  if (len > 64)
    return
  memcpy (buf, ...)
```
Signed comparison! exploitation: len = -1

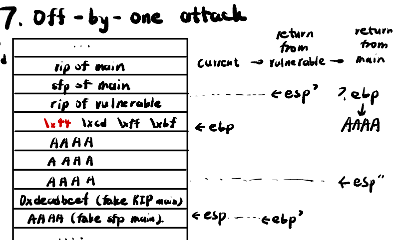
```
void func (size_t len)
  char *buf = malloc(len+2)
  if (buf == NULL)
    return
  buf [len+1] = '\0'
```
if len = 0xffffffff, len+2 = 1
Causing heap overflow!

5. Format string attack



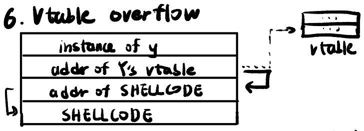
char buf [16]
char str [12]
fgets (buf, 16, f)
*Code: printf (buf)

7. Off-by-one attack



*Code: void vulnerable () {
char name [20];
fgets (name, 21, f);
}

6. Vtable overflow



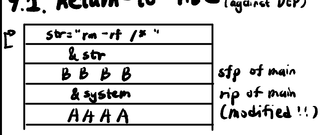
* Fix: correct way to call fgets: {
bytesRead = fgets (buf, 1, sizeof (buf) - 1, f)
buf [bytesRead] = '\0'
}

8. Mitigating memory vulnerabilities.

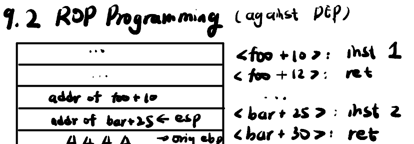
1. Safer language. (C#, Java...) 2. Safer function calling (fgets) 3. Stack canary
4. Non-executable Pages (W^X, DEP.) 5. PAC (E.g. for x64, use high 22-bits for MAC)
6. ASLR 7. Run-time check / monitor code behavior / run in sandbox

9. Subverting Mitigations.

9.1 Return-to-libC (against DEP)



9.2 ROP Programming (against DEP)



result: execute inst 1, inst 2

9.3 MISC (ret2ret, ret 2pop)

10. Introduction to cryptography.

- Confidentiality • Integrity • Authenticity. • Correct, Efficient, Secure
- Threat-Model: Ciphertext only / CPA / CCA / CPCA. *: CCA < CPA (weaker).

11. IND-CPA Game.

- Choose $len(M_0) = len(M_1)$. Can't guess $C = C_1$ or C_2. $Pr(success) = \frac{1}{2} + o(\frac{1}{2^n})$

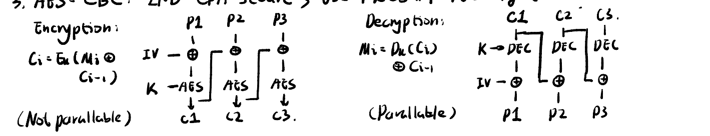
12. One-time-padding

- Keygen() ⇒ n-bit key; $Enc(K, M) = K \oplus M$; $Dec(K, C) = K \oplus C$

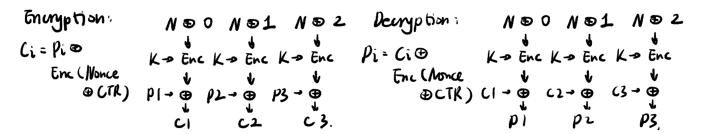
Problem: knows M_0 xor M_1 if $K_1 = K_2$ ⇒ not IND-CPA if $K_1 = K_2$; random is expensive; key exchanging

13. AES, ECB, CBC, CTR & CFB.

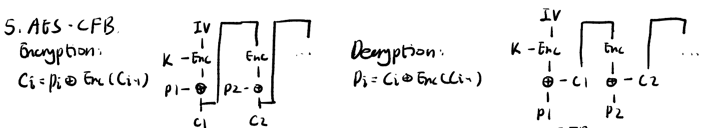
1. AES: looks like random permutation; can't tell diff in distinguishing games (from real random nums)
NOT IND-CPA (deterministic)
Key = 128 / 192 / 256; Block = 128 bit
2. AES-ECB: AES on each block
3. AES-CBC: IND-CPA Secure; Use PKCS #7 Padding (n n n ... n) (n×n)



4. AES-CTR



5. AES-CFB



6. Key-Reusing: Leaks some info for CBC, catastrophic for CTR. (can start with same blocks)

14. Hash

1. Properties: ① One-wayness: hard to find x s.t. $H(x) = y$. (E.g. $H(x) = 42$ is not one-way)
② Collision-resistant: hard to find x s.t. $H(x) = H(x')$, $x \neq x'$
Eg. $H(x) = x$ is collision resistant (birthday problem)
Remark: only need $2^{\frac{n}{2}}$ tries to collide n-bit output
2. Examples of hash: MD5 (128 bit), SHA-1 (Completely Broken). (160 bit)
SHA-2 (Vulnerable to length extension, $H(M \| M')$)
SHA-3 (Secure now). ; SHA 2/3 both 256/384/512 bits
3. Hash SOMETIMES provides integrity (if everyone knows correct hash).

15. MACs.

1. EU-CPA: existentially unforgeable: can't forge tag without key.
Game: Mallory ← Alice $\forall (M_i, T_i)$; Mallory → Alice M', T' where $M' \neq M_i$
2. NMAC: KeyGen() → K_1, K_2 (n-bits, same as output length).
$NMAC(K_1, K_2, M) = H(K_1 \| H(K_2, M))$.
EU-CPA Secure if $K_1 \neq K_2$ and H is secure.
Intuition: use 2 hashes against length extension attack
(otherwise, can know $NMAC(M \| M')$, not EU-CPA secure.
3. HMAC: First compute. $K' = \begin{cases} H(K) & \text{if } K \text{ is too long} \\ K \| \backslash x00 & \text{if } K \text{ is too short (careful! K can't be too short)} \end{cases}$
Output $H((K' \| opad) \| (K' \| ipad) \| M)$.
opad → 0x5c, ipad → 0x36 → repeated until its same length as K'
4. Properties: Integrity Yes, No authenticity.
Not IND-CPA Secure, Usually leaks information (HMACs DONT)

16. Authenticated encryption

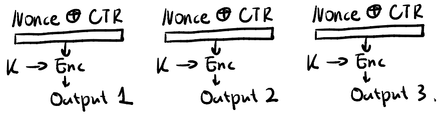
1. Enc (K_1, M) and MAC $(K_2, Enc(K_1, M))$. ✓ Good.
2. Enc $(K_1, M \| MAC(K_2, M))$ → allows side-channel attacks!!!
3. Don't reuse keys.!

17. PRNGs.

1. Properties: deterministic, Computationally indistinguishable from true randomness (consider distinguishing game! $Pr(success) < \frac{1}{2} + \frac{1}{2^n}$
✱ can't infer future output by previous ones
✱ Roll back resistance: knowing PRNG's internal state, can't infer previous output.
2. Functions: seed (init); reseed (add entropy); generate.

3. Example1: CTR-DRBG. → n random bits.

Gen(): → $E_K(IV \| 1) \mid E_K(IV \| 2) \mid \cdots \mid E_K(IV \| \text{ceil}(\frac{m}{128}))$



4. HMAC-DRBG.

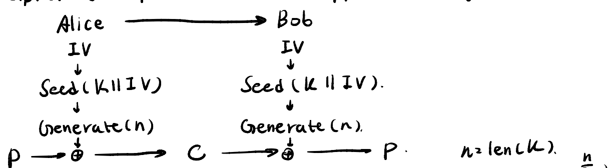
Seed(s): K := 0, V := 0; Reseed(s).

Reseed(s): K := HMAC(K, V‖0x00‖S), V := HMAC(K, V).
K := HMAC(K, V‖0x01‖S), V := HMAC(K, V).

Generate(n): while len(out) < n: { V := HMAC(K, V), Output += V }
K := HMAC(K, V‖0x00‖S), V := HMAC(K, V).

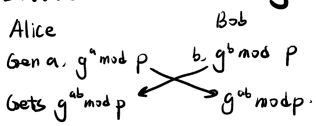
Example application: uuid

5. Stream-Cipher (example: AES-CTR; supports seeking).



Problem: may lose randomness if too much bytes encrypted. ($> 2^{\frac{n}{2}}$)

18. Diffie-Hellman Key Exchange



Properties: forward secrecy (if a, b, K are discarded after use)

Problem: MITM Attack (Alice ⟺(DH) Mallory ⟺(DH) Bob).

Extension: ECDH (elliptic-curve DH, harder than discrete log problem). 384 bit ECDH ~ 3072 bit DHE

19. ElGamal Encryption.

Keygen(): Bob → b, Pub key g^b mod p.

Enc(B, M): Alice → r, $R = g^r$ mod p, Send: ($C_1 = R$, $C_2 = M \times B^r$ mod p).

Dec(b, C_1, C_2): Bob ← $C_2 \times C_1^{-b} = M \times B^r \times R^{-b} = M$ mod p.

Problem: malleability (can be tampered with).

20. RSA Encryption.

- KeyGen(): Random choose p, q; test if prime;
  Calculate $N = pq \in [2048, 4096]$ bit length.
  Choose e (relatively prime to $(p-1)(q-1)$).
  Compute $d = e^{-1}$ mod $(p-1)(q-1)$ (Extended Euclidean)
  Pub key: N, e; Priv key: d.
- Enc(e, N, M): M^e mod N // Dec(d, C): C^d mod N
- Correctness Proof:
  ①. Chinese remainder $x \equiv y$ mod p, y mod q ⇒ $x \equiv y$ mod pq
  ②. $a^{p-1} = 1$ (mod p) (Fermat's Little)
  - $M^{ed} = M^{ed-1} M = M^{k(p-1)(q-1)} M = M^{c(p-1)} M = 1 \times M = M$
  ③. Euler's thm: $a^{\varphi(N)} \equiv 1$ (mod N).
  if N = prime, $\varphi(N) = N - 1$ (Fermat's little thm).
  $N = pq$, $\varphi(N) = (p-1)(q-1)$.
  - Notice $ed = 1$ (mod $\varphi(N)$) ⇒ $M^{ed} = M^{k\varphi(N)+1} = M$
- RSA Problem: Given N & M^e mod N. hard to find M
  Best solution: factor N. exponential time.
- Remark: RSA is deterministic. Must use RSA-OAEP. ⇒ IND-CPA secure.

21. RSA Signatures

- Sign(d, M) ⇒ $H(M)^d$ mod N;
  Verify(e, N, M, sig) ⇒ return. $H(M) == sig^e$ mod N

22. Tampering with Pub Key Distribution

Alice	Mallory	Bob.
Recv PK_M ⟵	PK_M ← PK_B ⟵	PK_B, SK_B
Send $\{M\}_{PK_M}$ ⟶	$\{M\}_{PK_M}$ ↓ $\{M\}_{PK_B}$ ⟶	Decrypt $\{M\}_{PK_B}$

23. Trust-Anchor (CA).

1. certificate: Identity + Pub key.
   example: suppose Eve is trust anchor. (trust PK_E).
   valid CA: {"Bob's public key is PK_B"}$_{SK_E}$
   abbreviation: encryption: {"Msg"}$_{PK}$; signing: {"Msg"}$_{SK^{-1}}$
2. Trusted Directory: from where to get anyone's public key.
   Suppose everyone knows PK_{TD}.

   Problem: Scalability + Single point of failure.
   Solution: Hierachical Trust.
   Problem: Revocation
   Solution: Expiration Date + Certification Revocation List (CRL)
   *: Shorter exp date, shorter CRL!.
3. Another method: Trust on First Use (example: SSH fingerprint)
   Never allow pub keys change (warn if it does).

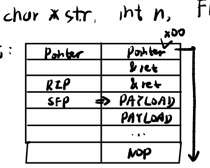
24 Password Storing

1. Store hashes: Problems: can see which user use same pwds.
   can't prevent dictionary attack (common pwd hashes).

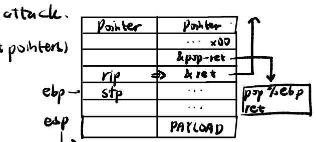
   Solution: hash(pwd, salt).
   Proof: Suppose M pwds, N users.
   w/o salt: hash all possible pwd & lookup N hashes. O(M+N)
   With salt: hash each salt's pwd ⇒ O(MN).
2. Use slower hash functions.
   Offline attack prevention: slow + Salted. functions. (Argon2Key)
   Online attack prevention: time-out defences.

25. MISC Supplemented.

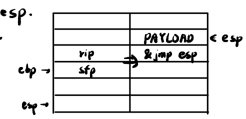
1. fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
   (size_t size: element size; nmemb: element num)
   * No "\0" is filled
2. fgets(char *str, int n, FILE *stream): read n-1 Bytes + fill "\0"
3. ret2ret attack (overwrite perfect pointers)



4. ret2pop attack. (use perfect pointers)



5. jump2esp. attack (objdump & grep & disas)



6. ret2eax attack (Suppose eax can be written by string func)