# x86

0x FFFF FFFF

| | |
|---|---|
| stack ↓ | function frames (RIP/SFP/local vars) |
| ↑ heap | dynamically allocated variables (malloc) |
| static | static variables (const. strings, static mem. buffers) |
| code | x86 instructions where EIP & RIP usually point |

0x 0000 0000

after RIP SFP

| | |
|---|---|
| EIP | instruction |
| ESP | top of the stack (lower addr) |
| EBP | base of current frame (higher addr) |

RIP : old EIP

SFP : prev. func.'s EBP

push puts reg value on stack & moves esp lower

pop puts value @ esp into reg

④ off by one vuln: write one byte past buf into LSB of SFP so it points into buf -> shlcd

⑥ ret 2 ret

| | |
|---|---|
| RIP main | little-endian: least significant byte is stored at the lowest mem |
| SFP main | 0x DEADBEEF |
| arg 2 | EF BF AD DE |
| arg 1 | |
| RIP foo | |
| SFP foo | |
| buf[5:8] | |
| buf[0:4] | |

1. push args onto stack (reverse)
2. push old EIP (RIP) onto stack
3. update EIP
4. push old EBP (sfp) onto stack
5. move EBP down to SFP
6. move ESP down for new frame
7. run function.
8. move ESP up to EBP
9. restore old EBP by popping SFP.
10. restore EIP pop RIP
11. remove arguments from stack by moving ESP up.

## MEMORY SAFETY ATTACKS

① buffer overflow

vuln: code uses unsafe gets, read, etc instead of fgets, fread can write to any region above buf (auth bool, *fnptr injection)

② stack smashing vuln: buffer overflow to overwrite RIP to point to shellcode. when func returns, exec will jump to RIP addr.

③ integer conversion vuln: checking len < 8, passing -1 (0xFF...) and it being interpretted as an unsigned int later (2s prev.)

⑤ format string vuln:

| | | |
|---|---|---|
| %c | ingests one char of args, | VALUE |
| %k u | prints as unsigned int & adds whitespace before to display k total characters | VALUE |
| %s | derefs & prints val as string | PTR |
| %n | writes # of bytes that have been printed as a 4-byte # to the mem addr in arg | PTR |
| %hn | same but 2 byte word | PTR |
| %x | prints words in hex | VALUES |

## defenses

① stack canary

RIP foo
SFP foo
CANARY
buf[4:8]
buf[0:4]

- can still heap overflow
- local vars can still be overwritten
- if atk happens before return, canary not checked
- can be leaked
- canary checked when return

② ASLR randomize the start of each segment of memory

- relative addresses still preserved
- if one stack addr leaked, other addresses can be determined
- can be subverted with ROP: return-oriented programming which allows you to look for useful segments of code called gadgets that can allow you to perform specific attacks

confidentiality: can't read
integrity: can't change
authenticity: can verify sender

IND-CPA:
1. eve sends alice $M_0, M_1$
2. alice randomly encrypts & sends one
3. eve guesses 0.5

deterministic -> not IND-CPA secure

1-time pad:
gen random n-bit key
enc = dec = K ⊕

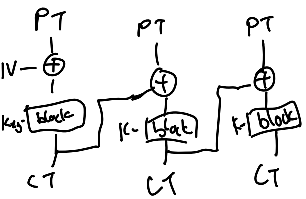
block ciphers by itself:
- not IND-CPA secure
- can't handle non-fixed size

ECB mode:
encrypt block-sized chunks
still not secure

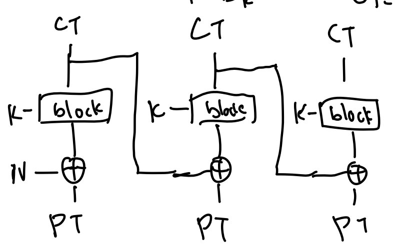
IV/nonce: random bits, public, not reuseable

CBC mode:

ENC $C_i = E_K(M_i \oplus C_{i-1})$, $C_0 = IV$



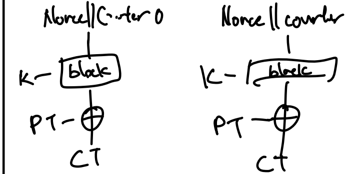

DEC $M_i = D_K(C_i) \oplus C_{i-1}$




non-exec pages
- local data manipulation still possible
- ROP may bypass non-exec pages

XOR
$1 \wedge 0 = 1$, $0 \wedge 0 = 1 \wedge 1 = 0$
commutative: $X \wedge Y = Y \wedge X$
associative: $X \wedge (Y \wedge Z) = (X \wedge Y) \wedge Z$
$X \wedge X = 0$
$0 \wedge Y = Y$

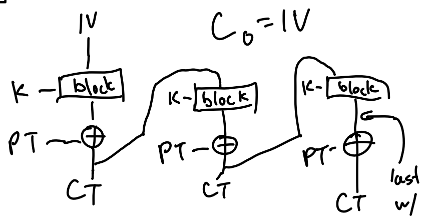
CTR (counter) can be parallel




DEC swap PT & CT

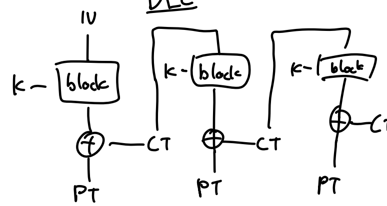
CFB mode: $C_i = E_k(C_{i-1}) \oplus M_i$
$C_0 = IV$

ENC




last block, just XOR w/ the necessary # of bytes from encrypted ciphertext, thus no padding needed

DEC




$M_i = E_k(C_{i-1}) \oplus C_i$

hashes
- H(M): M is arbitrary length outputs fixed length n-bit hash
- looks "random" - fast
- one way: hard to find x given a y such that $H(x) = y$
- collision-resistant: hard to find $x \neq x'$ such that $H(x) = H(x')$

MAC
KeyGen() -> k
MAC(K, m): generates tag T (fixed len)
properties:
- correctness: deterministic
- eff
- security EU-CPA (atk cannot create a valid tag on M w/o k)

diffie hellman
Alice & Bob agree on large prime $P$ and generator $G$, $1 < G < P-1$

Alice picks $a$, computes
$A = g^a \mod p$
Bob picks $b \to B = g^b \mod p$
announce A & B
Alice calculates $B^a = g^{ab} \mod p$
Bob calculates $A^b = g^{ab} \mod p$

relies on discrete log problem: given g & p & $g^a \mod p$, can't find a

throw away ab & S when done so forward secrecy

Pointer authentication
unused bits of 64 bit system are set to authentication bits like a canary for an address

| 48 bit addr | 16 bit PAC |

- can trick CPU into gen PAC
- brute force able

public key cryptography

A & B don't need to share key
but much slower

el gamal

Bob announces $B = g^b \bmod p$

Alice sends $C_1 = R = g^r \bmod p$
and $C_2 = M \times B^r \bmod p$

Bob calculates
$$C_2 \times C_1^{-b} = M \times B^r \times R^{-b}$$
$$= M \times g^{br} \times g^{-br}$$
$$= M \bmod p$$

# RSA

Alice | Bob

$\boxed{M} \to \boxed{C = M^e \bmod N} \to \boxed{C} \to \boxed{C^d \bmod N = M^{ed} \bmod N = M}$

key gen

- large primes $p$ & $q$
- $N = pq$
- choose $e$ that is relatively prime to $(p-1)(q-1)$
- compute $d = e^{-1} \bmod (p-1)(q-1)$ using EEA
- public key: $(N, e)$
- private key: $d$