FreeBSD-SA-09:05.telnetd と LD_PRELOAD 環境変数について

ＮＴＴ コミュニケーションズ株式会社
ＩＴ マネジメントサービス事業部
セキュリティオペレーションセンタ

2009 年 05 月 26 日

Ver. 2.1

1. 調査概要

2009 年 02 月 14 日、FreeBSD の Telnet デーモンに対して、0Day の攻撃方法が公開された(「5 参考の 1」)。
LD_PRELOAD 環境変数を使って、不正なライブラリ(任意のコード)を事前にロードさせることで、Telnet デーモンの権限で任意のコードを実行させてしまうという問題である。

本文書では、この 0Day の再現、さらに LD_PRELOAD 環境変数の仕組み、LD_PRELOAD 環境変数を使った攻撃(LD_PRELOAD 攻撃)の対策としてのセキュア・プログラミングについて考察した結果を記す。

2. FreeBSD-SA-09:05.telnetd の再現

2009 年 02 月 14 日、FreeBSD の Telnet デーモンに対して、LD_PRELOAD 環境変数を使うことで、リモートから認証なしに root 権限を取得することができる脆弱性が公開された(「5 参考の 1」)。
この脆弱性に関する修正パッチも 2009 年 02 月 16 日に公開された(「5 参考の 2」)。

本章では、この問題の検証結果について記す。

2.1. FreeBSD7.1 の場合

FreeBSD7.1 に対して、検証した結果を記す。
FreeBSD7.1 をインストール後、Telnet デーモンを起動した(図 2.1-1)。
また、事前に攻撃用のライブラリをインストールし、コンパイルし、/tmp 上の配置した(図 2.1-2)。
この検証用コードは、_init()関数の処理を、シェルを起動する内容に置き換えるライブラリであることが分かる(図 2.1-2)。
その後、別の FreeBSD を使って、対象に対して Telnet コマンドを使って、リモートからの root 権限取得に成功した(図 2.1-3)。

このように、今回公開された脆弱性は、以下の状況の時、リモートから root 権限を取得される危険性がある。

- 対象にファイルを配置可能である。
- 対象は、Telnet デーモンを起動し、Telnet 接続が可能である。

対策は、パッチの適用である。「5 参考の 2」を参考にしてほしい。

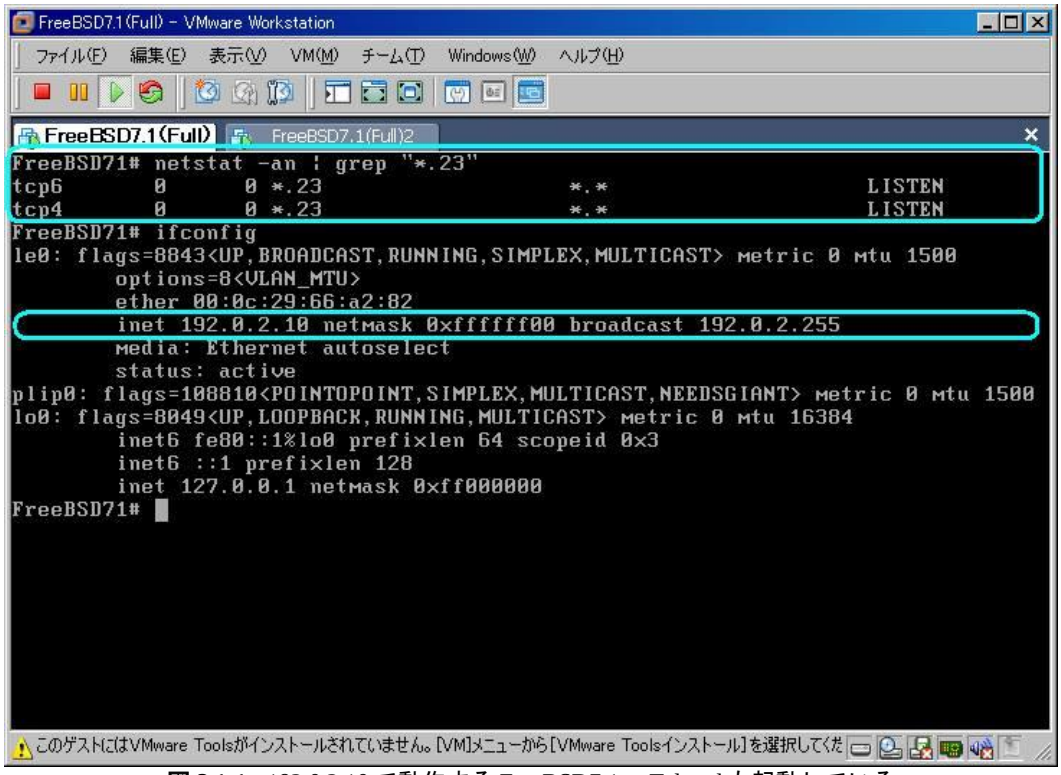

図 2.1-1 : 192.0.2.10 で動作する FreeBSD7.1 。Telnetd も起動している

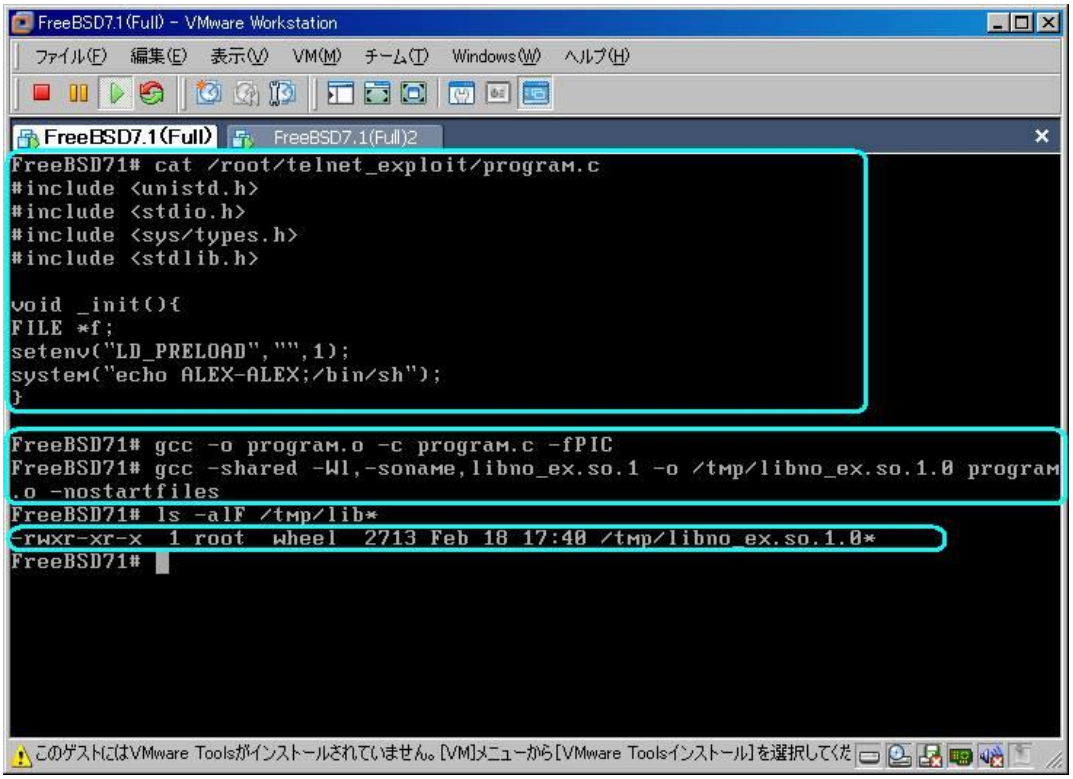

図 2.1-2 : 図 2.1-1 には、既に LD_PRELOAD 環境変数で
ロードさせるライブラリもインストール済である

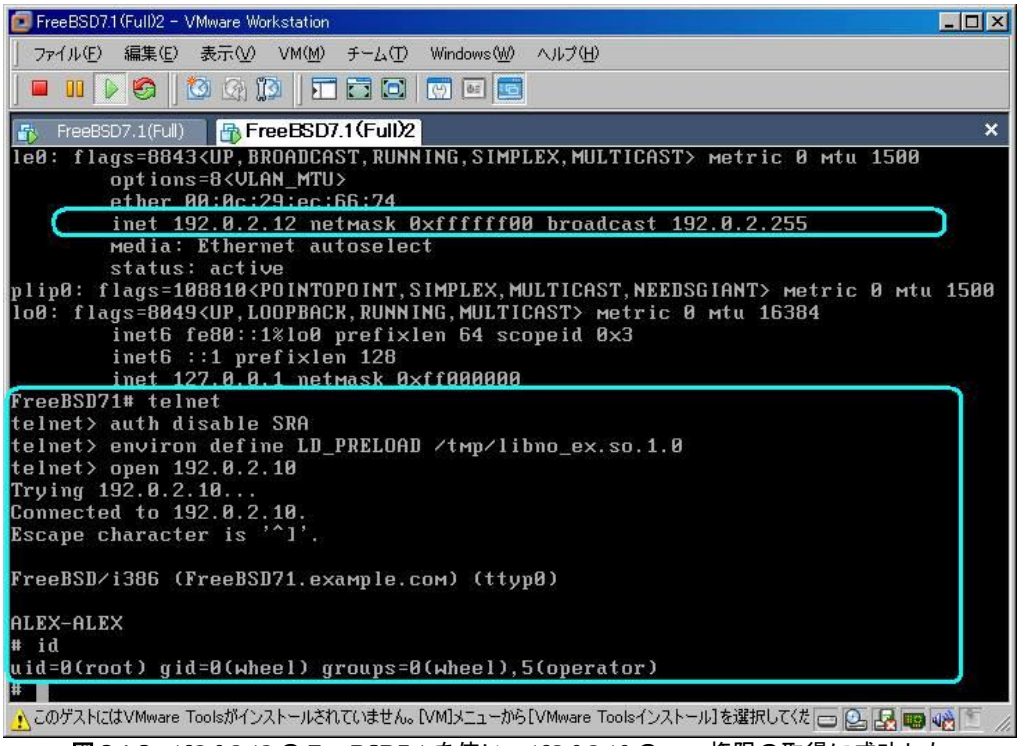

図 2.1-3 : 192.0.2.12 の FreeBSD7.1 を使い、192.0.2.10 の root 権限の取得に成功した

図 2.1-4 : Linux(CentOS5.1)の Telnet クライアントでは、
オプションを反映させることができないため成功しない

図 2.1-5 : WindowsXP の Telnet クライアントでも、
オプションを反映させることができないため成功しない

3. LD_PRELOAD 環境変数とセキュア・プログラミング

3.1. LD_PRELOAD 環境変数とは

LD_PRELOAD 環境変数とは、共有ライブラリ(.so)を事前にロードするために使われるものである。
LD_PRELOAD 環境変数に共有ライブラリを示すファイルパスを指定すると、メインのプログラムが起動した際に、他の共有ライブラリがロードされる前に呼び出される。
また、LD_PRELOAD 環境変数で指定した共有ライブラリに含まれる関数の名前と、他の共有ライブラリに含まれる関数の名前が重複した場合、LD_PRELOAD 環境変数で呼び出された共有ライブラリの関数が優先的に使用される。
このことから、本来呼び出される共有ライブラリの関数をフックする、または置き換える目的で使用されることが、一般的な LD_PRELOAD 環境変数の使い方である。

しかしながら、”呼び出される関数を置き換えられる” という特性から、悪意をもって置き換えられることによって不正使用に悪用される危険性もある。

例えば、図 3.1-1 のようなプログラムをサンプルとしてみる。
環境は、以下である。

- OS：CentOS5.1
- gcc4.1.2

図 3.1-1 は、プログラムの第一引数にパスワードを与え、そのパスワードが正しいかどうか(foobar かどうか)を確認するプログラムである。これを gcc でコンパイルする(図 3.1-2)。
図 3.1-3 でプログラムを動かし、第一引数が「foobar」かどうかで、出力するメッセージが異なることが確認できる。

図 3.1-1 は、文字列比較(認証処理)として、標準の strcmp()関数を用いているが、図 3.1-4 のように、文字列比較をしないが同名の strcmp()関数を定義し、共有ライブラリとしてコンパイルする。
この図 3.1-4 で作成された共有ライブラリを LD_PRELOAD 環境変数にセットした上で、図 3.1-1 のプログラムを起動すると、本来は標準の strcmp()関数が呼び出されるところを図 3.1-4 で新たに定義した関数が呼び出され、どんな文字列を第一引数に与えても、「foobar」を与えたと同じ結果となるように動作していることが確認できる(図 3.1-5)。
このように、認証処理や暗号化処理などの共有ライブラリが置き換えられるというセキュリティ上の危険性が存在する。

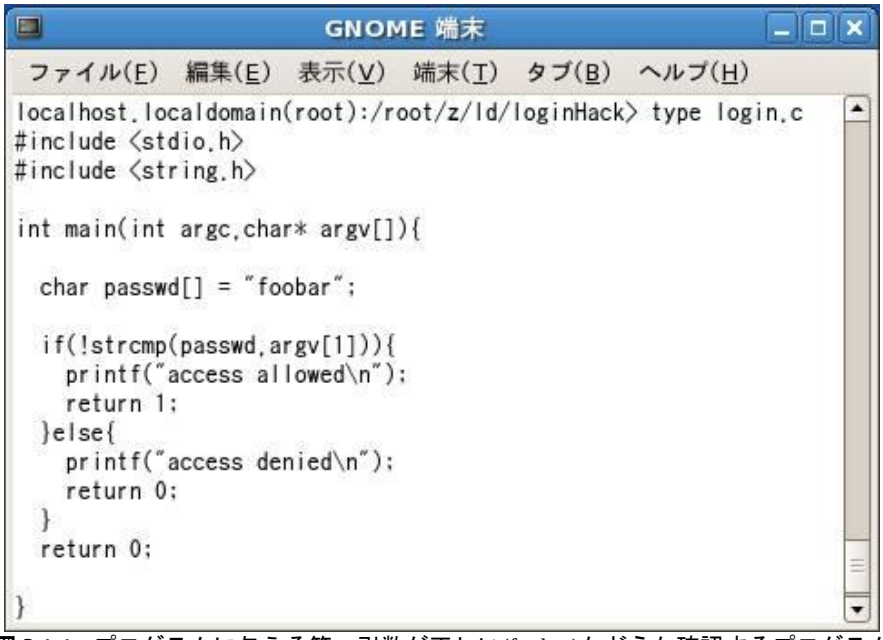

図 3.1-1 : プログラムに与える第一引数が正しい(foobar)かどうか確認するプログラム

単純に strcmp() 関数を使って比較している

図 3.1-2 : 図 3.1-1 を gcc4.1.2 CentOS5.1 でコンパイルした

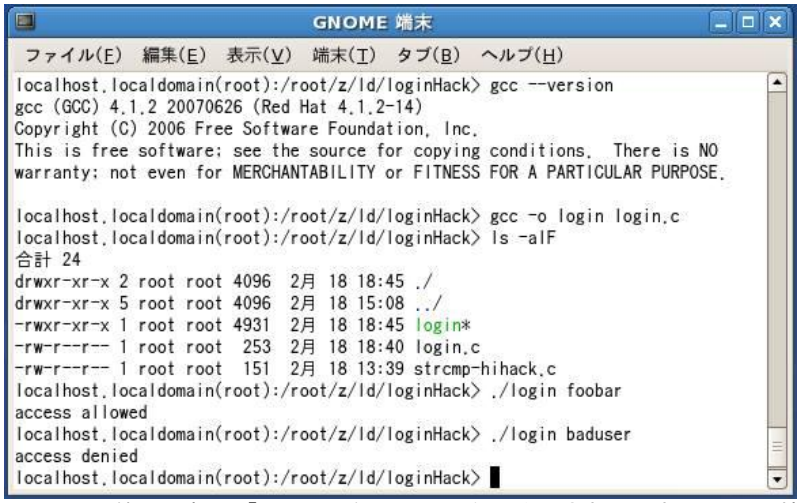

図 3.1-3 : 図 3.1-1 の第一引数に「foobar」を与えた場合は許可され、それ以外では禁止される

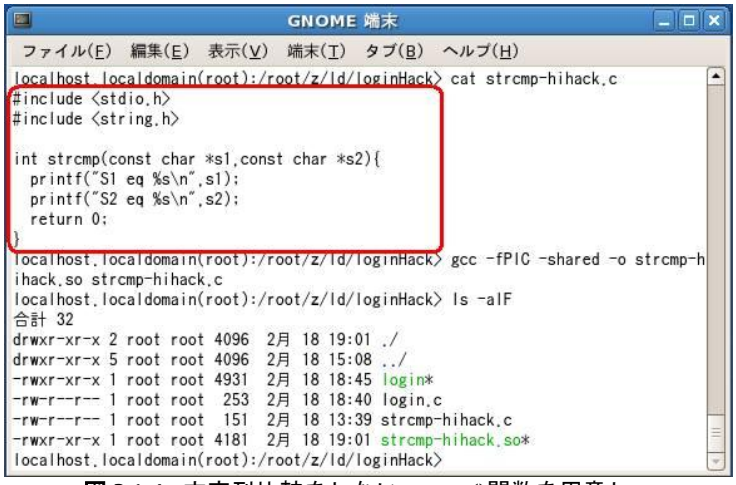

図 3.1-4：文字列比較をしない strcmp()関数を用意し、

共有ライブラリとしてコンパイルする

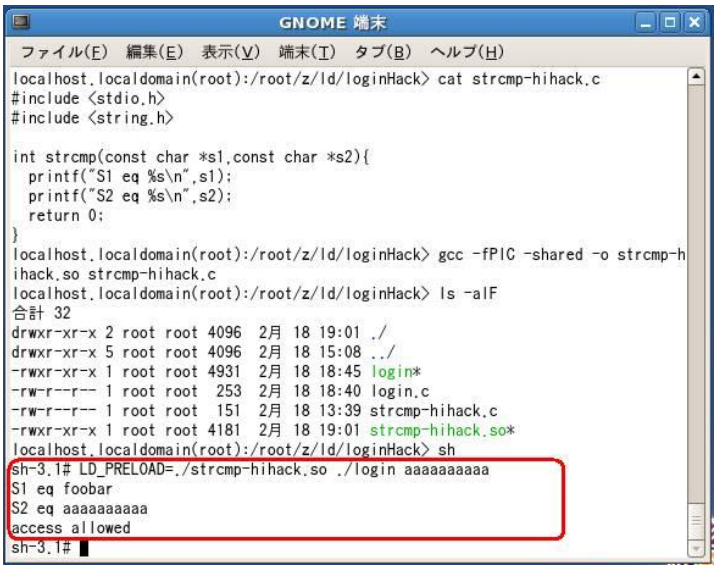

図 3.1-5：図 3.1-4 を LD_PRELOAD 環境変数によって事前ロードさせることで、

図 3.1-1 のプログラムの認証回避が可能となることが分かる

3.2. 過去の LD_PRELOAD 環境変数を使った脆弱性

「3.1 LD_PRELOAD 環境変数とは」で示したように、共有ライブラリを置換したりすることができるためこの手法自体は非常に有効なテクニックであるが、この LD_PRELOAD 環境変数に起因する脆弱性は、過去にいろいろなソフトウェアで報告されている(「5 参考の 3～5」)。
このように、UNIX/Linux に対しての Local Exploit という観点では、LD_PRELOAD 環境変数を使った攻撃方法は古典的であるが、今一度、次の節から LD_PRELOAD 環境変数の悪用を防止する対策について検討してみたい。

3.3. LD_PRELOAD 攻撃の対策 (environ 変数)

この節では、unistd.h で定義されている environ 変数を使って、LD_PRELOAD 環境変数が定義されているかどうかを、プログラムの先頭(main()関数の先頭)でチェックする方法を紹介する。
LD_PRELOAD 環境変数が定義されていれば、LD_PRELOAD 環境変数を破壊した上で、自らのプログラム自身を再起動させるようにした。
サンプルとなるソースコードは、図 3.3-1 である。
プログラム起動直後に unistd.h で定義されている環境変数を保持している領域のポインタ environ を取得し、LD_PRELOAD 環境変数の有無をチェックしている。
LD_PRELOAD 環境変数が定義されている場合は、LD_PRELOAD 環境変数を無効化したうえで自分自身を再読み込みしている。
環境変数を取得する関数 getenv() 関数を使っていない理由は、ただ一つ、関数だからである。
LD_PRELOAD 環境変数で指定されたライブラリによって、getenv()関数が危険な関数に置き換えられる危険性があるため、environ を使って main 関数内部に処理(while などを使って)を実装している。

この対策のデメリットとしては、対策のコードをライブラリ化できない点であろう。

また、再起動させるための execv()関数が LD_PRELOAD 環境変数によって置き換えられている可能性もあるため(図 3.3-4)、

- そもそも環境変数を任意に設定できない環境で動作させる。
- そういう環境で動作させることができない場合、LD_PRELOAD 環境変数が定義されているかどうかをプログラムの先頭でチェック後、定義されていれば終了する

というのが、現実的な解決策ではないだろうか。

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

int main(int argc,char* argv[]){
 char passwd[] = "foobar";       // 正しいパスワード
 char *envName = "LD_PRELOAD";   // 環境変数の領域で比較する文字列
 char **pp;
 char *p;
 char *p1;
 char *p2;
 int hint1;
 int hint2;

 /* Startup */
 pp = __environ;
 hint1 =0;
 while(*pp != NULL && hint1 == 0){
        p1 = *pp;               // 環境変数を一つずつ取得
        p2 = envName;           // LD_PRELOAD そのもの
        hint2 = 0;
        while(*p2 != '¥0' && hint2 == 0){       // 1Byte ずつ比較
                if(*p1 == '¥0'){
                        hint2++;
                }else{
                        if(*p1 != *p2){
```

```
                        hint2++;
                }else{
                        p = p1;         // 多分、LD_PRELOAD の最後の文字の
                        p1++;           // 'D'を示しているはず
                        p2++;
                }
        }
    }
    if(hint2 == 0){         hint1++;
    }else{                  pp++;           }
}
printf("hint1=%i¥nhint2=%i¥n",hint1,hint2);

if(hint2 == 0){
        printf("LD_PRELOAD found!¥nreloaded....¥n¥n");
        *p = '¥0';              // LD_PRELOAD 環境変数に NULL を入れて破壊する
        execv(argv[0],argv);    // 自分自身をリロードする
        return 0;
}else{  printf("LD_PRELOAD not found!¥n");       }

if(!strcmp(passwd,argv[1])){
        printf("access allowed¥n");
        return 1;
}else{
        printf("access denied¥n");
        return 0;
}               return 0;               }
```

図 3.3-1 : 図 3.1-1 を改造し、起動時に(char**)environ を使って

LD_PRELOAD 環境変数の有無をチェックするようにしたプログラム(login2.c)

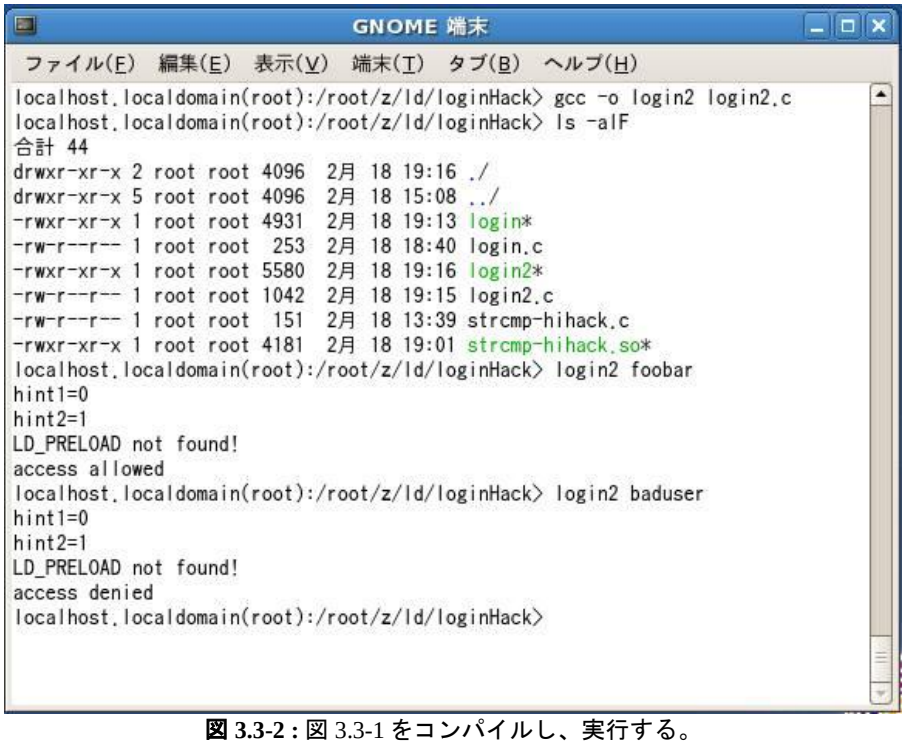

図 3.3-2 : 図 3.3-1 をコンパイルし、実行する。

第一引数が「foobar」の時とそれ以外の時でメッセージが異なることが確認できる

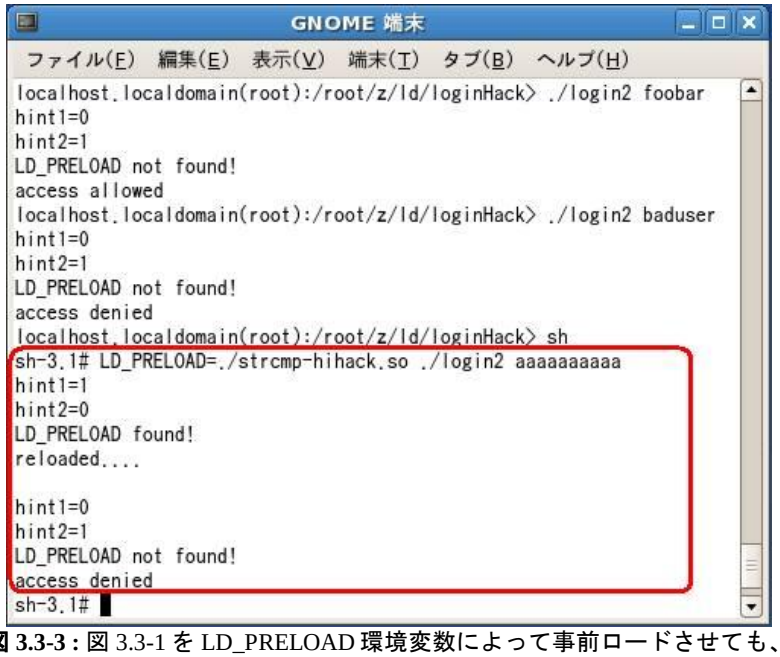

図 3.3-3 : 図 3.3-1 を LD_PRELOAD 環境変数によって事前ロードさせても、

main()関数の先頭で、LD_PRELOAD 環境変数の有無をチェックし、破壊後に再読み込みしているため、

strcmp()関数の置き換えがおこなわれない。

結果的に図 3.3-2 と同じ挙動となっていることが確認できる。

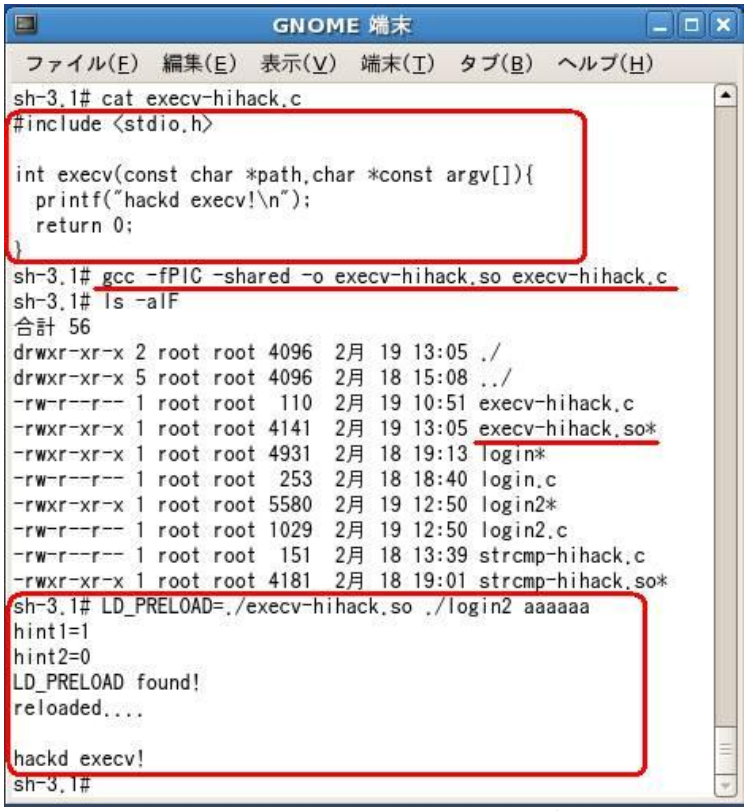

図 3.3-4 : 図 3.3-3 では LD_PRELOAD 環境変数でロードされたライブラリを無視することができたが、

リロードするために呼び出した execv()関数が置き換えられて、

プログラムが乗っ取られているのが確認できる

3.4. LD_PRELOAD 環境変数と gcc 拡張 (environ 変数の回避策)

「3.3 LD_PRELOAD 攻撃の対策」の方法では回避策があることが判明したので、ここに記述する(「5 参考の 27」)。
gcc の拡張機能を使って、main()関数より先に呼び出される関数を定義することができる。
この gcc の拡張機能を使うことで「3.3 LD_PRELOAD 攻撃の対策」で示した main()関数内の処理より先に回避コードを実行させることが可能である。

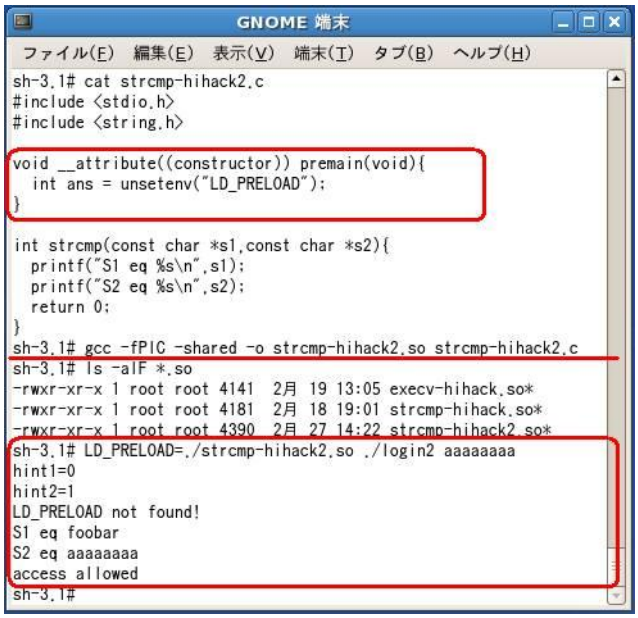

図 3.4-1 : gcc 拡張を使って main()関数より先に呼び出す関数を
LD_PRELOAD 環境変数で指定した共有ライブラリ内に定義することで、
「3.3 LD_PRELOAD 攻撃の対策 (environ 変数)」で考察した方法を回避することが可能である。

3.5. LD_PRELOAD 攻撃の対策 (setuid を使う)

この節では、setuid を使った方法を紹介する。
passwd コマンドなど setuid されたプログラムを保護するため、setuid されたプログラムでは、LD_PRELOAD 環境変数によるライブラリの置換ができないようになっている(図 3.5-1)。
とはいえ、図 3.5-2 のように、プログラムの所有者と実行者が同一の場合、LD_PRELOAD 環境変数によるライブラリの置換が行われてしまうことに留意しておく必要がある。
しかしながら、プログラムの所有者と実行者が異なる場合は、LD_PRELOAD 攻撃対策として、setuid を使うことが可能である。

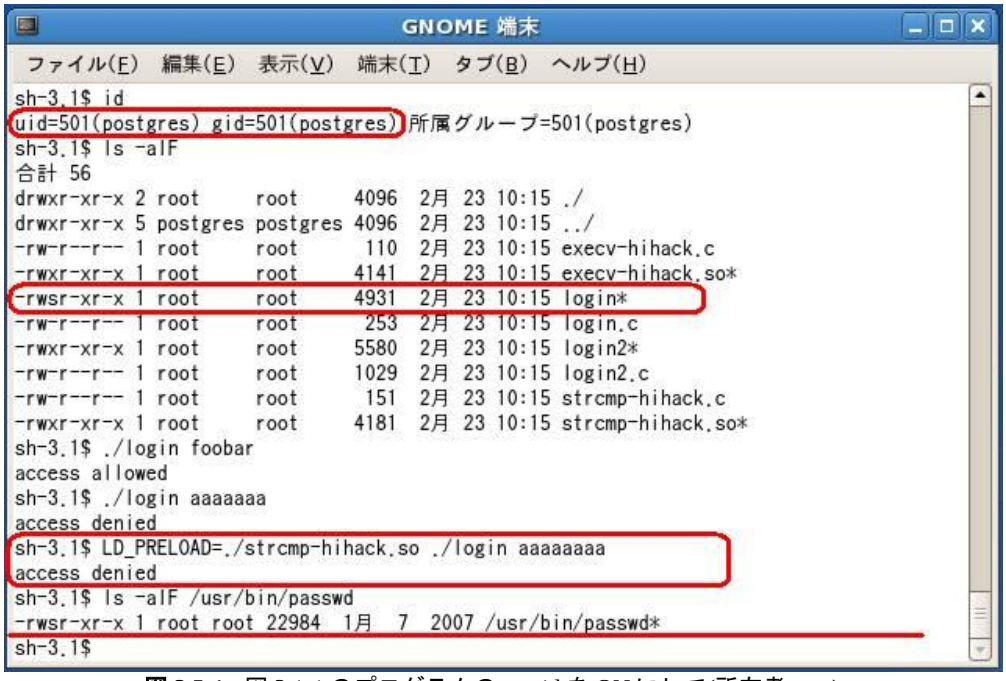

図 3.5-1 : 図 3.1-1 のプログラムの setuid を ON にして(所有者 root)、
一般ユーザ(postgres)で実行する場合、
LD_PRELOAD 環境変数によるライブラリの置換を行うことはできない。
また、パスワード変更コマンド(passwd)などには setuid されているため、
LD_PRELOAD 環境変数によるライブラリ置換ができないようになっている。

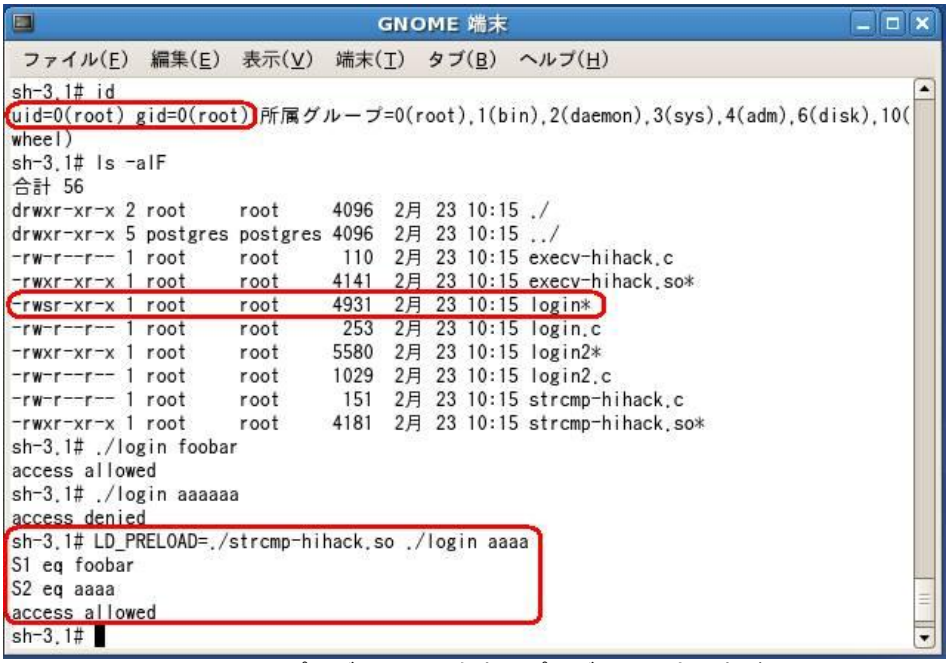

図 3.5-2 : プログラムの所有者とプログラムの実行者が
同一(図では root)の場合は、setuid を ON にするだけでは、
図 3.5-1 のような LD_PRELOAD 環境変数による置き換えが無効にはならない。

3.6. setuid()関数について

さらに、setuid()関数についても挙動を実験した結果をここに載せておく(実験したソースコードは図 3.6-1 である)。
setuid()関数を呼び出すことで、他の uid でコードを実行することができるのであるが、それにはファイルシステム上の「setuid」ビットが ON になっている必要がある(図 3.6-2～図 3.6-3)。
また、root 権限のプログラムが一度、setuid()関数によって他の権限に委譲すると、元の root 権限に戻れなくなる(図 3.6-4)。
ファイルシステムの setuid と setuid()関数の関係について確認したのが、図 3.6-5～図 3.6-8 である。

表にすると以下のようになる。

setuid=ON

実行者	ファイル所有者	起動時の uid/euid	setuid(0)	setuid(501)	setuid(48) (※)
root	root	0/0	0/0	501/501	
501	root	501/0	0/0	501/501	48/48
root	48	0/48	0/0	-1	0/48
501	48	501/48	-1	501/501	501/48

setuid=OFF

実行者	ファイル所有者	起動時の uid/euid	setuid(0)	setuid(501)	setuid(48) (※)
root	root	0/0	0/0	501/501	
501	root	501/501	-1	501/501	-1
root	48	0/0	0/0	501/501	48/48
501	48	501/501	-1	501/501	-1

- 0： ユーザ名=root
- 48： ユーザ名=apache
- 501： ユーザ名=postgres

(※)：一般ユーザ権限が他の一般ユーザ権限(実行者以外)へ遷移可能かどうかを目的にした列

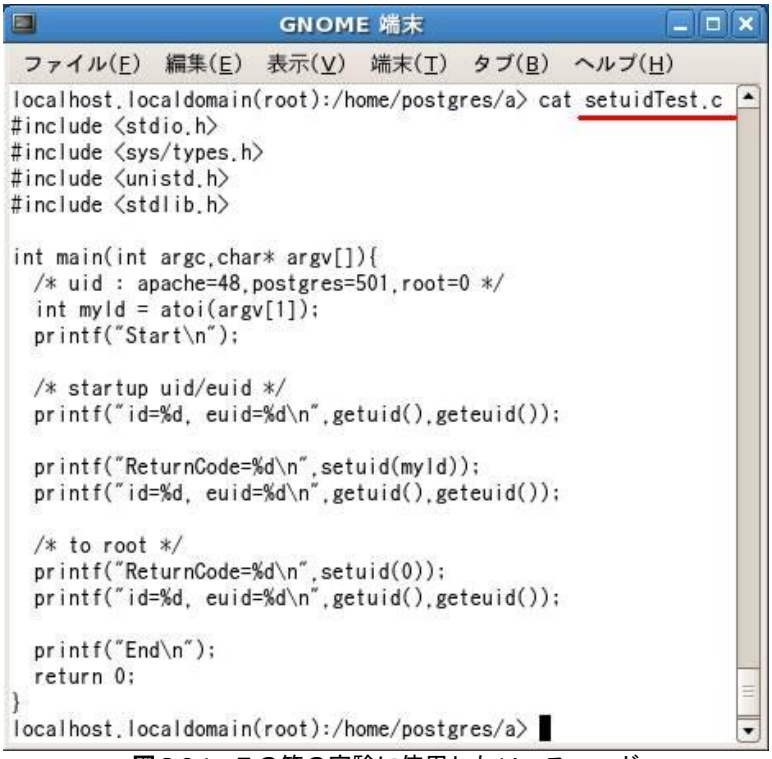

図 3.6-1：この節の実験に使用したソースコード

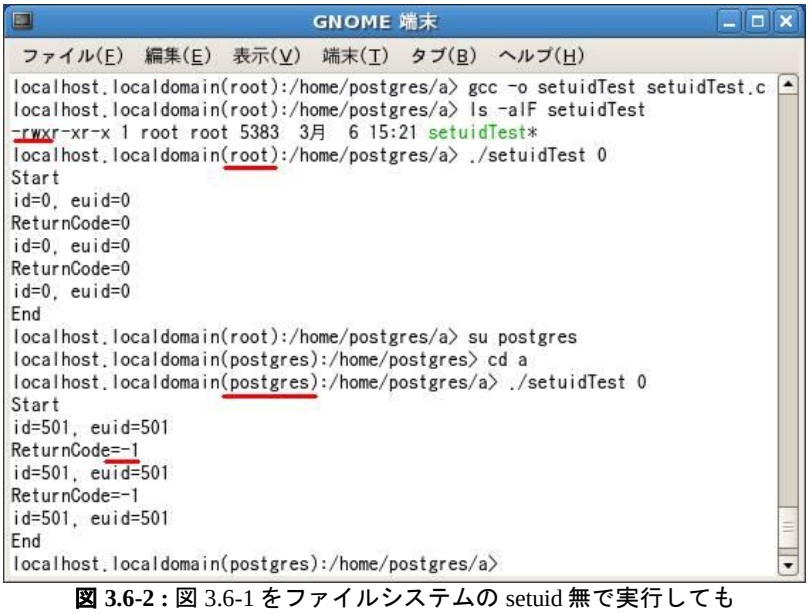

図 3.6-2：図 3.6-1 をファイルシステムの setuid 無で実行しても setuid()関数で root 権限になることはできない

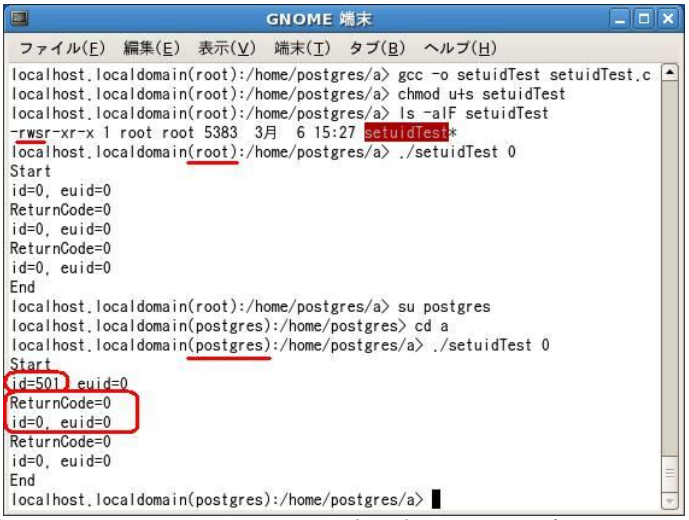

図 3.6-3 : 図 3.6-1 をファイルシステムの setuid 付で実行すると、euid がファイル所有者の ID となる。

また、ここで setuid()関数を使うことで uid/euid 共に root 権限になる。

さらに、ファイルシステムの setuid フラグだけで uid が root 権限(実行ファイルの所有者)に

なるわけではないことが、最初の「id=501」の表示から確認できる

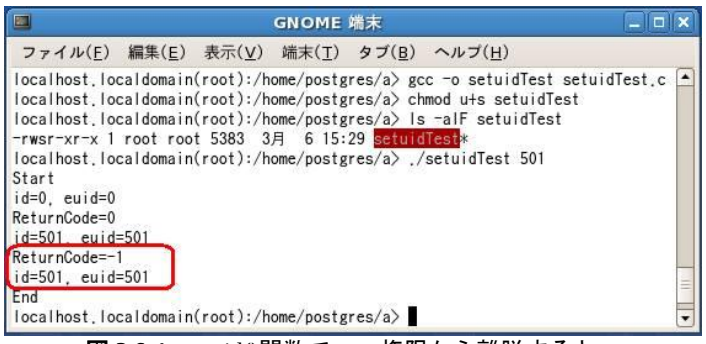

図 3.6-4 : setuid()関数で root 権限から離脱すると、

二度と root 権限を取得することができない

図 3.6-5 : 一般ユーザ所有で setuid が付与されている場合、

root 権限で setuid()関数を使っても、uid を root 権限から離脱することはできない。

(euid はファイル所有者の ID なので、最初から root 権限から離脱している)

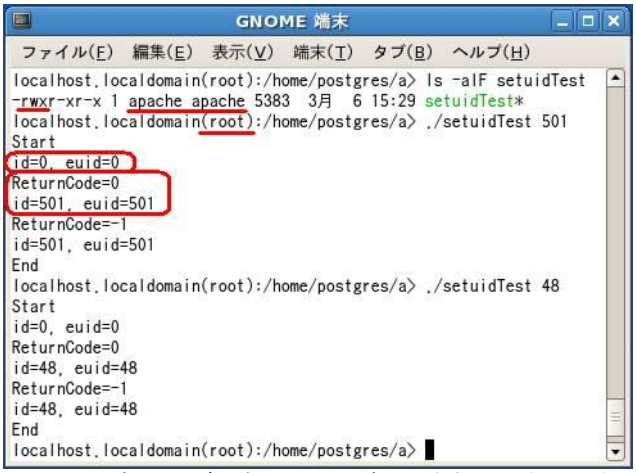

図 3.6-6 : 一般ユーザ所有で setuid が付与されていない場合、

root 権限で setuid()関数を使うことによって、uid/euid 共に root 権限から離脱することができる。

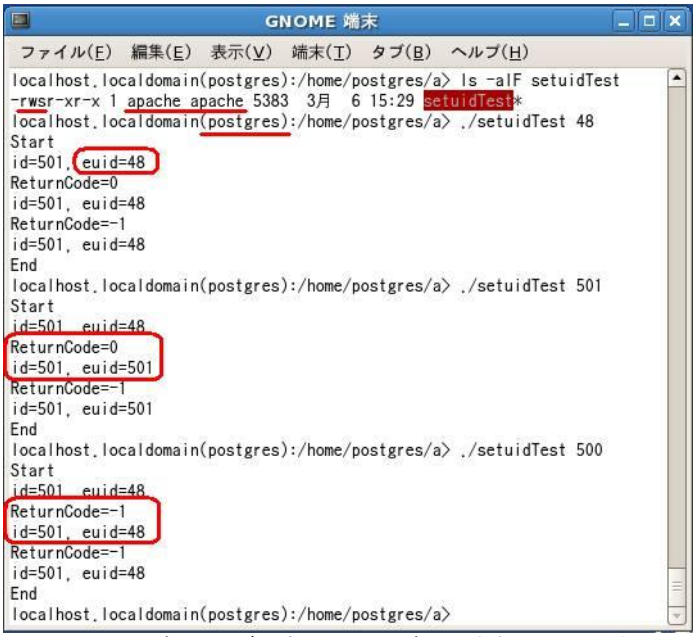

図 3.6-7 : 一般ユーザ所有で setuid が付与されている場合、

起動直後に euid はファイル所有者の ID に変わっている。

しかし、setuid()関数によって他者の権限になることはできない。

(euid を自分自身の権限に戻る事は可能(図の setuidTest 501 の箇所))

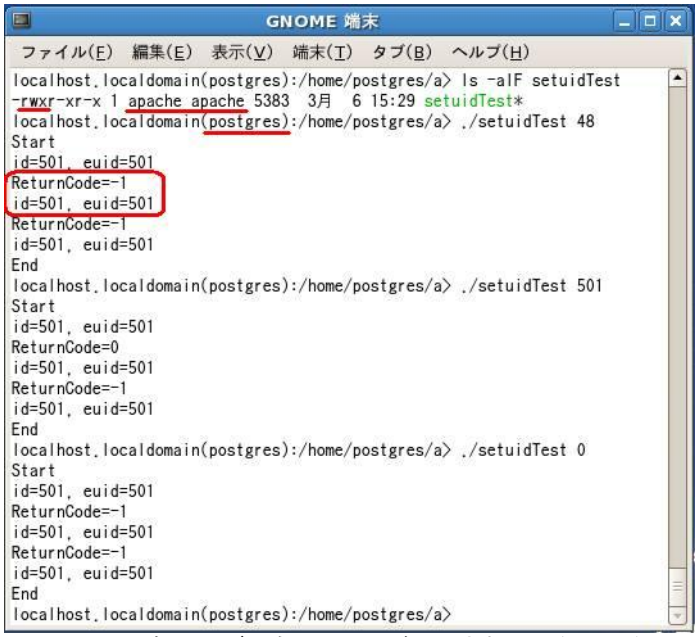

図 3.6-8：一般ユーザ所有で setuid が付与されていない場合、
一般ユーザ権限で setuid()関数を使って、権限の変更ができない。

3.7. setuid()関数と LD_PRELOAD 環境変数 1

この setuid()関数によって、権限が変更する前後で、LD_PRELOAD 環境変数で定義したライブラリがどのような扱いになるのかを確認してみた。
図 3.7-1～図 3.7-5 で確認できることは、LD_PRELOAD 環境変数と関係があるのは、setuid()関数ではなく、ファイルシステムの setuid フラグであるということである。

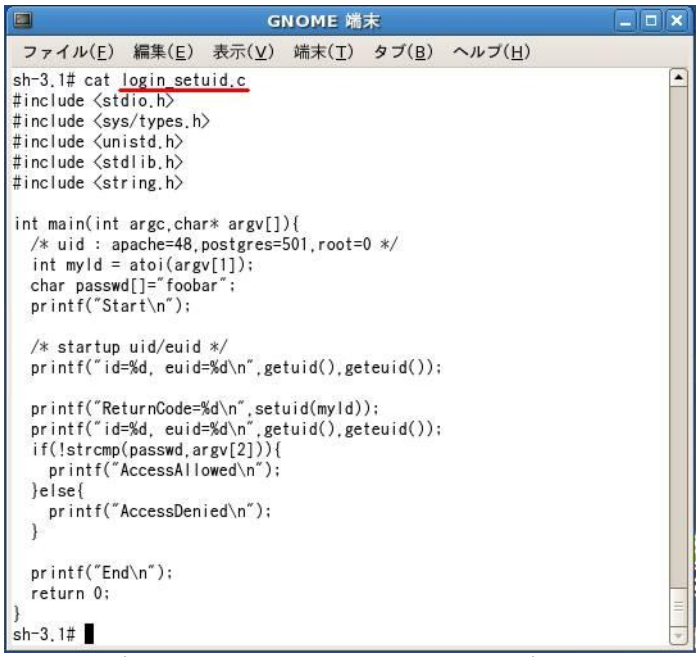

図 3.7-1：確認するためのコードは、この図で示しているソースコードと図 3.1-4 の共有ライブラリである

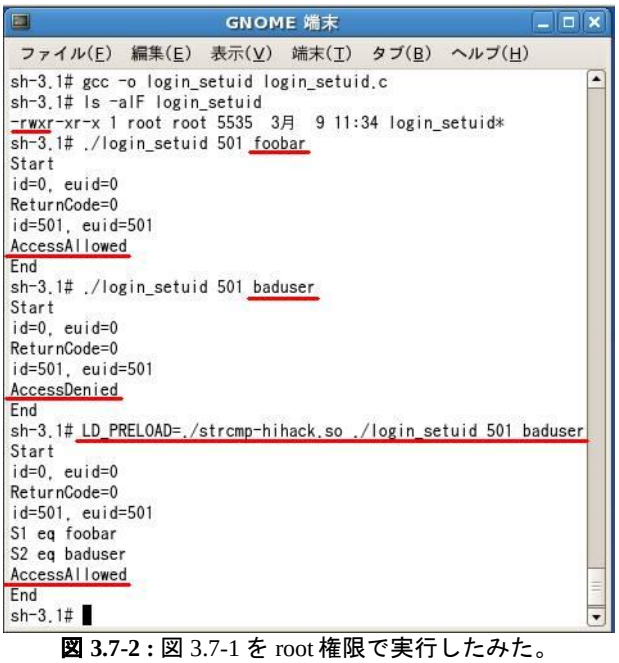

図 3.7-2 : 図 3.7-1 を root 権限で実行したみた。

一般ユーザ(501=postgres)に権限が離脱しているが、結局のところ

LD_PRELOAD 環境変数で定義した共有ライブラリ内の関数が呼び出されている

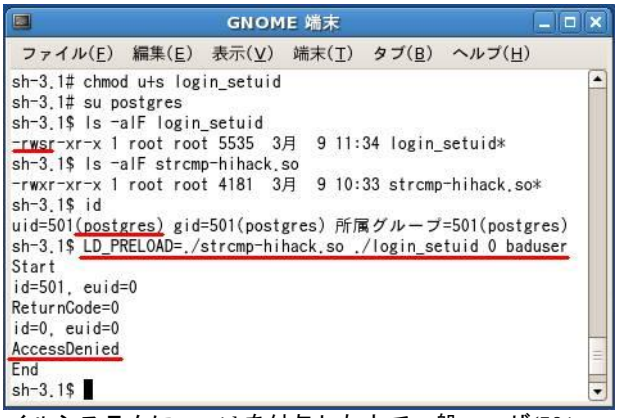

図 3.7-3 : 図 3.7-1 をファイルシステムに setuid を付与した上で一般ユーザ(501=postgres)で実行してみた。

setuid()関数によって、root 権限に昇格後は LD_PRELOAD 環境変数で

定義した共有ライブラリ内の関数が呼び出されていないことが確認できた。

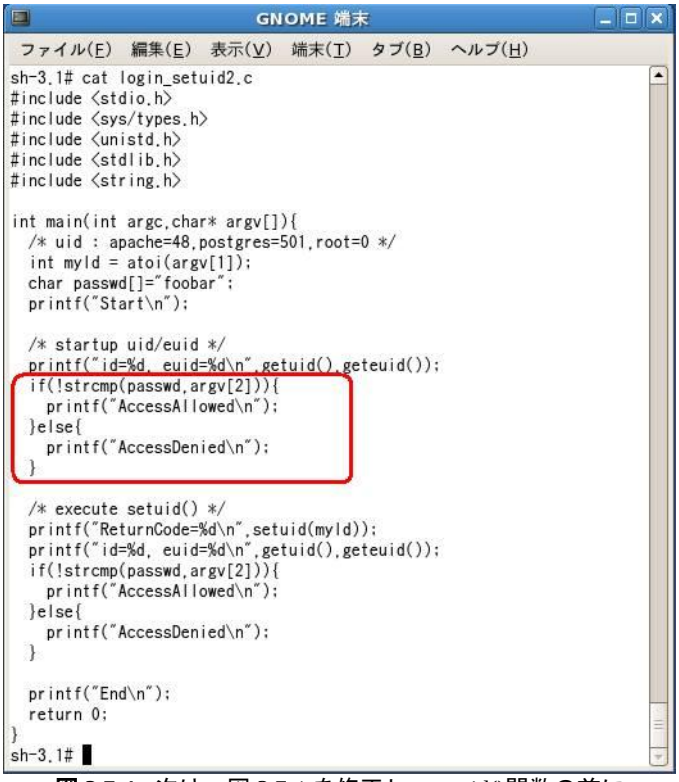

図 3.7-4 : 次は、図 3.7-1 を修正し、setuid()関数の前に
事前に strcmp 関数(置換対象の関数)を呼び出してみる

図 3.7-5 : setuid()関数よりも早く呼び出された strcmp()関数も置換されていないことから、
setuid()関数よりもファイルシステムの setuid フラグが
LD_PRELOAD 環境変数と関係があることが確認できる

3.8. setuid()関数と LD_PRELOAD 環境変数 2

「3.6 setuid()関数について」で setuid()関数が登場したので、setuid()関数が LD_PRELOAD 環境変数で置換可能かどうか確認してみた(図 3.8-1)。
結局、setuid()関数も LD_PRELOAD 環境変数の置換対象の関数であることが図 3.8-1 から確認できた。

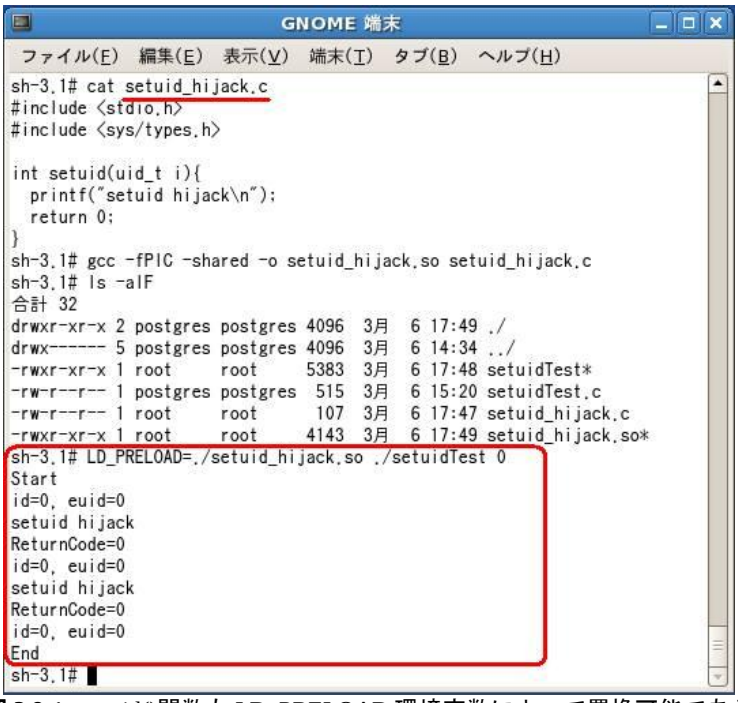

図 3.8-1 : setuid()関数も LD_PRELOAD 環境変数によって置換可能である

3.9. LD_PRELOAD 攻撃の対策 (Static Link を使う)

この節では、static link を使った方法を紹介する。
コンパイル時にスタティックリンクでコンパイルすると、動的ライブラリ(共有ライブラリ)を呼び出すことがなくなるため、LD_PRELOAD 環境変数を使った関数置換攻撃を防ぐことができる。

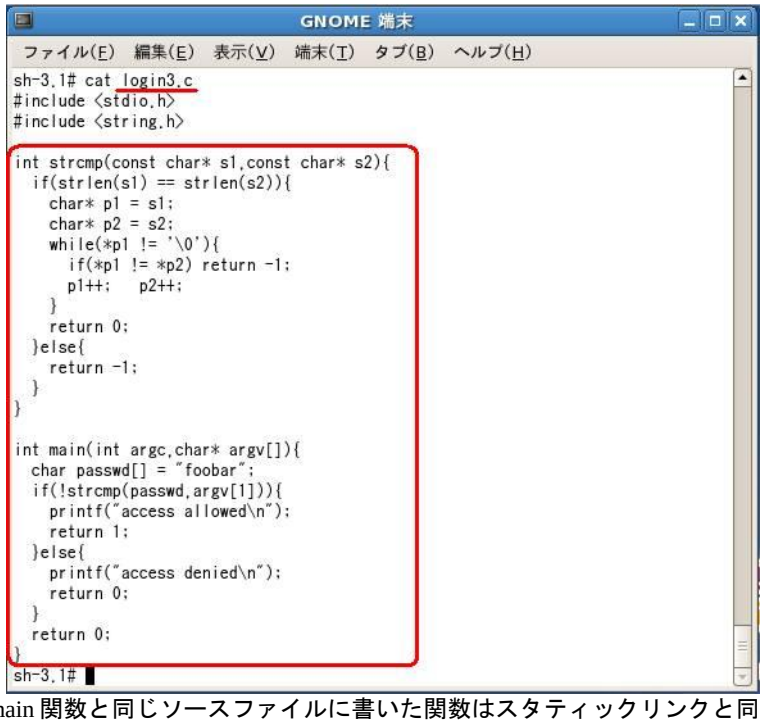

図 3.9-1 : main 関数と同じソースファイルに書いた関数はスタティックリンクと同じなので、LD_PRELOAD 環境変数による共有ライブラリ内の関数置換が行われない(その 1)

図 3.9-2 : main 関数と同じソースファイルに書いた関数はスタティックリンクと同じなので、LD_PRELOAD 環境変数による共有ライブラリ内の関数置換が行われない(その 2)

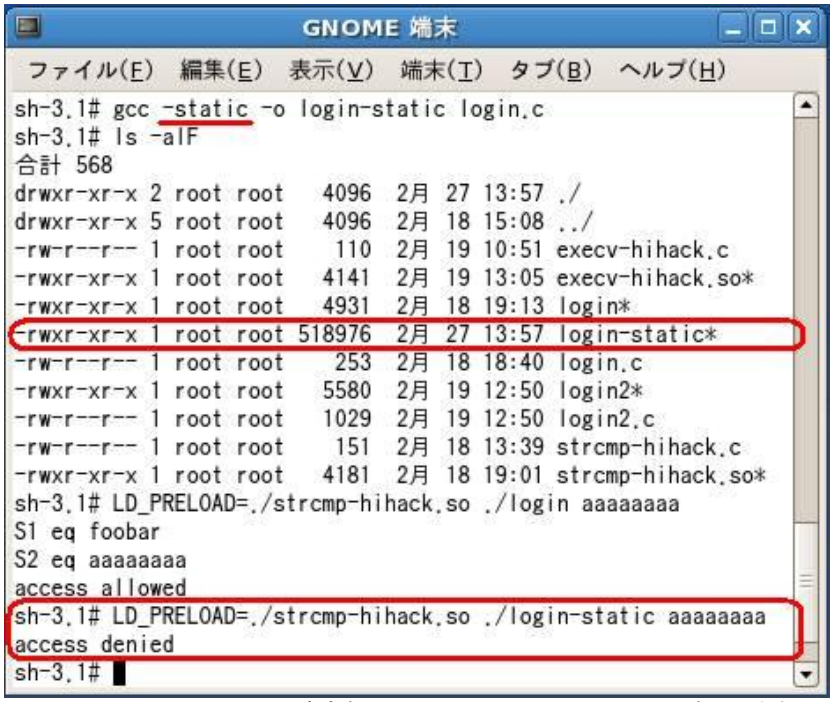

図 3.9-3 : LD_PRELOAD 環境変数はスタティックリンクでコンパイルされたプログラムに対しては、動作に影響を与えない。

3.10. 関数の呼び出し方によって LD_PRELOAD 攻撃が有効にならない場合

今回の検証中に、以下のソースコードで LD_PRELOAD 環境変数による置換が有効にならなかった。
これは、strcmp()関数特有の問題かも知れない。
また、他の関数でも同様に呼び出し方法によって、LD_PRELOAD 攻撃対策となるかも知れない。

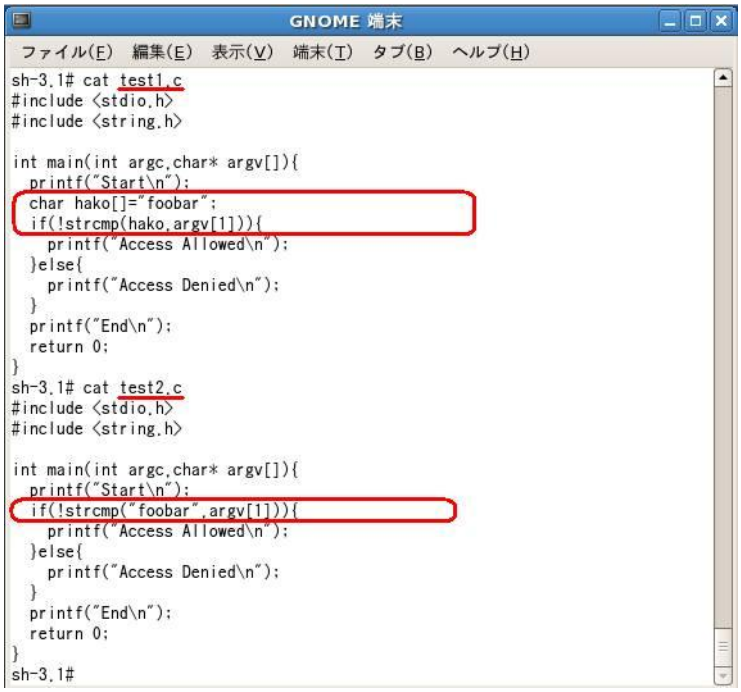

図 3.10-1 : 二つのソースコードの違いは、直接文字列リテラルを関数の引数にしているかどうかである

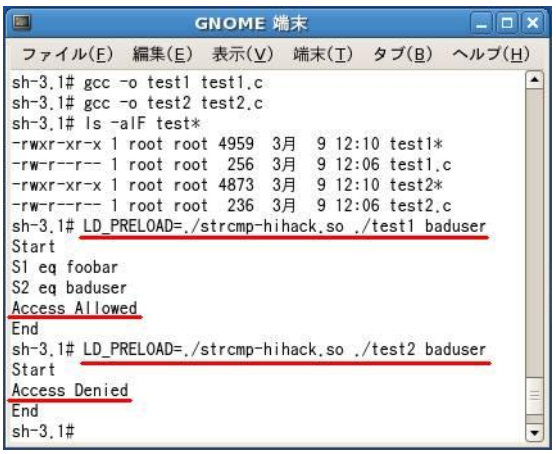

図 3.10-2 : 図 3.10-1 を実行してみたが、直接文字列リテラルを関数に指定すると、
LD_PRELOAD 環境変数による関数置換が有効にならなかった。

3.11. LD_PRELOAD 攻撃の対策のまとめ

今回は、LD_PRELOAD 攻撃対策をいくつか考察してみた。

- 変数「environ」を使って、LD_PRELOAD 環境変数が定義されているかどうか確認する方法
- ファイルシステムの setuid フラグを使う方法
- スタティックリンクしてしまう方法
- LD_PRELOAD 環境変数で置換される関数の呼び出し方を工夫することで、置換されないようにする方法

どれも一長一短があり、決め手とは言い難いのではないだろうか。

そもそも、不特定多数の利用者にプロセスの環境変数を自由に設定可能である状態そのものがセキュリティ上危険な状態であると評価してもいいのかも知れない。

よって、以下の対策を状況に応じて取捨選択すべきだろう

- プロセスの環境変数を自由に設定できない環境で、プログラムを動作させる
- システム上に任意のファイルを書き込めないような環境で、プログラムを動作させる
  (FTP サーバと同居している WebApplication サーバや、ファイルアップロード機能付き WebApplication サーバなどの場合は、厳しい条件かも知れない)
- ライブラリを全てスタティックリンクしてしまう
- ファイルシステムの setuid フラグを使って、LD_PRELOAD 環境変数を無効化してしまう
- プログラムが起動した最初の処理として、LD_PRELOAD 環境変数が定義されているかどうかチェックする
- 必要な環境変数以外を全て削除する処理を、プログラムが起動した最初の処理として行う

UNIX/Linux 上でアプリケーションを開発している読者諸氏は、今一度本文書で取り上げた古典的な共有ライブラリの置換方法である LD_PRELOAD 環境変数を使った攻撃への対策が、自ら開発中のソフトウェアに必要かどうか、必要である場合は対策しているかどうかを再確認してみてはいかがだろうか。

3.12. LD_PRELOAD 環境変数と libsafe

libsafe という C 言語で書かれたプログラムのバッファオーバーフロー(BoF)などの脆弱性を防止するラッパー・ライブラリがあるが、このライブラリの使用に LD_PRELOAD 環境変数が使用されることがある(「5 参考 26」)。
libsafe もセキュリティ対策として必要な場合は、LD_PRELOAD 環境変数に libsafe 以外のライブラリが指定されないようなチェックが必要になる。
(「3.3 LD_PRELOAD 攻撃の対策 (environ 変数)」では、LD_PRELOAD 環境変数の有無のチェックのみであるが、少しの改造で上記のチェック処理を実現することは可能であろう)

4. 検証作業者

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター
　佐名木 智貴

5. 参考

1. [Full-disclosure] FreeBSD zeroday
   http://lists.grok.org.uk/pipermail/full-disclosure/2009-February/067954.html
2. FreeBSD-SA-09:05.telnetd
   http://lists.freebsd.org/pipermail/freebsd-security/2009-February/005141.html
3. glibc LD_PRELOAD File Overwriting Vulnerability
   http://www.securityfocus.com/bid/2223
4. ld-linux.so LD_PRELOAD
   http://www.ca.com/jp/securityadvisor/vulninfo/Vuln.aspx?ID=405
5. Oracle LD_PRELOAD Privilege Escalation
   https://www.securinfos.info/english/security-advisories-alerts/20031021_Oracle_LD_PRELOAD_Privilege_Escalation.php
6. Libsafe を利用した Buffer Overflow 防止
   http://www.bflets.dyndns.org/Security/Libsafe.html
7. セキュアなプログラマー: 入力に目を光らす
   http://www.ibm.com/developerworks/jp/linux/library/l-sp3/index.html
8. 環境変数 LD_PRELOAD - 技術メモ帳
   http://d.hatena.ne.jp/lurker/20060511/1147354551
9. Linux Intrusion Detection System FAQ
   4.17. LIDS を使う時は、LD_PRELOAD 環境変数に注意した方がいいですか？
   http://www.linux.or.jp/JF/JFdocs/LIDS-FAQ/ld-preload-warning.html
10. IPA ISEC セキュアプログラミング講座「7-3 setuid は慎重に」
    http://www.ipa.go.jp/security/awareness/vendor/programming/b07_03_main.html
11. Articles:Izik : Reverse Engineering with LD_PRELOAD - security vulnerabilities database
    http://securityvulns.com/articles/reveng/
12. リンクと同名のシンボル - bk ブログ
    http://0xcc.net/blog/archives/000060.html
13. ウノウラボ Unoh Labs: LD_PRELOAD を使って任意の関数呼び出しにフックしてみる
    http://labs.unoh.net/2008/04/ld_preload.html
14. Linux Function Interception
    http://uberhip.com/people/godber/interception/index.html
15. execv()
    http://www.linux.or.jp/JM/html/LDP_man-pages/man3/exec.3.html
16. environ
    http://www.linux.or.jp/JM/html/LDP_man-pages/man7/environ.7.html

17. getenv()
http://www.linux.or.jp/JM/html/LDP_man-pages/man3/getenv.3.html
18. setuid()
http://www.linux.or.jp/JM/html/LDP_man-pages/man2/setuid.2.html
19. getuid()
http://www.linux.or.jp/JM/html/LDP_man-pages/man2/getuid.2.html
20. CREDENTIALS
http://www.linux.or.jp/JM/html/LDP_man-pages/man7/credentials.7.html
21. セキュリティホール memo メーリングリスト
投稿 ID=9592

6. 履歴

- 2009 年 02 月 20 日 : ver1.0 最初の公開
- 2009 年 03 月 13 日 : ver2.0
  “環境変数「LD_PRELOAD」” を “LD_PRELOAD 環境変数” に統一
  「2.1 FreeBSD7.1 の場合」に図を追記
  大幅に「3 LD_PRELOAD 環境変数とセキュア・プログラミング」を構成変更し、加筆修正
- 2009 年 05 月 26 日 : ver2.1 Web サイト移転に伴う最新版公開 URL の変更

7. 最新版の公開 URL

http://www.ntt.com/icto/security/data/soc.html#security_report

8. 本レポートに関する問合せ先

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター

e-mail: scan@ntt.com

以 上