sold: A linker for shared objects

https://github.com/akawashiro/sold

Akira Kawata
bio: https://akawashiro.github.io/
twitter: @a_kawashiro
2021/11/20 Kernel/VM探検隊online part4

どうやってエディタをインストールしていますか?

- OSにプリインストールされたものを使う
- apt/yum/pacmanで入れる
- 自分でビルドする
  - 最新バージョンでないと動かないプラグインがあった

最新のneovimを自分でビルドして使いたい!

- 新しくビルド環境と整えるのは面倒
  - 特にrootがない場合
- 一度ビルドしたものを他のマシンにコピーして使いまわしたい
- 共有ライブラリ(shared object, *.soのこと)への依存がありできない
  - soも一緒にコピーするのは社内的な事情で不可

```
[new machine] > scp build-server:~/nvim .
[new machine] > ./nvim
./nvim: error while loading shared libraries: libnsl.so.3
```

Preferred Networks

静的リンクすれば良いのでは?

- 静的リンクすると共有ライブラリへの依存は消える
- しかし静的リンクはサポートされていないことがある
  - neovimを静的リンクできるようにするPR はマージされていない

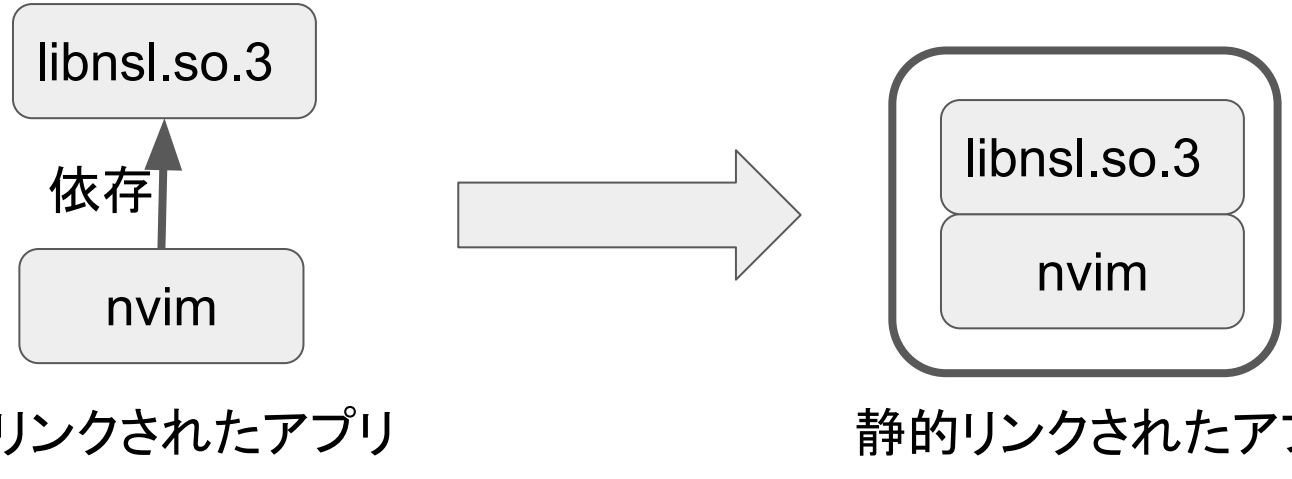
動的リンクされたバイナリに共有ライブラリをリンクすれば良い!

Preferred Networks

sold

- 依存している動的リンクライブラリを静的リンクする
- hamaji-sanが2020年の1月に作り始めた

```
sold -i nvim -o nvim.soldout
```



soldの動作例

```
> ldd a.out
        libfuga.so
        libhoge.so
> sold -i a.out -o a.out.soldout
> ldd a.out.soldout
>
```

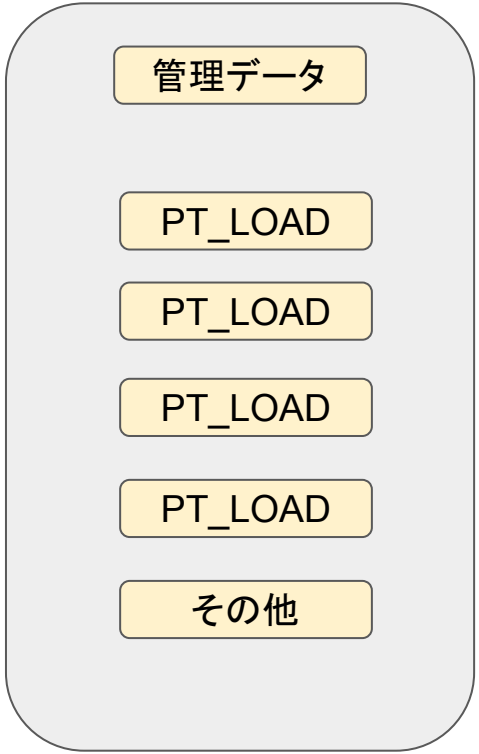
※ lddは依存する動的リンクライブラリを列挙するコマンド

Preferred Networks

soldの仕組み

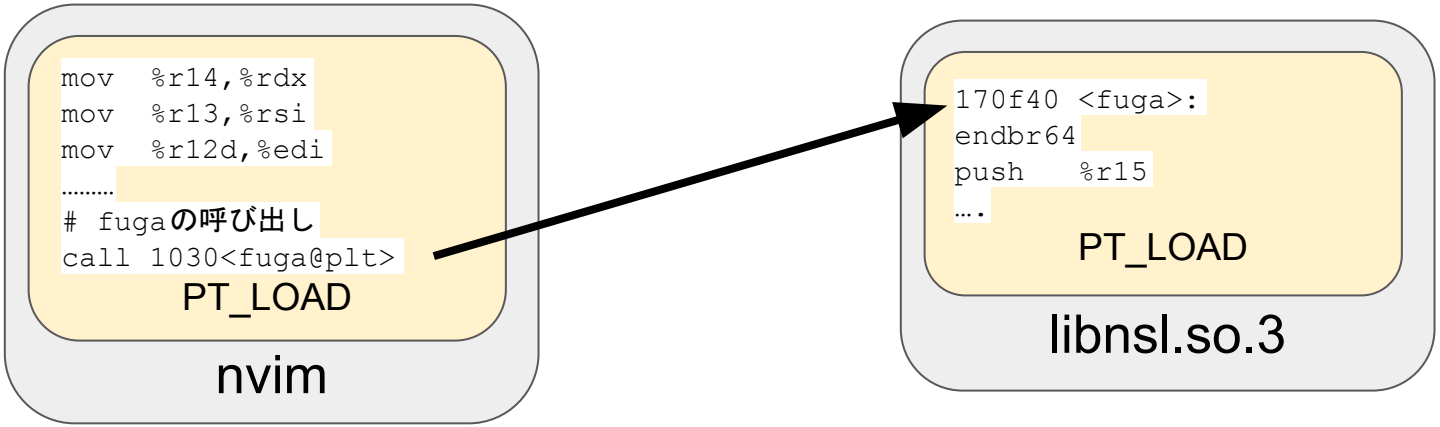
ELF: Executable and Linkable Format

- ELFはLinuxのバイナリフォーマット
  - 実行可能ファイル
  - 動的リンクライブラリ
- いくつかのセグメントからなる
  - メタデータが入っているセグメント
  - PT_LOADセグメント
  - その他
- 命令列はPT_LOADセグメントに入っている



動的リンクライブラリの呼び出し

- シンボルと再配置情報を使って動的リンクライブラリの関数を呼び出す

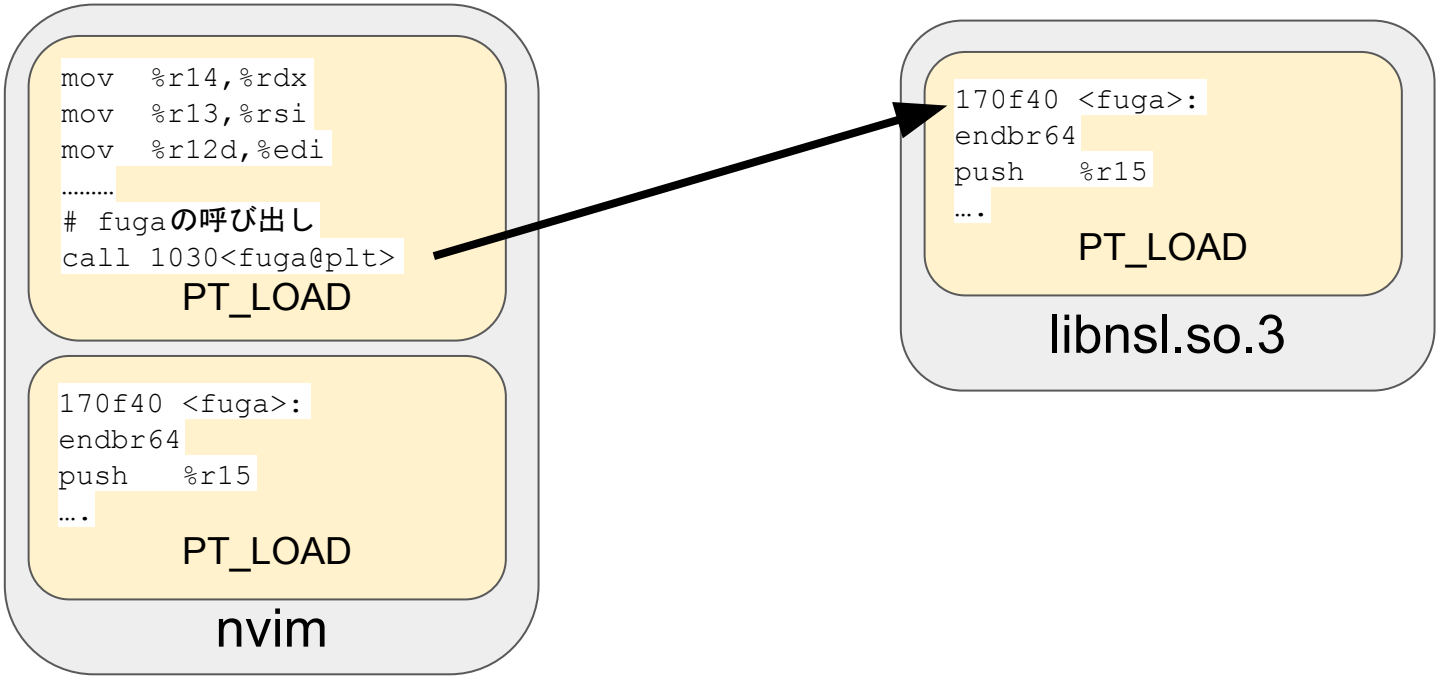


※: シンボルと再配置情報に関する説明は省略

Preferred Networks

soldの基本メカニズム①

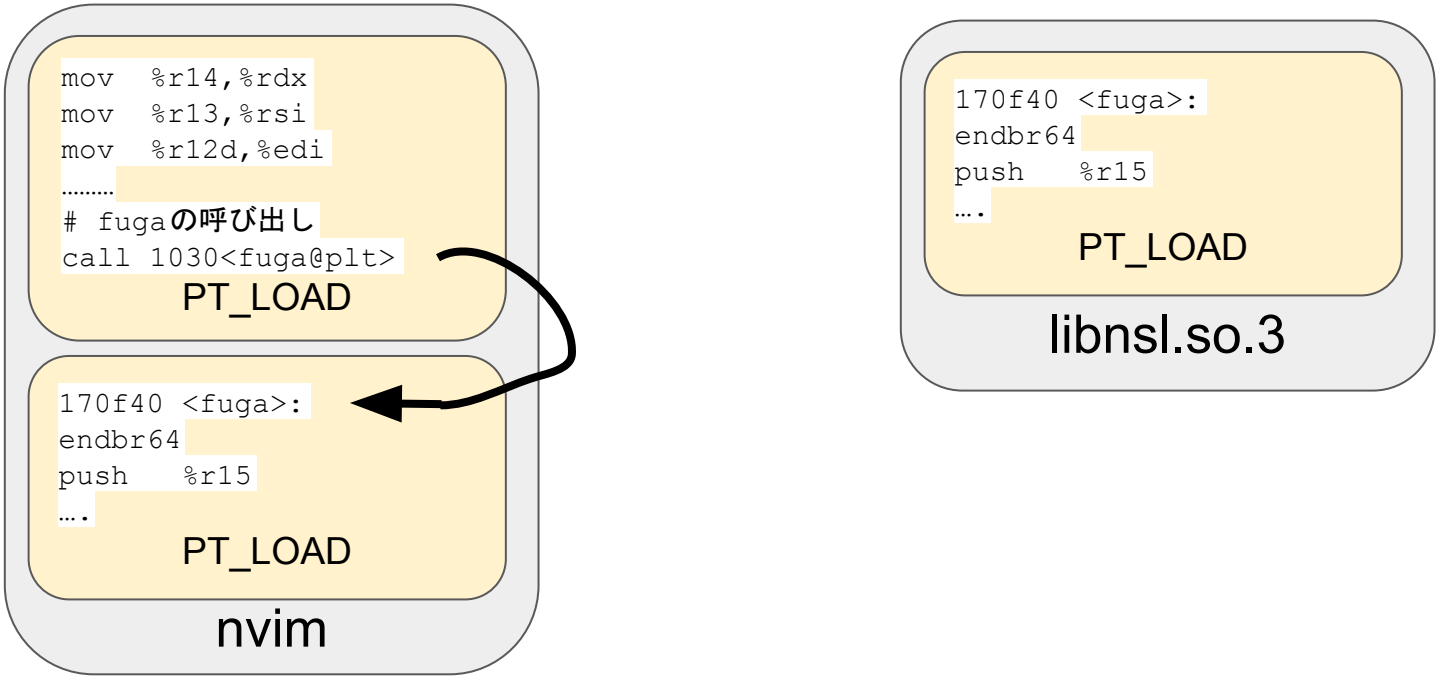
- 必要なPT_LOADをコピーする



Preferred Networks

soldの基本メカニズム②

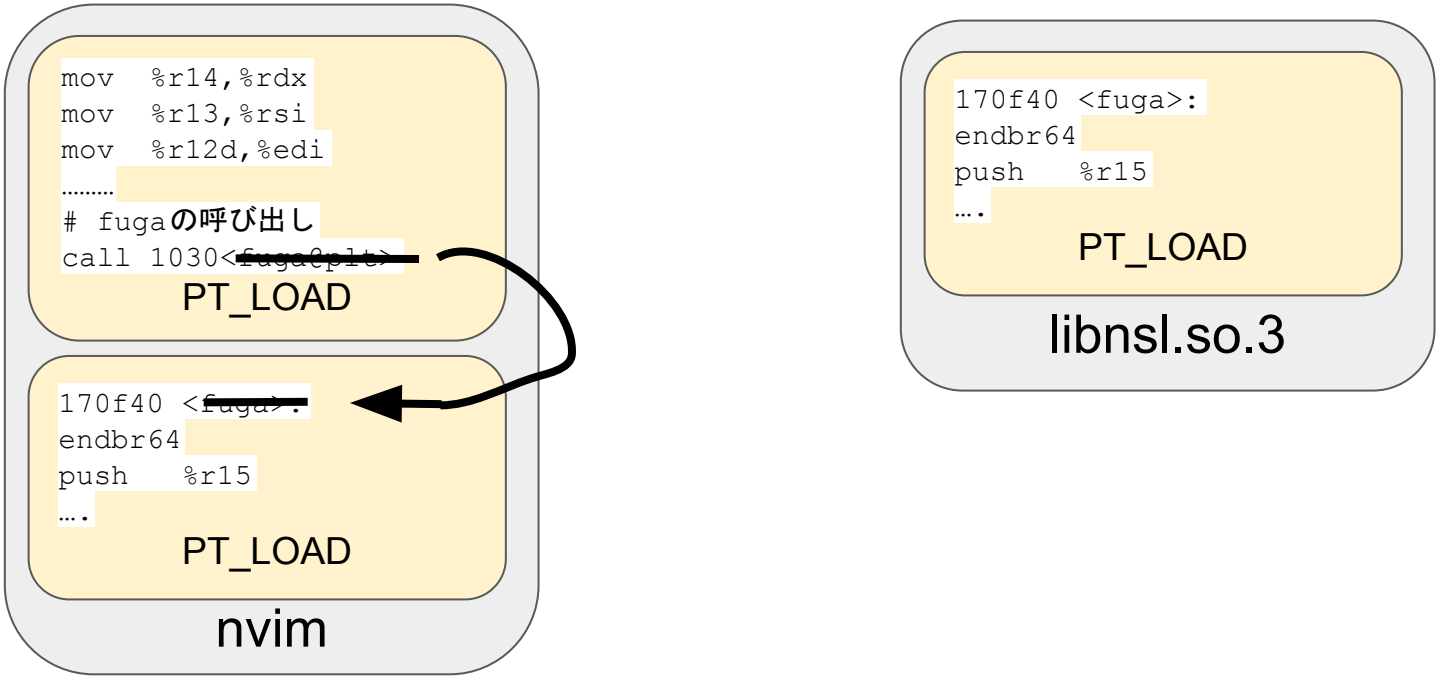
- 再配置情報を書き換える



Preferred Networks

soldの基本メカニズム③

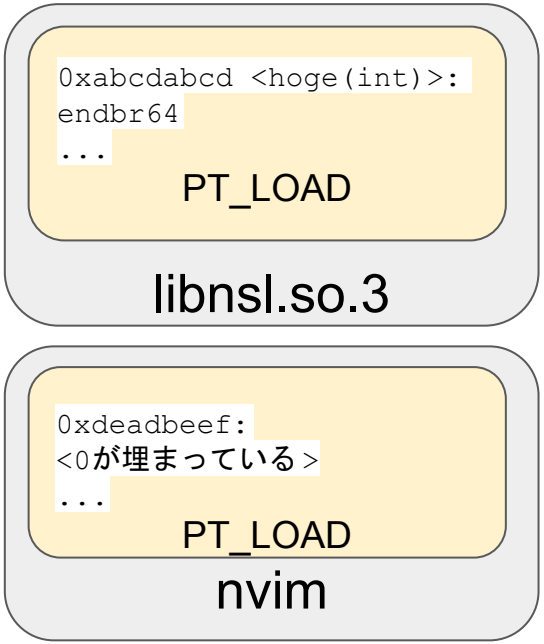
- シンボル情報を消す



Preferred Networks

再配置情報の書き換えをより詳しく(R_X86_64_64の場合)

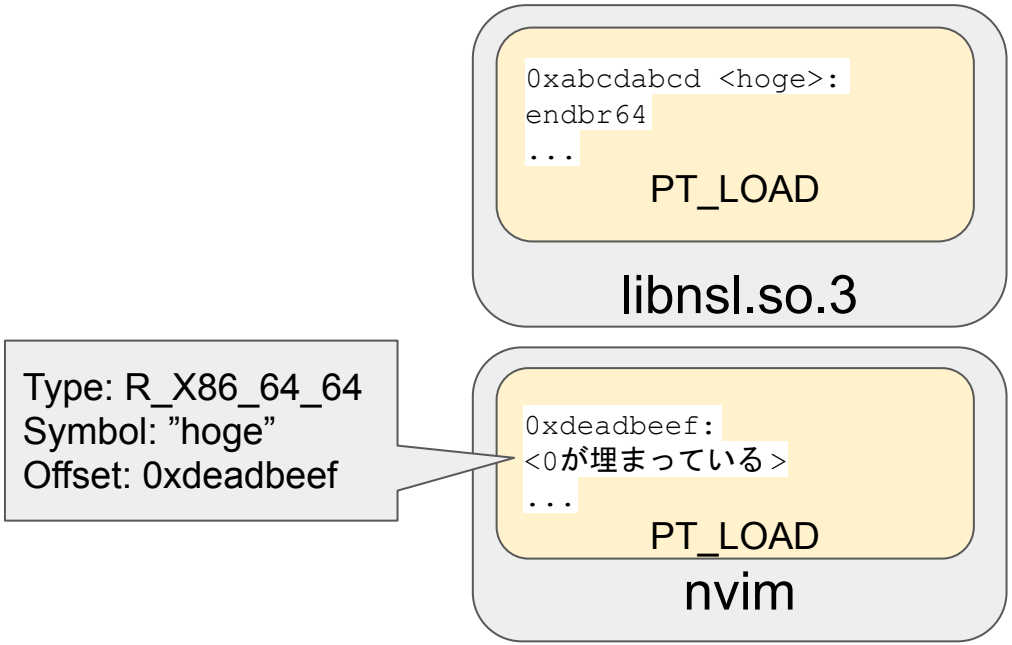
- R_X86_64_64とは?
  - 指定したアドレス(0xdeadbeef)にシンボル(hoge)のアドレスを埋める



Preferred Networks

再配置情報の書き換えをより詳しく(R_X86_64_64の場合)

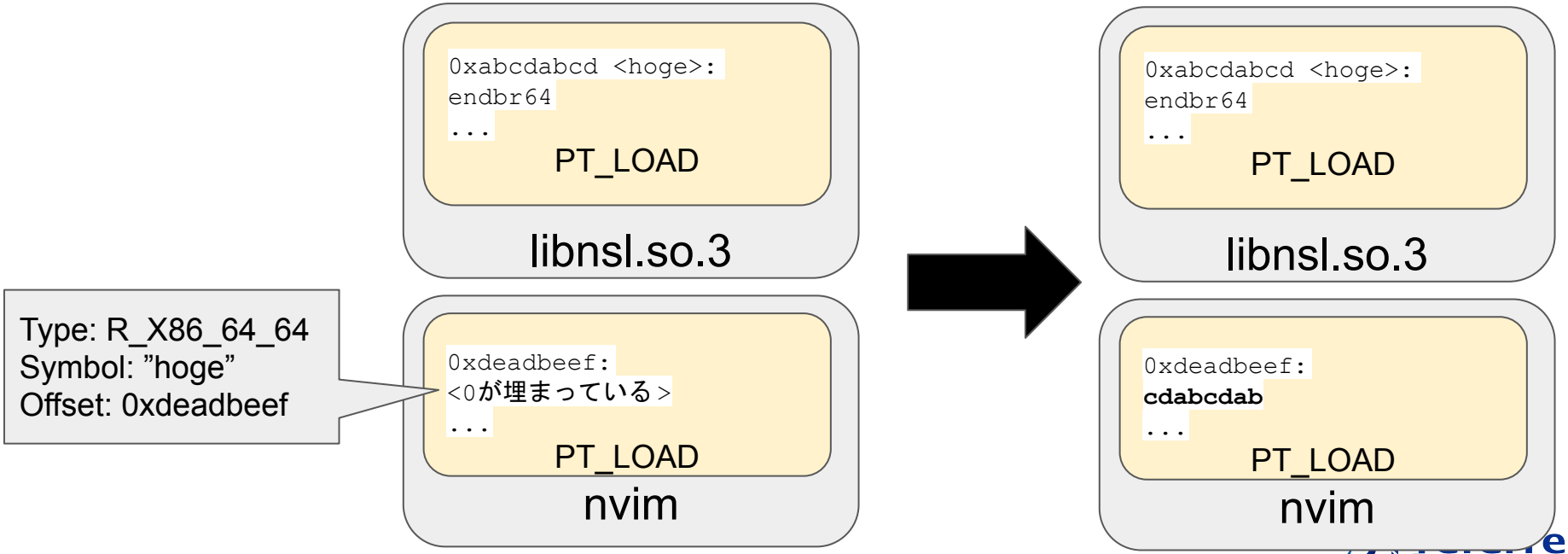
- R_X86_64_64とは?
  - 指定したアドレス(0xdeadbeef)にシンボル(hoge)のアドレスを埋める



Preferred Networks

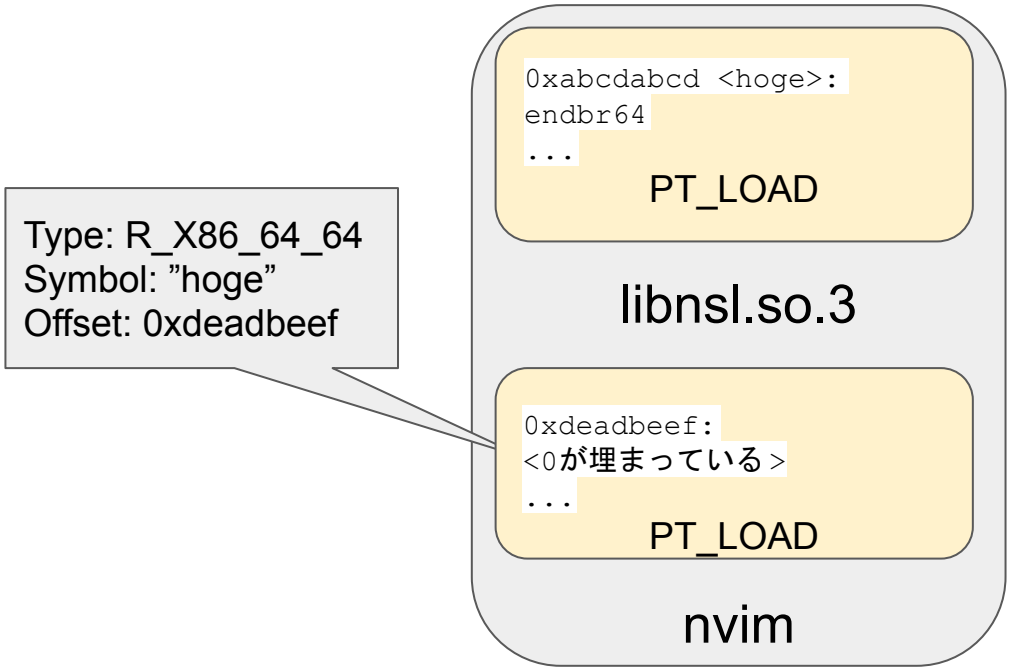
再配置情報の書き換えをより詳しく(R_X86_64_64の場合)

- R_X86_64_64とは?
  - 指定したアドレス(0xdeadbeef)にシンボル(hoge)のアドレスを埋める



再配置情報の書き換えをより詳しく(R_X86_64_64の場合)

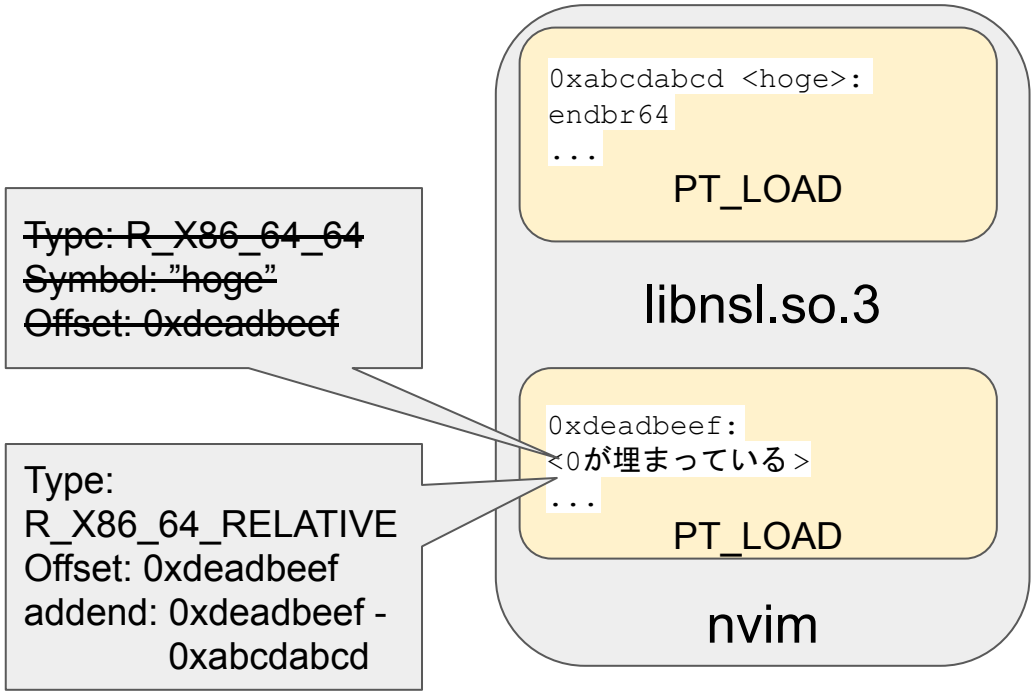
- soldによるR_X86_64_64の処理
  - シンボルが定義されていたらR_X86_64_RELATIVEに書き換える



Preferred Networks

再配置情報の書き換えをより詳しく(R_X86_64_64の場合)

- soldによるR_X86_64_64の処理
  - シンボルが定義されていたらR_X86_64_RELATIVEに書き換える

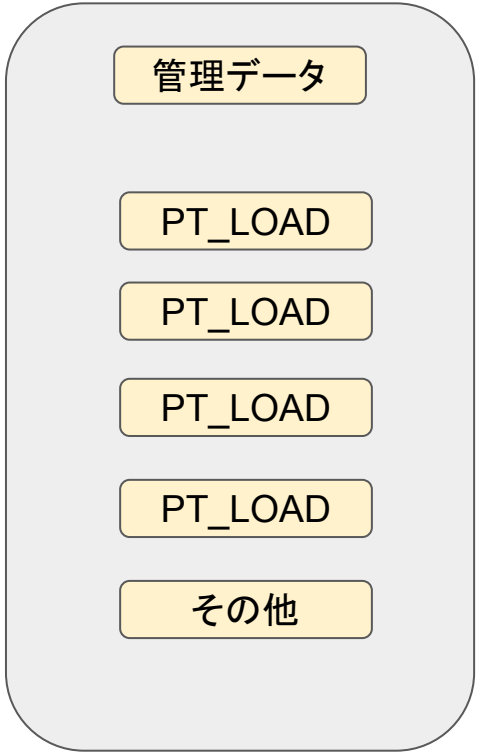


補足: R_X86_64_64
のままだとシンボルが
未解決でロードに
失敗します

Preferred Networks

実はその他も結構複雑

- PT_GNU_RELRO
  - 特定のメモリ空間の保護
  - mprotect_builder.h
- PT_TLS
  - thread local 変数
- PT_GNU_EH_FRAME
  - 例外
  - ehframe_builder.h



Preferred Networks

PT_GNU_RELRO

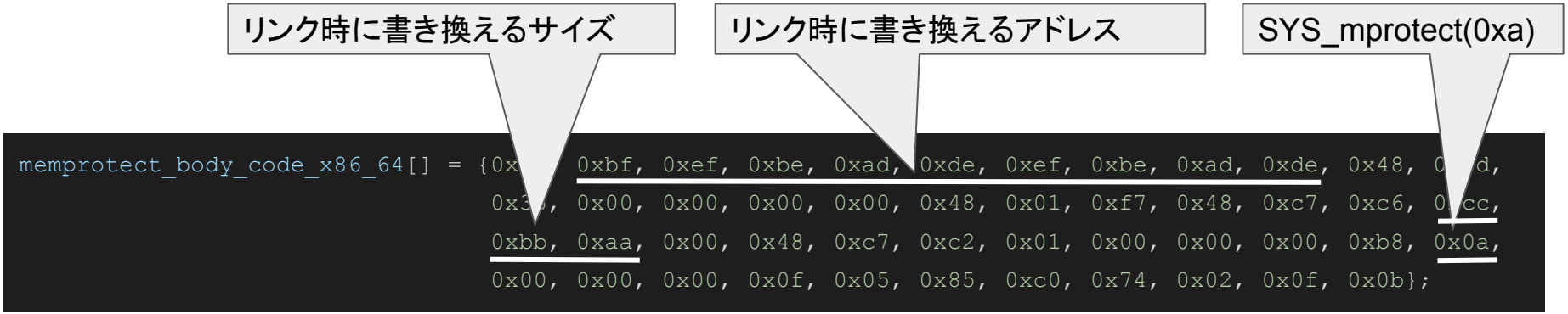
- 再配置後に特定のメモリ空間を書き込み不可にするセグメント
  - addrとsizeを保持している
  - 動作はmprotect(addr, size, PROT_EXEC) 相当
  - GOT(Global Offset Table)に対して使われる [1]
- 1つのELFで有効なのは1つのPT_GNU_RELRO
- しかし、soldでリンクしたバイナリには複数のGOTがある
- 1つのPT_GNU_RELROでは全てのGOTを保護できない
  - GOTの間に書き込み可のメモリ範囲がある可能性があるため

[1]: https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read-only-relro

Preferred Networks

Link-time コード生成

- 該当するメモリ範囲を1つ1つmprotectするバイナリをリンク時に生成
  - syscall命令を直で呼ぶバイナリ
- init_arrayにいれておく
  - Init_arrayとはshared objectのロード時に呼ばれる関数ポインタ群
  - C++のコンストラクタとかが入っている



Preferred Networks

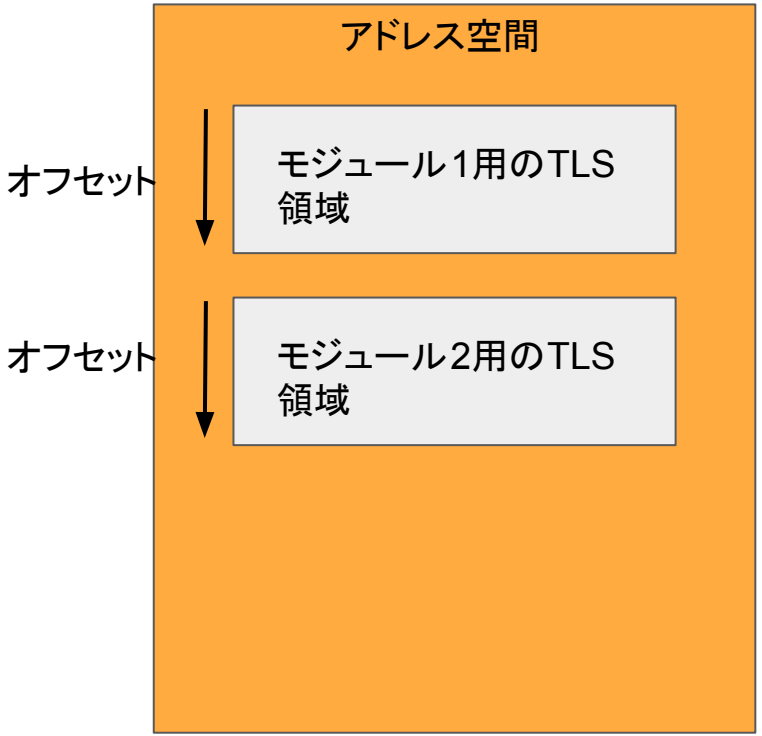
Thread Local Storage

```
thread_local int tls_a;
__thread int tls_b;
```

- スレッド間ではアドレス空間は共有
- TLSはスレッド毎の固有変数
- ローダによってスレッドごとにメモリ割付される

Thread Local Storageへのアクセス

```
void* __tls_get_addr
(size_t m, size_t offset){
    ....
    return address;
}
```



- TLSへのアクセスはモジュール番号とオフセットを使って行われる
- soldで処理する際は両方書き換える

Preferred Networks

soldでTLSを処理する際の問題点

- モジュール番号とオフセットの両方を書き換える必要がある
  - 一つのTLSについて2つの再配置情報が必要
    - R_X86_64_DTPMOD64 (モジュール番号)
    - R_X86_64_DTPOFF64 (オフセット)
- モジュール番号だけに再配置情報が生えているケースがある
  - TLS local dynamic modelと呼ばれる
- 対応できないとTLS変数への不正なアクセスが発生しSEGV

解決策: オフセットの位置を推測する

```
{
 uint64_t ti_module; /* <--- モジュール番号 */
 uint64_t ti_offset; /* <--- オフセット */
} tls_index;
```

- オフセットの位置はモジュール番号の8バイト先で固定
- R_X86_64_DTPMOD64だけでTLSを処理できる

デモ

まとめ

- https://github.com/akawashiro/sold
- soldは依存する動的リンクライブラリをリンクするリンカ
- ls / find / tree / grep などリンク可能・動作を確認
- neovimもリンク可能
  - LD_DEBUG=unused付きで動作を確認
- libc.so, libm.so, libpthread.so がリンクできない
  - 重要で歴史あるライブラリほど複雑

スライド終わり

FAQ

FAQ

Q: 依存しているライブラリも一緒にコピーすれば良いのでは?

A: 社内的な事情で要件を満たさなかった

Q: 動的リンクライブラリを静的リンクする困難さとは?

A: そもそも動的リンクライブラリは静的リンクするためのものではない

- Static link of shared library function in gcc
- 静的リンクするするための情報(セクション)がないケースがある

FAQ

Q: LD_DEBUG=unusedとは?

A: “Determines unused DSOs.” from man だが...

- elf: Do not run IFUNC resolvers for LD_DEBUG=unused @ 4db71d
  あたりが関係しているのではないかと思っている

Preferred Networks

FAQ終わり