バッファーオーバーフロープロテクションのFreeBSDへの対応について

江藤博明
etoh@jp.ibm.com

バッファーオーバーフロー問題

- 今年見つかった、影響の大きいバッファーオーバーフロー
  - 2001-05-01: Microsoft Windows 2000 IIS 5.0 IPP ISAPI 'Host:' Buffer Overflow Vulnerability
  - 2001-04-04: Ntpd Remote Buffer Overflow Vulnerability
  - 2001-01-29: ISC Bind 8 Transaction Signatures Buffer Overflow Vulnerability

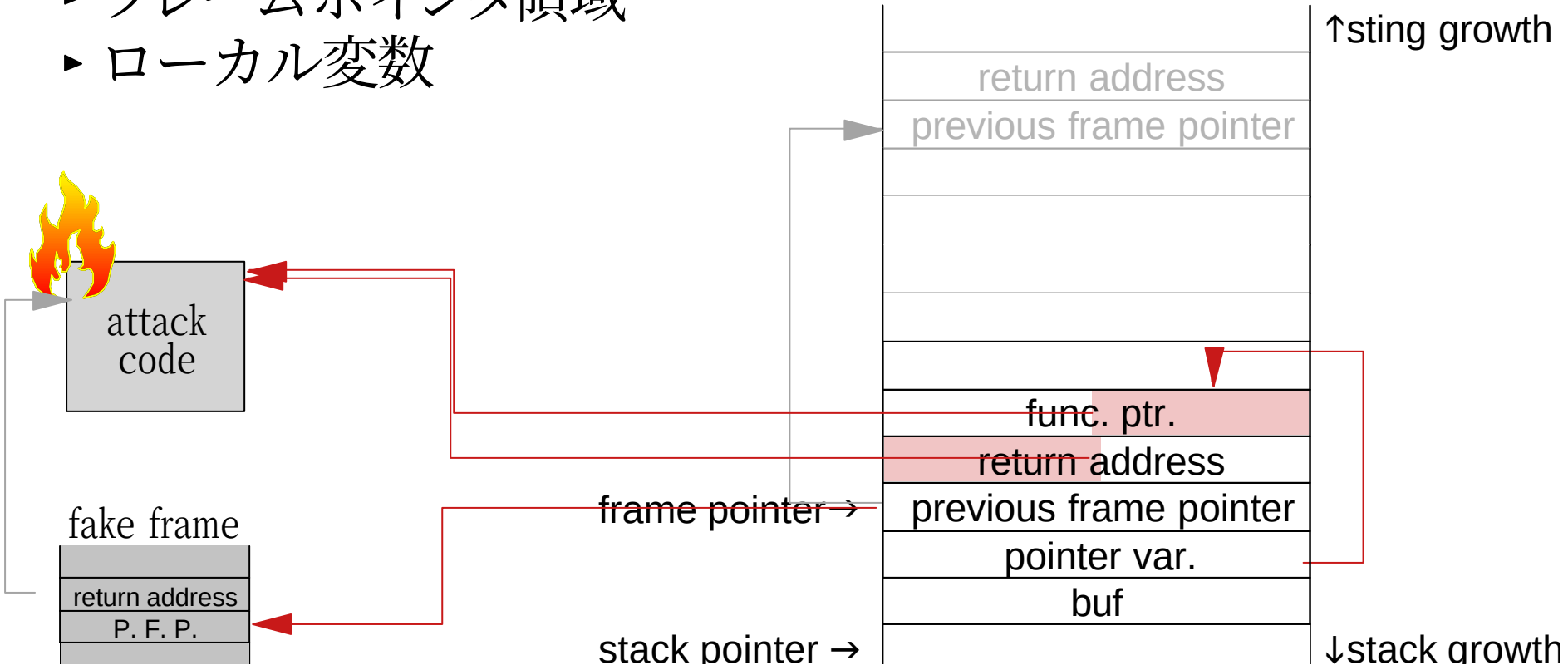
バッファーオーバーフローを利用した攻撃

- SUIDプログラムへの攻撃
- サーバープログラムへの攻撃
- 個人プログラムへの攻撃
- HTMLコンテンツを利用しての攻撃
- ユーザー権限の奪取
- プログラムの実行

スタック内のいろいろな攻撃方法

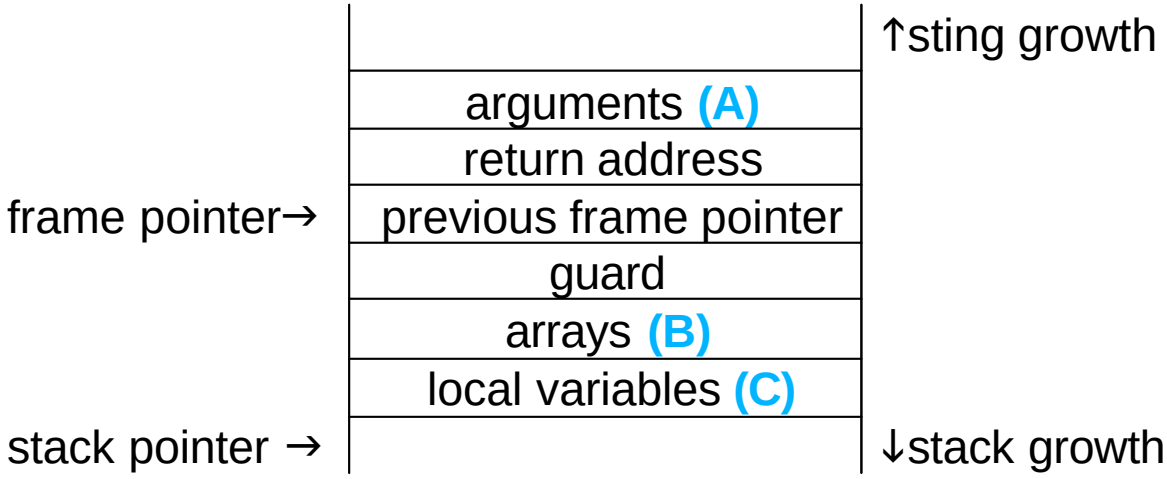
- 保護すべき場所
  - 関数引数
  - 戻りアドレス
  - フレームポインタ領域
  - ローカル変数

```
int  foo (void (*fn)())  {
     char* ptr = bar;   /* bar: global string */
     char buf[128];
     *ptr = -1;
     gets (buf);
}
```



スタックを安全に利用するモデル

- A 配列やポインターを含まない
- B 配列や配列を含む構造体だけ
- C 配列を含まない。ポインターを含んでも良い

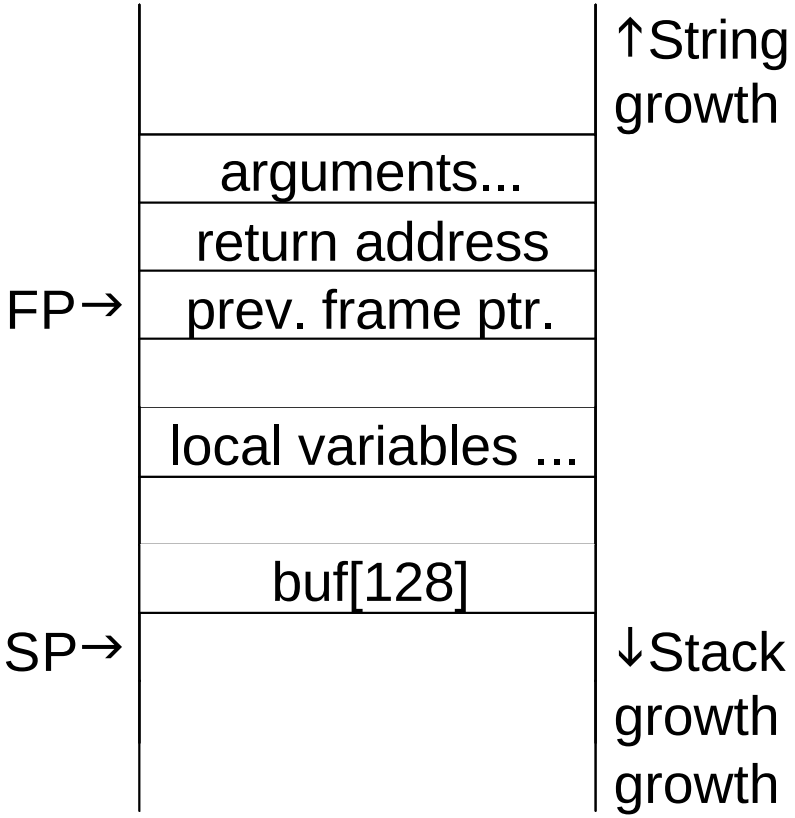


- 呼び出し側関数はバッファーオーバーフローに関して安全である。
  - The protection can be emitted for the function without string

Stack Protection (1)

- 防御用コードの自動挿入
- "Guard" をローカル変数として挿入
- ストリング配列の有無で制御

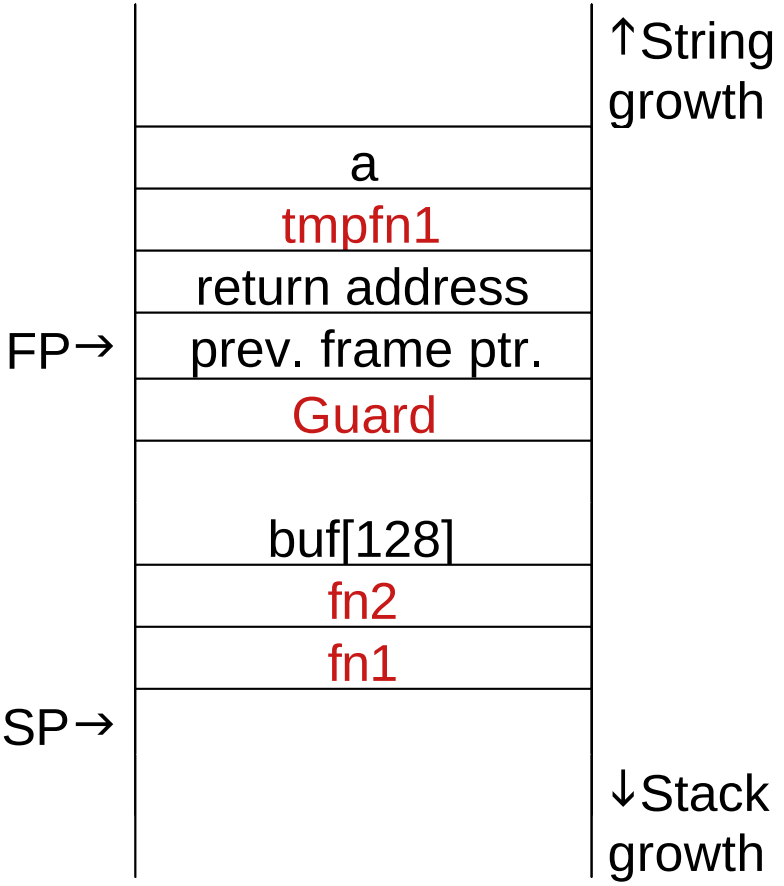
```
foo () {
  volatile int Guard
  char buf[128];
  Guard = 'random number'

  gets (buf);

  if (Guard != 'random number')
}     /* program halts */
```



Stack Protection (2)

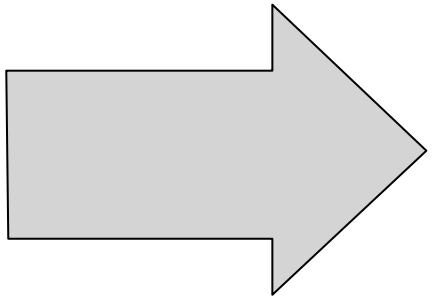
1. ローカル変数の位置を入れ替える
2. ポインター引数を配列の後ろへ

```
foo (int a, void (*tmpfn1)()) {
  volatile int Guard
  char buf[128];
  void (*fn2)();

  Guard = 'random number'
  gets (buf); (*fn1)(); (*fn2)();
  if (Guard != 'random number')
      /* program halts */
}
```



Translation Details

```
;; Function stkchk

(note 2 0 3 "" NOTE_INSN_DELETED)
(note 3 2 27 "" NOTE_INSN_FUNCTION_BEG)

(note 4 28 6 "" NOTE_INSN_DELETED)

(note 6 4 9 0 NOTE_INSN_BLOCK_BEG)

(insn 9 6 12 (set (mem:SI (plus:SI (reg:SI 18)
            (const_int -4)))
        (const_int 0)) -1 (nil)
    (nil))

(insn 12 9 14 (set (reg:SI 21)
        (plus:SI (reg:SI 18)
            (const_int -132))) -1 (nil)
    (nil))

(insn 14 12 16 (set (mem:SI (pre_dec:SI (reg:SI 7 %esp)))
        (reg:SI 21)) -1 (nil)
    (nil))
```



```
;; Function stkchk

(note 2 0 3 "" NOTE_INSN_DELETED)
(note 3 2 27 "" NOTE_INSN_FUNCTION_BEG)

(insn 27 3 28 (set (reg:SI 22)
        (mem:SI (symbol_ref:SI ("__canary")))) -1 (nil)
    (nil))

(insn 28 27 4 (set (mem/v:SI (plus:SI (reg:SI 18)
            (const_int -4)))
        (reg:SI 22)) -1 (nil)
    (nil))

(note 4 28 6 "" NOTE_INSN_DELETED)

(note 6 4 9 0 NOTE_INSN_BLOCK_BEG)

(insn 9 6 12 (set (mem:SI (plus:SI (reg:SI 18)
            (const_int -136)))
        (const_int 0)) -1 (nil)
    (nil))

(insn 12 9 14 (set (reg:SI 21)
        (plus:SI (reg:SI 18)
            (const_int -132))) -1 (nil)
    (nil))

(insn 14 12 16 (set (mem:SI (pre_dec:SI (reg:SI 7 %esp)))
        (reg:SI 21)) -1 (nil)
    (nil))
```

- code insertion
- address modification
- debug info. maintenance

攻撃耐性

Program	Description	Result of Attack	Protected Result
xlockmore 3.10	Screen saver	root shell	terminated
Perl 5.003	Perl script lang.	root shell	terminated
elm 2.003	Mail agent	root shell	terminated
SuperProbe2.11	Probes videoHW	root shell	terminated

- xlockmore, Perl, elm exploits:
  attacks against the return address
- SuperProbe exploit:
  attacks against argument

他の防御法との比較

Description	None	propolice	libsafe	StackGuard	StackShield
Protection Effectiveness					
return address	No	Yes	Yes	Yes	Yes
prev. frame pointer	No	Yes	Yes	No	Yes
argument	No	Yes	Yes	No	No
local variable	No	Yes	No	No	No
string overation coverage	No	All	Not all	All	All
Implementation characteristics					
OS independence	-	Yes	No	Yes	Yes
Processor independence	-	Yes	Yes	No	No
Other characteristics					
performance overhead	None	Very low	Very low	Low	Low
source code needed	-	Yes	No	Yes	Yes

防御コードのオーバーヘッドは？

- Benchmark program:
  50,000,000 runs on a 600 MHz Pentium III
  - int test()
    {
        char buf[128];
        strcpy(buf, "1234567890");
        return strncmp(buf, "1234", 4);
    }

Original run time	propolice run time	Overhead (%)
4.67 sec	5.05 sec	8%

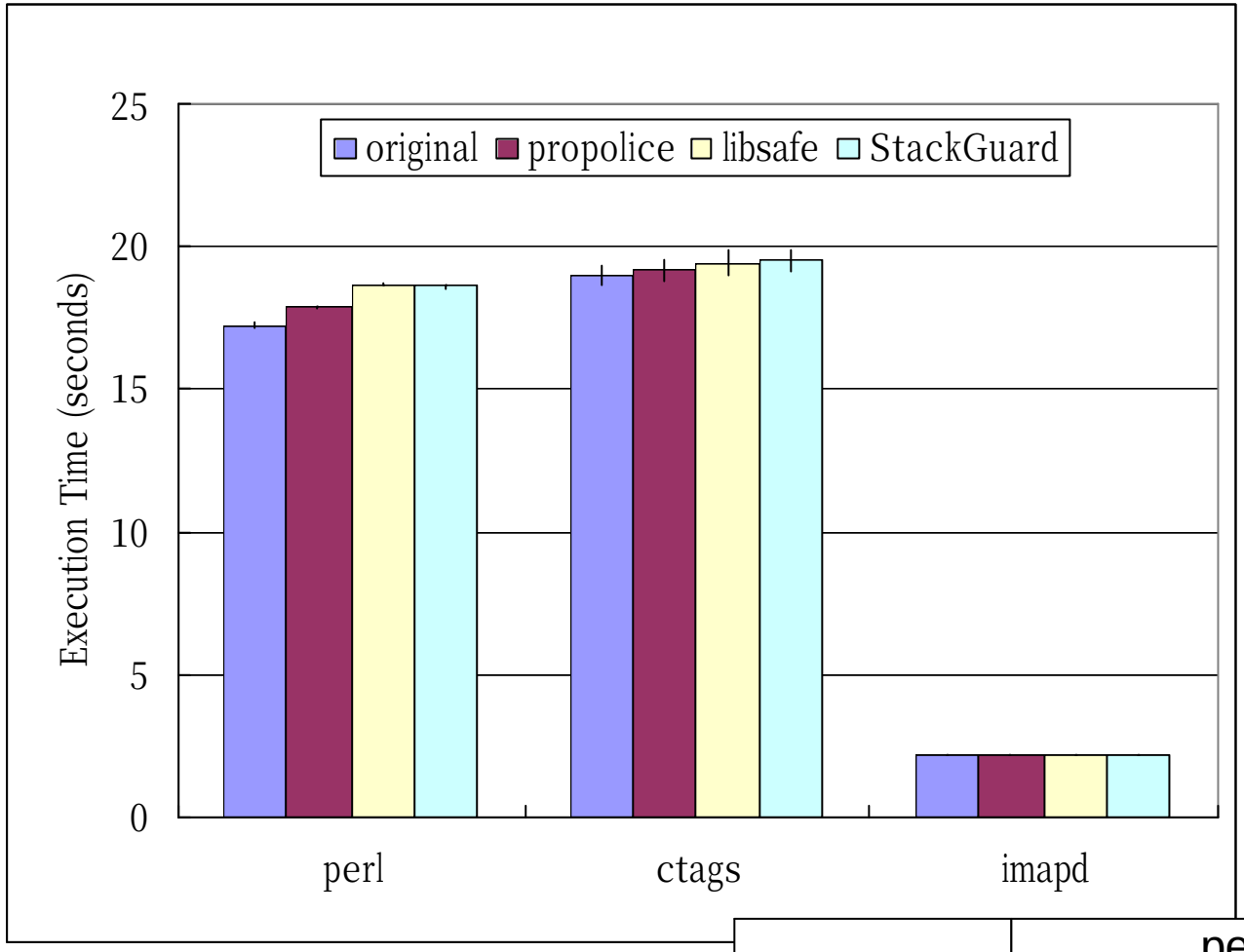
ストリング配列の使用率

Program	# of function	with string	ratio (%)
Perl 5.005	9590	54	0.6%
ctags	850	4	0.5%
imapd 4.7-5	915	259	6.4%
XFree86 3.3.6	22198	1145	5.1%
kernel 2.2.14	3881	112	2.8%
egcs 1.1.2	27019	244	0.9%
glibc 2.1.3	719	106	14.7%

✔Weak correlation to the actual execution ratio

- No structure passing found as an argument

ベンチマーク



performance overhead:

	perl	ctags	imapd
propolice	4%	1%	0%
libsafe	8%	2%	0%

ツールの公開場所

- http://www.trl.ibm.com/projects/security/ssp
  - gcc-2.95.2, gcc-2.95.3, latest_snapshotの差分
  - OS毎の手順
- GCC使用方法
  - GCC オプション： -fstack-protector
  - オプションをつけない場合はオリジナルGCCと同じ動作

プロテクト化のおねがい

- 多くのユーザーは自らビルドすることはない
- 個人用アプリケーションやベータコードのプロテクトも必要

- ポートのプロテクト化
- プロテクト化されたものの配布
- ＊BSDのプロテクト化
- プロセッサごとのテスト
  - 確認されたプロセッサ： iX86, powerpc, sparc

オペレーティングシステムの移植概要

- FreeBSD 4.3-release
  - カーネル、/usr/srcのプロテクト化完了
- 作業手順
  - GCCのパッチ当て
  - GCCのビルド&インストール
  - カーネルへのパッチ作成、(10行程度)
    シェアードライブラリへのパッチ作成
  - BUILD ALL

まとめ

- バッファーオーバーフローの概要を話しました。
- 防御方法を提案しました。
- バッファーオーバーフローに関して、防御したOSを皆さん作りませんか？