Segurança da Computação

Matheus Venturyne Xavier Ferreira
Universidade Federal de Itajubá
21 de Outubro de 2015

Network Security

- Aspectos
  - Confidencialidade: acesso apenas aos autorizados
  - Integridade: somente alterações autorizadas
  - Autenticação: é realmente quem/o que afirma se?
  - Disponibilidade: posso acessar o serviço?
- Ameaças
  - Interceptação: espionagem
  - Interrupção: destruição, DoS (Denial of Service)
  - Modificação: inserir dados ou programas
  - Fabricação: novo data/programa, respondendo mensagens

Autenticação

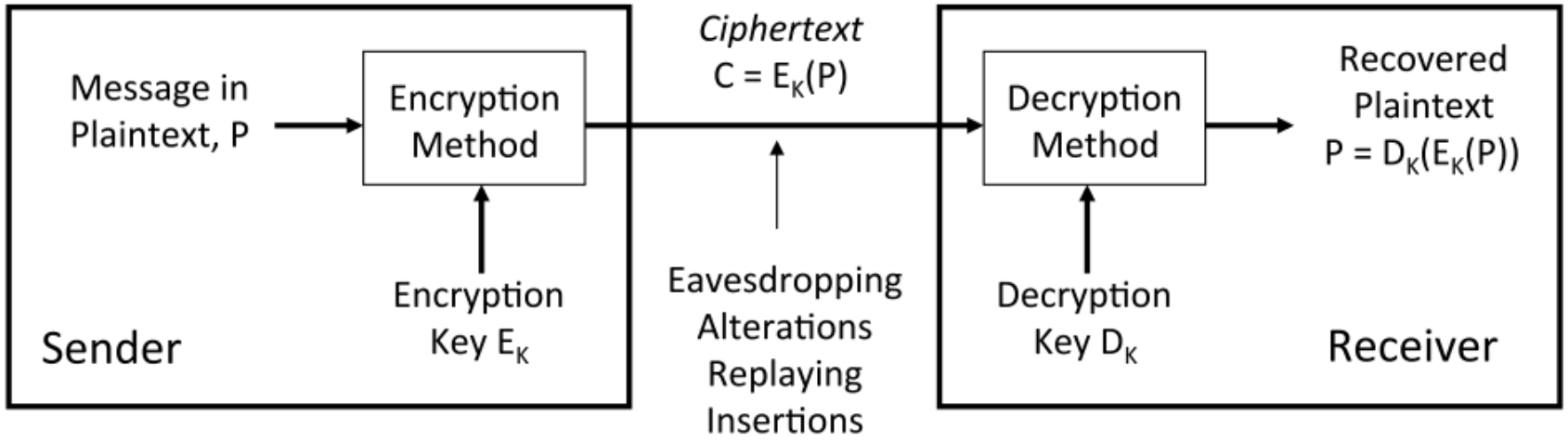
- Senhas são o método mais comum
  - Usuário e o computador conhecem o segredo
  - Usuário prova o conhecimento do segredo
  - Computador checa
- Senhas criptografadas
  - Computador armazena apenas a senha criptografada
  - Usuário fornece a senha
  - Computador criptografa e checa

Problema das senhas

- Assuma 1 msec para checar senhas alfabéticas
- 5 chars 52^5 =360 milhões 6.3 minutos
- 6 chars 52^6 = 19.7 bilhões 5.5 horas
- 7 chars 52^7 = 1.03 trilhões 12 dias
- 8 chars 52^8 = 53.5 trilhões 1.7 anos
- Mas a maioria é incomum, difícil de lembrar
- Usando 200000 palavras do dicionário: 0.2 sec!

Criptografia

- Codificar mensagens para
  - Limitar quem pode ver a mensagem original
  - Determinar quem enviou a mensagem



Secret Key Criptosystem

- Chave secreta (simétrica)
  - Mesma chave K é usada para criptografar e de-criptografa
  - Sender criptografa $E_k(P)$, Receiver de-criptografa $D_k\big(E_k(P)\big)$
- DES: Data Encryption Standard (1977)
  - Fraca por usar chaves de 56-bit
- AES: Advanced Encryption Standard (2001)
  - Chaves de 128, 192, 256-bit

Public Key Encryption

- Chave pública (assimétrica)
  - Chaves diferentes para criptografa e de-criptografa
  - Cada usuário tem duas chaves: uma pública, uma privada
- Se A quer enviar para B
  - A criptografa usando a chave pública de B
  - B de-criptografa usando sua chave privada
- RSA (Rivest, Shamir, and Adelman)

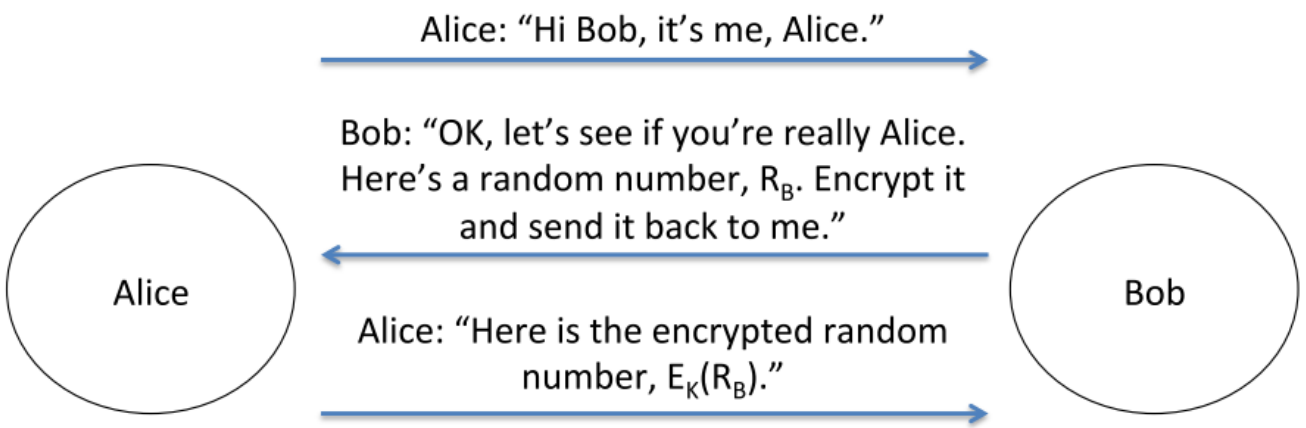
Public Key vs. Secret Key

- Chave secreta
  - Operação rápida
  - Difícil distribuir chaves
- Chave pública
  - Operação lenta
  - Conveniente para distribuição de senhas
- Combinação
  - Chave publica para iniciar seção: trocar chave secreta
  - Durante seção, se usa chave secreta

Autenticação Usando Chave Secreta

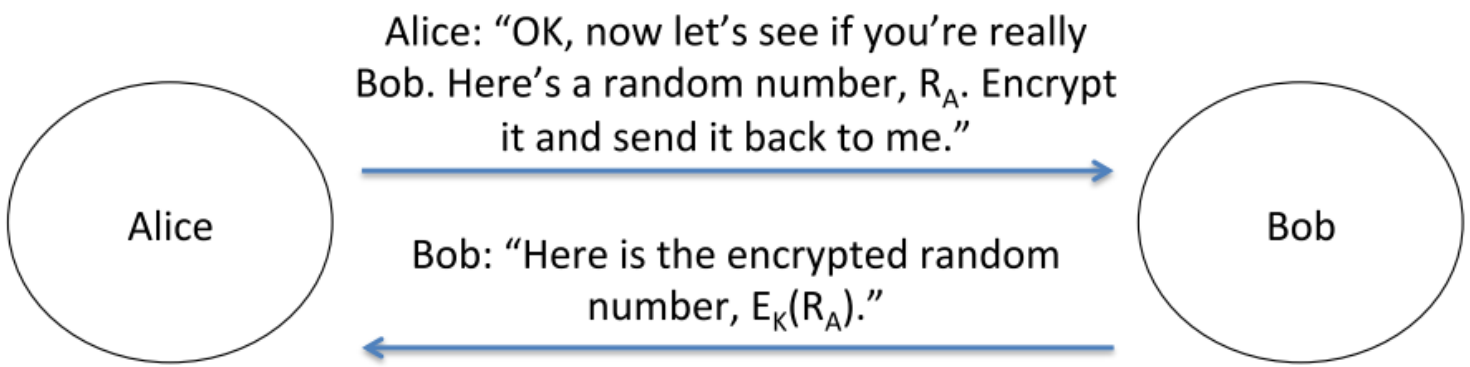
- Alice envia para Bob identificador: “Hi Bob, it’s me, Alice”
- Bob envia desafio para Alice: número randômico R_B
- Alice criptografa R_B usando K: $E_K(R_B)$; envia para Bob
- Bob de-criptografa $E_K(R_B)$ usando K; deve se Alice
- Alice envia desafio para Bob: número randômico R_A
- Bob criptografa R_A usando K: $E_K(R_A)$; envia para Alice
- Alice de-criptografa $E_K(R_A)$ usando K; deve ser Bob

Bob autentica Alice



Bob decrypts $E_K(R_B)$ and gets R_B. Since this is what he sent to Alice, and only he and Alice know the key K, only Alice could have sent R_B. And since R_B was never and will never be sent again, it could not be a replay of a previous message. Bob now knows he’s talking to Alice. But Alice doesn’t know if she’s really been talking to Bob.

De forma Similar, Alice autentica Bob

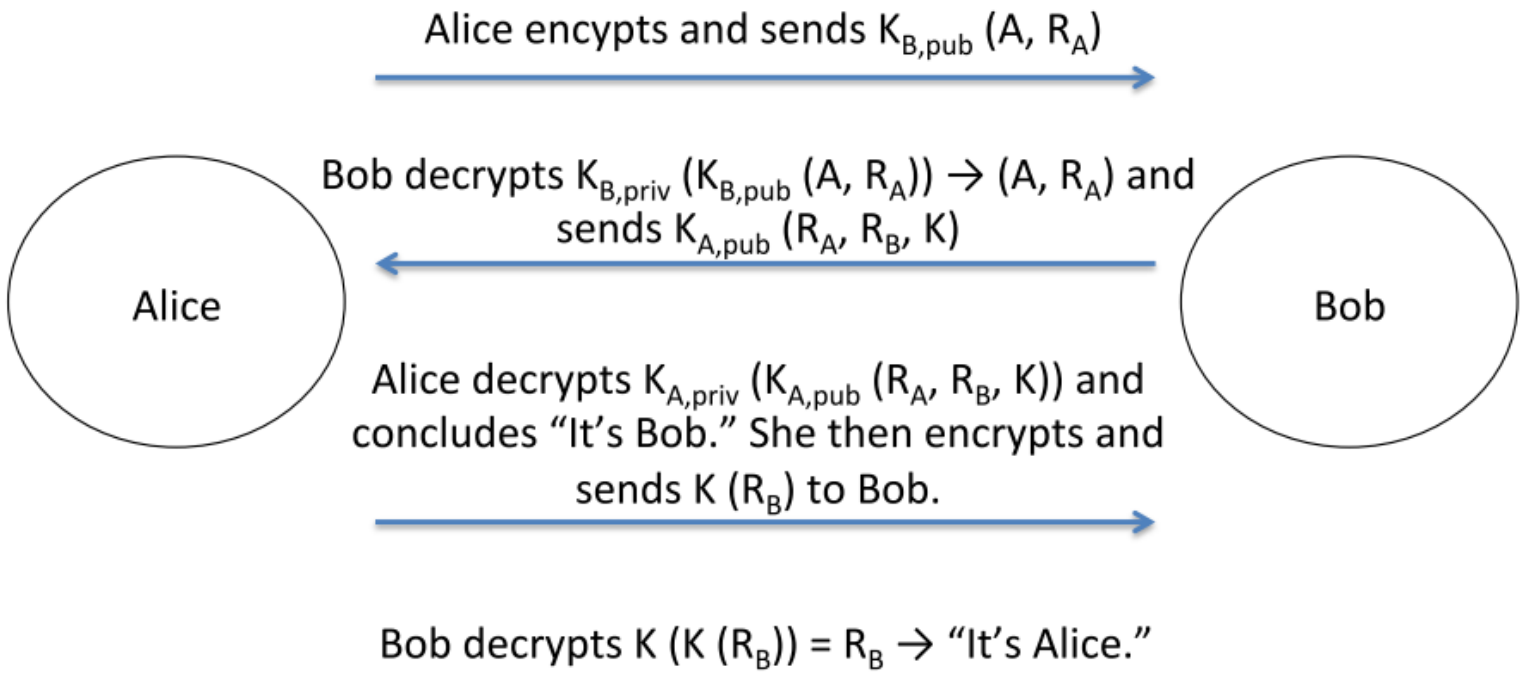


Alice decrypts $E_K(R_A)$ and gets R_A. Since this is what she sent to Bob, and only she and Bob know the key K, only Bob could have sent R_A. And since R_A was never and will never be sent again, it could not be a replay of a previous message. Alice now knows she's been talking to Bob.

Autenticação Usando Chave Pública

- Alice
  - Envia $K_B, pub(A, R_A)$ para Bob (usa chave pública de Bob)
- Bob
  - De-criptografa: $K_B, priv\big(K_B, pub(A, R_A)\big) \rightarrow (A, R_A)$
  - Criptografa e envia $K_{A,pub}(R_A, R_B, K)$ para Alice
- Alice
  - De-criptografa: $K_{A,priv}\left(K_{A,pub}(R_A, R_B, K)\right) \rightarrow "it's\ Bob"$
  - Criptografa e envia $K(R_B)$ para Bob
- Bob
  - De-criptografa $K\big(K(R_B)\big) = R_B \rightarrow "It's\ Alice"$

Autenticação Usando Chave Pública

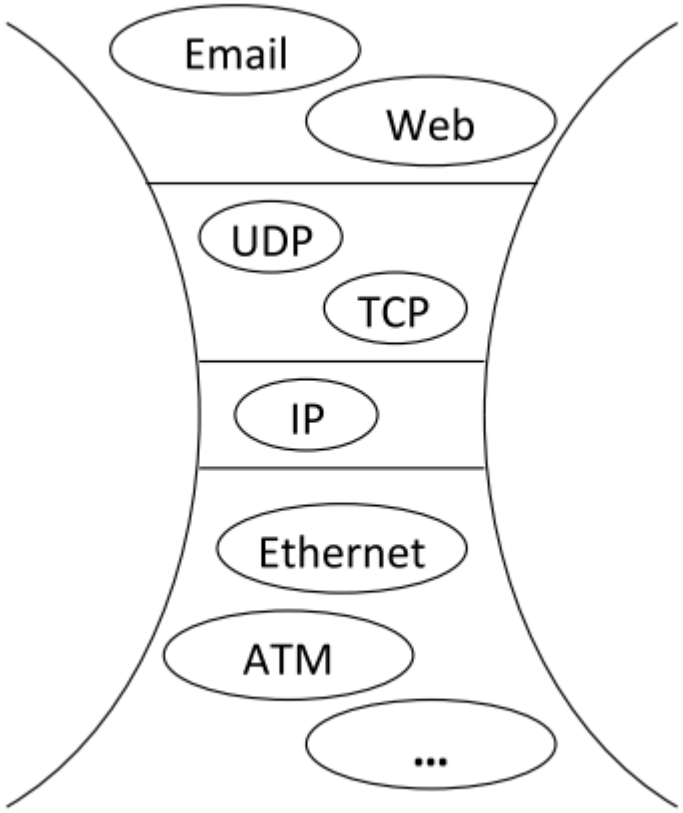


Assinaturas Digitais

- Se Alice quer assinar digitalmente uma mensagem para Bob
  - Criptografa M usando $K_{A,priv}$ e envia $K_{A,priv}(M)$ para Bob
- Quando Bob recebe, de-criptografa usando $K_{A,pub}$
  - Pode de-criptografa somente se veio de Alice
- Para assinar e manter privado
  - Alice envia $K_{B,pub}(M, K_{A,priv}(M))$ para Bob
  - Somente Bob pode criptografar: $K_{B,priv}(K_{B,pub}\left(M, K_{A,priv}(M)\right))$
  - De-criptografa usando $K_{A,pub}$ provando que Alice assinou

Networking

- IP: Qualquer dispositivo que fale IP pode se conectar a camada física e se conectar à rede.
- Se adiciona um endereço (chamado IP address) único a cada entidade na rede.
- Ipv4: 32 bits
- Ipv6: 128 bits
- Packet switched vs circuit switched (phone)



src → r1 → r2 → r3 → dst

Routing

- TTL (Time to Live): quantos roteadores um pacote é permitido passar ou caso contrário é abandonado (threw in the floor)
  - Evita que pacotes se acumulem na rede quando há existência de ciclos
- Best Effort: tenta entregar o pacote mas IP não fornece nenhuma garantia
- Algoritmos: Distance-Vector, Link-State Routing, BGP (Border Gateway Protocol)

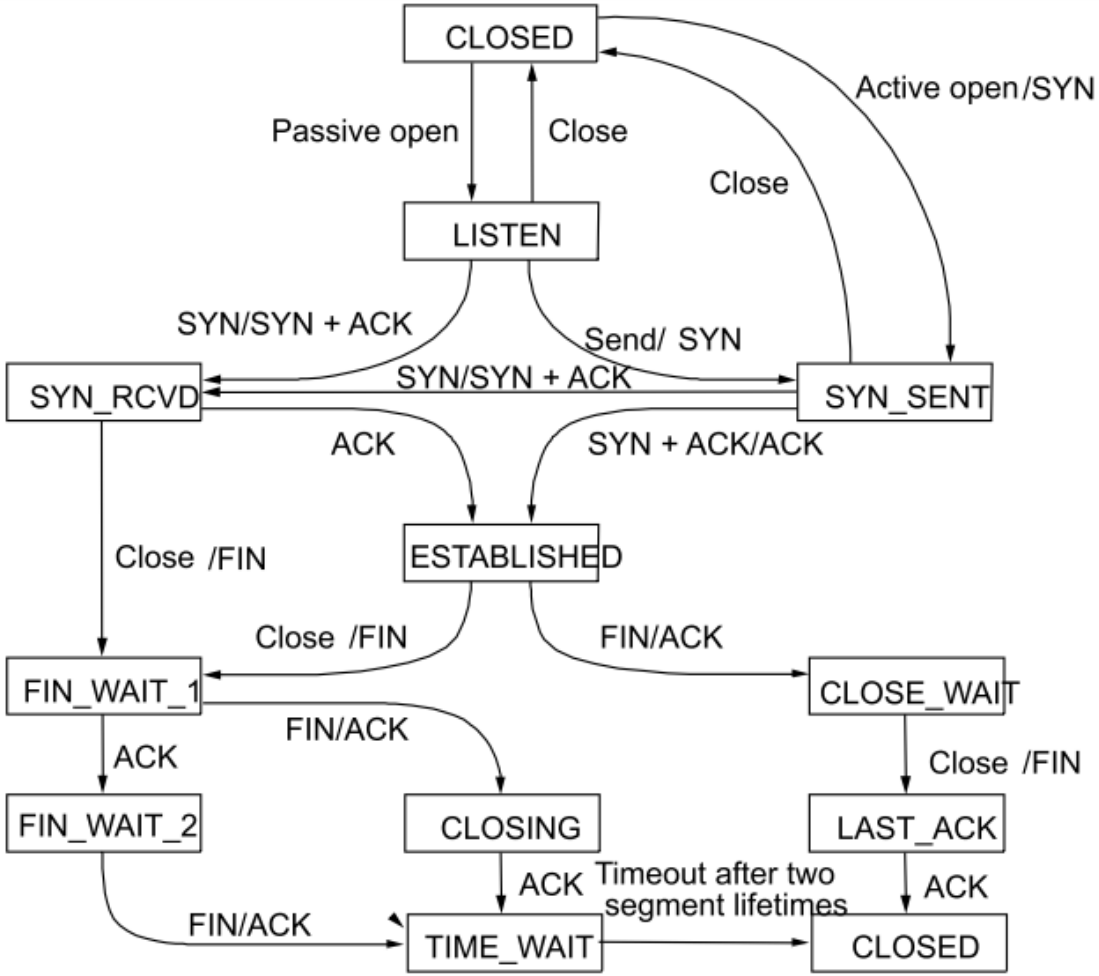
Transport Layer

- Portas (TCP/UDP)
  - HTTP: 80/TCP
  - HTTPS: 4343/TCP
- Nada força a ideia de cliente/servidor. Toda inteligência está nas extremidades.
- Sem decisões centralizadas
- UDP
  - Datagram-oriented
  - Não garante entrega
  - Possivelmente entrega fora de ordem (pode receber mais de uma cópia)
  - Sem controle de congestionamento

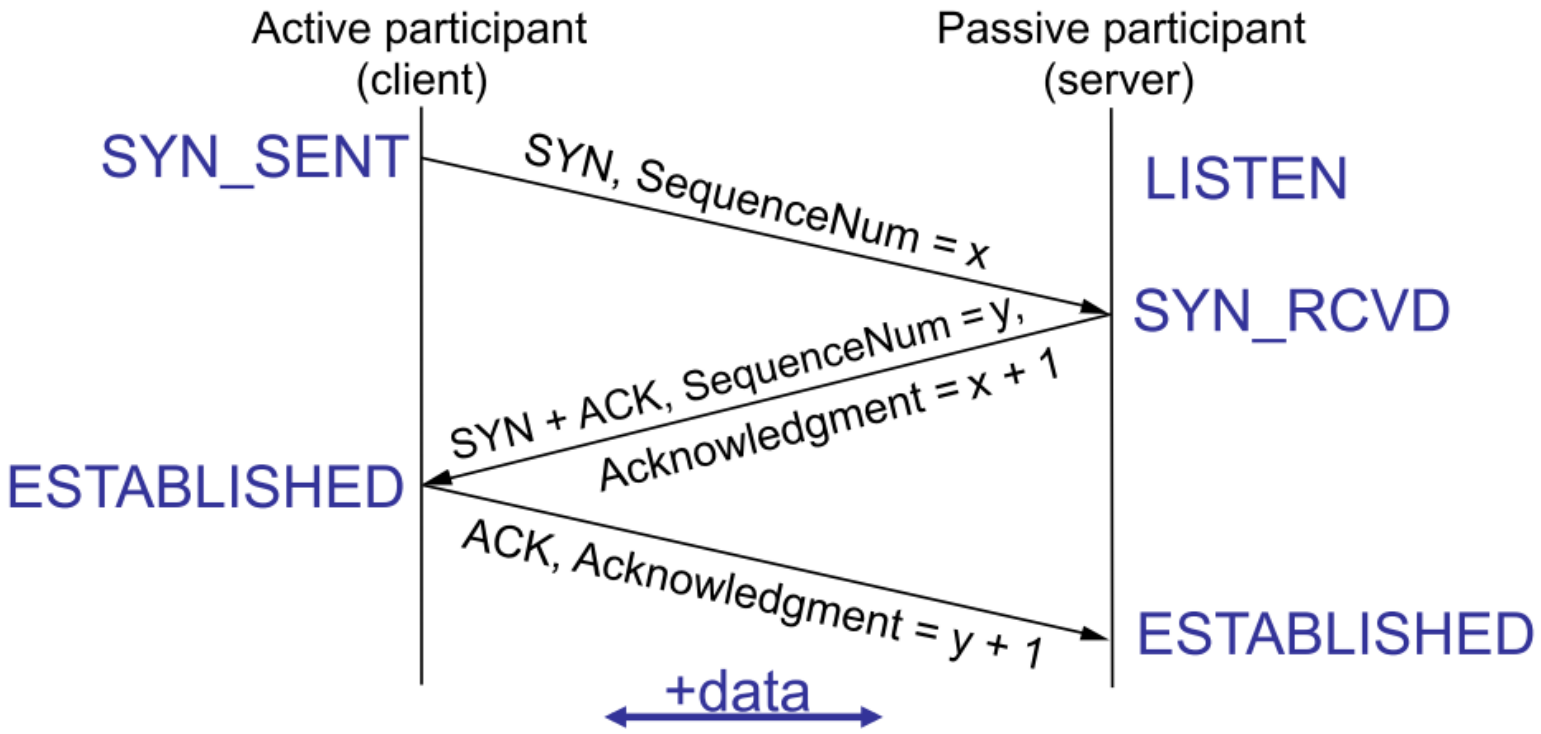
Transport Layer

- TCP
  - Garantia de entrega
  - Orientado a conexão
  - Orientado a byte stream
  - Possui controle de congestionamento

TCP Transição de Estados



TCP - SYN



Firewalls

- Sistema de segurança, hardware ou software, que inspeciona pacotes fluindo para e de dispositivos da rede. Checa as concordâncias dos pacotes com um conjunto de regras e decide se o pacote pode ser enviando para o destino
- Níveis de operação:
  - Filtro de pacote: baseado no conteúdo do cabeçalho dos pacotes
  - Inspetor de estado: baseado no estado de uma conexão TCP
  - Filtros na camada de aplicação: baseado na informação da camada de aplicação carregada em cada pacote

Firewalls

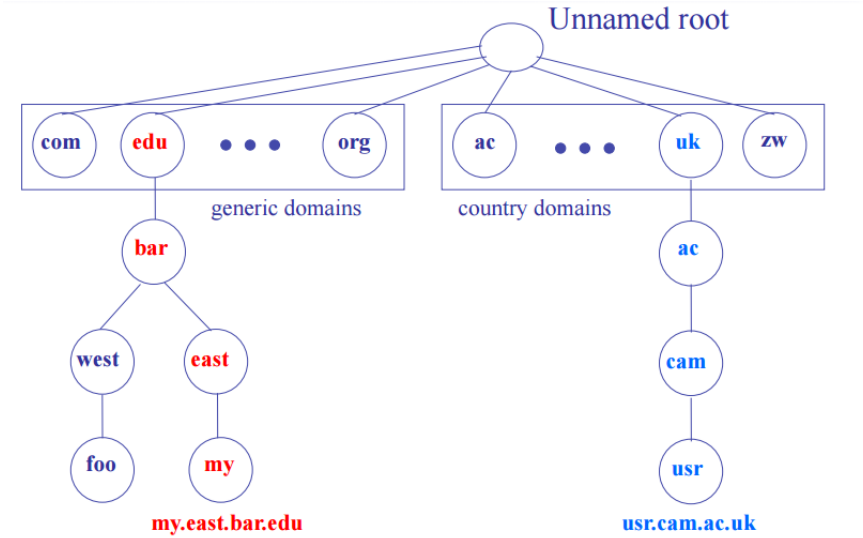
- Regras
  - Deny tcp from 20.0.0.0/8 to any 22 out
    - 22 (SSH port)
  - Allow tcp from any to 20.0.0.0/8 80
    - 80 (HTTP port)
  - Deny tcp from any to 20.0.0.0/8 in
  - Allow tcp from any to any

NAT (Network Address Translation)

- Forma de enganar todo o tráfico saindo e entrando da rede como se eles se originaram e destinados para um único endereço IP
- Todo a rede local utiliza um único endereço IP
  - Pode alterar o endereço de qualquer dispositivo na rede local sem notificar o mundo externo
  - Pode alterar ISP (Internet Service Provider) sem alterar os endereços dos dispositivos na rede local
  - Dispositivos dentro da rede local não podem ser explicitamente endereçados (não são visíveis do mundo exterior)
- Endereços Privados
  - 10.0.0.0/8 (16.777.216 dispositivos)
  - 172.16.0.0/12 (1.048.576 dispositivos)
  - 192.168.0.0/16 (65.536 dispositivos)

DNS (Domain Name Server)

- Conversão de nomes para endereços IP (phonebook)
- Hierarquia de nomes
- Caching (Evita sobrecarga)
- Todo pedido é acompanhado de uma query ID

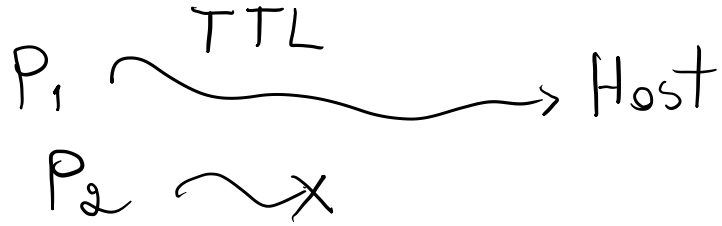


IDS (Intruder Detection System)

- Princípio end-to-end: as partes inteligentes na internet estão nas extremidades
- Conjunto de regras executando tentando detectar um padrão de um ataque
- Deseja-se identificar o que está acontecendo na rede
- Inspeção passive de pacotes
  - Ataques ao roteador
  - Estudar pacotes, procurar por ataques
- O que fazer ao detector?
  - Importante detector sedo
- Dificuldade
  - Portabilidade

IDS - Problemas

- Tentar entender comportamentos de alto nível com informação de baixo nível
- Ao receber um pacote para um alvo A
  - O alvo irá receber o pacote?
  - Como o alvo interpreta o pacote?

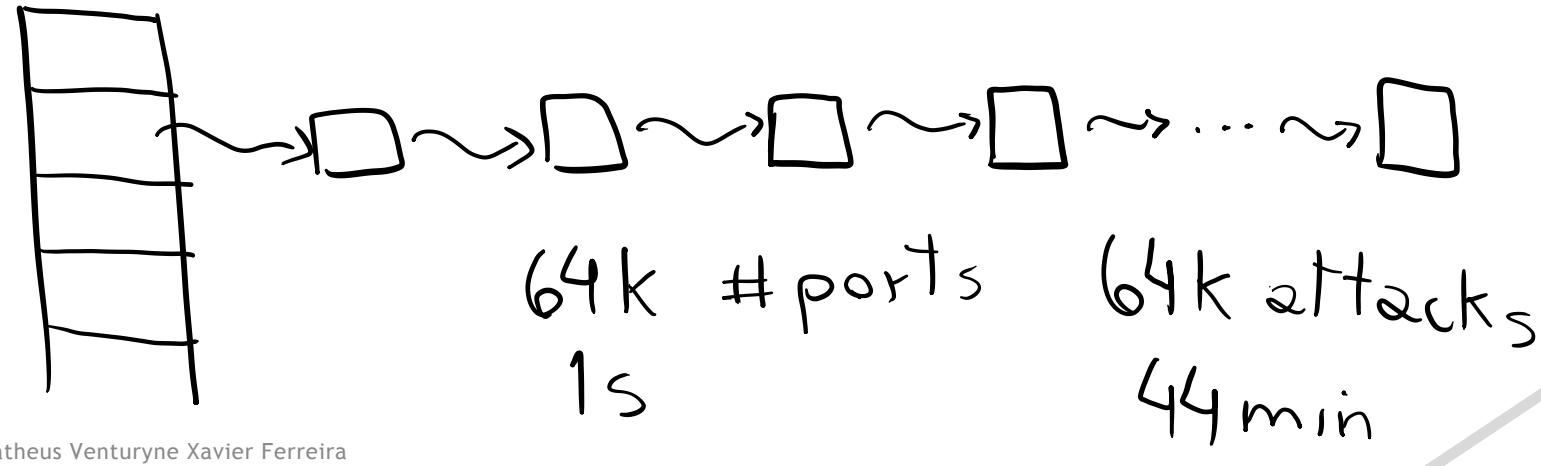


Detector de Intruso - Problemas

- Checksum
  - Em geral se o checksum esta errado o pacote não deve ser processador; no entanto, alguns sistemas operacionais podem processor o pacote mesmo se o checkum está errado
- Máquina/Roteador sem memória (sobrecarga no IDS)
  - Para pacotes (Fail-shut)
    - Sobrecarregar uma máquina agora ode derrubar toda a rede (amplified DoS)
  - Permitir pacotes (fail-open)
    - Se quer atacar só é necessário sobrecarregar o IDS e iniciar o ataque

IDS - Port Scan

- Deseja-se scanner todas as portas de um computador para saber quais portas estão abertas
- IDS utiliza uma hash table para rastrear as entradas da forma <src IP, dst port>
- Hash function é previsível (algo não desejável na presença de um invasor)]

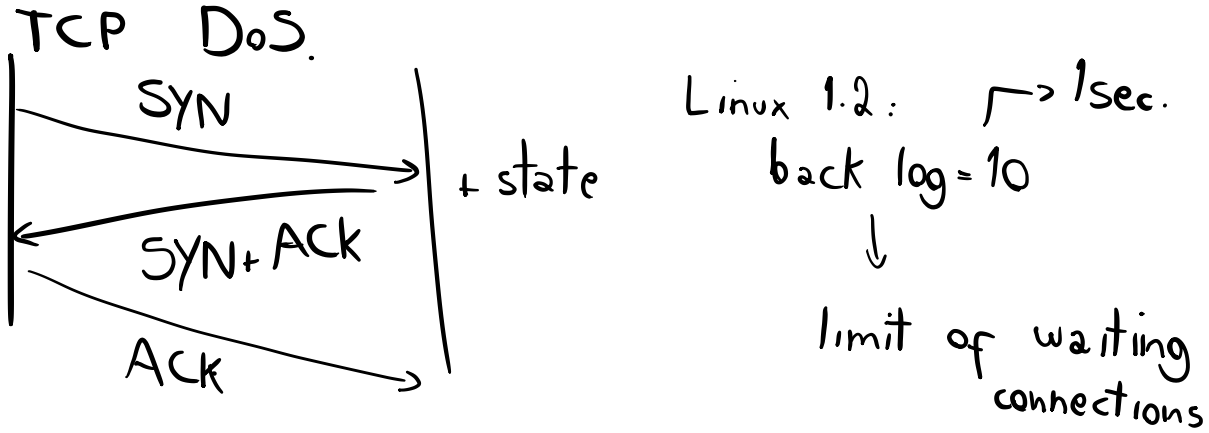


DoS (Denial of Service)

- Ataque à disponibilidade do Sistema
  - Vulnerabilidades lógicas
    - Teardrop: envia-se de um único pacote que faz o kernel seg. Fault (Microsoft)
  - Enviar um excesso de dados para o Sistema
    - Amplificação: em conexões UDP alguns serviços podem receber um pedido e enviar uma resposta com muitos dados

DoS – TCP SYN

- Envia várias requisições de conexão TCP, bloqueando futuras conexões legitimas

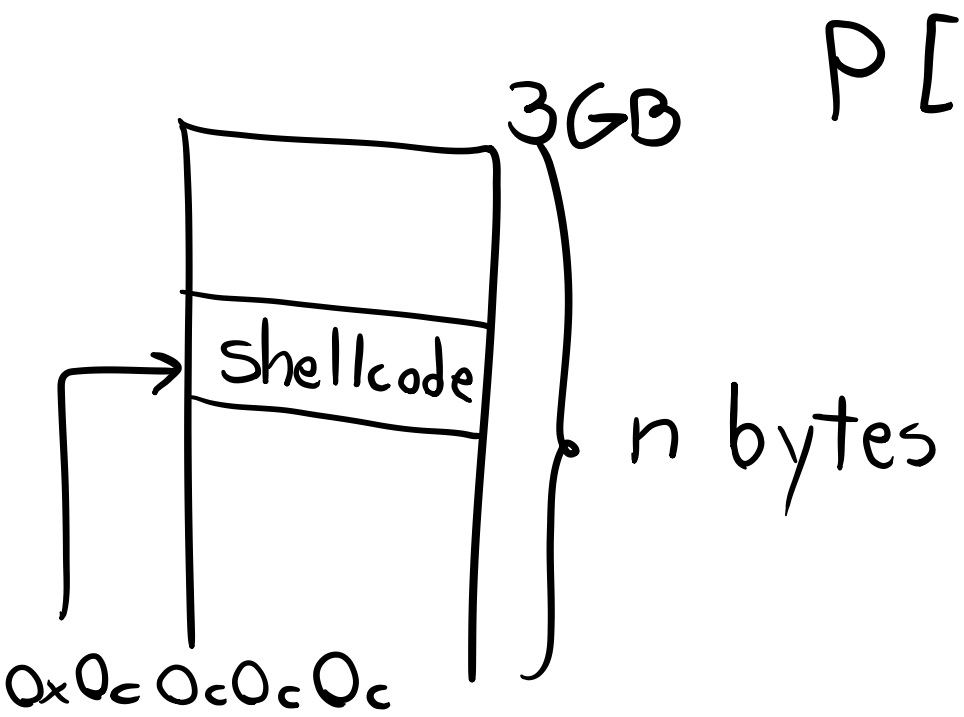


DoS – TCP SYN - Solução

- SYN cookies (DJB)
  - Evita o armazenamento de estado (ofload para o cliente)
  - Codifica dado que devíamos ter deixado no servidor
  - Clientes honestos irão ACK de volta
  - A integridade do cookie é garantida com criptografia

Ataque a ASLR + DEP (Data Execution Prevention)

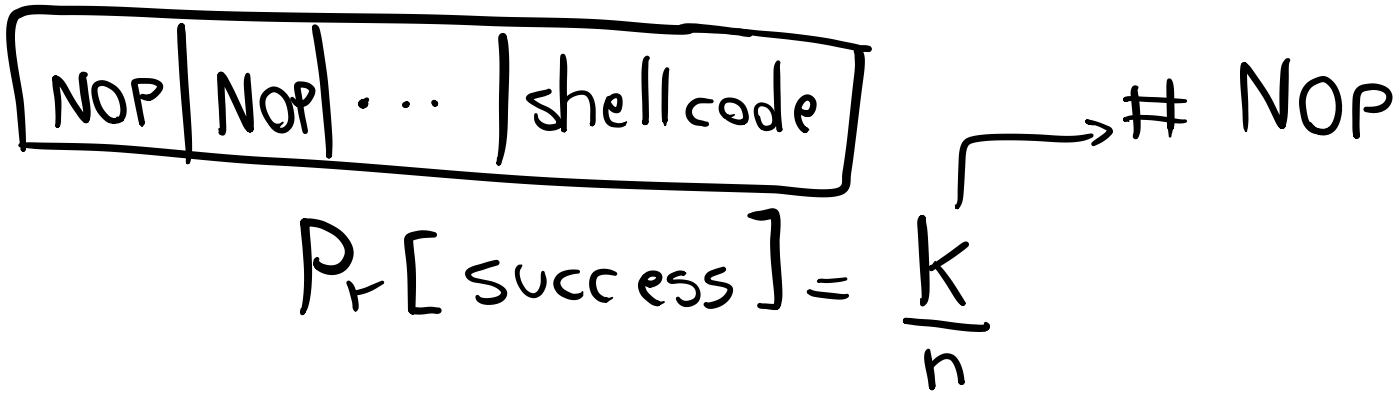
- Ataque probabilístico



$$P[\text{jump shellcode } 0x0c\ldots] = \frac{1}{n}$$

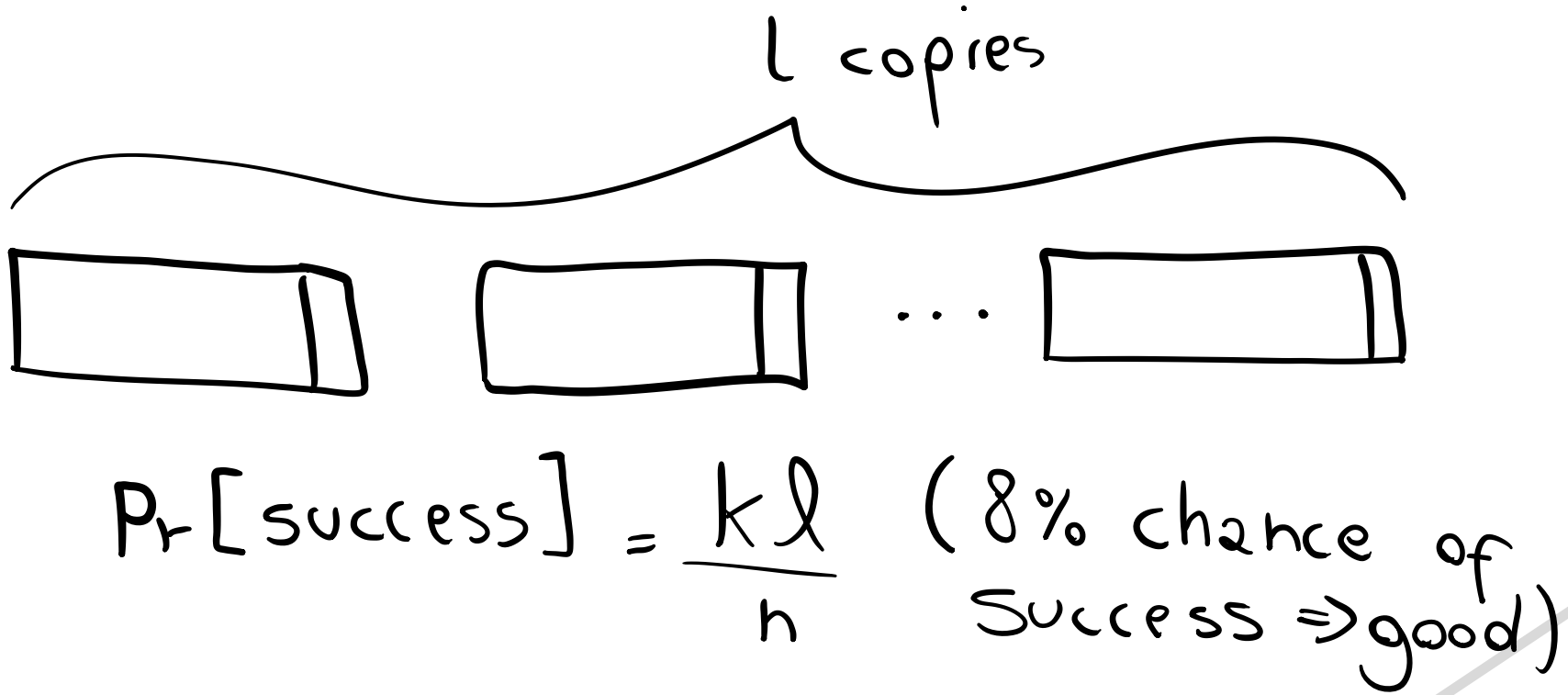
Ataque a ASLR + DEP – Melhora 1

- NOP (0x90) sleds



Ataque a ASLR + DEP – Melhora 2

- Muitas cópias de NOPs + Shellcode



Double Free

- Malloc é utilizado para alocar memória dinámica em tempo de execução. Toda memória alocada é dealocada em algum momento da execução do programa
- Memória dinâmica é armazenada na Heap
  - Uma estrutura de dados implementadas em C. Não necessariamente possui suporte do Sistema operacional.
- Algumas implementações de Malloc são vulneráveis a memória que é liberada duas vezes

Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/*
 * the chunk header
 */
typedef double ALIGN;

typedef union CHUNK_TAG
{
  struct
    {
      union CHUNK_TAG *l;   /* leftward chunk */
      union CHUNK_TAG *r;   /* rightward chunk + free bit (see below) */
    } s;
  ALIGN x;
} CHUNK;
```

Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/*
 * we store the freebit -- 1 if the chunk is free, 0 if it is busy --
 * in the low-order bit of the chunk's r pointer.
 */

/* *& indirection because a cast isn't an lvalue and gcc 4 complains */
#define SET_FREEBIT(chunk) ( *(unsigned *)&(chunk)->s.r |=  0x1 )
#define CLR_FREEBIT(chunk) ( *(unsigned *)&(chunk)->s.r &= ~0x1 )
#define GET_FREEBIT(chunk) ( (unsigned)(chunk)->s.r & 0x1 )
```

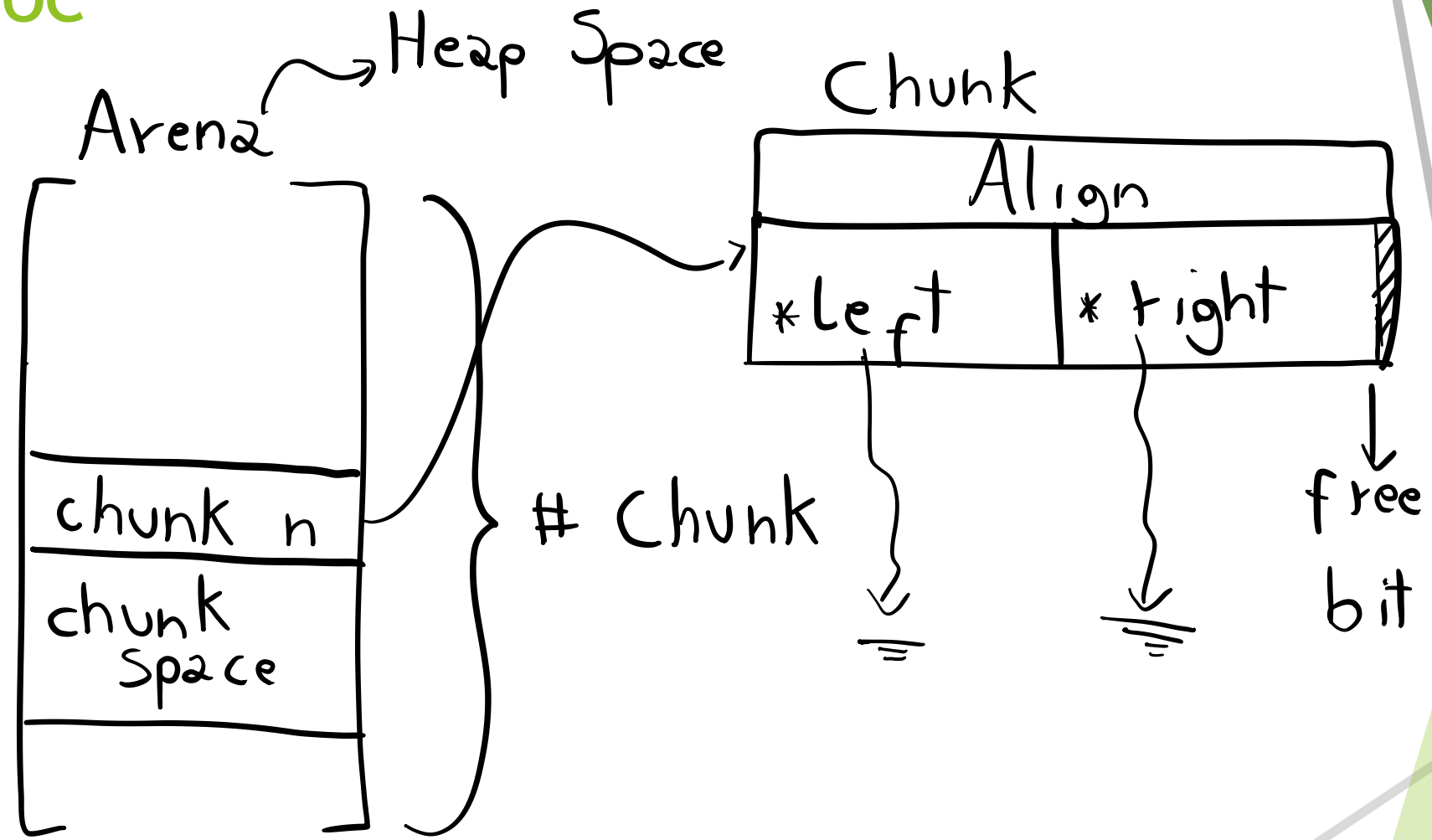
Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/* it's only safe to operate on chunk->s.r if we know freebit
 * is unset; otherwise, we use ... */
#define RIGHT(chunk) ((CHUNK *)(~0x1 & (unsigned)(chunk)->s.r))

/*
 * chunk size is implicit from l-r
 */
#define CHUNKSIZE(chunk) ((unsigned)RIGHT((chunk)) - (unsigned)(chunk))

/*
 * back or forward chunk header
 */
#define TOCHUNK(vp) (-1 + (CHUNK *)(vp))
#define FROMCHUNK(chunk) ((void *)(1 + (chunk)))
```

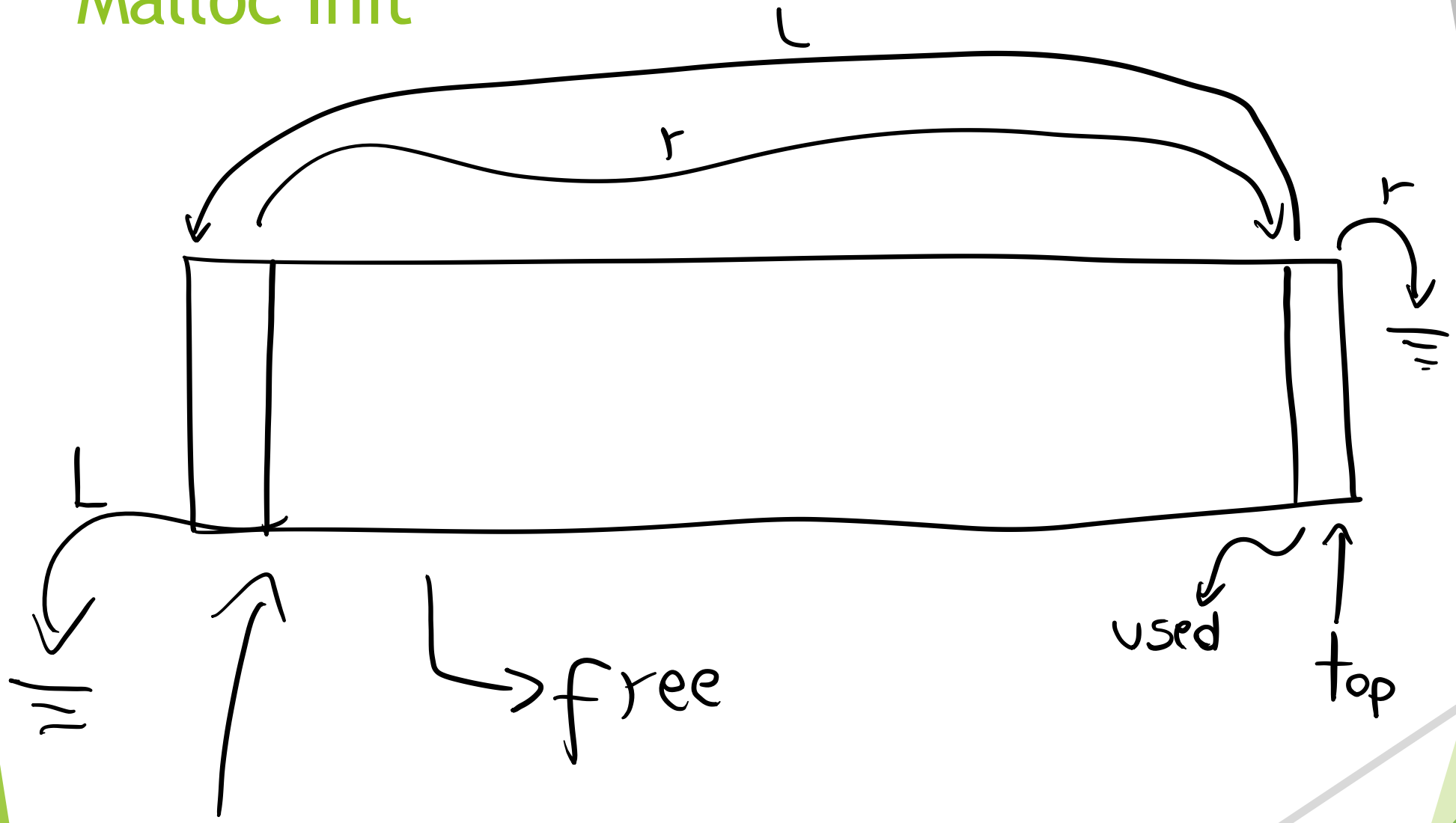
Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/* for demo purposes, a static arena is good enough. */
#define ARENA_CHUNKS (65536/sizeof(CHUNK))
static CHUNK arena[ARENA_CHUNKS];

static CHUNK *bot = NULL;   /* all free space, initially */
static CHUNK *top = NULL;   /* delimiter chunk for top of arena */
```

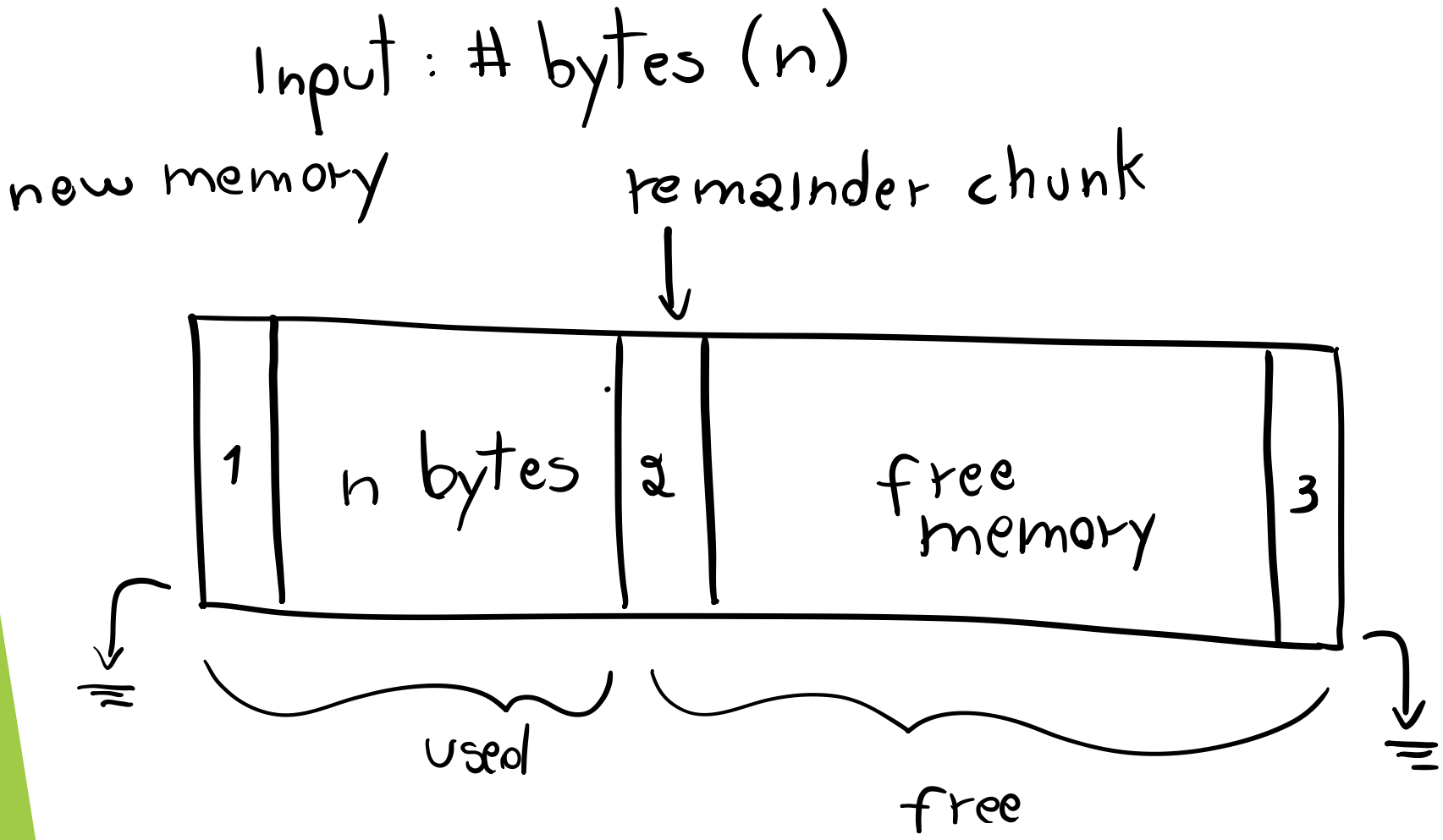
Malloc



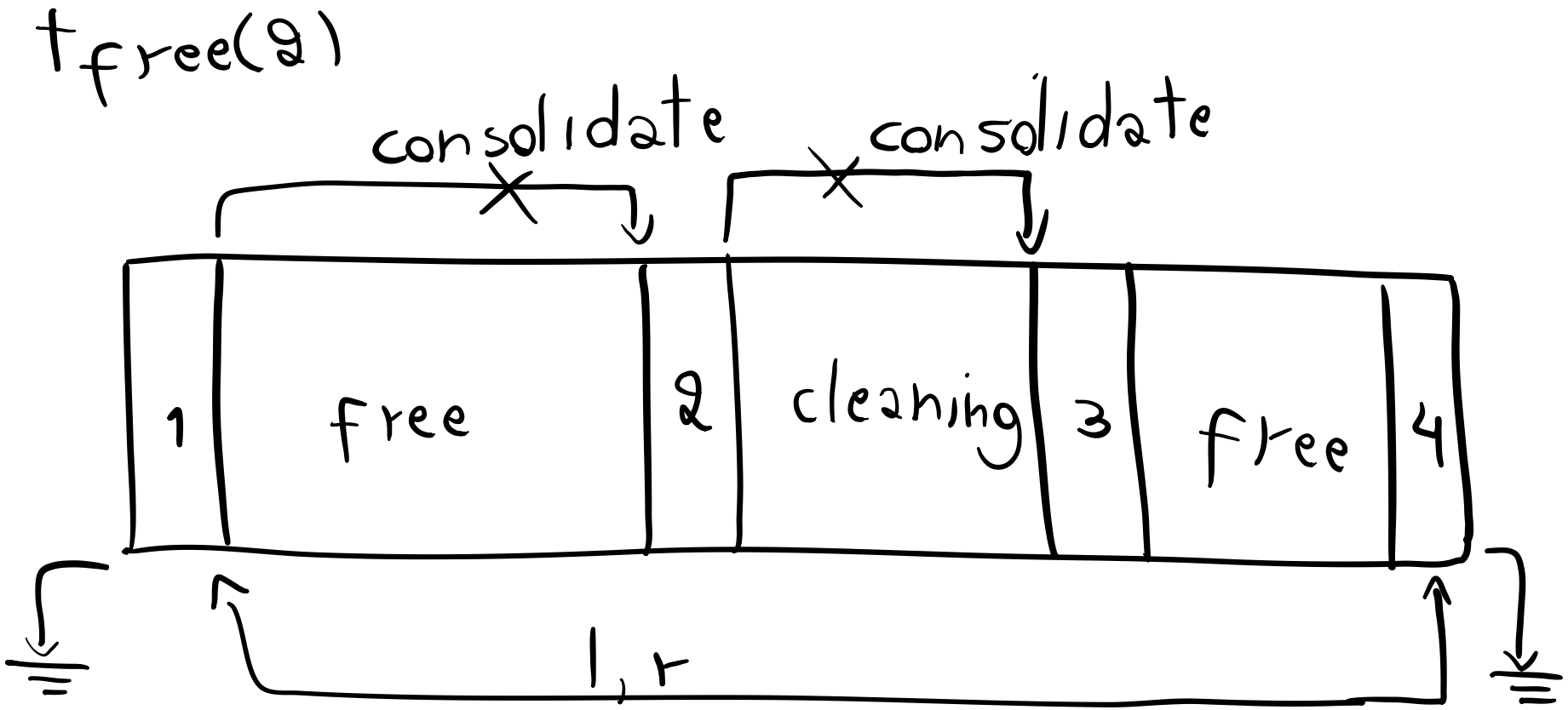
Malloc Init



Malloc tmalloc

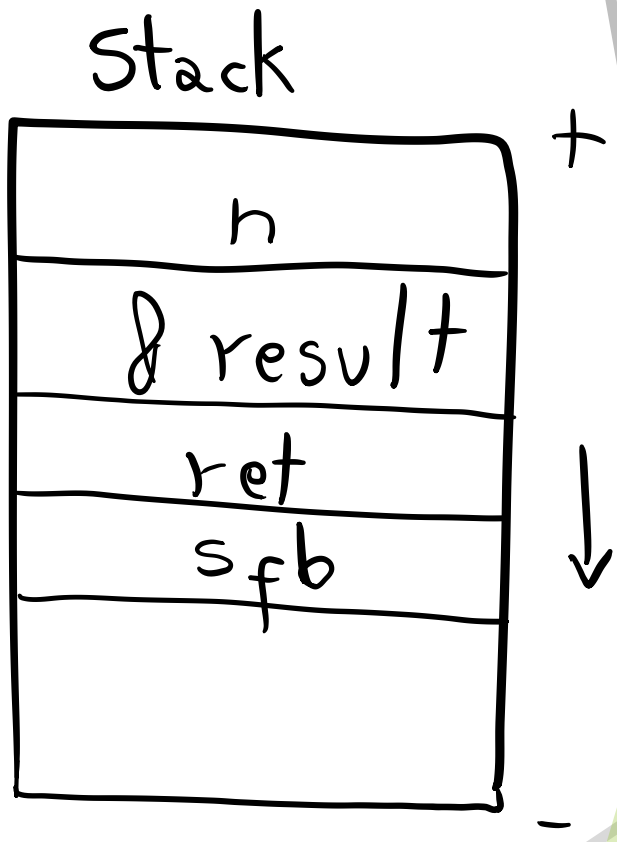


Malloc tfree



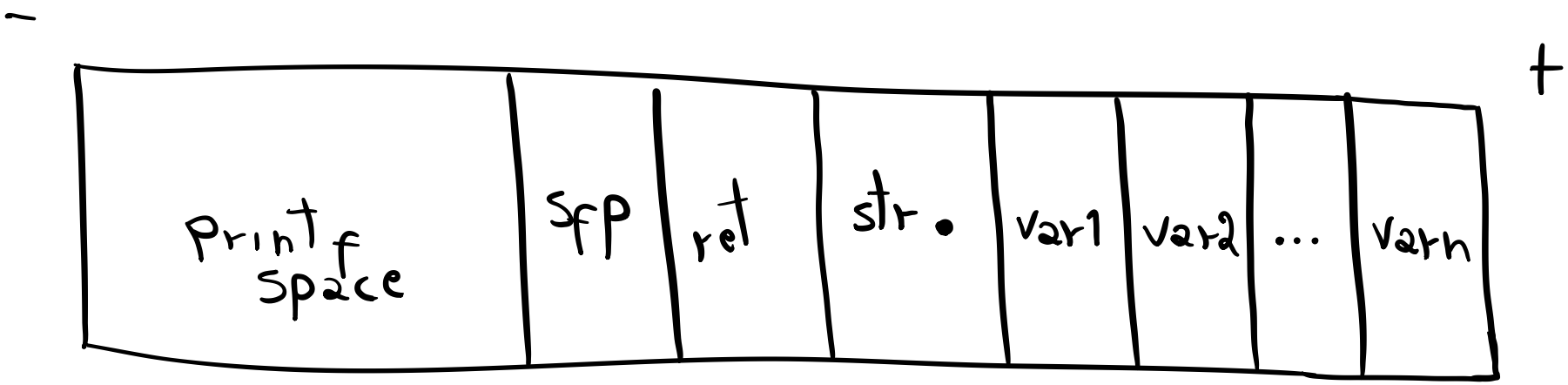
Ellipsis

```
void func(int n, ...)
{
    va_list list;
    va_start(list, n);
    int arg = va_arg(list, int); // what am I
suppose to do here ._.
    arg = n + 7;
    va_end(list);
}

int main() {
    int result;
    func(10,&result);
    if (result == 17); // my goal
}
```



Printf

- Void printf(const char* str, …)



Printf

- A string passada para o printf possui a capacidade de ler e escrever informações na memória
- O que acontece se o usuário é permitido definer essa string?

```
string str;
cin >> str;
printf(str.c_str());
```

Printf - Parametros

- %n recebe um ponteiro da stack e escreve o número de bytes escritos até agora

parameter	*output*	*passed as*
%d	decimal (int)	value
%u	unsigned decimal (unsigned int)	value
%x	hexadecimal (unsigned int)	value
%s	string ((const) (unsigned) char *)	reference
%n	number of bytes written so far, (* int)	reference

Referências

- J. Pasquale: CSE 120: Principles of Operating Systems – Lecture 13: Protection and Security. University of California, San Diego, Winter, 2014
- A. Snoeren: CSE 123: Computer Networks – Lecture 22: TCP and NAT. University of California, San Diego, Fall, 2014.
- M. Zalewski: Browser Security Handbook, chapters 1 (basic concepts) and 2 (standard security features).
- D. Blazakis: Interpreter Exploitation
- Anonymous, “Once Upon a free()...,” Phrack 57 #0x09.
- Anonymous, “Bypassing PaX ASLR Protection,” Phrack 59 #0x09.