Process Infection

Carlos Eduardo O. A. Barros
barros@barrossecurity.com

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

O que é *Process Infection*?

"Injetar código na imagem de um processo em execução de forma a modificar o comportamento do mesmo, geralmente com o intuito de obter algum tipo de privilégio."

Introdução

Utilidades:

- Implantação de backdoor;
- Captura de senhas;
- Burlar firewall;
- Outros.

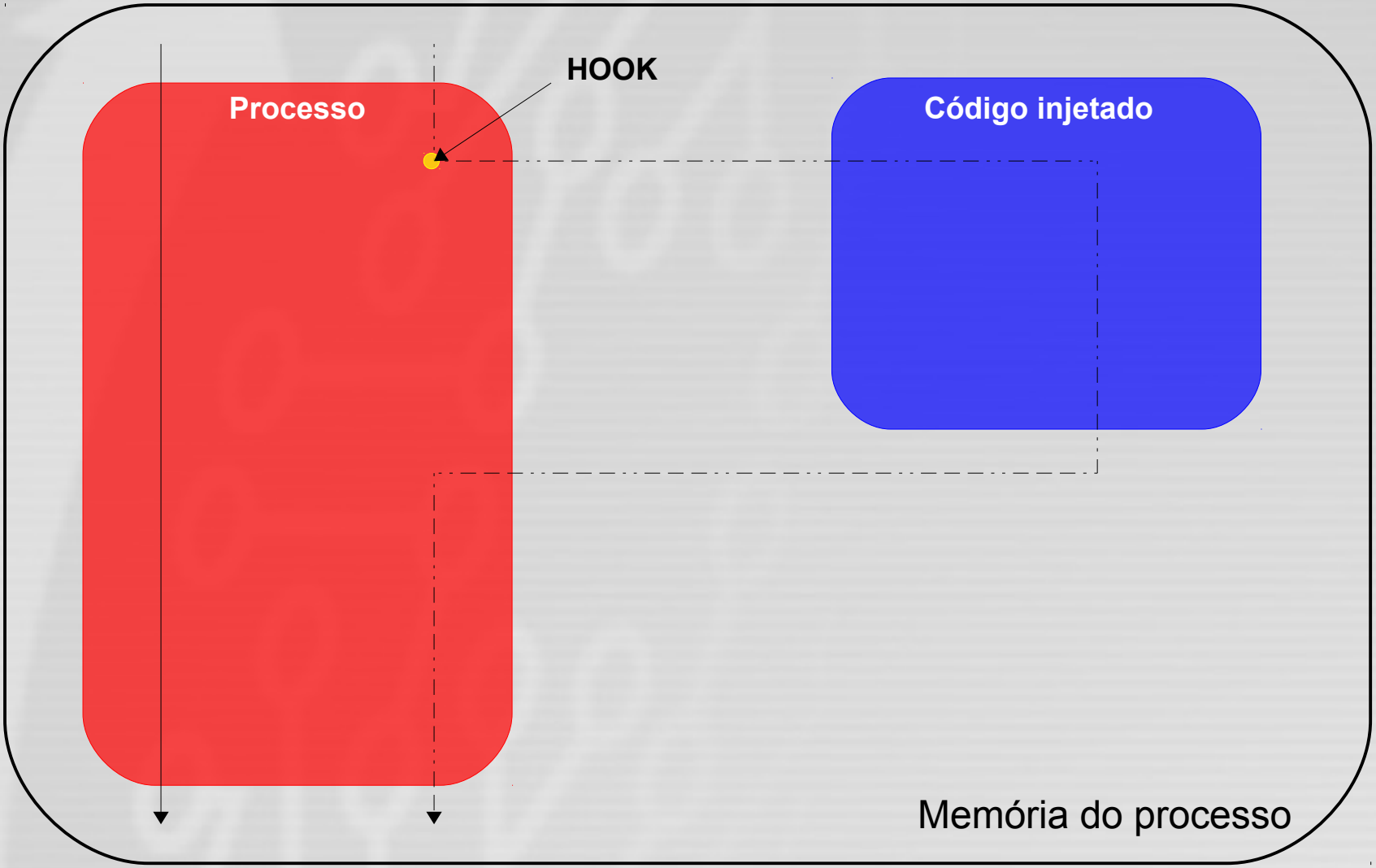
Exemplo:

- SSHeater.

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Princípio de Funcionamento



Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Resolução de Símbolos

```
int main()
{
   printf("Hello H2HC"\n");
   return 1;
}
```

hello.c

Compilando e executando:

```
$ gcc hello.c -o hello
$ ./hello
Hello H2HC
```

Resolução de Símbolos

```
int main()
{
   printf("Hello H2HC"\n");
   return 1;
}
```

hello

```
int printf(const char *fmt, ...)
{
   ...
   ...
}
```

libc

Resolução de Símbolos

Como o compilador sabe o endereço exato dos símbolos externos?

R: Ele não sabe

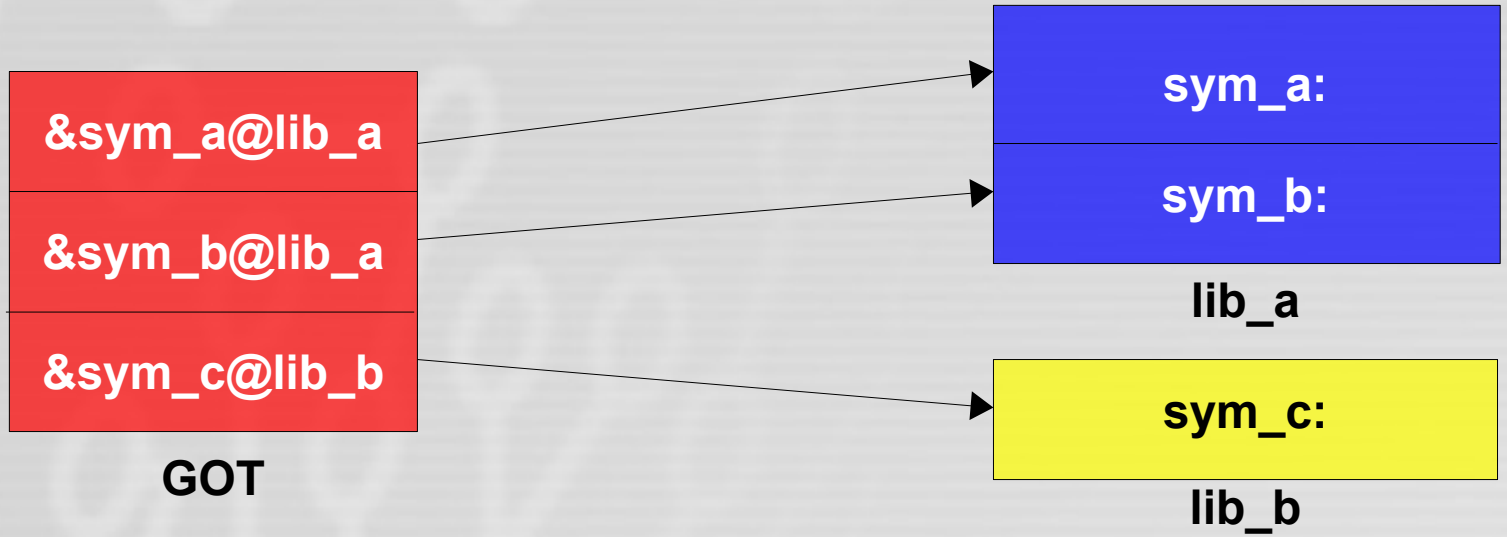
Duas estruturas são utilizadas:

- GOT – Global Offset Table
- PLT – Procedure Linkage Table

GOT – Global Offset Table

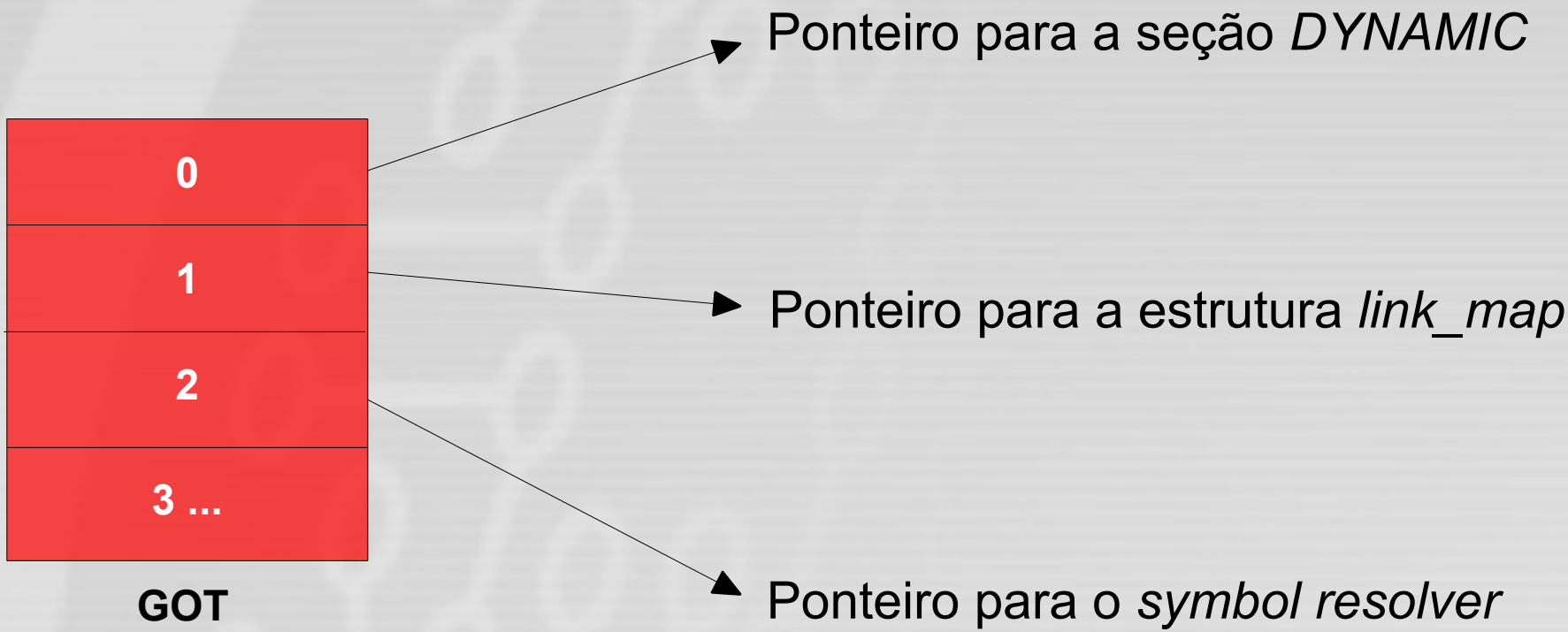
Definição:

GOT é uma tabela de endereços absolutos onde os símbolos externos são mapeados na memória do processo, garantindo a independência de posição do código.



GOT – Global Offset Table

As três primeiras posições da tabela GOT são reservadas:



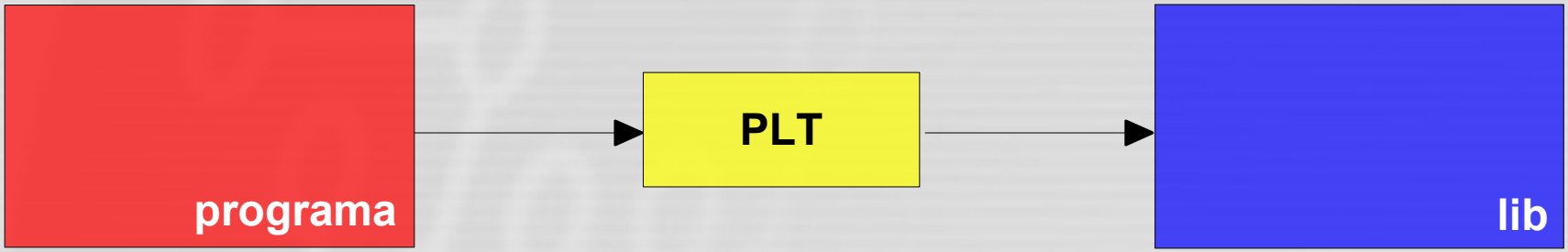
Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

PLT – Procedure Linkage Table

Definição:

PLT é a ponte de ligação entre o código do programa em execução e os símbolos externos, localizados nas bibliotecas (*shared libraries*).



PLT – Procedure Linkage Table

Disassembly:

```
...
<main+35>:    call    0x80482a0 <printf@plt>
...
<printf@plt>:    jmp     *0x804957c
<printf@plt+6>:  push    $0x0
<printf@plt+11>:jmp     0x8048290 <_init+24>
```

GOT Entry

Symbol Index

PLT0

PLT0:

Responsável por chamar o *"symbol resolver"*.

Disassembly:

```
0x8048290 <_init+24>:    pushl   0x8049574
0x8048296 <_init+30>:    jmp     *0x8049578
```

&link_map

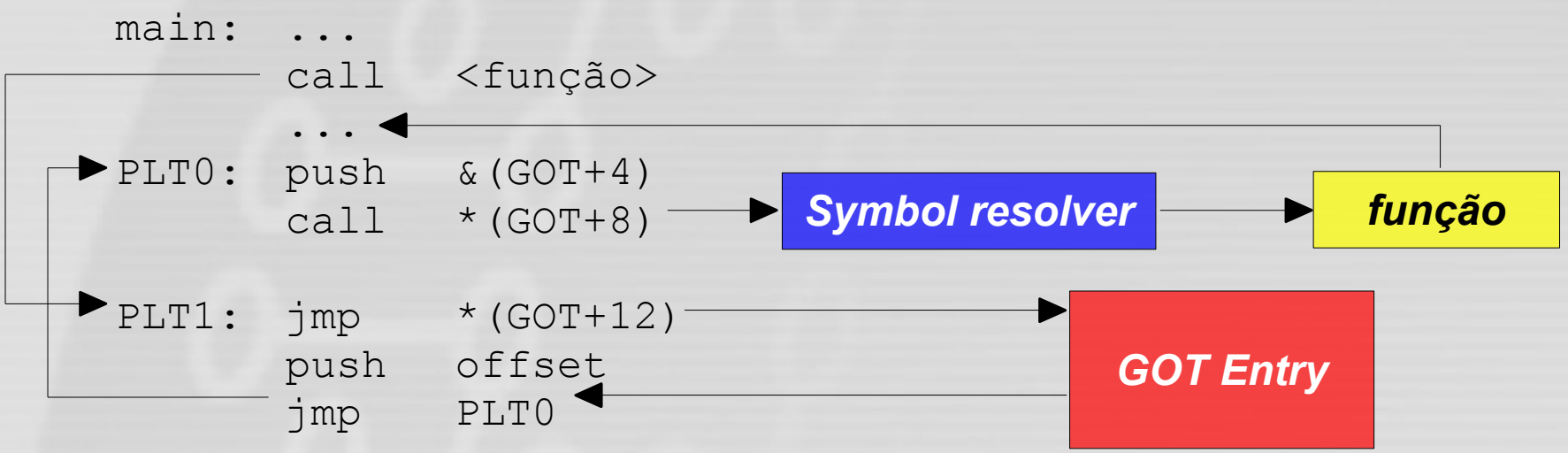
Symbol resolver

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

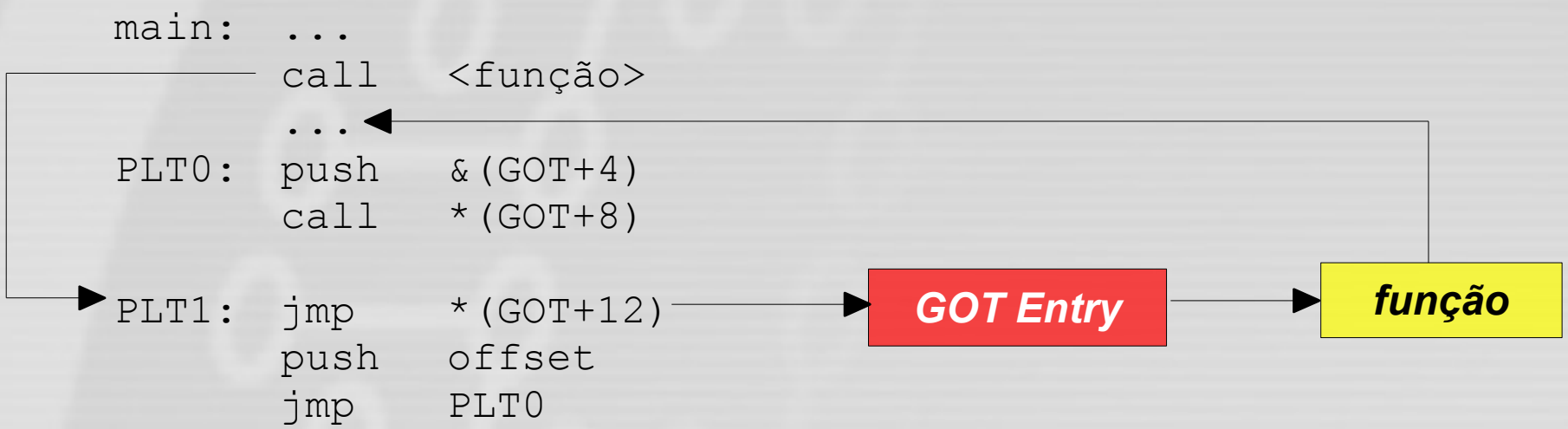
PLT + GOT

Primeira vez que um símbolo é referenciado:



PLT + GOT

Segunda vez que um símbolo é referenciado:



Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

ptrace()

Definição:

É uma chamada de systema (*syscall)* que permite a um processo observar e controlar a execução de um outro processo, além de examinar e modificar sua imagem e registradores.

Usamos a chamada de sistema ptrace() para obter o controle do processo alvo e injetar nosso código.

ptrace()

```
long  ptrace(enum __ptrace_request request, pid_t pid, void *addr,
             void *data);
```

Request:

- PTRACE_ATTACH: Assume o controle do processo apontador por *pid*;
- PTRACE_DETACH: Libera o processo apontado por *pid*;
- PTRACE_PEEKTEXT: Le uma *word* localizado em *addr* e retorna;
- PTRACE_POKETEXT: Escreve uma *word* em *addr*;
- PTRACE_GETREGS: Copia os registradores de uso geral para o endereço em *data*;
- PTRACE_SETREGS: Copia os registradores de uso geral de *data* para o processo;
- PTRACE_CONT: Reinicia um processo parado;

Pid: Indica o *pid* do processo onde será executada a operação;

Addr: Ponteiro genérico. Função depende da operação;

Data: Ponteiro genérico. Função depende da operação;

Executando código via ptrace

Após assumir o controle do processo com PTRACE_ATTACH:

- Ler o valor do registrador %eip com PTRACE_GETREGS;
- Calcular o tamanho do shellcode a ser executado;
- Copiar *size* bytes a partir do endereço apontado por %eip com PTRACE_PEEKTEXT;
- Escrever o shellcode no endereço apontado por %eip com PTRACE_POKETEXT;
- Continuar a execução do processo com PTRACE_CONT;
- Aguardar até que o processo pare;
- Restaurar a memória do programa com PTRACE_POKETEXT;
- Restaurar os registradores com PTRACE_SETREGS;

Observações:

Para utilizar essa técnica, o shellcode **deve** conter a instrução *int3* para suspender a execução do processo alvo.

O processo alvo pode estar **bloqueado**. Nesse caso devemos interagir com o processo de forma a liberá-lo.

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Passos da Infecção

Determinar as funções a serem subvertidas;

Escrever o *shellcodes* para cada função;

Injetar os *shellcodes* na memória do processo;

Localizar as entradas na seção GOT e/ou PLT das funções;

Modificar as entradas GOT e/ou PLT.

Determinar as Funções

A simplicidade desta etapa é inversamente proporcional a complexidade do software sendo infectado;

Pontos importantes:

- Determinar o que se deseja fazer;
- Entender o código alvo;
- Identificar as funções.

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Escrever os Shellcodes

Geralmente deve-se escrever um shellcode para cada função;

Nunca altere o conteúdo dos seguintes registradores:

- %ebp;
- %ebx;
- %edi;
- %esi;
- %esp;

Dois pontos importantes:

- Chamar a função original;
- Armazenar dados temporários.

Escrever os Shellcodes

Chamando a função original

Dois métodos distintos:

- Função subvertida modificando a PLT;
- Função subvertida modificando a GOT.

Escrever os Shellcodes

Função subvertida modificando a PLT:

```
<printf@plt>:   jmp     *0x804957c
<printf@plt+6>: push    $0x0
<printf@plt+11>:jmp     0x8048290 <_init+24>
```

PLT

Fazer uma cópia da instrução JMP e inserir no shellcode;

É necessário retomar o controle da aplicação?

```
        call   jmp_para_função
retoma_controle:
        ...
        ...
jmp_para_função:
        jmp    *0x804957c
```

Shellcode

É necessário ajustar os parâmetros na *stack.*

Escrever os Shellcodes

Função subvertida modificando a GOT:

Salvar o endereço da GOT e utilizar no shellcode;

```
mov    $ORIGINAL_ADDR,%eax
jmp    *%eax
```

Shellcode

É necessário retomar o controle da aplicação?

```
mov    $ORIGINAL_ADDR,%eax
call   *%eax
```

Shellcode

É necessário ajustar os parâmetros na *stack;*

Lazy Resolution gera problemas ao chamar a função original;

```
mov    $SHELLCODE_ADDR,%ecx
mov    %ecx,GOT_ENTRY
ret
```

Shellcode

Escrever os Shellcodes

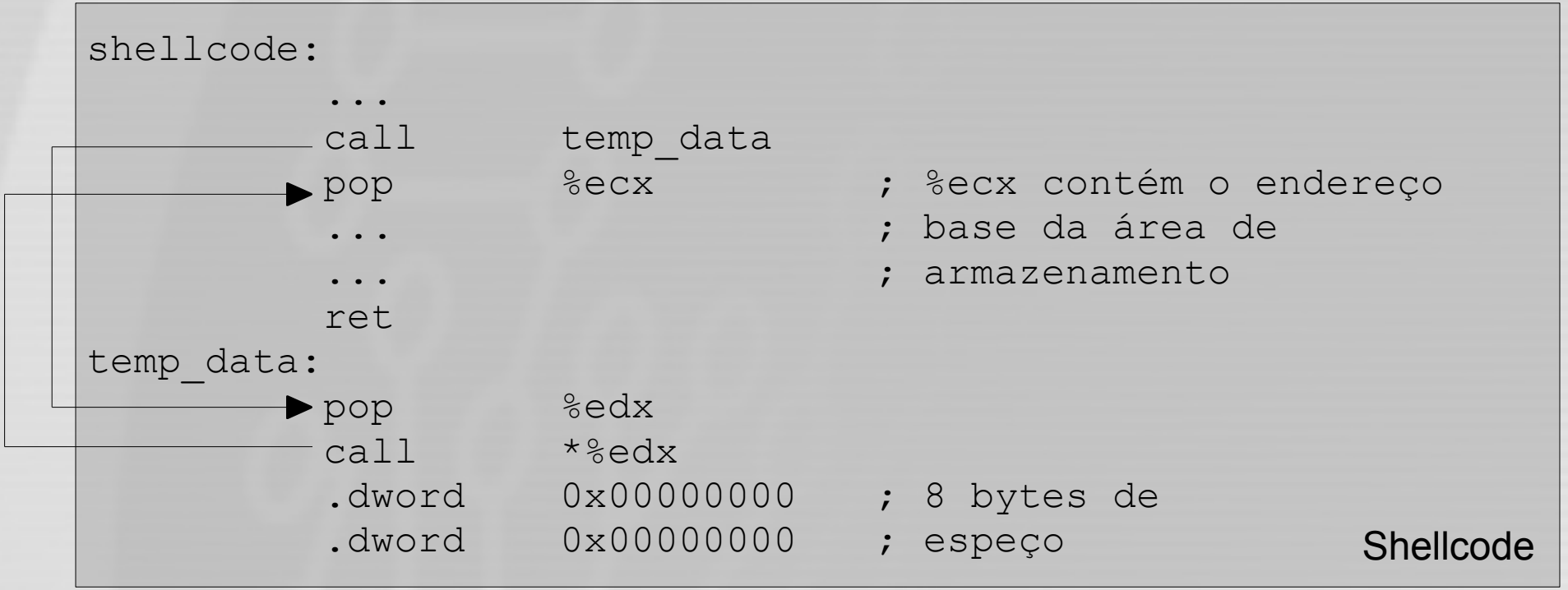
Armazenando dados temporários

Porque armazenar dados?

Dois métodos:

Forçar o processo a alocar memória e utilizar o endereço no *shellcode*;

Armazenar junto com o *shellcode*;



```
shellcode:
            ...
            call    temp_data
            pop     %ecx            ; %ecx contém o endereço
            ...                     ; base da área de
            ...                     ; armazenamento
            ret
temp_data:
            pop     %edx
            call    *%edx
            .dword  0x00000000      ; 8 bytes de
            .dword  0x00000000      ; espeço
```

Shellcode

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Injetar os Shellcodes

Onde armazenar o shellcode?

Na *stack*;
Nos *program headers*;
Em *anonymous mappings.*

A melhor escolha é a 3a. opção. Porque?

Pode-se marcar a área de memória como *executável;*
Sem restrições de tamanho.

Injetar os Shellcodes

```
mmap(NULL,size,PROT_READ|PROT_WRITE|PROT_EXEC,
                MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
```

```
<main+31>:   push   $0x0           // offset
<main+33>:   push   $0xffffffff    // fd
<main+35>:   push   $0x22          // flags
<main+37>:   push   $0x7           // prot
<main+39>:   push   $0x41414141    // size
<main+44>:   push   $0x0           // start
<main+46>:   call   0x804ed50 <mmap>
   ...
   ...
<mmap+0>:    mov    %ebx,%edx
<mmap+2>:    mov    $0x5a,%eax
<mmap+7>:    lea    0x4(%esp),%ebx
<mmap+11>:   int    $0x80
<mmap+13>:   mov    %edx,%ebx
<mmap+15>:   cmp    $0xfffff000,%eax
<mmap+20>:   ja     0x8050a90 <__syscall_error>
<mmap+26>:   ret
```

Parâmetros

Código do mmap()

Parâmetros

Chamada de sistema

Injetar os Shellcodes

```
mmap(NULL,size,PROT_READ|PROT_WRITE|PROT_EXEC,
                MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
```

Transformando em shellcode:

```
mmap:       mov %0x5a,%eax
            jmp mmap_params
mmap1:      pop %ebx
            int $0x80

mmap_params:  call   mmap1
            .dword 0x00000000    // start
            .dword 0x41414141    // length
            .dword 0x07000000    // prot
            .dword 0x22000000    // flags
            .dword 0xffffffff    // fd
            .dword 0x00000000    // offset
```

Hardcoded

Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Localizar as entradas GOT/PLT

É o passo mais importante e também o mais complexo de todo o processo de infecção.

Nessa etapa deve ser definido o método de subversão:

- Alterando a GOT;
- Alterando a PLT.

Existem duas técnicas distintas:

- Analisando o binário do processo:
  - /proc/<pid>/exe é um *symlink* para o binário;
  - Mais simples de ser realizada;
- Analisando a memória do processo:
  - Algumas estruturas do binário estão na memória do processo;
  - Um pouco mais complexo.

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar *sections* dentro do binário:

```
typedef struct
{
      unsigned char e_ident[EI_NIDENT];
      Elf32_Half    e_type;
      Elf32_Half    e_machine;
      Elf32_Word    e_version;
      Elf32_Addr    e_entry;
      Elf32_Off     e_phoff;
      Elf32_Off     e_shoff;
      Elf32_Word    e_flags;
      Elf32_Half    e_ehsize;
      Elf32_Half    e_phentsize;
      Elf32_Half    e_phnum;
      Elf32_Half    e_shentsize;
      Elf32_Half    e_shnum;
      Elf32_Half    e_shstrndx;
}Elf32_Ehdr;
```

→ Primeiro *program header*

→ Primeiro *section header*

→ Número de *sections*

→ Index da Headers *String Table section*

ELF Header

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar *sections* dentro do binário:

```
typedef struct
{
      Elf32_Word    sh_name;
      Elf32_Word    sh_type;
      Elf32_Word    sh_flags;
      Elf32_Addr    sh_addr;
      Elf32_Off     sh_offset;
      Elf32_Word    sh_size;
      Elf32_Word    sh_link;
      Elf32_Word    sh_info;
      Elf32_Word    sh_addralign;
      Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

→ Offset na headers *string table*

→ Offset da *section* no binário

→ Tamanho total da *section*

→ Tamanho de cada entrada

Section Header

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar *sections* dentro do binário:

```
[1] sh_header <- e_shoff;
[2] str_table <- sh_header[e_shstrndx].sh_offset
[3] for I = 0 to Elf32_Ehdr.e_shnum do
[4] begin
[5]    if str_table[sh_header[I].sh_name] == "desired_section"
[6]       then found_section <- &sh_header[I];
[7] end;
```

Algoritmo para localizar *sections*

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar a entrada na GOT de um símbolo:

A *section* .rel.plt é utilizada;

Armazena informações sobre os símbolos externos;

```
typedef struct
{
    Elf32_Addr    r_offset;   ──► Ponteiro para a entrada na GOT
    Elf32_Word    r_info;     ──► Informações sobre o símbolo
} Elf32_Rel;
```

.rel.plt *section*

r_info contem o índice do símbolo na *section* .dynsym;

Pode ser recuperado utilizando a macro ELF32_R_SYM:

```
#define ELF32_R_SYM(val)                ((val) >> 8)
```

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar a entrada na GOT de um símbolo:

```
typedef struct
{
    Elf32_Word    st_name;   ──► Offset na dynamic string table
    Elf32_Addr    st_value;
    Elf32_Word    st_size;
    unsigned char st_info;
    unsigned char st_other;
    Elf32_Section st_shndx;
} Elf32_Sym;
```

.dynsym section

Dynamic string table:

É uma tabela de strings, igual a *headres string table*, onde são armazenados as strings dos nomes de todos os símbolos do programa.

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar a entrada na GOT de um símbolo:

```
[1] rel_plt_header <- locate_section(".rel.plt");
[2] rel_plt <- rel_plt_header.sh_offset;
[3] symcount <- rel_plt_header.sh_size / rel_plt_header.sh_entsize;
[4] dynstr <- locate_section(".dynstr").sh_offset;
[5] dynsym <- locate_section(".dynsym").sh_offset;

[6] for I = 0 to symcount do
[7] begin
[8]    if dynstr[sym[ELF32_R_SYM(rel_plt[I].r_info)].st_name] == \
[9]       "desired_symbol" then
[10]      found_got <- rel_plt[I].r_offset;
[11]end;
```

Algoritmo para localiza uma entrada na GOT

Localizar as entradas GOT/PLT

Analisando o binário do processo

Como localizar a entrada na PLT de um símbolo:

- O processo é semelhante ao de localizar a entrada na GOT;
- Percorrer a *section* .rel.plt e localizar o índice referente ao símbolo;
- O índice do símbolo na *section* .rel.plt é exatamente o mesmo da *section* .plt:

```
address = .plt_address + (symindex + 1) * 16;
```

```
[1] rel_plt_header <- locate_section(".rel.plt");
[2] rel_plt <- rel_plt_header.sh_offset;
[3] symcount <- rel_plt_header.sh_size / rel_plt_header.sh_entsize;
[4] dynstr <- locate_section(".dynstr").sh_offset;
[5] dynsym <- locate_section(".dynsym").sh_offset;

[6] for I = 0 to symcount do
[7] begin
[8]    if dynstr[sym[ELF32_R_SYM(rel_plt[I].r_info)].st_name] == \
[9]       "desired_symbol" then
[10]      break;
[11]   end;
[12]   found_plt <-  locate_section(".plt").sh_addr + (I + 1) * 16;
```

Localizar as entradas GOT/PLT

Analisando a memória do processo

Como localizar *sections* dentro do processo:

- Os *section headers* não são mapeados na memória do processo;
- Os *program headers* são utilizados (Elf32_Ehdr->e_phoff);
- *ELF Header* é mapeado no endereço 0x08048000 (Linux).

```
typedef struct
{
  Elf32_Word    p_type;   ──► Tipo de segmento
  Elf32_Off     p_offset;
  Elf32_Addr    p_vaddr;  ──► Endereço virtual
  Elf32_Addr    p_paddr;
  Elf32_Word    p_filesz;
  Elf32_Word    p_memsz;
  Elf32_Word    p_flags;
  Elf32_Word    p_align;
} Elf32_Phdr;
```

program header

p_type = PT_DYNAMIC: p_vaddr aponta para a *section* .dynamic

Localizar as entradas GOT/PLT

Analisando a memória do processo

Como localizar *sections* dentro do processo:

```
typedef struct
{
      Elf32_Sword    d_tag;
      union
      {
            Elf32_Word d_val;
            Elf32_Addr d_ptr;
      } d_un;
} Elf32_Dyn;
```

Tipo da entrada

.dynamic *section*

Valores interessantes de d_tag:

- DT_PLTGOT - d_un.d_ptr aponta para a *section* .got.plt;
- DT_JMPREL - d_un.d_ptr aponta para a *section* .rel.plt;
- DT_PLTRELSZ - d_un.d_val armazena o tamanho em bytes da *section* .rel.plt;
- DT_INIT - d_un.d_ptr aponta para a *section* .init;
- DT_SYMTAB - d_un.d_ptr aponta para a *section* .dynsym;
- DT_STRTAB - d_un.d_ptr aponta para a *section* .dynstr;

Localizar as entradas GOT/PLT

Analisando a memória do processo

Como localizar a entrada na GOT de um símbolo:

O processo é o mesmo utilizado para localizar analisando o binário do processo.

```
[1] rel_plt_header <- locate_section(".rel.plt");
[2] rel_plt <- rel_plt_header.sh_offset;
[3] symcount <- rel_plt_header.sh_size / rel_plt_header.sh_entsize;
[4] dynstr <- locate_section(".dynstr").sh_offset;
[5] dynsym <- locate_section(".dynsym").sh_offset;

[6] for I = 0 to symcount do
[7] begin
[8]    if dynstr[sym[ELF32_R_SYM(rel_plt[I].r_info)].st_name] == \
[9]       "desired_symbol" then
[10]      found_got <- rel_plt[I].r_offset;
[11]end;
```

Algoritmo para localiza uma entrada na GOT

Localizar as entradas GOT/PLT

Analisando a memória do processo

Como localizar a entrada na PLT de um símbolo:

Esse processo é mais complicado pois não temos o endereço da *section* .plt;

Disassembly da entrada GOT:

```
80482b0:        ff 25 c0 95 04 08        jmp    *0x80495c0
```

Algoritmo:

```
[1]    got_entry <- locate_got("our_symbol");
[2]    search_addr <- init_addr;
[3]    while true
[4]    begin
[5]        pattern <- (int)*search_addr;
[6]        if pattern == got_entry then
[7]            break;
[8]        pattern = pattern + 1;
[9]    end;
[10]   found_plt <- search_addr - 2;
```

Roteiro

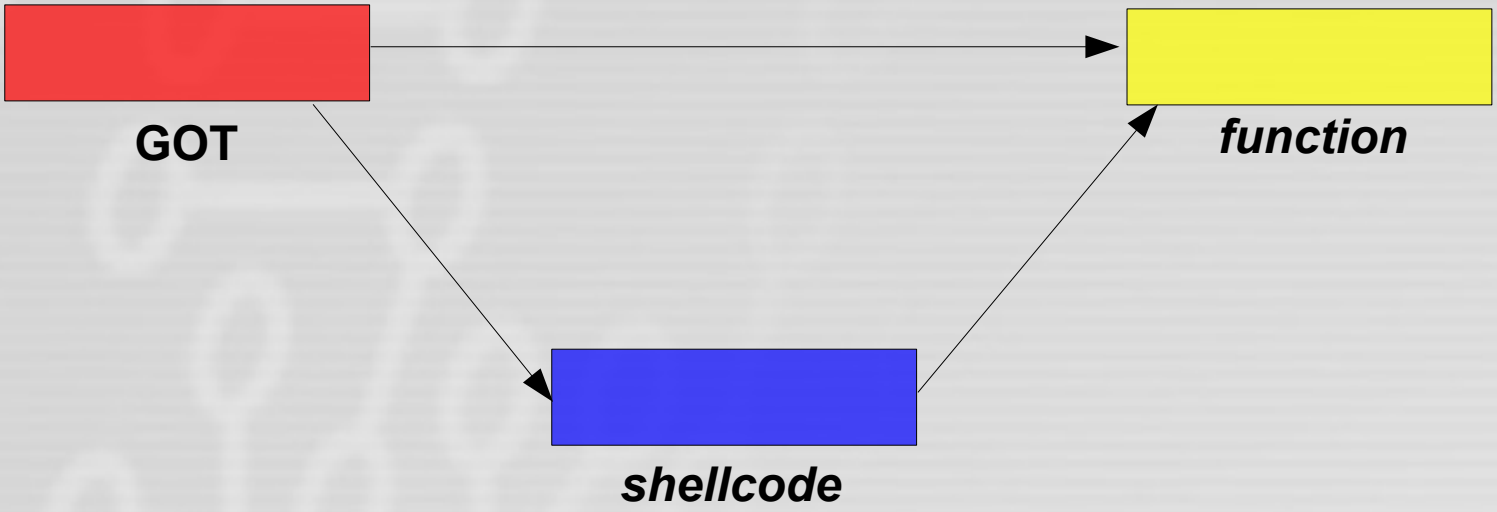
Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Alterar as entradas GOT/PLT

Alterar a entrada na GOT:

Salvar o valor original da entrada;

Alterar o ponteiro original por um ponteiro para o *shellcode*, injetado anteriormente;



Alterar as entradas GOT/PLT

Alterar a entrada na PLT:

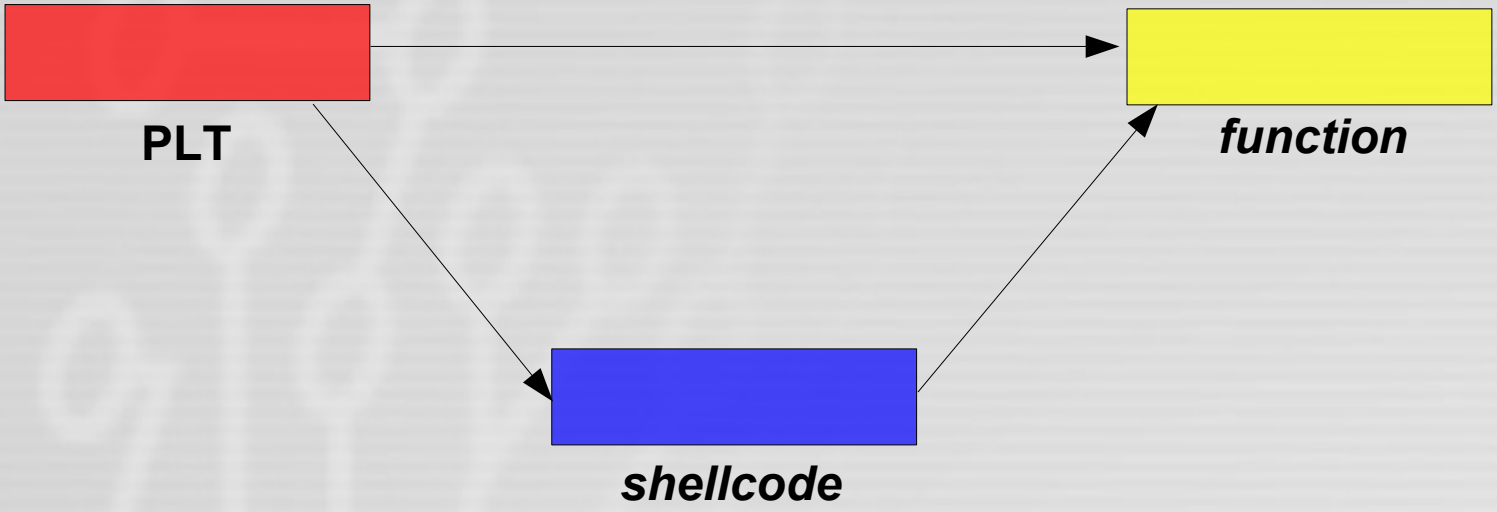
Substituir a instrução jmp *GOT para desviar para o *shellcode*

Antes de substituir, salvar a instrução original para usa-la no *shellcode*

```
80482b0:        ff 25 c0 95 04 08        jmp    *0x80495c0
```

Até seis bytes podem ser substituídos:

```
0:   68 41 41 41 41        push   $0x41414141
5:   c3                    ret
```



Roteiro

Introdução;
Princípio de funcionamento;
Resolução de símbolos;
  GOT – Global Offset Table;
  PLT – Procedure Linkage Table;
  PLT + GOT;
ptrace();
  Executando código via ptrace;
Passos da infecçao;
  Determinar as funções;
  Escrever os shellcodes;
  Injetar os shellcodes;
  Localizar as entradas GOT/PLT;
  Alterar as enrtadas GOT/PLT;
Referências;
Perguntas?

Referências

Complete Guide to Process Infection
Carlos Barros - http://www.barrossecurity.com/download/24/

Shared Library Call Redirection Using ELF PLT Infection
Silvio Cesare - http://www.phiral.net/lib-redirection.txt

Runtime Process Infection
Anonymous - http://www.phrack.org/phrack/59/p59-0x08.txt

Runtime Process Infection 2
Ares - http://ares.x25zine.org/ES/txt/0x4553-\
Runtime_Process_Infecting.htm

Intel386 Architecture Processor Supplement Fourth Edition
http://www.caldera.com/developers/devspecs/abi386-4.pdf

Elf specification
http://x86.ddj.com/ftp/manuals/tools/elf.pdf

Libinfector v1.0
Carlos Barros - http://www.barrossecurity.com/download/25/

Perguntas?

??????