SecHack365

環境に手を加えず動作するセキュリティ機構の開発

ptraceを用いた別プロセスからのエクスプロイト検出機構実験集

学習駆動コース 坂井ゼミ 藤森大潤

ptraceとは

・デバッガに使われるLinuxのシステムコール
・他プロセスのメモリ、レジスタの取得操作が可能

↓

つまりptraceはプロセスに対して**なんでもできる**システムコール

開発のきっかけ

・これだけの可能性を秘めたシステムコールがデバッガにしか使われないのはもったいない
・もっとアクティブな使い方はできないのか？と考えた結果

↓

エクスプロイト検出に活かせそうだと気付いた

成果物

ptraceを使い3つのツールを実験集として制作

- PSCFilter
- PEPFilter

　　┣ プロセスのサンドボックス化を実現

- PDFDetector

　　┗ エクスプロイト(DoubleFree)を検出

特徴

・ユーザーランドで動作する
　➡カーネルモジュールの導入が必要ない
・ライブラリ側の対策が必要ない
　➡ライブラリの更新が必要ない
・実行したいコマンドを書き換える必要がない
　➡検知のためにアプリを改造する必要がない

↓

環境に手を加える必要がない！

既存のセキュリティ機構はカーネルやライブラリで対策を行っているものが多い
そのためそれらのツールに対する深い知識が必要であったり　プログラムの依存関係によって導入がしづらい場合がある
しかし

ptraceを使えばもっとカジュアルに**環境を汚さず**攻撃を防げる！

ツール紹介

PSCFilter (Ptrace SystemCall Filter)

```
{
  "config": {
    "type":"blacklist"
  },
  "path" : [
    "/home/",
    "/lib/x86_64-linux-gnu/",
    "/etc/"
  ],
  "systemcall": [
    {
      "systemcall_name": "read",
      "option": "alert"
    },
    {
      "systemcall_name": "execve",
      "option": "kill"
    },
    {
      "systemcall_name": "write",
      "option": "alert"
    }
  ]
}
```

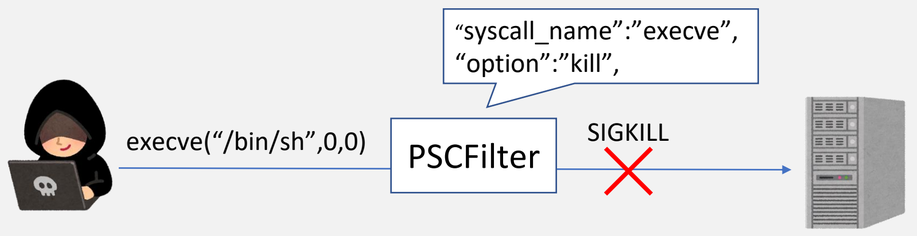
実行中のプロセスにアタッチし、システムコールを制限する(PSCFilter)。
jsonと呼ばれる言語で実行を許可又は阻止するシステムコールを記述。
ptraceを用いて発行されたフックしたシステムコールと、ファイルに記述されたシステムコールを比較し、フィルタリングを行う。
記述されたシステムコールが呼び出された場合の挙動を、ユーザーが指定することも可能。
SIGKILLシグナルをプロセスに送信し処理を停止するkillオプションと、alertを出力しログを記録した後処理を続行するalertオプションの２つを選択できる。
これにより危険なシステムコールはブロックし、特定のシステムコールはログを取り観測するといったことができる。

又レジスタの値を監視し、システムコールが操作するパスを制限する機能(PEPFilter)も実装。
特定のシステムコール(現在はopen,execve対応)が指定のパス以外で読み書きを行った場合、SIGKILLを発行する。

PEPFilter (Ptrace Execution Path Filter)

これらを使用することで攻撃の影響を軽減する、いわゆるサンドボックスを簡単に構築することができる。

例えば攻撃者が何らかの脆弱性を突き、システムに侵入後シェルコード実行を試みていたとする。



execveをkillオプションで指定してプロセスにアタッチしていると、PSCFilterがexecveの実行を検知し、シェルの起動を阻止することができた。

PDFDetector (Ptrace DoubleFree Detector)

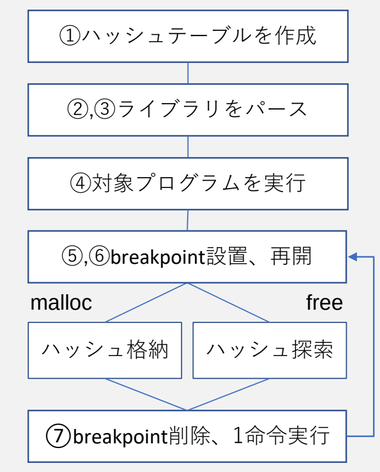
DoubleFreeとは、解放されたメモリを二重に解放することで生じる脆弱性。
攻撃者によってポインタ操作が可能になり、任意コード実行の恐れがある。
PDFDetectorでは、レジスタの値を読み取り確保されたメモリアドレスを
ハッシュテーブルに格納することで、DoubleFreeの実行を検出する。

①mmapを使用しPDFDetector側でハッシュテーブルを用意
②.symtabセクションを解析しmalloc,free関数のアドレスを取得
③malloc関数を解析しret命令のアドレスを取得
④fork,execveで子プロセスとして実行しptraceからattachを行う
⑤2と3で取得したアドレスにint3 breakpointを設置
⑥PTRACE_CONT (ptraceでの動作継続)を実行

mallocが呼ばれたら：レジスタの値からchunkの先頭アドレスを、
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chunkアドレスがハッシュテーブルに
格納されていることを確認。格納されていなければSIGKILLを発行し
プログラムを終了する。

⑦breakpointを削除しPTRACE_SINGLESTEPで1命令実行
現時点では対象プログラムがライブラリと静的リンクされている
必要があるが、今後は共有ライブラリにも対応したい。




```
continue
0x402236 0x4bca30 0x4bca30
list_addr = 0x5555555676a0
free  hash_addr -> 0x4bca30 0xca30 5555555676a0
|*|chunk_list -> 0x0
0x401f18 : doublefree detected
```

学習プロセスとSecHack365で学んだこと

6月 ptraceを知る	デバッガを学んでいたときに初めてptraceを知った。当時はシステムコール自体の知識がなかったため、ゼロから学習を行った。
8月 C言語学習開始	C言語を学び始め、データ構造等の理解に苦しみながら学習を進めた。
9月 PSCFilter開発	学んできたシステムコールの知識を活かし、PSCFilterを開発した。学んだC言語を使って開発したプログラムが完成したときには大きな達成感と自信が得られた。
10月 PEPFilter開発	PEPFilterを実装した。システムコールについて学習をしながら実装を進めていくことで、理解がより深まった。
1月 PDFDetector開発	PDFDetectorを開発した。もともと解析が好きだったため、デバッグをしながら開発を進めていく事がすごく楽しかった。開発を通してメモリ管理に関する理解がより深まった。

今回技術や知識以外にも開発の進め方も学ぶことができた。
今まで高度なプログラムを作りきったことがなく、
インプットを繰り返し、出てきた課題で挫折していた。
しかし今回はまずプロトタイプを作ってみて、その先で方向性を考え
行き詰まったら別の方法で試してみるといった方法で開発を進めた。
その結果、考えるだけでなくとりあえず手を動かすことができるようになった。
これからも手を動かすことを意識して開発を続けたい。

Twitter:@Gregar1ous
GitHub:Gregarious-git

ソースコード

https://github.com/Gregarious-git/ptracetools

今後

- これらのツールの改良
- 新たなツールの開発(ptraceを用いたFuzzerを開発予定)
- カンファレンス等での発表

NICT National Institute of Information and Communications Technology