環境に手を加えず動作する セキュリティ機構の開発

学習駆動コース　坂井ゼミ　29Ss 藤森 大潤

デモ

```
  }else{

    new_list->next = p;
    return new_list;
  }
}

struct name_list *list_delete(struct namelist *name_list,int i)
{
  struct name_list *next_pointer;
  struct name_list *node = name_list;
  struct name_list *pointer = name_list;

  while (pointer->next != NULL){
    next_pointer = pointer ->next;
    if(pointer->i == i){
      pointer->next = next_pointer->next;
      free(next_pointer);
      free(next_pointer);
      return 1;
    }
    pointer = next_pointer;
  }
  return 2;
}

struct name_list *list_print(struct name_list *name_list)
{
  struct name_list *next_pointer;
```

DoubleFree

libc : https://kozos.jp/nlcc

開発内容・取り組み

- ptrace()を用いた実験集の制作
- システムコールやELF等低レイヤーの学習
- C言語の学習

ptraceとは

- Linuxにあるシステムコール、主にデバッグ用途で使われる
- 子プロセスに対してメモリやレジスタの読み書きができる

ptraceを使ったアイデア

ptraceはデバッガ以外の用途で使われにくい

もっとアクティブな使い方はできないのか？

エクスプロイト検出に活かせそう！

SecHack365での制作物

- PSC Filter
- PEP Filter
- PDFDetector

サンドボックス化を実現

エクスプロイトを保護

SecHack365での制作物

- PSC Filter
- PEP
- PDFDetector

カーネル や ライブラリ

を書き換えずに実現

現

エクスプロイトを保護

カーネルとは

- ハードウェアとプロセスの仲介を行う部分
- カーネル以外の領域をユーザーランドと呼ぶ
- 全ての権限を持っている

https://www.hitachi.co.jp/products/it/server/ha8500/column/no004_p01.html

特徴

環境に手を加えずに動作する

- ユーザーランドで動作するため、カーネルを改造する必要がない
- ライブラリに処理を追加する必要がない
- 実行したいコマンドを書き換える必要がない

1.PSC Filter、PEP Filter

既存ツール

- サンドボックス化を行う既存ツールは全てカーネルの機能
- SELinux、AppArmor、Seccomp等

Seccompを利用する場合

- カーネル側でシステムコールの制限をおこなう
- BPFを用いてフィルタリング
- ChromiumやDocker等で利用されている

Seccompを利用する場合

- カーネル側でシステ
- BPF
- ChromiumやDocker等で利用されている

ユーザーランドで行いたい！

PSC Filter (Ptrace System Call Filter)

- ptraceを用いて発行されるシステムコールを制限
- システムコールをブラックリスト又はホワイトリスト登録
- システムコールが呼び出されたら
  1. シグナルSIGKILLをプロセスに送信
  2. アラートとログを出力して処理を続行
- ROP等攻撃の難化につながる ⇨ プロセスのセキュリティを強化

Jsonで記述

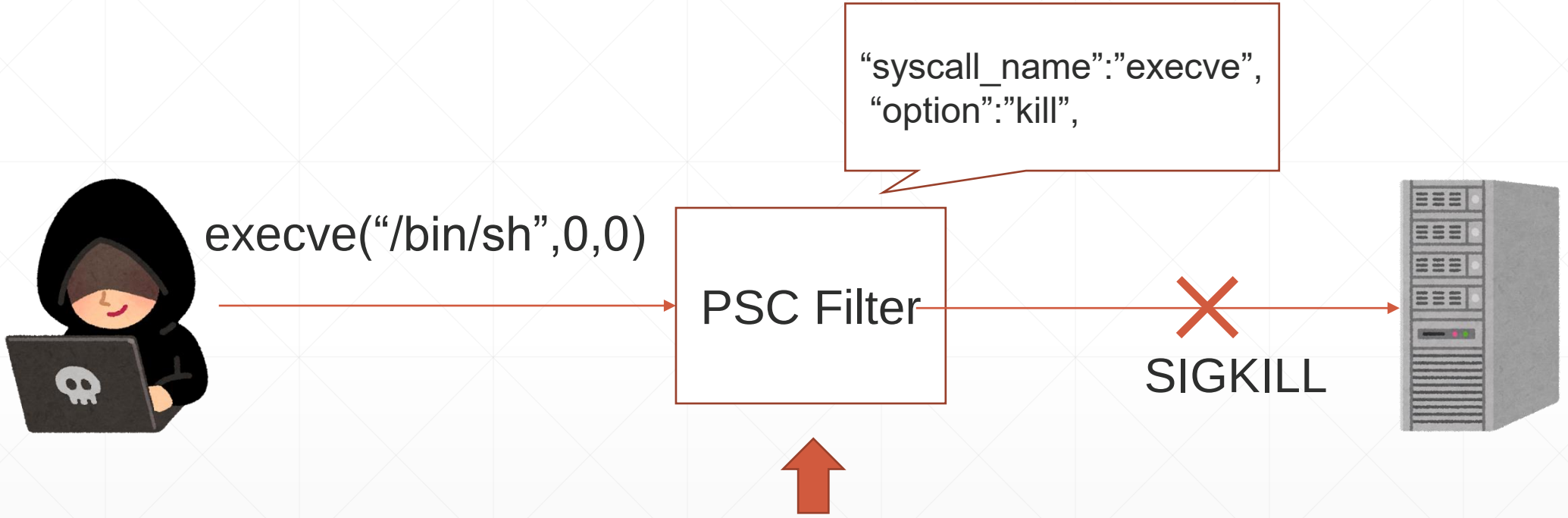
config.json

```
{
  "config": {
    "type":"blacklist"
  },

  "path" : [
    "/home/",
    "/lib/x86_64-linux-gnu/",
    "/etc/"
  ],

  "systemcall": [
    {
      "systemcall_name": "read",
      "option": "alert"
    },
    {
      "systemcall_name": "execve",
      "option": "alert"
    },
    {
      "systemcall_name": "write",
      "option": "alert"
    }
  ]
}
```

PSC Filterの目的

- 攻撃者は脆弱性をついた後、任意のコードを実行しようとする
- 余計なシステムコールをブロックしコード実行のリスクを減らす

実現できること



execveシステムコールをPSC Filterが阻止

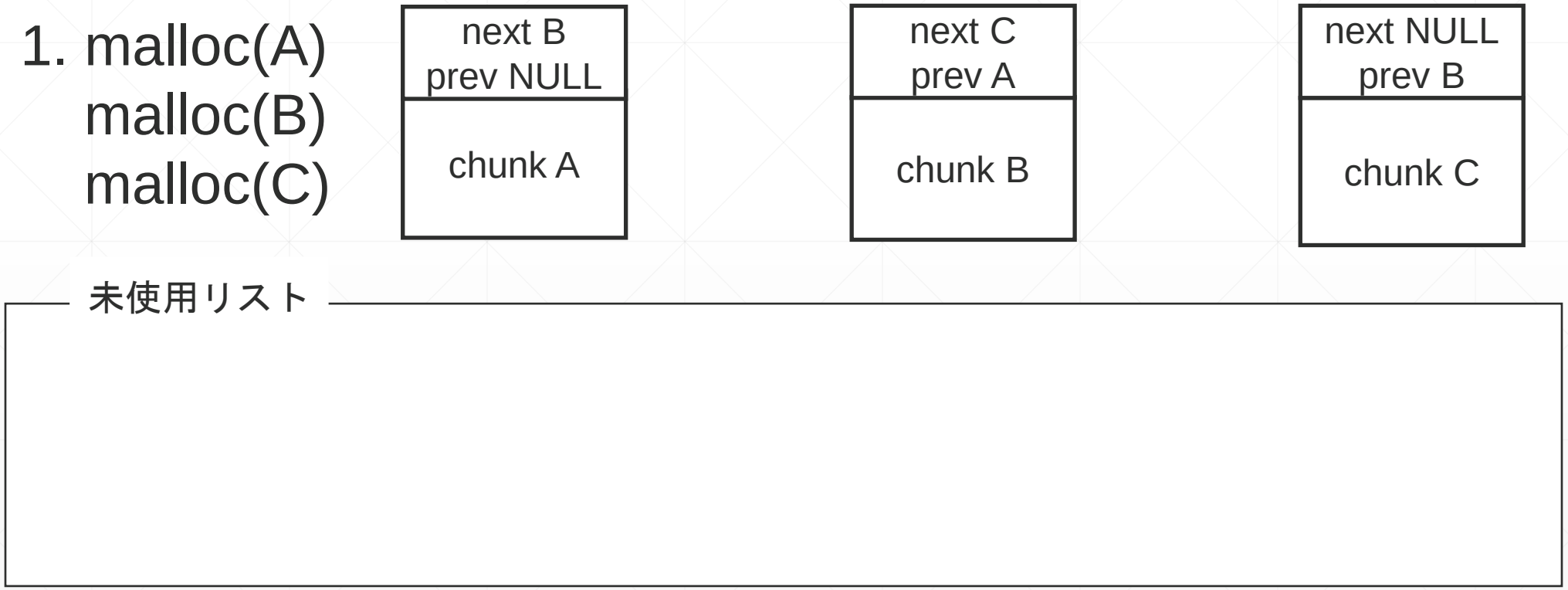
PEP Filter （Ptrace Execution Path Filter）

- 特定のシステムコールが読み書きするパスを制限
- 許可するパスをconfigファイルにホワイトリスト登録
- 許可されていないパスでシステムコールが動作するとSIGKILLを発行

2. PDFDetector

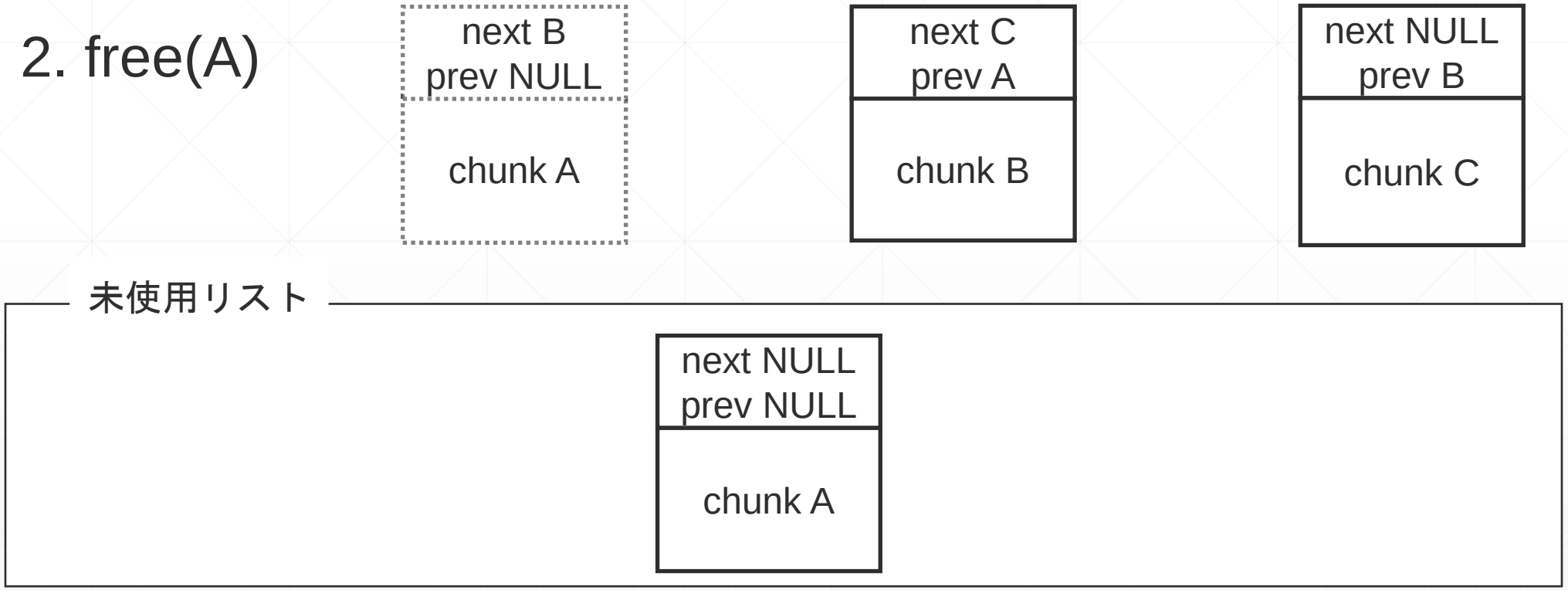
DoubleFree（二重解放）とは

1. malloc(A)
   malloc(B)
   malloc(C)



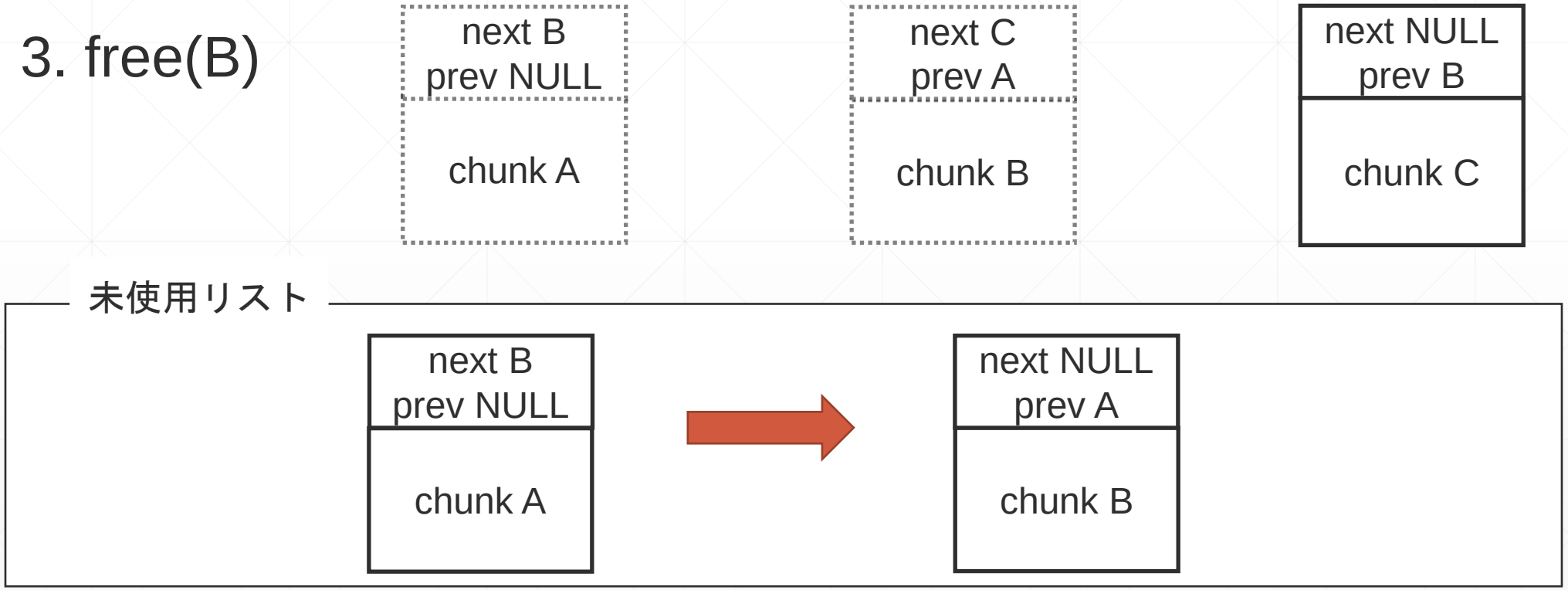
DoubleFree（二重解放）とは

2. free(A)



DoubleFree（二重解放）とは

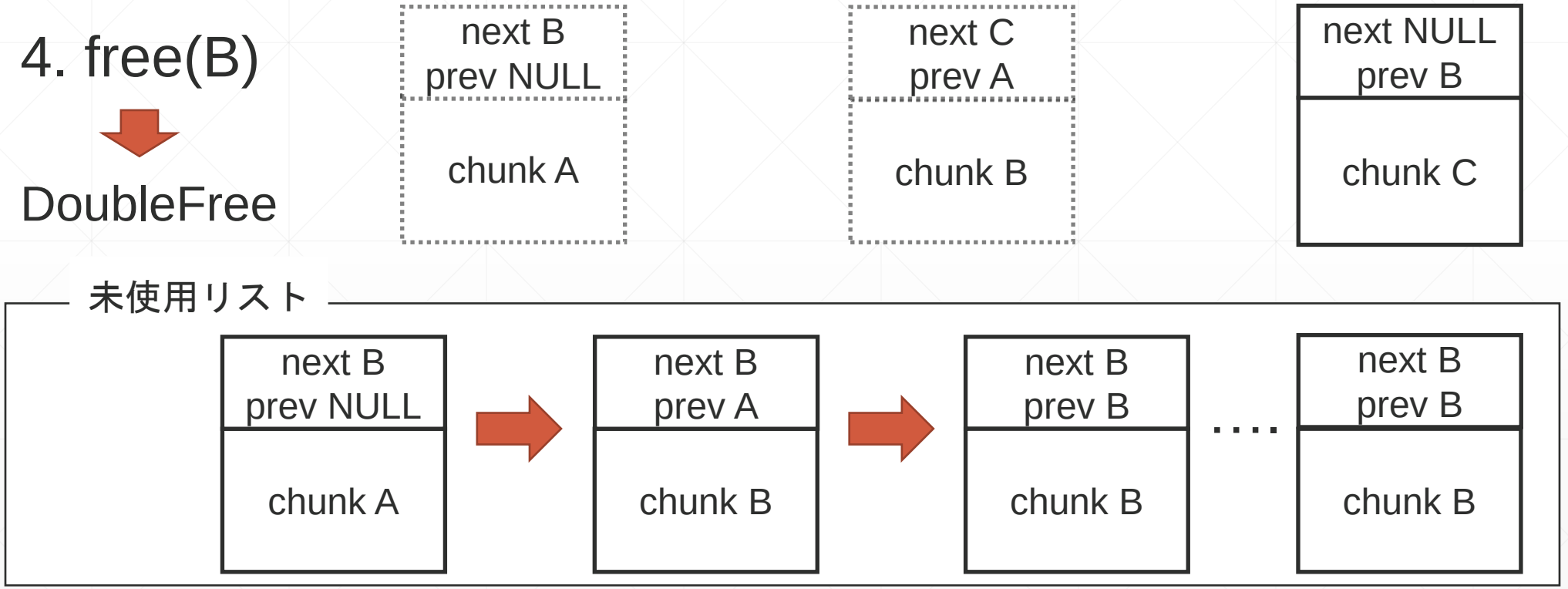
3. free(B)



DoubleFree（二重解放）とは

4. free(B)

DoubleFree



DoubleFree（二重解放）とは

- 解放(free)済みのメモリを再度解放してしまう脆弱性
- 任意のアドレスに書き込むことができてしまう
- 複雑なプログラムでは脆弱性に気づかない

プロトタイプ：線形リストを用いたDoubleFree検出

- malloc、freeをブレークポイントで監視
- mallocが呼ばれると線形リストにchunkアドレスを格納
- freeが呼ばれるとリストを探索しfree予定のアドレスがあればノードを削除
- なければDoubleFreeと判断

さらに高速化

解放処理が行われるたびにリストの探索を行う

メモリ操作を繰り返すプログラムだと処理速度が大幅に低下する

ハッシュテーブルと組み合わせることで速度の低下を防ぐ

PDFDetector （Ptrace DoubleFree Detector）

- mmapを用いてハッシュテーブルを作成
- シンボル情報からmallocとfree関数のアドレスを取得、監視
- mallocが呼ばれたらchunkアドレスを線形リストでつなげ

  chunkアドレスの下5桁とハッシュテーブルのindexが対応するように格納
- freeが呼ばれたら解放予定のアドレスがリストに存在することを確認、削除
- このときアドレスが存在しなければDoubleFreeと判断

```
  long hash_addr = chunk_addr % 0x100000;

  if(mmap_pointer[hash_addr] == 0){
    struct chunk_list *list = NULL;
    chunk_list_create(list,mmap_pointer,chunk_addr);
  }else{
    struct chunk_list *list = mmap_pointer[hash_addr];

    chunk_list_create(list,mmap_pointer,chunk_addr);
    chunk_list_print(list,chunk_addr);
  }

  regs.rip = malloc_addr;
  ptrace(PTRACE_SETREGS, pid, 0, &regs);

}else if (regs.rip == free_func + 1){

  regs.rip = free_func;
  ptrace(PTRACE_SETREGS, pid, 0, &regs);

  long hash_addr = regs.rax % 0x100000;

  long chunk_addr = regs.rax;

  list = mmap_pointer[hash_addr];
```

PDFDetectorの利点

- 外部から強制的に値を書き換えて攻撃を検知できる
  - ライブラリ側に処理を追加する必要がない
- ユーザーが行う動作は引数を積んで実行するだけ

SecHack365で学んだこと

複雑なプログラムを
作りきったことがなかった

インプットを繰り返して
出てきた課題で挫折

「とりあえず作ってみる」
から始めることを意識し、手を動かせるようになった

今後の開発

- ツールの継続開発
- ptraceを用いたツールの開発（ptraceを使ったfuzzer等）
- カンファレンス等での発表

まとめ

- PSC Filter
- PEP Filter
- PDFDetector

これら３つはカーネルやライブラリを弄らない

つまり

ユーザーの環境に手を加えず動作する！