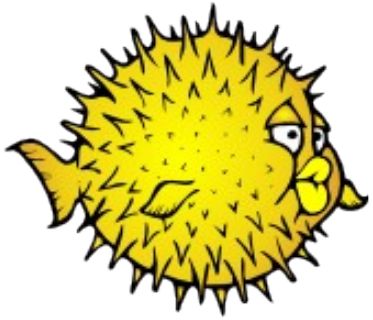
Mecanismos de Segurança Default no OpenBSD



OpenBSD

Fábio Olivé
(fabio.olive@gmail.com)


Tópicos

- A cultura de desenvolvimento do OpenBSD
- O foco em código correto, claro e simples
- Segurança é um sub-produto do código correto
- O uso de criptografia forte e números aleatórios
- Address Space Layout Randomization “extreme”
- Separação de privilégios e confinamento

Disclaimer: sou apenas um [ex-]usuário entusiasta do OpenBSD, que tem um entendimento razoável das várias técnicas utilizadas.


Cultura de Desenvolvimento do OpenBSD

Shut up and hack!

livre
OeSC


Cultura de Desenvolvimento

- Descendente direto do Unix original, vindo do 4.4BSD através do NetBSD e alguns outros
- Feito para desenvolvedores e usuários experientes
- A documentação é ótima, então RTFM (Fine)


Cultura de Desenvolvimento

- Descendente direto do Unix original, vindo do 4.4BSD através do NetBSD e alguns outros
- Feito para desenvolvedores e usuários experientes
- A documentação é ótima, então RTFM (Fine)
- Se mantém o mais fiel possível à filosofia e padrões do Unix (POSIX, SUS, ANSI, etc)
- É um Sistema Operacional completo, desenvolvido de forma integral
  - Diferente das “coleções de pacotes” do GNU/Linux
  - Fontes de **todo o sistema** em `/usr/src`


Cultura de Desenvolvimento

- Free, Functional, Secure
  - Ser um Unix livre, cheio de funcionalidades e seguro
- Suporta várias arquiteturas de hardware
  - Ajuda a encontrar bugs!


Cultura de Desenvolvimento

- Free, Functional, Secure
  - Ser um Unix livre, cheio de funcionalidades e seguro
- Suporta várias arquiteturas de hardware
  - Ajuda a encontrar bugs!
- Sistema base contém tudo o que se esperaria encontrar em um sistema Unix típico
  - Aplicações externas (gnome, etc) mantidas em separado, através do ports e pacotes
- Ênfase em serviços e protocolos de rede
  - pf, ipsec, openssh e outros daemons open*


Foco em Código Correto, Claro e Simples

“One of my most productive days
was throwing away 1000 lines of code.”
-- Ken Thompson, criador do Unix


Código Correto, Claro e Simples

- Design simples, código simples, fácil de verificar
  - Simples e ao mesmo tempo poderoso (paradoxo?)
- Jogar fora código que não é usado
  - Repositório versionado, nada se perde de verdade
- Conhecer as APIs que se usa


Código Correto, Claro e Simples

- Design simples, código simples, fácil de verificar
  - Simples e ao mesmo tempo poderoso (paradoxo?)
- Jogar fora código que não é usado
  - Repositório versionado, nada se perde de verdade
- Conhecer as APIs que se usa
- Padronizar estilo e design patterns do sistema
- Ao achar um bug, procure-o em todo o sistema
  - Se possível, asbtraia o “pattern” do bug
- Tratar erros retornados pela API
  - Sério! É óbvio mas não é comum tratar todos os erros.


Código Correto, Claro e Simples

- O compilador é seu amigo, não um obstáculo
  - Código que compila com -Wall -Wextra -Wpedantic não é automaticamente seguro, mas é bem provável que bugs óbvios serão pegos
- Resista à tentação da POG
  - Os revisores do código vão perceber as gambiarras e não vai entrar, e sua reputação ficará manchada


Código Correto, Claro e Simples

- O compilador é seu amigo, não um obstáculo
  - Código que compila com -Wall -Wextra -Wpedantic não é automaticamente seguro, mas é bem provável que bugs óbvios serão pegos
- Resista à tentação da POG
  - Os revisores do código vão perceber as gambiarras e não vai entrar, e sua reputação ficará manchada
- Código esperto em geral é incorreto em algum caso
  - “Premature optimization is the root of all evil.” -- D. Knuth
- Profiling só depois que estiver correto e rodando
  - Onde realmente vale a pena otimizar?


Segurança é um Sub-Produto do Código Correto

A maioria dos furos de segurança é uma combinação de simples bugs.


Segurança: Código Correto

- “Secure by default” (lema desde 1996)
- O design é furado? Corrija o design
- O código é furado? Refaça o código
- A documentação está errada? Melhore os docs!
  - Uma ferramenta segura porém mal documentada é quase o mesmo que uma ferramenta insegura


Segurança: Código Correto

- “Secure by default” (lema desde 1996)
- O design é furado? Corrija o design
- O código é furado? Refaça o código
- A documentação está errada? Melhore os docs!
  - Uma ferramenta segura porém mal documentada é quase o mesmo que uma ferramenta insegura
- APIs alternativas, se a original é confusa ou ambígua
  - strcpy, responsável por muitos buffer overflows
- Use tipos corretos e evite integer {over,under}flows
- Detectou um erro? Na dúvida, termine o processo


Uso de Criptografia Forte e Números Aleatórios

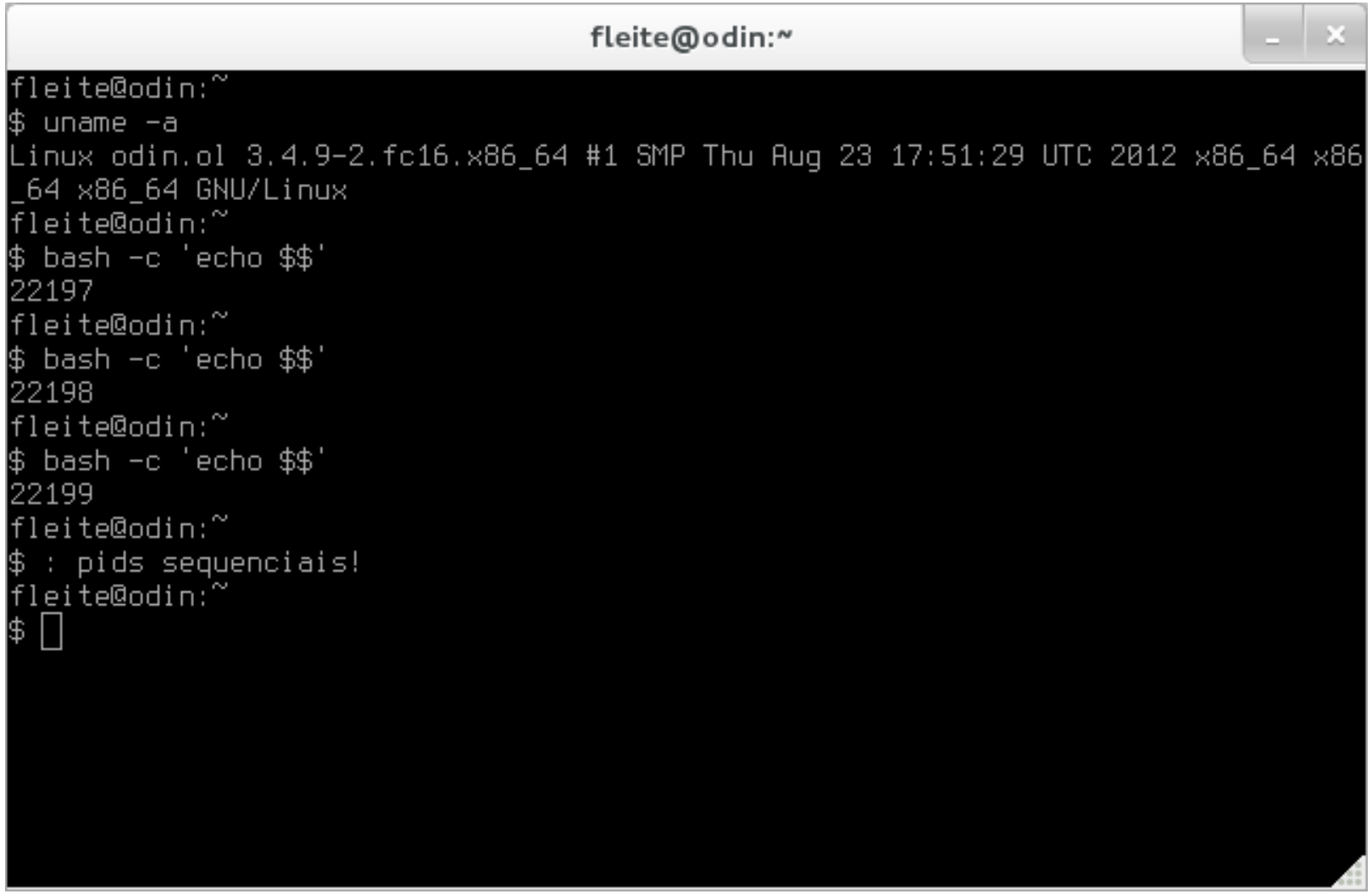
Ambiente hostil para exploits.

Cripto Forte e Números Aleatórios

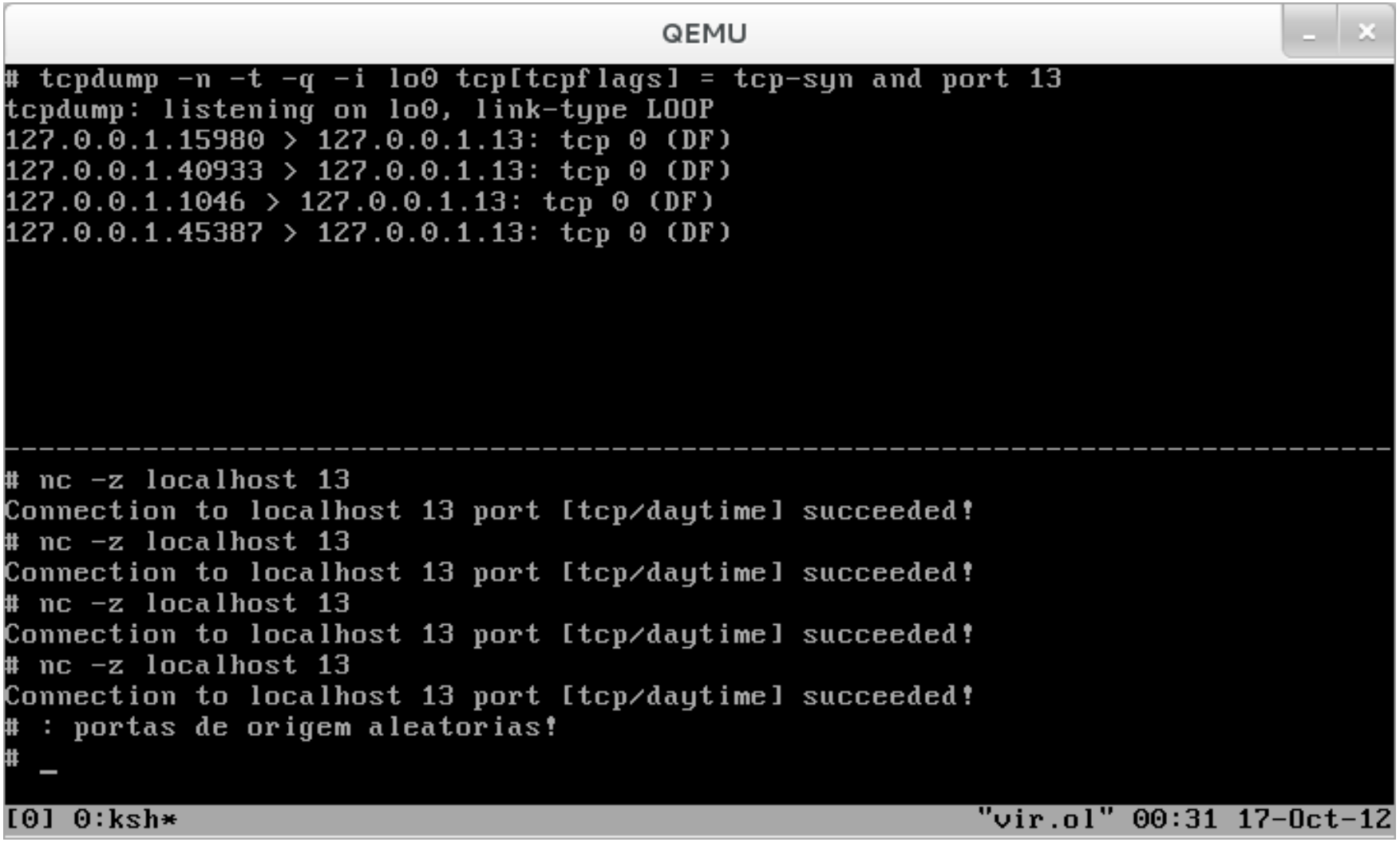
- Muitos ataques dependem de poder prever o comportamento do sistema
  - Pids sequenciais, portas e IDs em protocolos de rede, ...
  - OpenBSD usa números aleatórios sempre que possível

Cripto Forte e Números Aleatórios

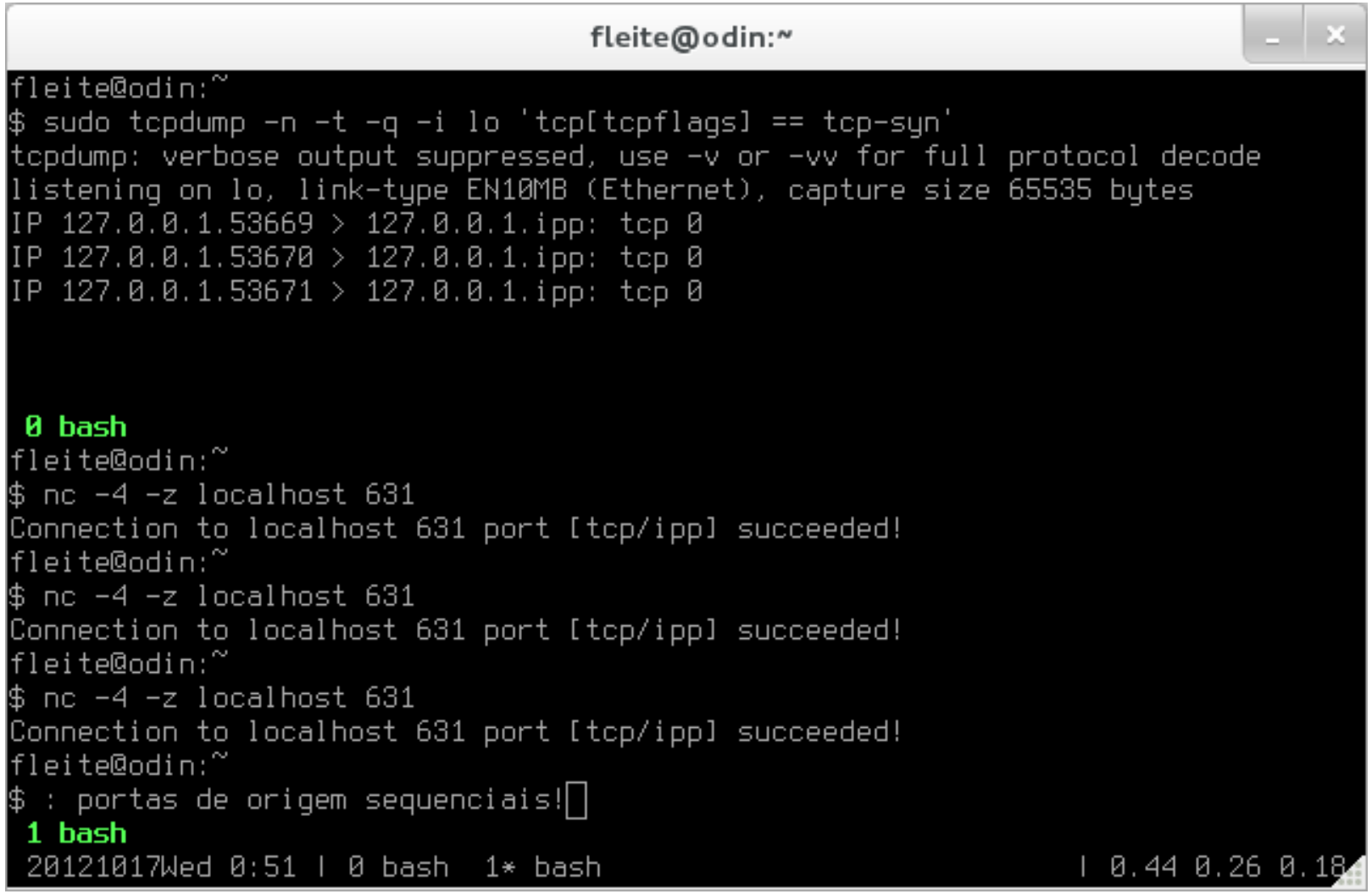
Cripto Forte e Números Aleatórios



Cripto Forte e Números Aleatórios



Cripto Forte e Números Aleatórios




Cripto Forte e Números Aleatórios

- Muitos ataques dependem de poder prever o comportamento do sistema
  - Pids sequenciais, portas e IDs em protocolos de rede, ...
  - OpenBSD usa números aleatórios sempre que possível
- Muitos sistemas comerciais e fechados usam criptografia fraca, ou cripto forte mal implementada
  - Play Station Network, por exemplo
  - Ninguém faz auditoria, pois o software é fechado
  - OpenBSD começou cedo a incorporar algoritmos fortes
    - Antiga(?) restrição de exportação de algoritmos de cripto nos EUA


Cripto Forte e Números Aleatórios

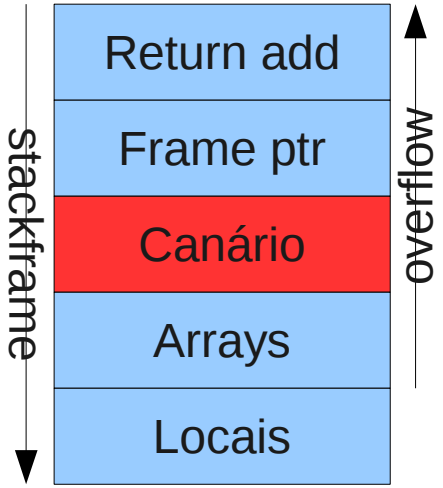
- Sistemas sem proteção vazam muita informação em cada conexão com o sistema
  - Timestamps, conjuntos de flags ou padrão de progressão das portas e IDs permitem identificar o sistema remotamente (nmap, etc)
- Observando os processos na máquina se pode prever os pids
  - Ataques antigos de mktemp, etc


Cripto Forte e Números Aleatórios

- Sistemas sem proteção vazam muita informação em cada conexão com o sistema
  - Timestamps, conjuntos de flags ou padrão de progressão das portas e IDs permitem identificar o sistema remotamente (nmap, etc)
- Observando os processos na máquina se pode prever os pids
  - Ataques antigos de mktemp, etc
- O uso de números aleatórios impede a identificação do Sistema Operacional da máquina e sua versão
- OpenBSD como firewall pode “injetar aleatoriedade”
  - Protege outros sistemas menos seguros

Cripto Forte e Números Aleatórios

- Proteção de buffer-e stack-overflow
  - ProPolice, Stack Smashing Protection
  - Atacante precisa adivinhar um número de 32 ou 64bits
  - Processo morre em caso de erro



Estourar um array (buffer overflow) vai matar o canário se conseguir mudar o endereço de retorno. As instruções no final de cada função verificam o canário antes de retornar e matam o processo se tiver mudado.

Cripto Forte e Números Aleatórios

fleite@odin:~/src

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
        int magic = 42;
        char buf[10];

        if ((void *)&magic < (void *)buf)
                printf("magic está antes de buf.\n");
        else
                printf("magic está DEPOIS de buf.\n");

        /* buffer overflow clássico com strcpy */
        strcpy(buf, argv[1]);
        printf("O número mágico é %d.\n", magic);

        return 0;
}
~
~
~
~
```

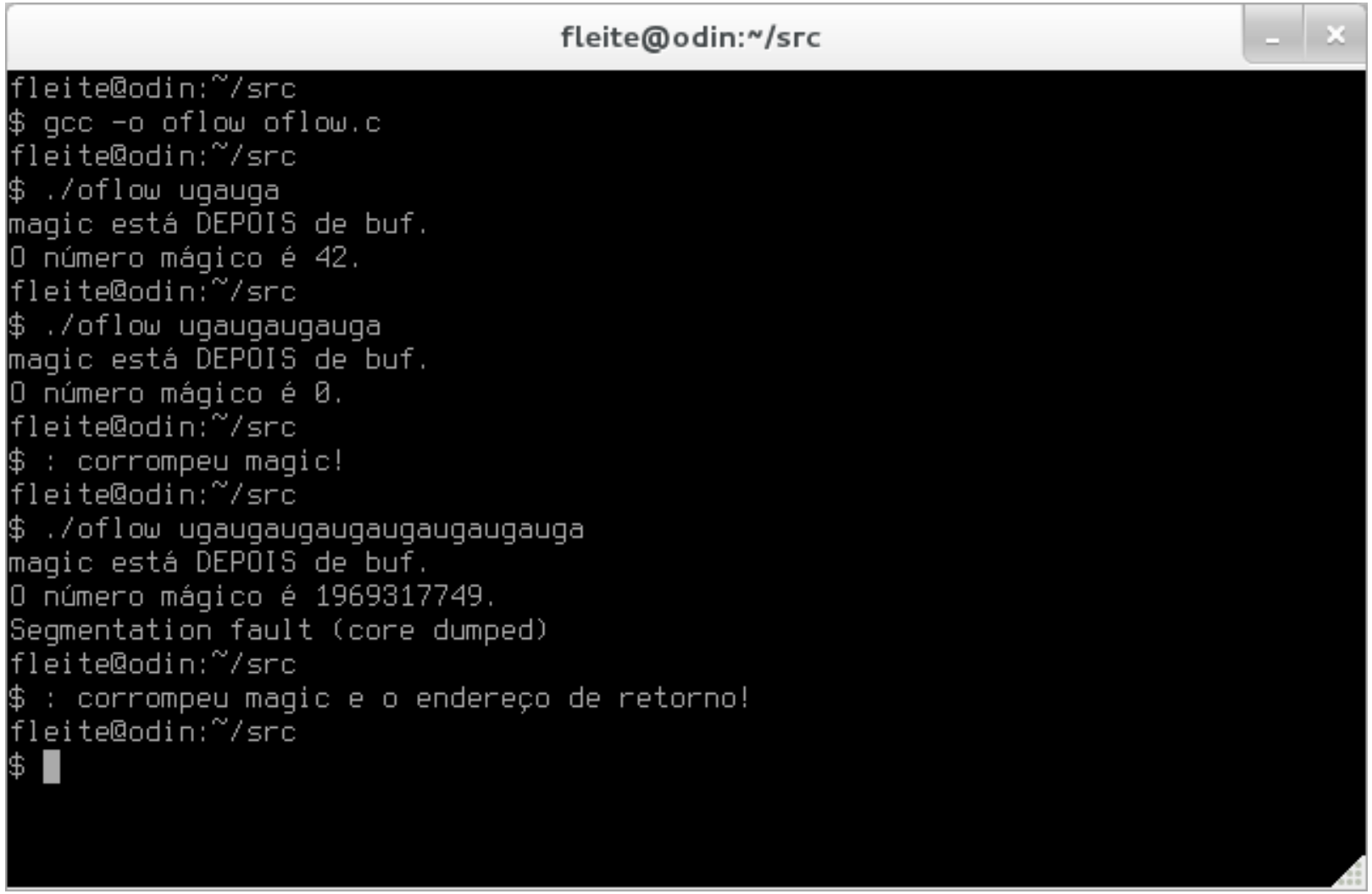
14,1-8 All

Cripto Forte e Números Aleatórios

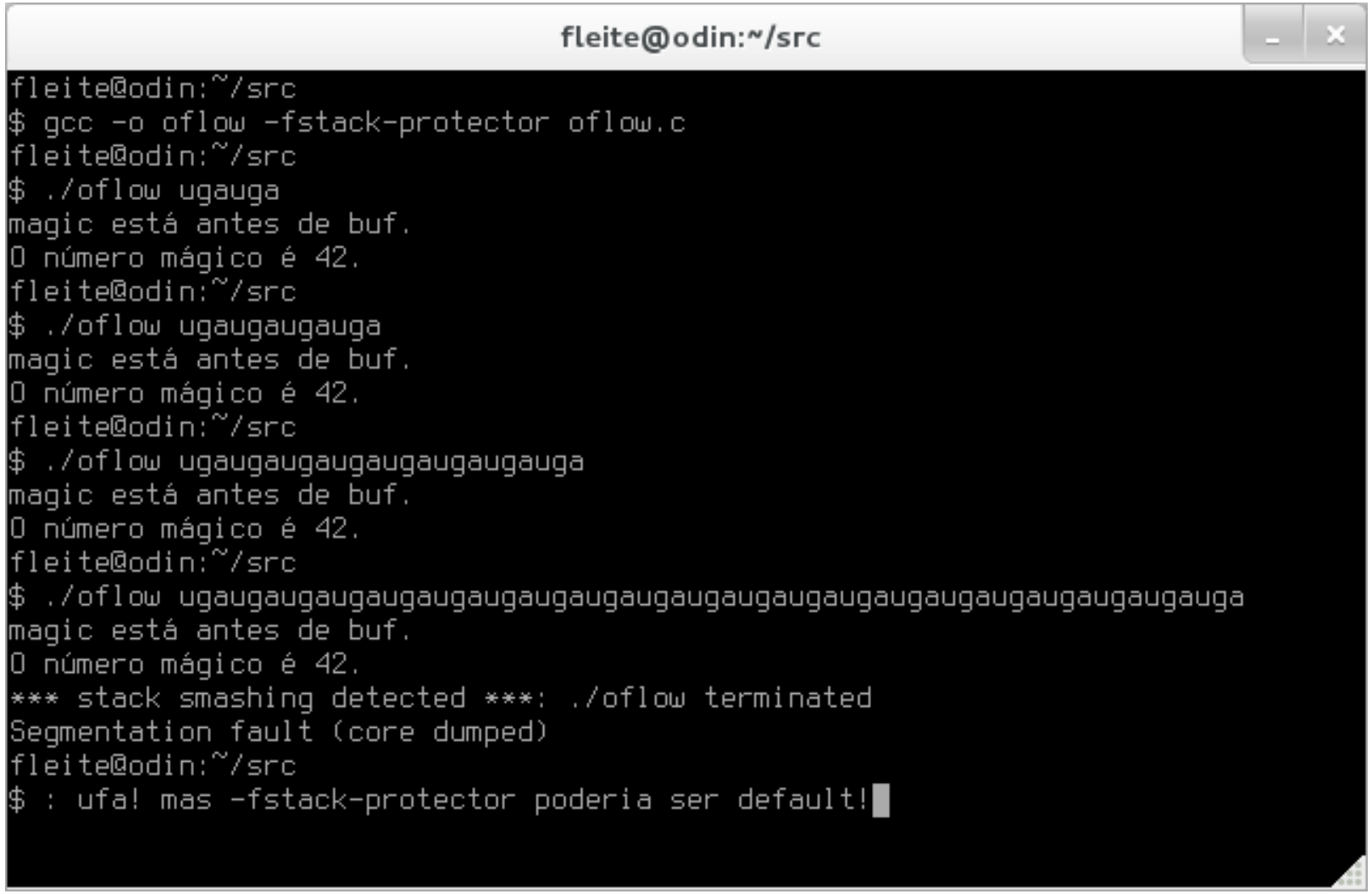
QEMU

```
# gcc -o oflow oflow.c
/tmp//ccP5UfRk.o(.text+0x5a): In function `main':
: warning: strcpy() is almost always misused, please use strlcpy()
# ./oflow ugauga
magic esta antes de buf.
O numero magico e 42.
# ./oflow ugaugaugauga
magic esta antes de buf.
O numero magico e 42.
# ./oflow ugaugaugaugaugauga
magic esta antes de buf.
O numero magico e 42.
# ./oflow ugaugaugaugaugaugaugaugaugaugaugauga
magic esta antes de buf.
O numero magico e 42.
Abort trap (core dumped)
# ./oflow ugaugaugaugaugaugaugaugaugaugaugaugaugaugaugaugaugaugaugauga
magic esta antes de buf.
O numero magico e 42.
Abort trap (core dumped)
# : magic nao foi corrompido e o processo morreu quando corrompeu a pilha
# _
```

Cripto Forte e Números Aleatórios



Cripto Forte e Números Aleatórios




Address Space Layout Randomization “extreme”

Nada fica no mesmo lugar duas vezes.

ASLR “extreme”

- Muitos ataques dependem de saber a posição na memória em que certos buffers, pilhas ou libs estão
  - Linker dinâmico típico é previsível e faz tudo sempre igual
  - Endereços continuam os mesmos em outras máquinas
- OpenBSD atualmente varia o endereço de tudo
  - Pilha, heap, bibliotecas, até o executável agora é PIE
  - Reorganiza parâmetros e variáveis de funções na pilha

ASLR “extreme”

fleite@odin:~/src

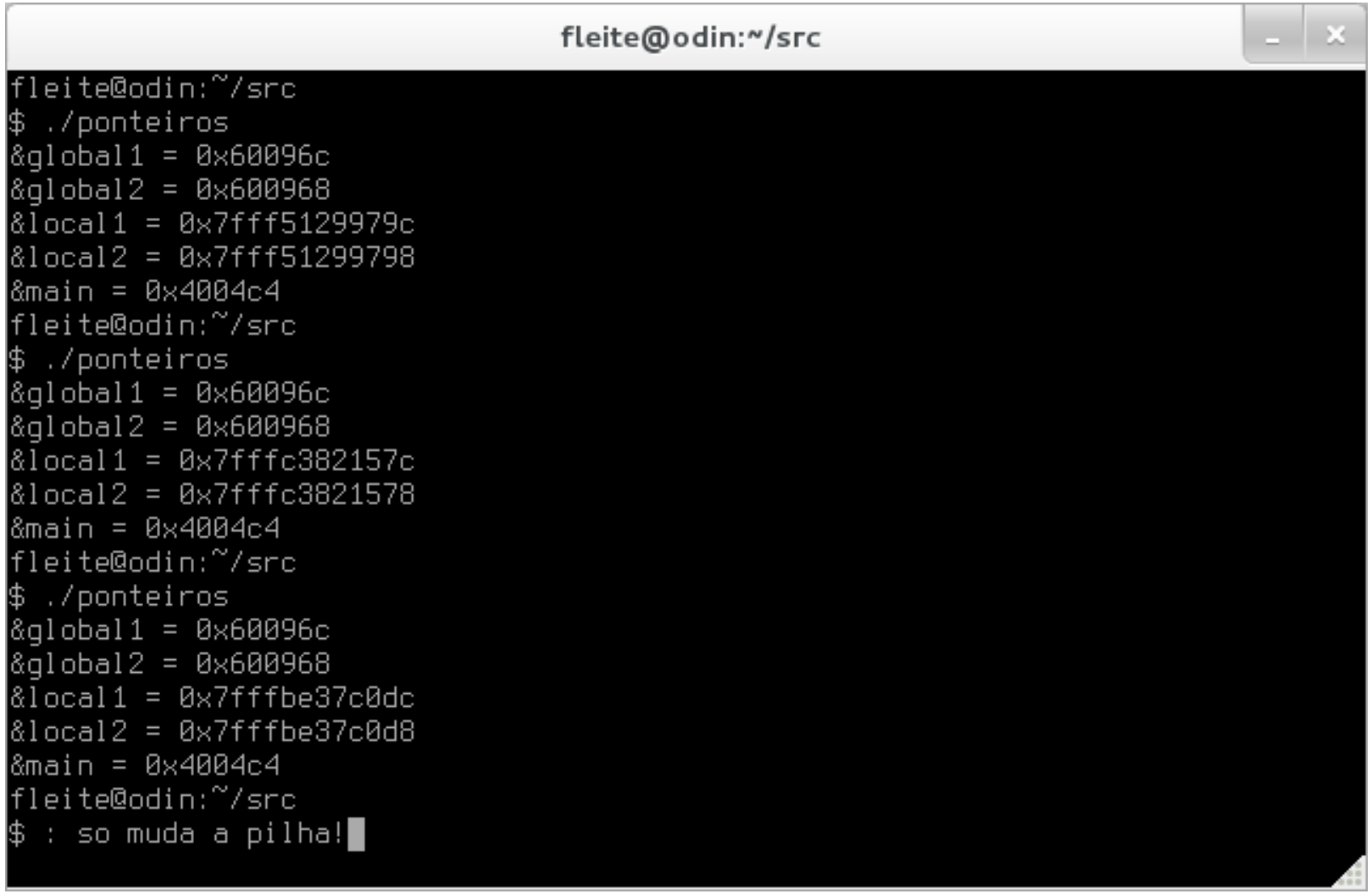
```
#include <stdio.h>

int global1, global2 = 0;

void main(void) {
        int local1, local2 = 0;

        printf("&global1 = %p\n", &global1);
        printf("&global2 = %p\n", &global2);
        printf("&local1 = %p\n", &local1);
        printf("&local2 = %p\n", &local2);
        printf("&main = %p\n", main);
}
~
~
~
~
~
~
~
~
~
~
"ponteiros.c" 14L, 273C                                12,2-9        All
```

ASLR “extreme”

QEMU

```
# uname -a
OpenBSD vir.ol 5.2 GENERIC#10 amd64
# ./ponteiros
&global1 = 0x290fdd01460
&global2 = 0x290fdb01020
&local1 = 0x7f7ffffd8aec
&local2 = 0x7f7ffffd8ae8
&main = 0x290fd900ce0
# ./ponteiros
&global1 = 0x3576fb01460
&global2 = 0x3576f901020
&local1 = 0x7f7fffffad6c
&local2 = 0x7f7fffffad68
&main = 0x3576f700ce0
# ./ponteiros
&global1 = 0x37299e01460
&global2 = 0x37299c01020
&local1 = 0x7f7ffffca6dc
&local2 = 0x7f7ffffca6d8
&main = 0x37299a00ce0
# : enderecos sempre diferentes!
# _
```

ASLR “extreme”

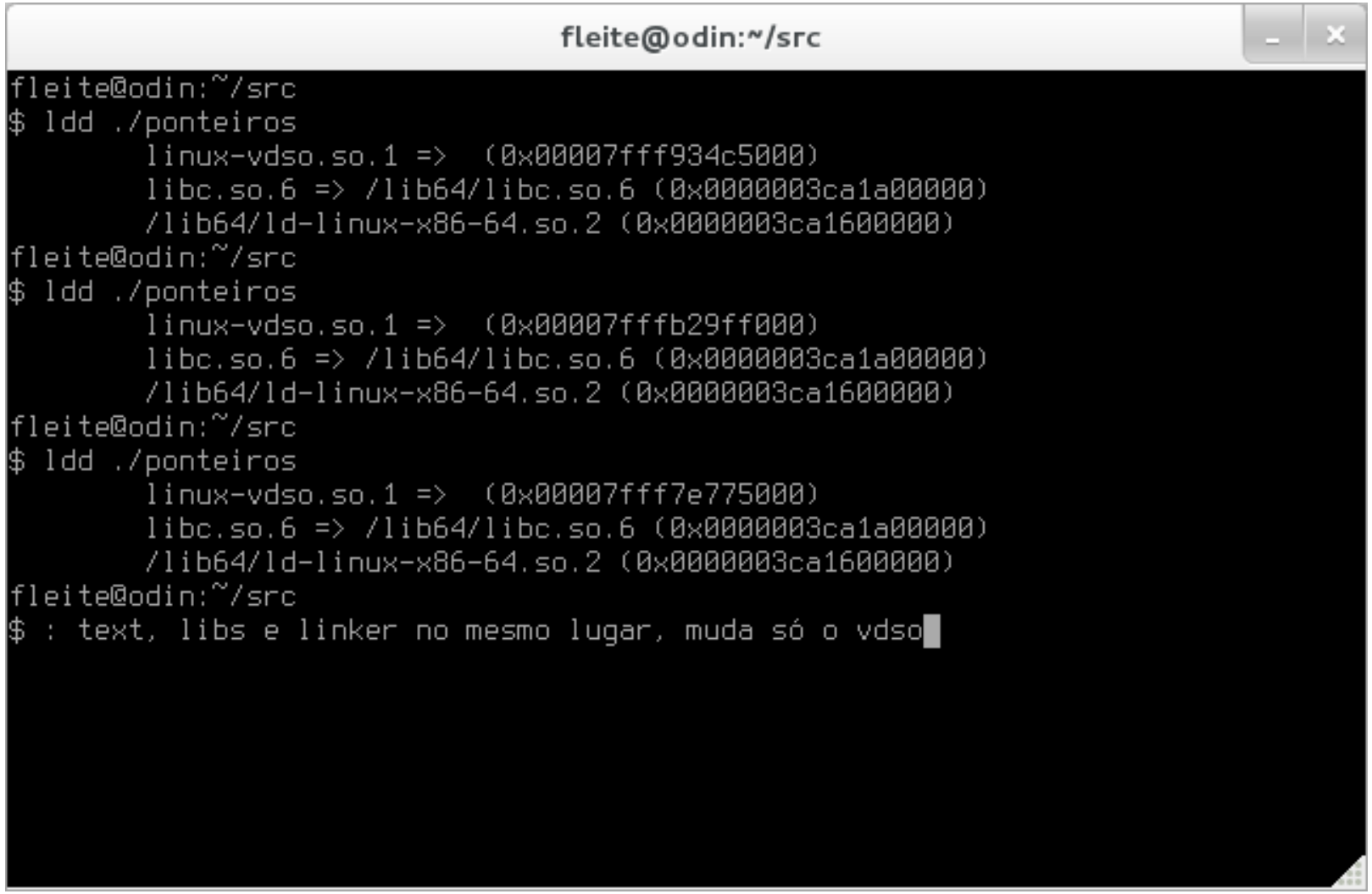


ASLR “extreme”

QEMU

```
# ldd ./ponteiros
./ponteiros:
        Start            End              Type Open Ref GrpRef Name
        000018fe04800000 000018fe04c02000 exe  1    0   0      ./ponteiros
        000019000baa4000 000019000bf8c000 rlib 0    1   0      /usr/lib/libc.so.
66.0
        000019000a900000 000019000a900000 rtld 0    1   0      /usr/libexec/ld.s
o
# ldd ./ponteiros
./ponteiros:
        Start            End              Type Open Ref GrpRef Name
        0000173c5f800000 0000173c5fc02000 exe  1    0   0      ./ponteiros
        0000173e6bd55000 0000173e6c23d000 rlib 0    1   0      /usr/lib/libc.so.
66.0
        0000173e63100000 0000173e63100000 rtld 0    1   0      /usr/libexec/ld.s
o
# ldd ./ponteiros
./ponteiros:
        Start            End              Type Open Ref GrpRef Name
        000001a589300000 000001a589702000 exe  1    0   0      ./ponteiros
        000001a79291b000 000001a792e03000 rlib 0    1   0      /usr/lib/libc.so.
66.0
        000001a798b00000 000001a798b00000 rtld 0    1   0      /usr/libexec/ld.s
o
# : text, libs e ate o linker muda de lugar!
```

ASLR “extreme”




ASLR “extreme”

- Muitos ataques dependem de saber a posição na memória em que certos buffers, pilhas ou libs estão
  - Linker dinâmico típico é previsível e faz tudo sempre igual
  - Endereços continuam os mesmos em outras máquinas
- OpenBSD atualmente varia o endereço de tudo
  - Pilha, heap, bibliotecas, até o executável agora é PIE
  - Reorganiza parâmetros e variáveis de funções na pilha
- Com ASLR um atacante que consegue descobrir endereços interessantes tem uma janela muito curta de validade destes endereços (vida útil do processo)


Separação de Privilégios e Confinamento

Cada um na sua caixinha.


Separação de Priv. e Confinamento

- Uso muito limitado de root e setuid
  - Adquire os recursos privilegiados e solta os privilégios
  - Solta privilégios antes de ter qualquer contato com a rede
- Se não precisa acessar arquivos, chroot pra um diretório vazio no qual não tem privilégio de escrita
  - Daemons confinados no /var/empty, sem uid privilegiada


Separação de Priv. e Confinamento

- Uso muito limitado de root e setuid
  - Adquire os recursos privilegiados e solta os privilégios
  - Solta privilégios antes de ter qualquer contato com a rede
- Se não precisa acessar arquivos, chroot pra um diretório vazio no qual não tem privilégio de escrita
  - Daemons confinados no /var/empty, sem uid privilegiada
- OpenBSD criou padrão comum para daemons
  - Processo pai (root) abre sockets e lê configs, forka filhos
    - Filho se confina em /var/empty, solta privilégios e atende o socket
    - Filho solta privilégios e atua no sistema de arquivos, se necessário
    - Processos trocam apenas mensagens fixas entre si


Reflexão Final

Nenhuma destas técnicas garante segurança por si só, mas todas juntas tornam as coisas muito difíceis para os atacantes.


Referências

- http://openbsd.org/papers (artigos e apresentações)
  - Puffy at Work
  - Protection Measures in OpenBSD
  - Advanced OpenBSD Hardening
- http://marc.info/ (listas openbsd-{cvs,misc,tech})
- http://freshbsd.org/project/openbsd (commits)
- http://openbsd.org/faq (FAQ:)