قاپیدن کنترل برنامه
(Program Control Hijacking)

قاپیدن کنترل برنامه

- کد برنامه رشته ای از دستورات CPU است که کار مورد نظر یک برنامه کاربردی را انجام می دهد.
- کد برنامه خرابکار، رشته ای از دستورات CPU است که کار مورد نظر حمله کننده را انجام می دهد.
  - مثل کد اجرای شل exec(“/bin/sh”)
- برای بدست گرفتن کنترل برنامه مراحل زیر لازم است:
  1. قرار دادن کد خرابکار در فضای آدرس برنامه
  2. پرش به کد جهت اجرا

قرار دادن کد در فضای آدرس برنامه

- تزریق کد:
  - کد به صورت ورودی به برنامه داده شده و در بافر قرار می گیرد.
    - بافر Stack
    - بافر Heap
- کد در برنامه موجود است.
  - مثل exec(“/bin/sh”)
  - حمله کننده باید به کد مورد نظر پرش کند.

پرش به کد جهت اجرا

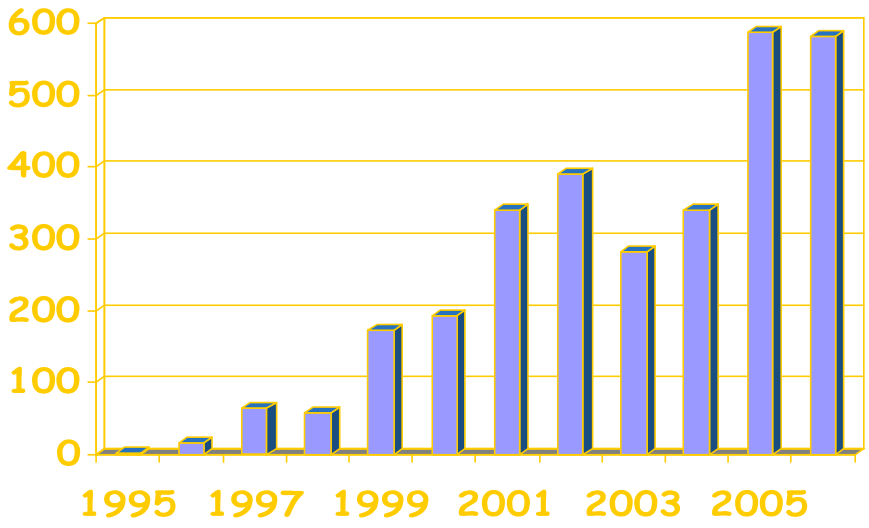
- برای پرش به کد باید حالت برنامه را تغییر داد.
- استفاده از سرریز بافر (Buffer Overflow) برای تغییر حالت
- سرریز بافر می تواند به سه طریق انجام شود.

1. رکورد فعال (Activation Record): تغییر آدرس بازگشت در پشته (Stack) و اشاره آن به کد
2. اشاره گر تابع: تغییر آدرس تابع در پشته یا Heap و اشاره آن به کد
3. Longjmp
   - setjmp و longjmp دو تابع سیستمی لینوکس برای تعیین Checkpoint
   - تغییر اطلاعات ورودی تابع longjmp در پشته یا Heap و اشاره آن به کد

سرریز بافر
(Buffer Overflow)

سرریز بافر

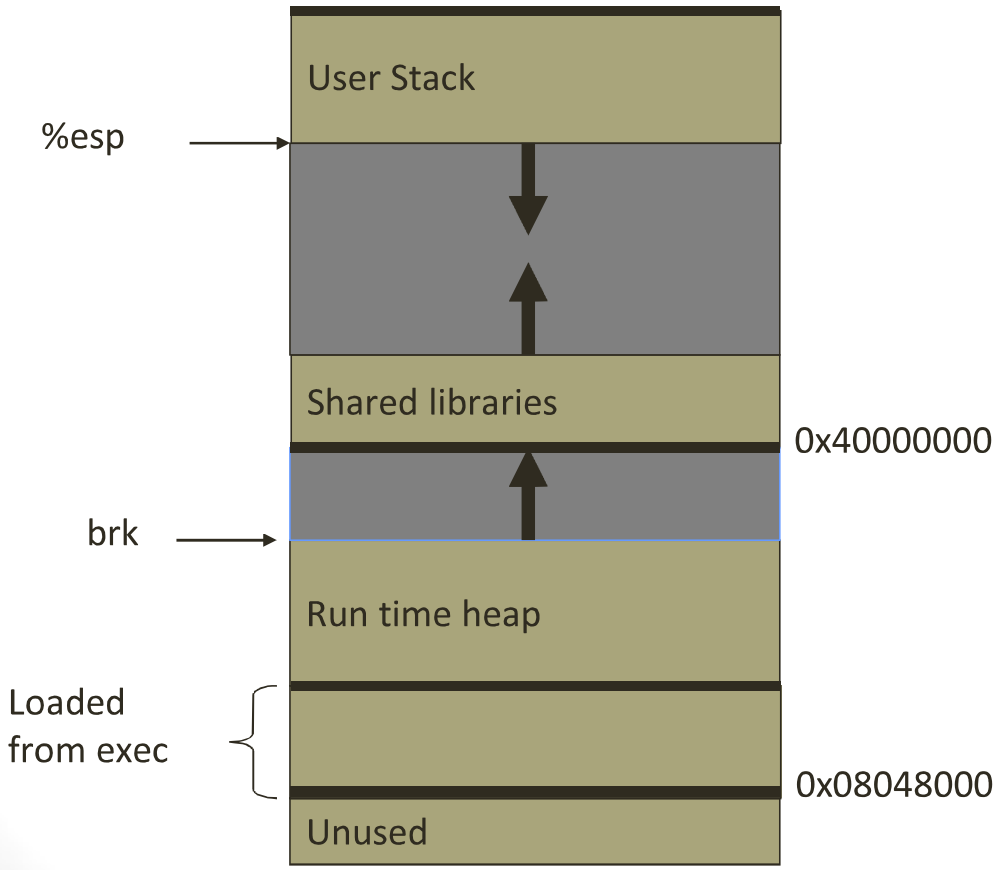
- تا ۲۰۰۵ ۲۰٪ از آسیب پذیریها از نوع سرریز بافر
- از ۲۰۰۵-۲۰۰۷ ۱۰٪



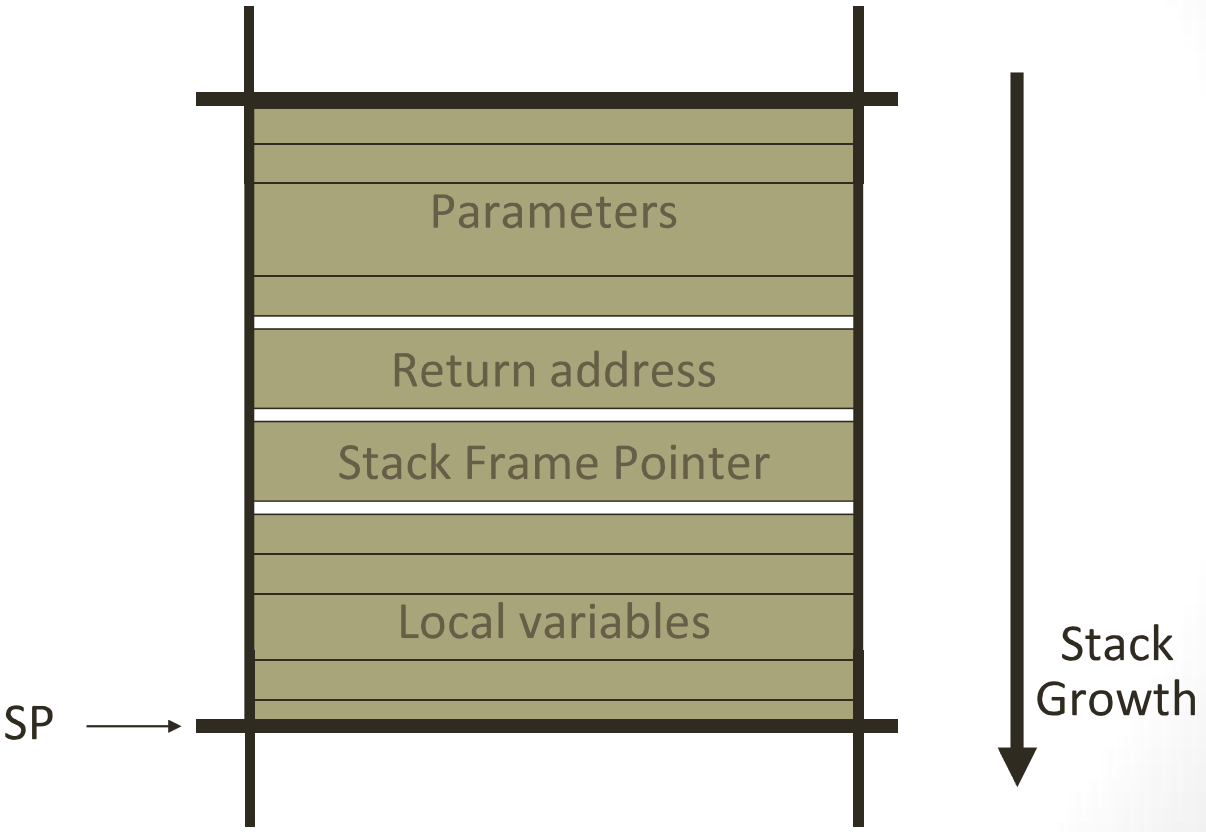
سرریز بافر

- اطلاعات مورد نیاز برای انجام سرریز بافر
  - آشنائی با زبان های برنامه ریزی بخصوص C
  - آشنائی با Application Binary Interface(ABI) شامل ساختار stack، و جهت رشد آن،
  - روش پاس کردن آرگومان های تابع و متغییر های محلی
  - آشنائی با سیستم عامل و فراخوان های سیستمی مانند EXEC()
  - آشنائی با CPU

سازماندهی پروسس در حافظه(Linux)



Stack Frame


Parameters
Return address
Stack Frame Pointer
Local variables
SP
Stack
Growth

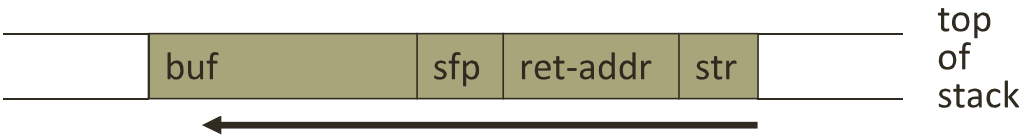
رکورد فعال

- هنگام فراخوانی تابع، یک رکورد فعال برای آن ساخته می شود.
- رکورد فعال در پشته (Stack) نگهداری می شود.
- رکورد فعال شامل موارد زیر است:
  - پارامترهای ورودی
  - آدرس بازگشت
  - آدرس رکورد فعال تابع فراخوان کننده
  - متغیرهای محلی

سرریز بافر

- یک نمومه کد برنامه و شکل stack

```
void func(char *str) {
   char buf[128];
  strcpy(buf, str);
do-something(buf);
                    }
```

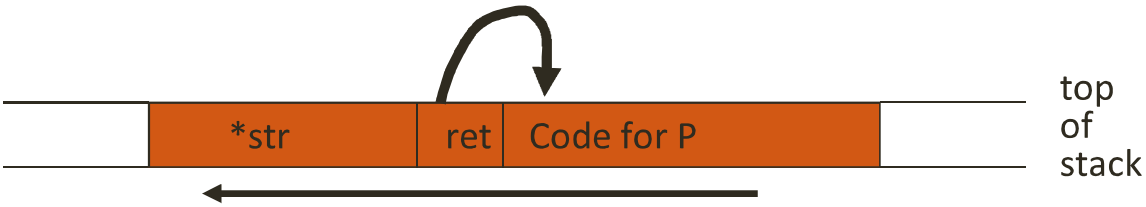


- اگر طول *str برابر ۱۳۶ بایت باشد. شکل stack بعد از اجرای strcpy:

سرریز بافر

- بافر: رشته ای از بایت های متوالی در حافظه با اندازه مشخص
  - بافر ایستا در پشته
  - بافر پویا در Heap
- هنگامی که بیشتر از اندازه بافر، در بافر داده قرار دهیم، سرریز بافر رخ داده است.
- سرریز بافر در پشته می تواند باعث خراب شدن آدرس بازگشت شود (Stack Smashing).
- حمله کننده با سرریز بافر حساب شده، آدرس بازگشت را به کد مورد نظر اشاره می دهد و کنترل برنامه را به دست می گیرد.

- مشکل اصلی: در تابع strcpy() ، محدوده کپی تست نمی شود.
- شکل stack و str* بعد از اجرای تابع strcpy()



Program P: **exec("/bin/sh")**

- به محض خروج از تابع برنامه P اجراء می شود.
- برای محاسبه محل ret باید محل stack را بدانیم.

strcpy (char *dest, const char *src)

strcat (char *dest, const char *src)

gets (char *s)

scanf (const char *format, …)

- “Safe” versions strncpy(), strncat()
  - strncpy()ممکن است که در انتهای رشته 0\ نگذارد.
  - strncpy(), strncat()

- فرض کنید که یک web server تابع func() را صدا می زند. و ورودی های تابع از URL بدست می آید. پس با ارسال کد درست می توان ، remote shell گرفت.
- برنامه قبل از اجرای تابع نباید crash کند.
- کد ارسالی نباید \0 داشته باشد.
- نمونه۱: ۲۰۰۵، فیلد MIME type در برنامه MS outlook
- نمونه۲: ۲۰۰۵، سریز بافر در بر نامه symantec virus detection

```
Set test = CreateObject("Symantec.SymVAFileQuery.1")
test.GetPrivateProfileString  "file",  [long string]
```

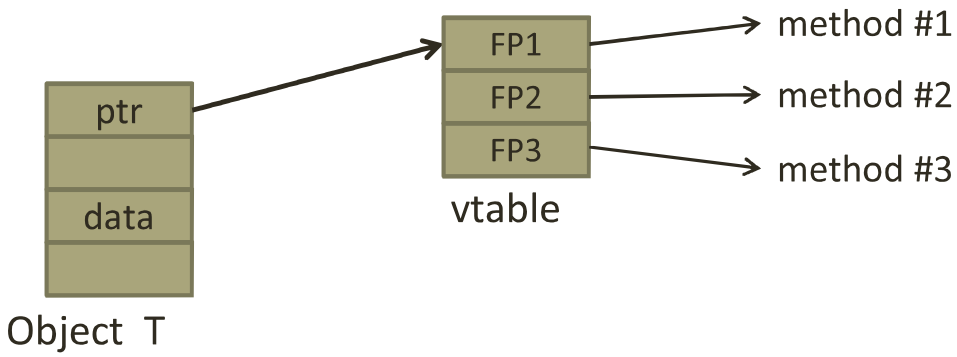
حمله سرریز بافر

- حمله stack smashing
  - نوشتن فیلد RET با سرریز بافر
- نوشتن بر روی فیلد اشاره گر با سرریز بافر
  - (e.g. PHP 4.0.2, MS MediaPlayer Bitmaps)

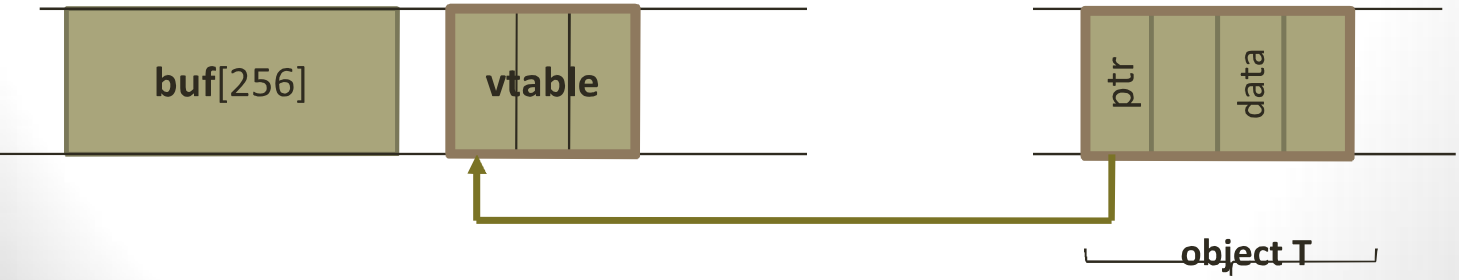
- نوشتن بر روی متغیر آرگومان Longjmp(pos)
  - مثال: Perl 5.003

کنترل برنامه از طریق Heap

- اشاره گرهای تولید شده در heap توسط کامپایلر

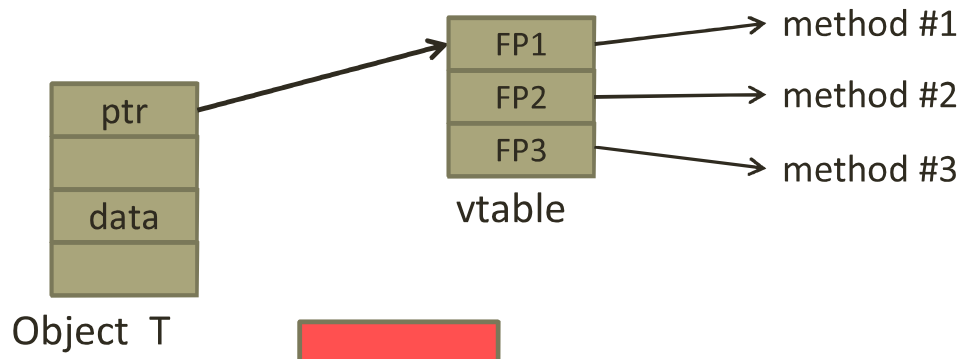


- حال اگر vtable بعد از یک بافر باشد با سریز بافر می توان FP1 را عوض کرد.

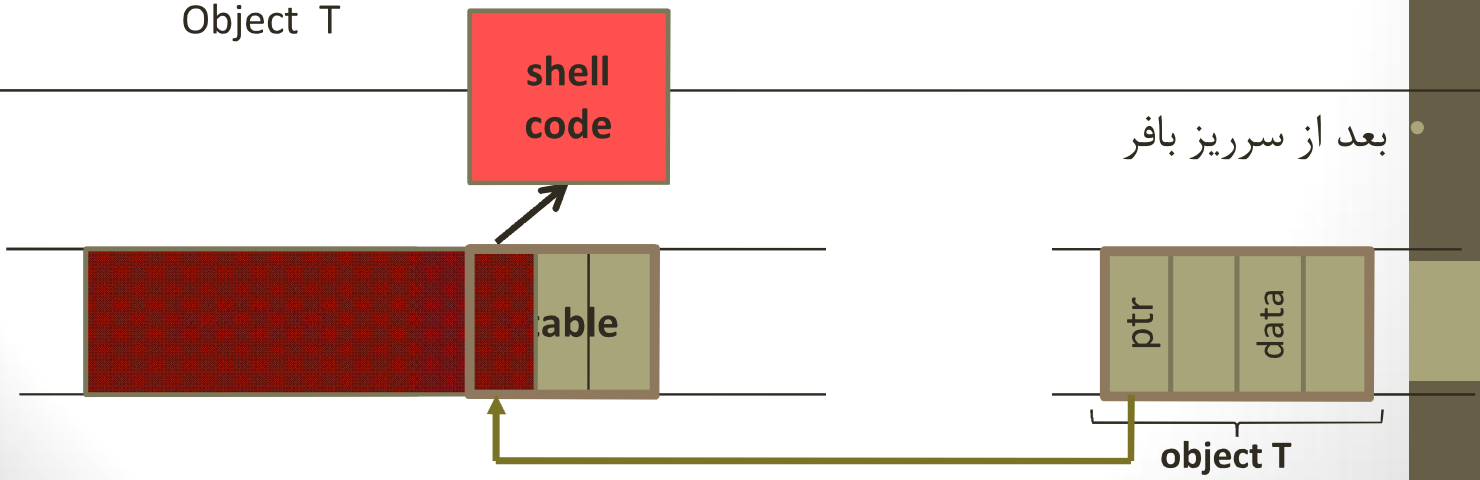


اشاره گرهای تولید شده در heap توسط کامپایلر

- اشاره گرهای تولید شده در heap توسط کامپایلر



- بعد از سرریز بافر



دفاع در برابر سرریز بافر

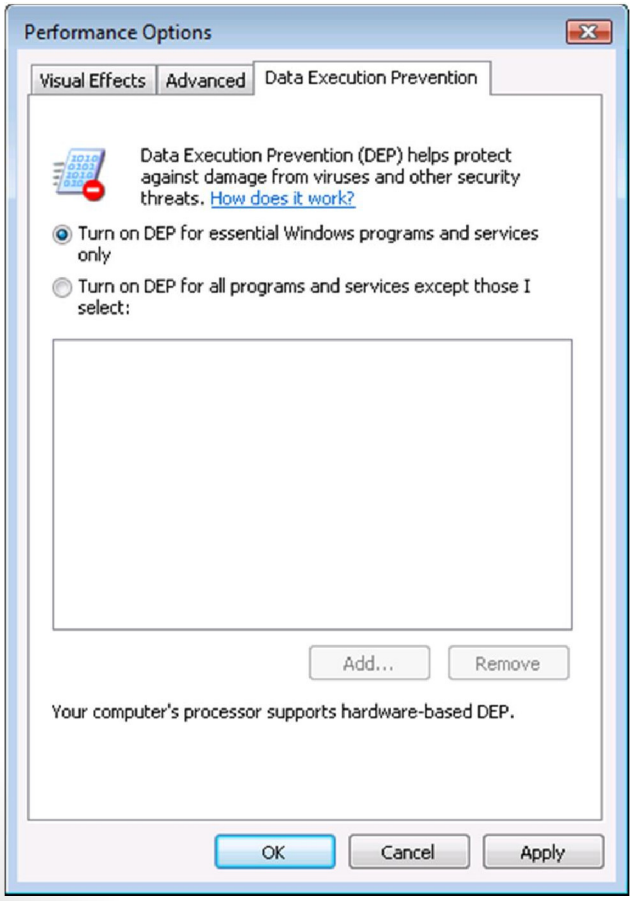
وگیری از حمله های کنترل برنامه

- رفع bugها
  - ممیزی نرم افزار ها
  - ایزار ها خودکار: Coverity, Prefast/Prefix
  - دوباره نویسی برنامه ها با زبانهای امن مانند Java
    - هزینه بالا برای برنامه های موجود
- اجاره دادن برای سرریز بافر ولی جلوگیری از اجراء کد خرابکار
- اضافه کردن کد های هنگام اجراء برای اشکار سازی کنترل برنامه(exploits)
  - Halt پردازنده بعد از آشکارسازی overflow exploits
  - StackGuard, LibSafe, ...

غیر قابل اجراء کردن فضای حافظه (W^X)

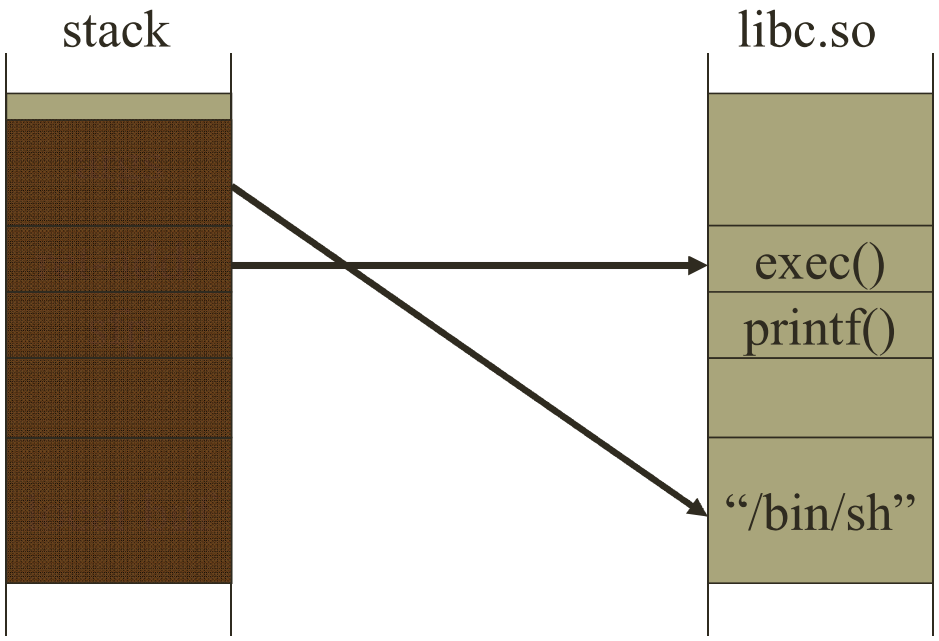
- برای جلوگیری از اجرای برنامه stack و بخشی از heap را غیر قابل اجرا می کنیم.
  - NX-bit on AMD Athlon 64, XD-bit on Intel P4 Prescott
    - NX bit in every Page Table Entry (PTE)
  - پیاده سازی شده
    - Linux (via PaX project); OpenBSD
    - Windows since XP SP2 (DEP)
      - Boot.ini : **/noexecute=OptInorAlwaysOn**
- محدودیت ها
  - بعضی از کاربردها نیازبه قابیت اجرا heap را دارند
  - از کنترل برنامه توسط فراخوانی برنامه ازفضای libc جلوگیری نمی کند

Examples: DEP controls in Vista



DEP terminating a program

حمله: فراخوانی از فضای libc

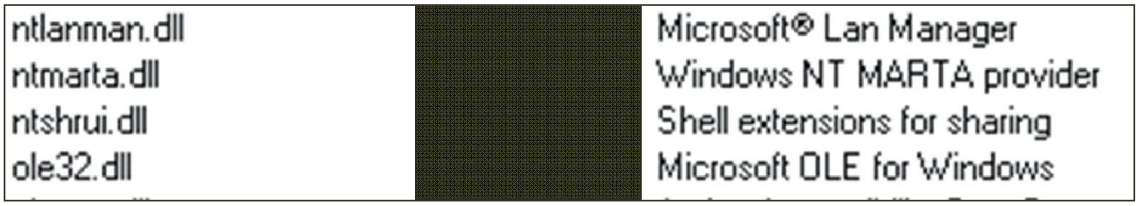


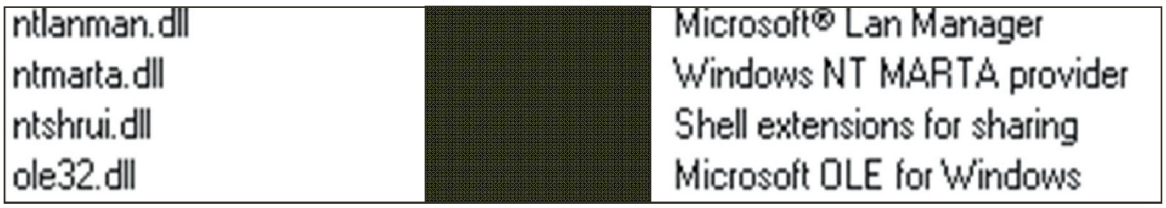
روش جلوگیری:randomization

- **ASLR**: (Address Space Layout Randomization)
  - تصادفی کردن محل قرار کتابخانه libc
  - <== حمله کننده مستقیما نمی تواند به توابع اجرائی بپرد
  - پیاده سازی شده
- Windows Vista: 8 bits of randomness for DLLs
  - aligned to 64K page in a 16MB region ⇒ 256 choices
- **Linux** (via PaX): 16 bits of randomness for libraries
  - روشهاس دیگر تصادفی سازی
    - تصادفی سازی id های فراخوانی توابع سیستمی

مثالASLR

Booting Vista twice loads libraries into different locations:



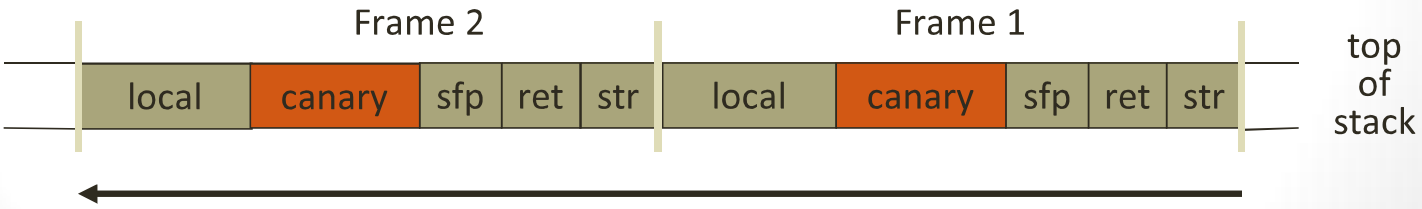


ASLR is only applied to images for which the dynamic-relocation flag is set

بررسی زمان اجراء

StackGuard

- روش های زیادی برای بررسی در زمان اجراء وجود دارد که در اینجا فقط روش های مورد نظر برای محافظت از سرریز بافر ارائه می گردد
- روش ۱: StackGuard
- بررسی عدم تغییر Stack در زمان اجراء
- قرار دادن کد canary در stack قبل از محل آدرس برگشت تابع و بررسی درستی آن قبل از بازگست تابع



انواع کد قناری

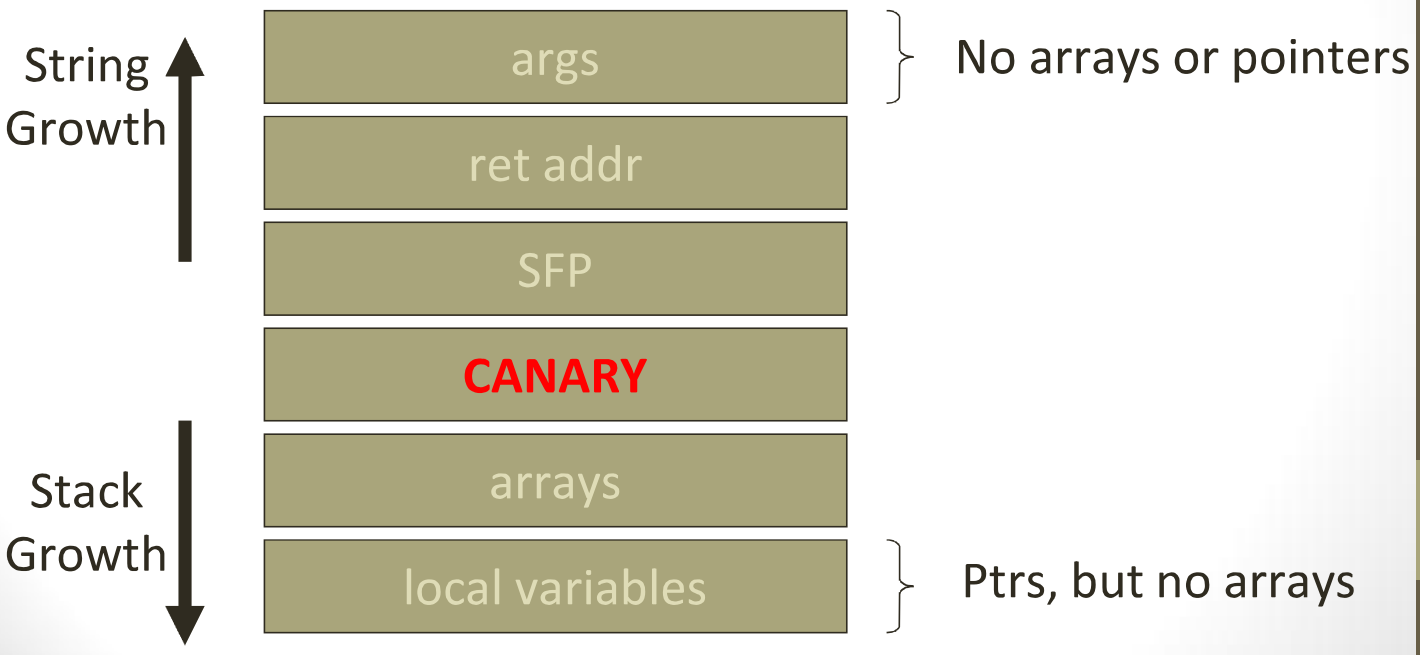
- کد قناری تصادفی : یک کد تصادفی در شروع اجرای برنامه در نظر گرفته شده و به ازای هر فراخوانی توابع داخل برنامه در پشته قرار داده شده و قبل از برگشت از تابع بررسی می شود.
- کد قناری Terminator
  Canary = 0, newline, linefeed, EOF
- با قرار دادن این کد امکان کپی کردن بعد از کد در پشته توسط تابع strcpy() امکان پذیر نیست.
- این روش در کامپایلر gcc پیاده سازی شده است و برنامه ها باید دوباره کامپایل شوند.
- باعث کاهش کارائی برنامه ها می شود. مثلا کارآئی برنامه apache، ۸٪ کم شده است.
- بعضی از روش ها کد قناری را دورمی رنند.

PointGuard

- محافظت از Heap
- اشاره گر به توابع و بافر setjmp رمز می شود.(با یک عدد تصادفی xor می شود).
- کارائی را به صورت قابل توجهی کم می کند.

ProPolice

- همان روش stackguard با کمی تغییر است.
- ProPolice (IBM) - gcc 3.4.1. (**-fstack-protector**)
  - تغییر ترتیب فیلد ها در پشته



MS Visual Studio /GS [2003]

- سوئیچ GS/ برای حفاظت از سرریز بافر
- ترکیبی از قناری تصادفی و Propolice
- در صورتی که کد قناری تغییر یابد یک کد ExceptionHandler برای shutdown کردن پروسس استفاده می شود.

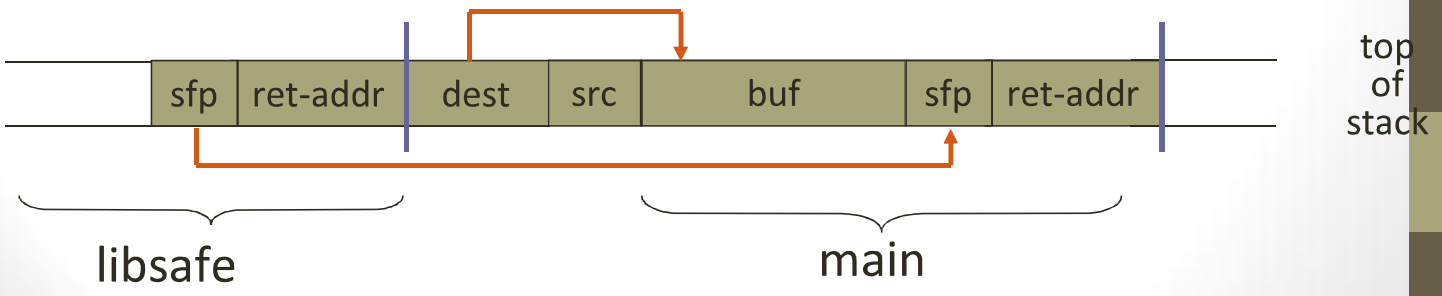
```
mov  eax,dword ptr [___security_cookie]
xor  eax,ebp
mov  dword ptr [ebp-8],eax
...
mov  ecx,dword ptr [ebp-8]
xor  ecx,ebp
call __security_check_cookie@4
```

- آسیب پذیری Litchfield
- با سرریز بافر ExceptinHandler عوض شده و exception به کد حمله کننده هدایت می شود

بررسی زمان اجراء

- راه حل ۲: LibSafe (Avaya Labs)
- به صورت یک کتابخانه پویا به حافظه بار می شود.
- هر وقت تابع strcpy فراخوانی شود میزان فضای کافی در پشته را بررسی می کند. اگر مشکلی نبود strcpy انجام می شود و در غیراینصورت برنامه قطع می شود.

|frame-pointer – dest| >strlen(src)



روش های دیگر...

- StackShield
  - در ابتدای تابع فراخوانی شده (Function Prologue) مقدار فیلد RET و SFP پشته در یک محل مناسب در حافظه کپی شده و قبل از برگشت مفدار RET و SFP با مقادیر ذخیره شده مقایسه می شود.
  - در GCC یه عنوان assembler پیاده سازی شده است.

بررسی فلوی برنامه به صورت استاتیک و پویا و در زمان اجراء فلوی برنامه کنترل می شود.

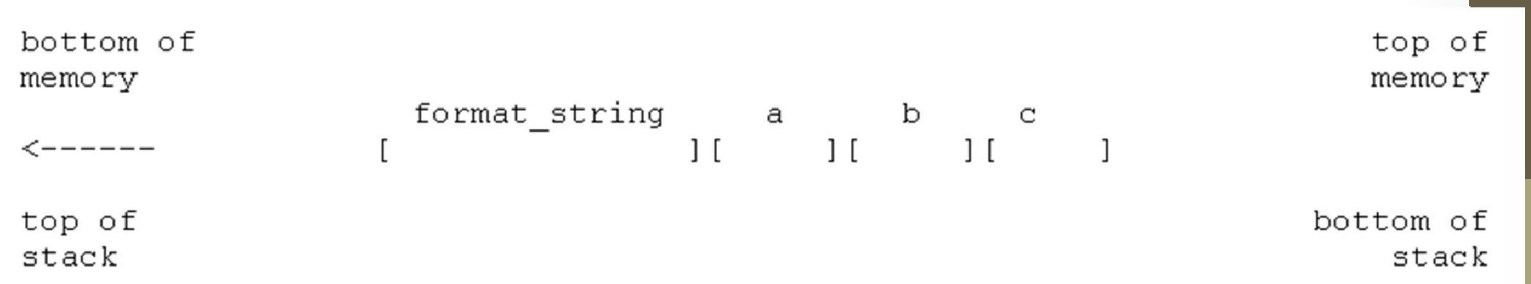
آسیب پذیری رشته فرمت (Format String Vulnerability)

توابع فرمتی

- توابعی که تعداد متغیری پارامتر به عنوان ورودی می گیرند و بر اساس یک رشته فرمت آن ها را چاپ می کنند.
- توابع فرمتی در زبان C
  - printf
  - fprintf
  - sprintf, snprintf
  - vprintf, vfprintf, vsprintf, vsnprintf

نحوه کار توابع فرمتی

- printf(“%d %u %s”, a, b, c)
- شکل زیر نحوه قرارگیری آرگومان های توابع فرمتی را در پشته نشان می دهد.
- توابع فرمتی رشته فرمت را بررسی کرده و متغیرهای مربوطه را از پشته pop می کنند.



نحوه آسیب پذیر شدن رشته فرمت

```
void function(char* data) {
        printf(data); // wrong

        printf("%s", data); // correct
}
```

استفاده از آسیب پذیری رشته فرمت

- crash کردن برنامه
  - data = “%s%s%s%s%s%s%s”
  - تابع فرمتی سعی می کند از پشته pop کند تا جایی که برنامه از محدوده آدرس خود خارج شده و crash می کند.
- خواندن محتویات حافظه (پشته)
  - data = “%x%x%x%x%x%x%x”
  - تابع فرمتی از پشته pop کرده و اطلاعات درون پشته را چاپ می کند.

منابع

- Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade
- Smashing The Stack For Fun And Profit
- How do buffer overflow attacks work?
- Basic Integer Overflows
- Exploiting Format String Vulnerabilities