システムコールフックを使用した攻撃検出

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティー技術研究所
http://www.fourteenforty.jp

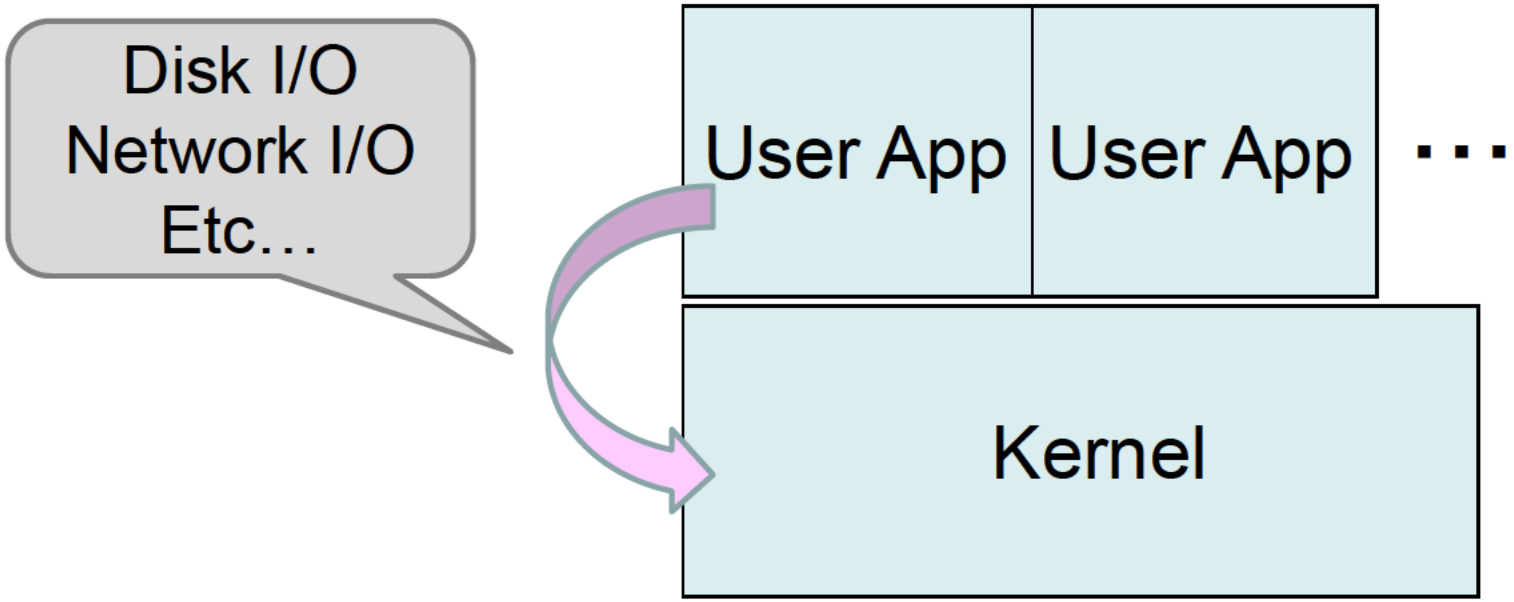
取締役技術担当　金居 良治

お題目

- System Call について
- System Call Protection
- System Call Hook
- 考察

System Call とは？

ユーザ アプリケーションからカーネル
のサービスルーチンを呼び出す



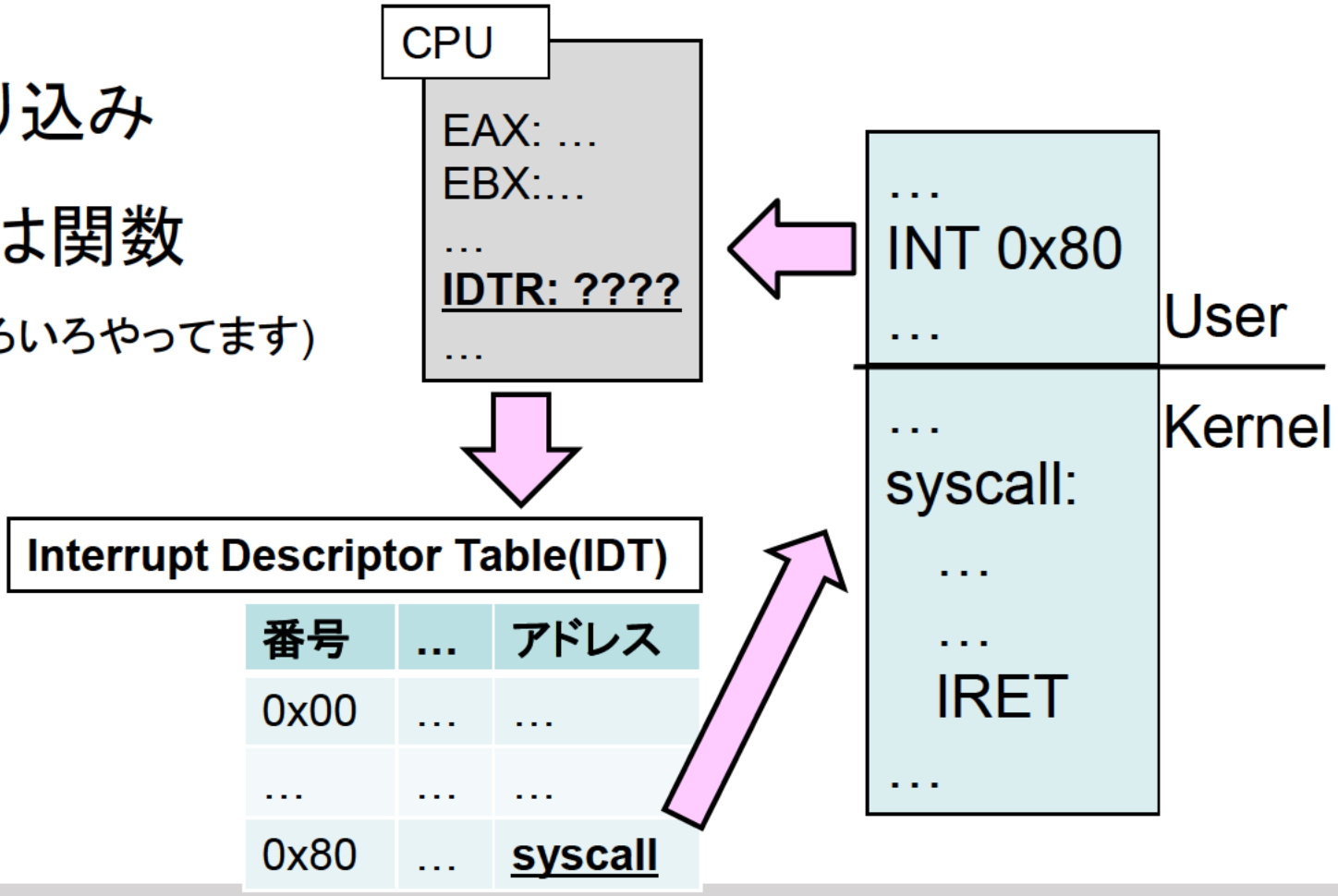
System Call 実装方法 (i386)

- INT (Linux, FreeBSD, Windows 2000)
  割り込み
- SYSENTER (Linux?, Windows XP)
  高速システムコール

INT による System Call の実装概略図

- INT = 割り込み
- 感覚的には関数

(ホントはもっといろいろやってます)



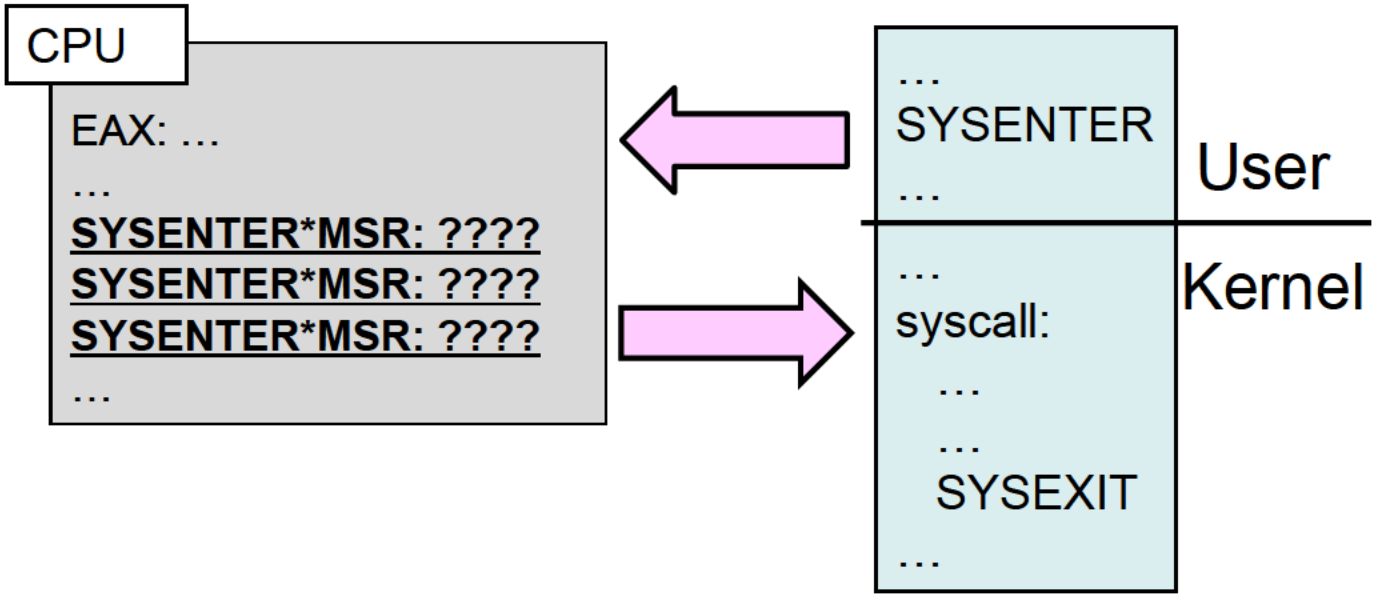
INT による System Call の特徴

- IDTというメモリ領域に割り込みハンドラのアドレスがある
- Windows でも 2000 以前は INT を使用
- trap gate(FreeBSD), interrupt gate(Linux)という2種類のバリエーションがある

SYSENTER の動作概略図

- System Call 用に特化、高速化してある

(ホントはもっといろいろやってます)



SYSENTER による System Call の特徴

- INTのような冗長なチェックが無く、メモリアクセスも無いので高速
- Pentium II で登場した命令
- Windows XP以降(Linuxも?)はこっち

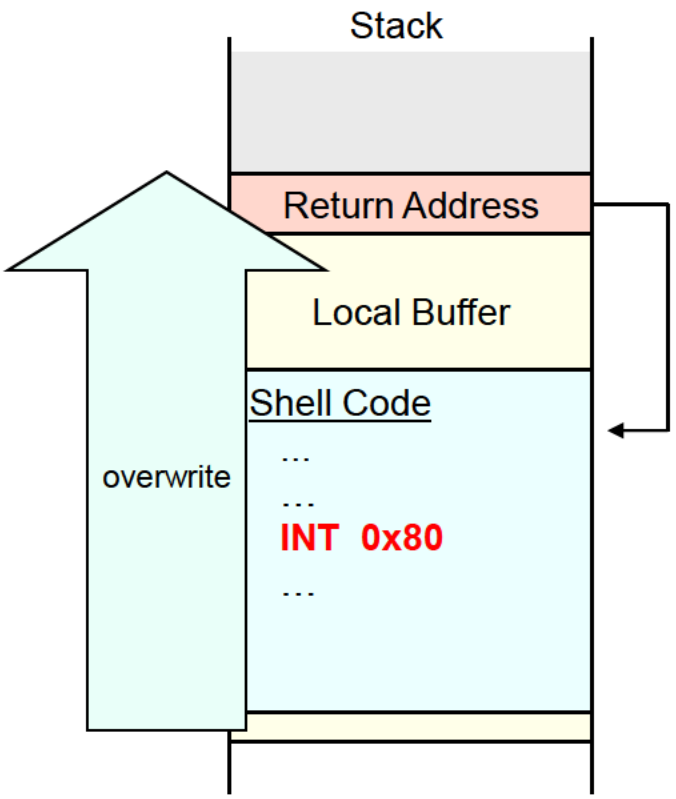
- ここら辺の挙動が詳しく知りたい人は Intel のページへ

System Call で遊んでみよう!

- 以外と簡単
- System Call Protection というのを考案
- FreeBSD でカーネルモジュールとして実装
- アーキテクチャは i386 に限定

攻撃に System Call が利用される例

- 例として、スタックベースのバッファオーバーフローについて検討
- だいたいシェルコード内でシステムコールを使用する
- これが禁止できれば、シェルコードが実行できても被害は最小に
- OpenBSD の W^X みたいなやつ

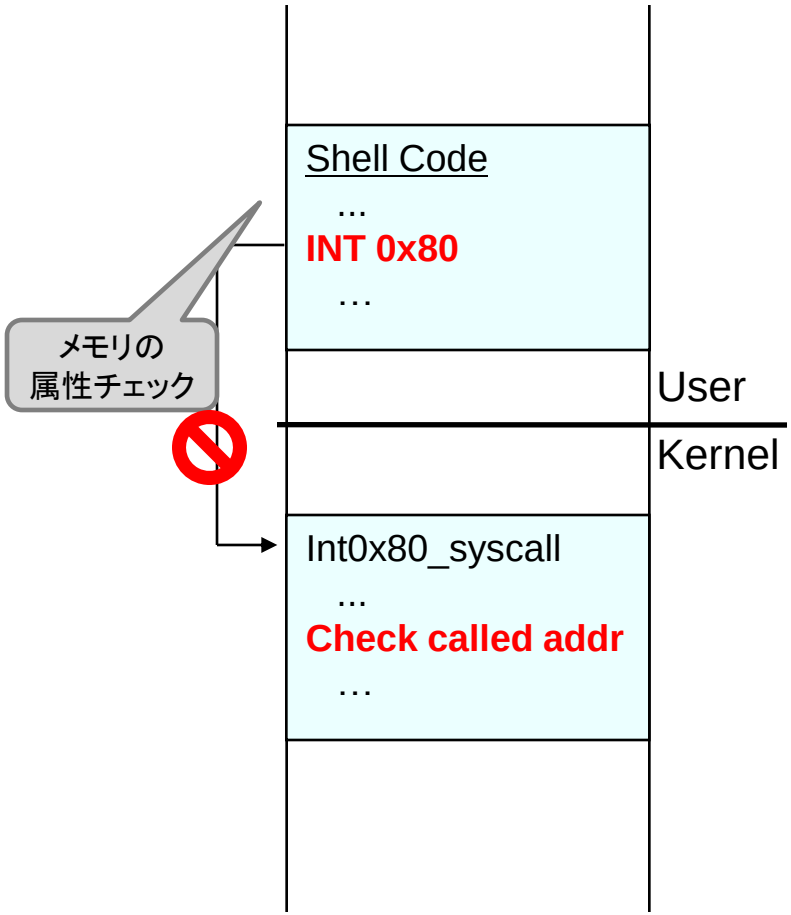


スタック上でのコード実行は禁止できるか

- 一部の例外を除いてスタック上でコードを実行するようなコードは不要
- OpenBSD は出来てる
- i386では出来ないOSが多い(Windowsとか)
- FreeBSD i386 も出来ない

System Call Protection

- 書き込み属性のあるメモリ領域から System Call の呼び出しを禁止
- 呼び出し元のアドレスはスタック上にあるので、これを調べる
- コード領域に書き込み属性が設定されていない事を利用
- スタック/ヒープベースのオーバーフローが発生しても、悪用が困難に

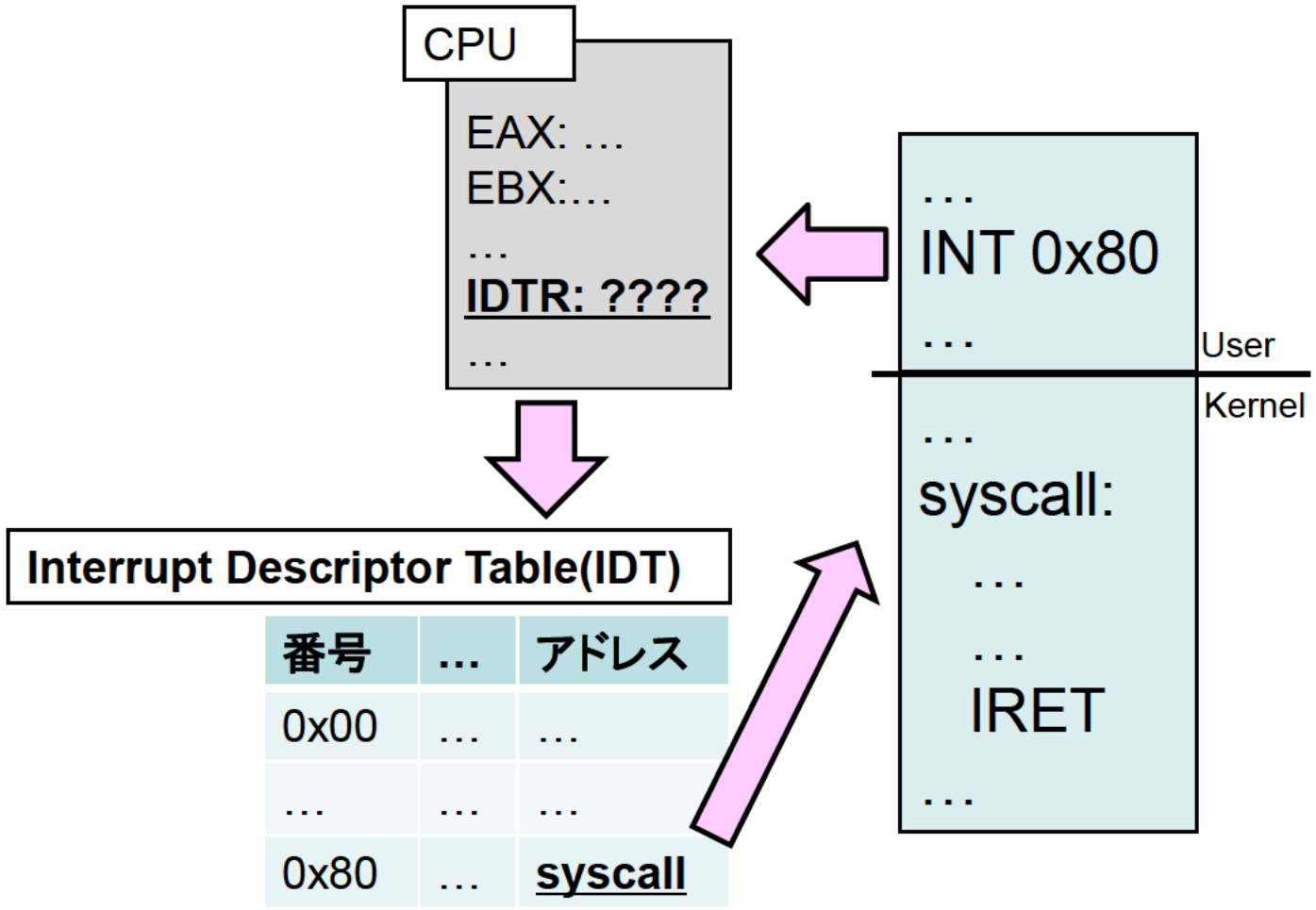


メモリの属性(パーミッション)

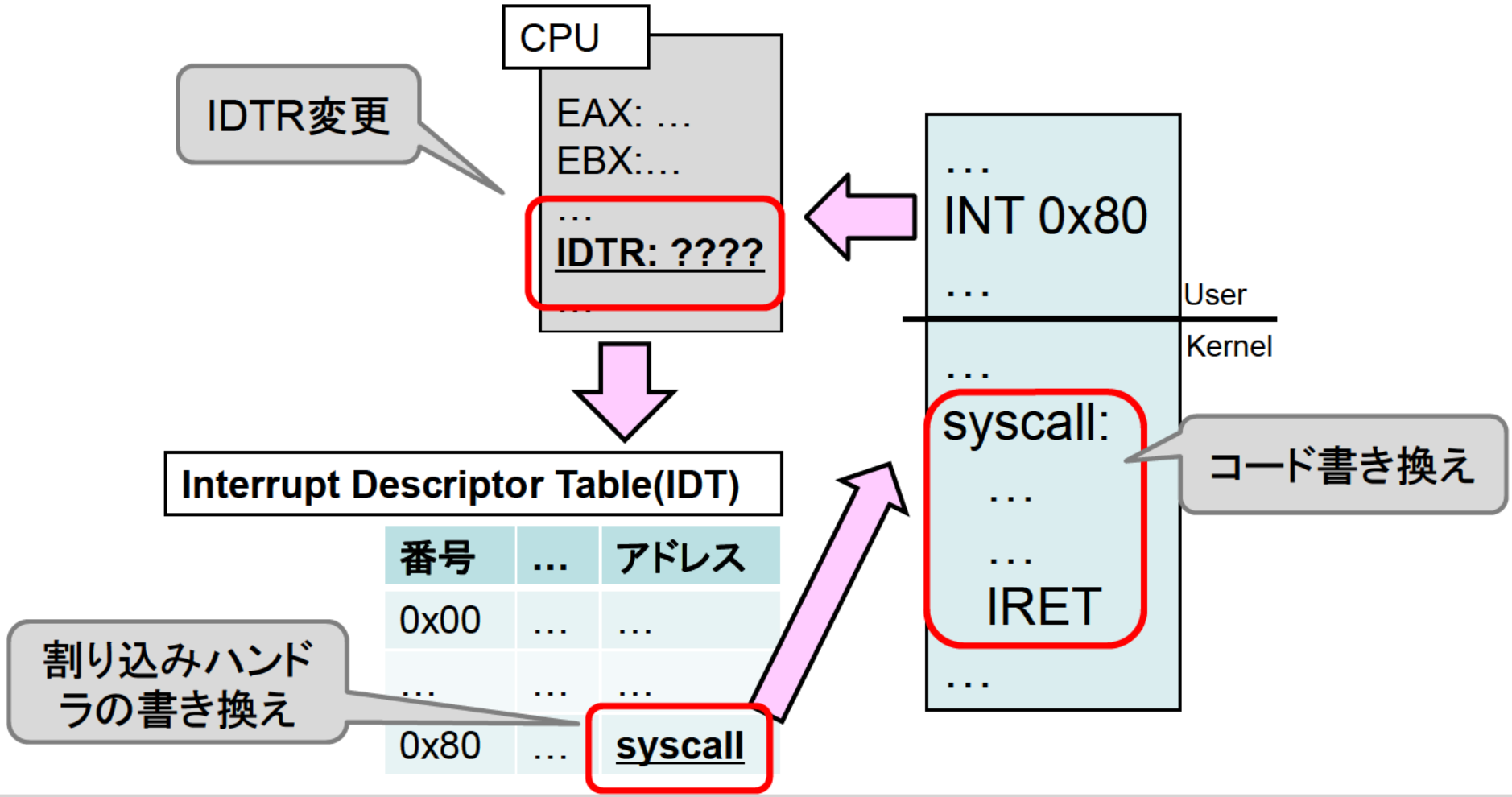
- FreeBSD では ports の pmap を使用して、メモリの属性をチェックできる

```
freebsd# pmap -n codeexec
1253:   /usr/home/kanai/work/codeexec
Address   Kbytes     RSS   Shared    Priv Mode   Mapped File
08048000       4       4       -       4 r-x    /usr/home/kanai/work/codeexec
08049000       4       4       -       4 rw-      [ anon ]
28049000     136     136       -     136 r-x    /libexec/ld-elf.so.1
2806B000       8       8       -       8 rw-    /libexec/ld-elf.so.1
2806D000      20      20       -      20 rw-      [ anon ]
28072000      32      28       -      32 rwx      [ anon ]
2807A000     800     176     800       - r-x    /lib/libc.so.6
28142000       4       4       -       4 r-x    /lib/libc.so.6
28143000      24      24       -      24 rwx    /lib/libc.so.6
28149000      88       4       -      88 rwx      [ anon ]
BFBE0000     128       8       -     128 rwx      [ anon ]
-------- ------- ------- ------- -------
Total Kb    1248     416     800     448
```

System Call Hook の方法



System Call Hook の方法



割り込みハンドラのアドレスを書き換える

- カーネルモジュールとしての実装が可能
- 簡単 (setidt()を呼ぶだけ)

```
setidt(IDT_SYSCALL, &IDTVEC(my_int0x80_syscall), SDT_SYS386TGT,
        SEL_UPL, GSEL(GCODE_SEL, SEL_KPL));
```

- ユーザ側からすれば、カーネルの再ビルドが不要なので、手軽に試すことが出来る

→ コンセプトコードに最適!

実行した様子

```
freebsd# sysctl -w scprotect.enable=0
scprotect.enable: 1 -> 0
freebsd# ./codeexec stack
code executed: ret=0
freebsd# ./codeexec heap
code executed: ret=0
freebsd# sysctl -w scprotect.enable=1
scprotect.enable: 0 -> 1
freebsd# ./codeexec stack
EXECPROTECT: pid 1418, eip 0xbfbfea57, code 36
Killed
freebsd# ./codeexec heap
EXECPROTECT: pid 1419, eip 0x804b007, code 36
Killed
freebsd# uname -a
FreeBSD freebsd 6.2-RELEASE FreeBSD 6.2-RELEASE #0: Fri Jan 1
2 10:40:27 UTC 2007     root@dessler.cse.buffalo.edu:/usr/obj
/usr/src/sys/GENERIC  i386
```

System Call Protection Off

System Call Protection On

スタック/ヒープ上で System Call が呼べない!

パフォーマンス – その1

- 10秒間に何回 getpid() を呼べるか計測
- System Call によるオーバーヘッドを計測
- 7.7% のパフォーマンス低下

System Call Protect		
無効	有効	パフォーマンス
24,059,220	22,195,394	92.3%

パフォーマンス – その2

- ab(apache benchmark) で1秒あたりに処理できるリクエスト数を計測
- ab -n 10000 –c 5 http://target:80/
- 3.1% のパフォーマンス低下

System Call Protect			
無効	有効	パフォーマンス	
873.7	847.0	96.9%	

パフォーマンスまとめ

- 実際のアプリケーションでは 3～7%程度のパフォーマンス低下が予想される
- ターゲットを vmware 上で動かしたので、値は不正確かも
- connect()やexec()等の危険なSystem Callが呼ばれた時のみチェックをするようにすれば、比較的、安全なまま高速化が可能

考察

- OpenBSD W^X を実装すれば済む話
- ただし、mips 等では実装が不可能。そういった環境では役に立つかも
- バイパスする方法があるが、ASLR 等の他のセキュリティ機能と組み合わせる事で防御できる
- いろいろ応用が可能

ありがとうございました

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティー技術研究所
http://www.fourteenforty.jp

取締役技術担当　金居 良治
kanai@fourteenforty.jp