FreeBSD-SA-09:05.telnetd と LD_PRELOAD 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 02 月 20 日

Ver. 1.0

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 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 の仕組み、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 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。

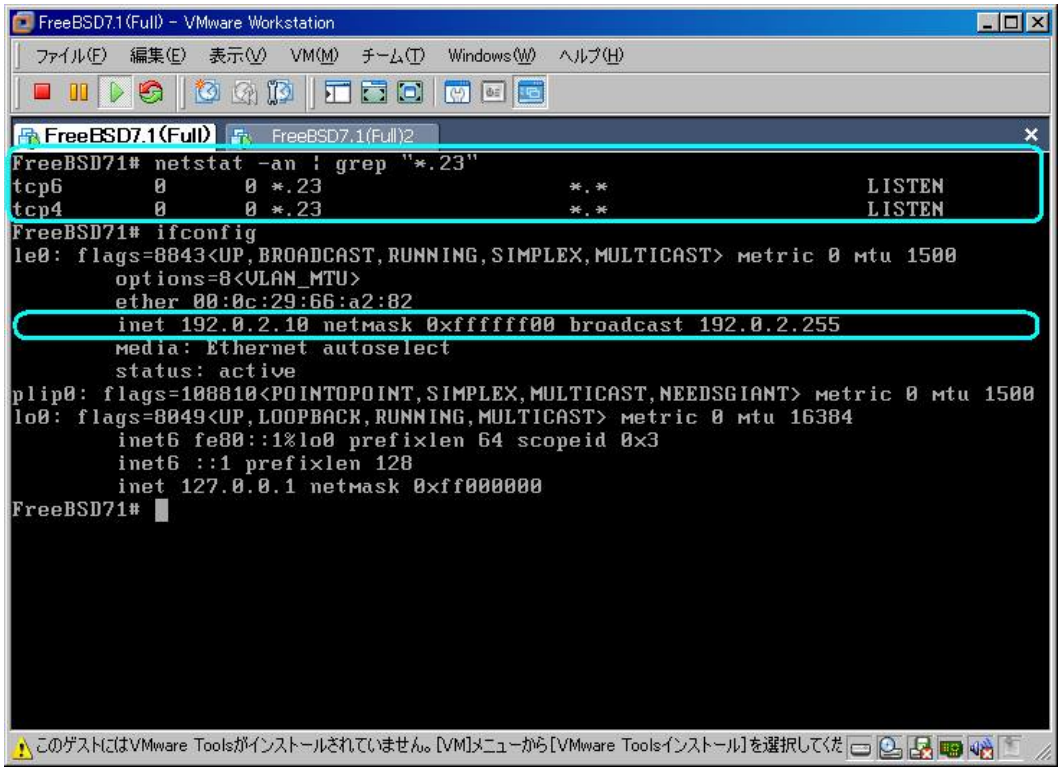

図 2.1-1 : 192.0.2.10 で動作する FreeBSD7.1 。Telnetd も起動している

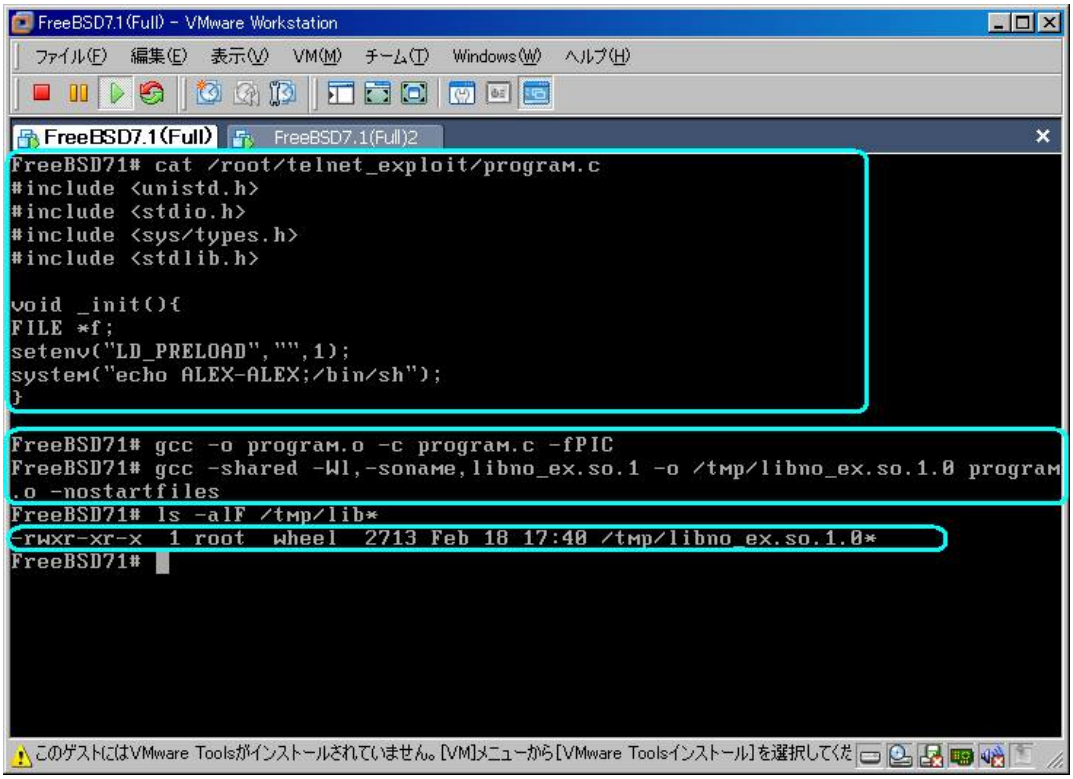

図 2.1-2 : 図 2.1-1 には、既に LD_PRELOAD で
ロードさせるライブラリもインストール済である

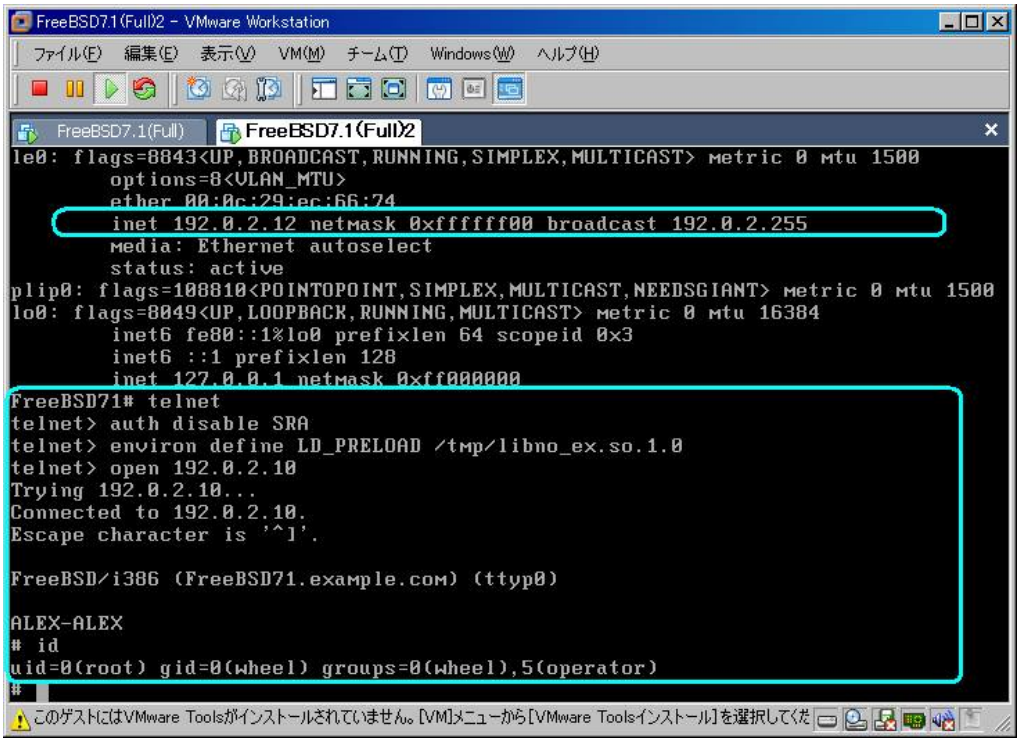

図 2.1-3 : 192.0.2.12 の FreeBSD7.1 を使い、192.0.2.10 の root 権限の取得に成功した

3. LD_PRELOAD とセキュア・プログラミング

3.1. LD_PRELOAD とは

環境変数「LD_PRELOAD」とは、ライブラリを事前にロードするために使われるものである。
環境変数「LD_PRELOAD」に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、他のライブラリがロードされる前に呼び出される。
また、環境変数「LD_PRELOAD」で指定したライブラリに含まれる関数の名前と、他のライブラリに含まれる関数の名前が重複した場合、環境変数「LD_PRELOAD」で呼び出された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、一般的な環境変数「LD_PRELOAD」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などす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の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。

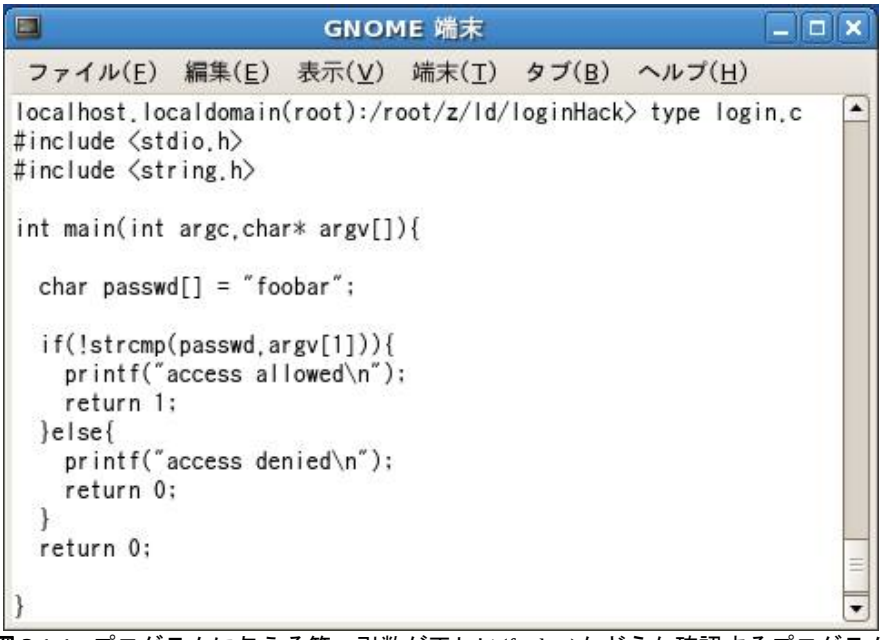

図 3.1-1 : プログラムに与える第一引数が正しい(foobar)かどうか確認するプログラム

単純に strcmp() 関数を使って比較している

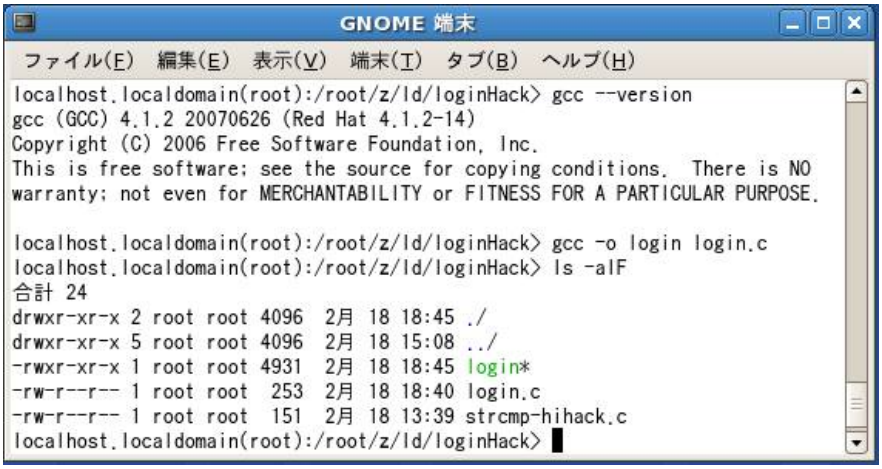

図 3.1-2 : 図 3.1-1 を gcc4.1.2 CentOS5.1 でコンパイルした

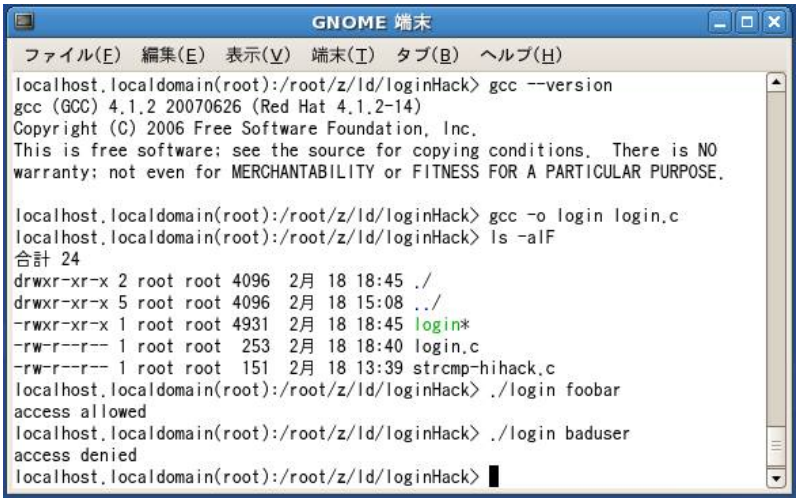

図 3.1-3 : 図 3.1-1 の第一引数に「foobar」を与えた場合は許可され、それ以外では禁止される

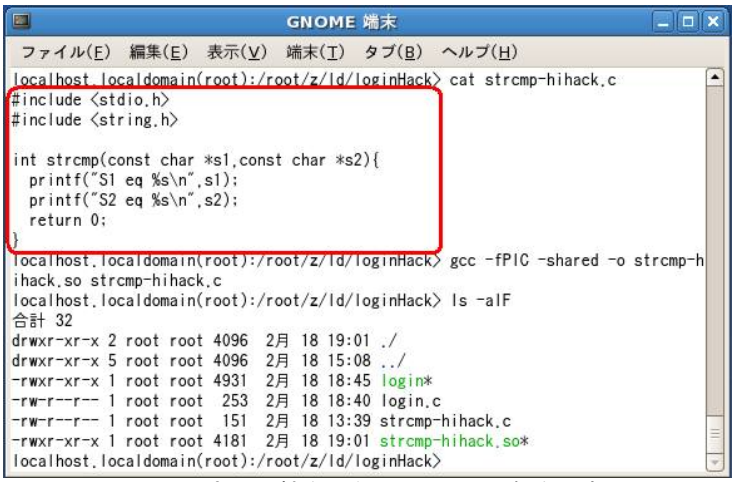

図 3.1-4 : 文字列比較をしない strcmp()関数を用意し、

共有ライブラリとしてコンパイルする

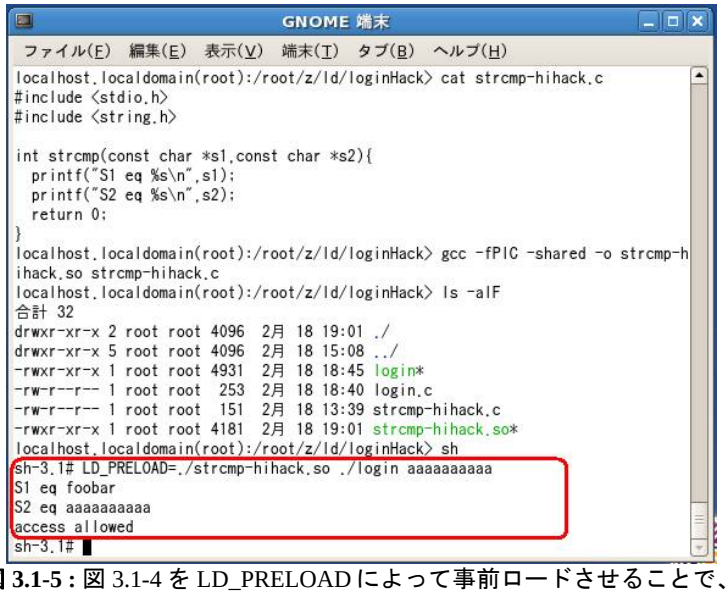

図 3.1-5 : 図 3.1-4 を LD_PRELOAD によって事前ロードさせることで、

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 を使った脆弱性

「3.1 LD_PRELOAD とは」で示したように、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 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 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 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

今回は、複数ある対策方法のうち、unistd.h で定義されている environ 変数を使って、環境変数「LD_PRELOAD」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。
環境変数「LD_PRELOAD」が定義されていれば、環境変数「LD_PRELOAD」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、環境変数「LD_PRELOAD」の有無をチェックしている。
環境変数「LD_PRELOAD」がある場合は、環境変数「LD_PRELOAD」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() を使っていない理由は、ただ一つ、関数だからである。
LD_PRELOAD 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';              // 環境変数 LD_PRELOAD 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

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

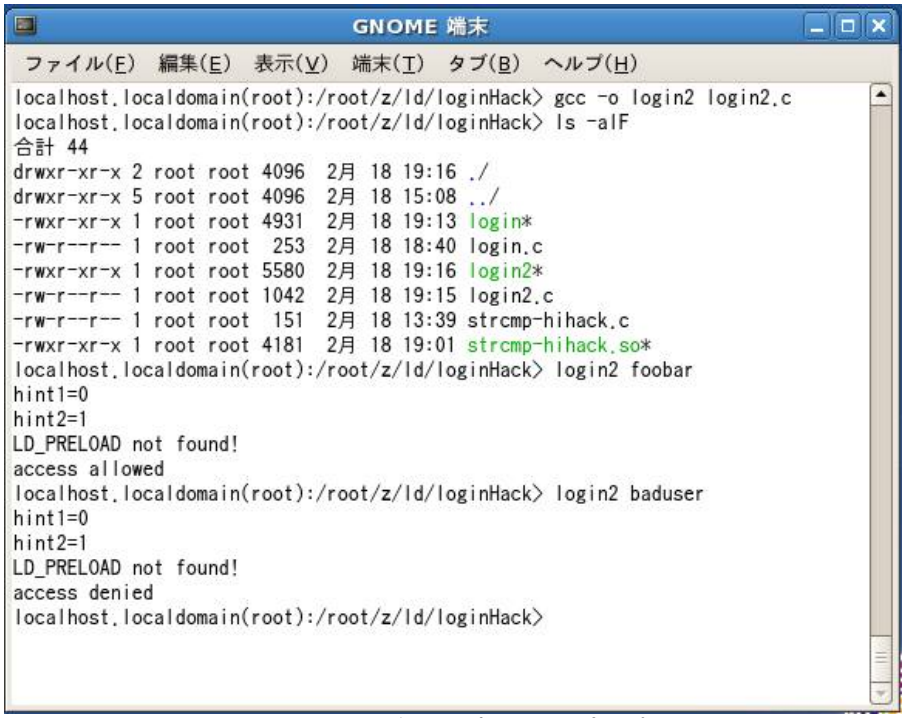

図 3.3-2 : 図 3.3-1 をコンパイルし、実行する。

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

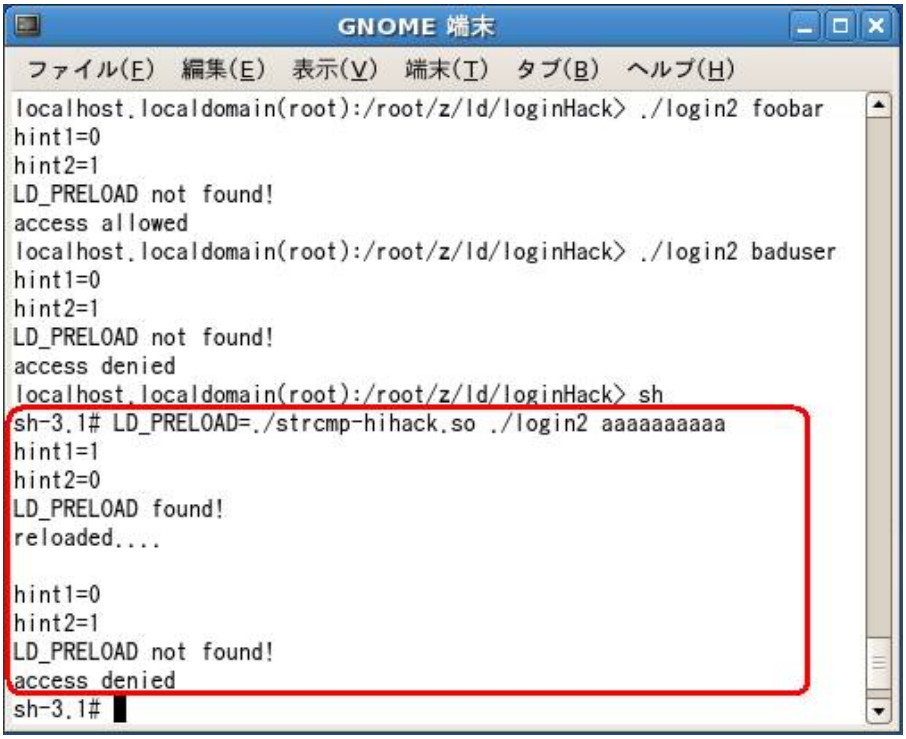

図 3.3-3 : 図 3.3-1 を LD_PRELOAD によって事前ロードさせても、main()関数の先頭で、環境変数「LD_PRELOAD」の有無をチェックし、破壊後に再読み込みしているため、strcmp()関数の置き換えがおこなわれない。

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。

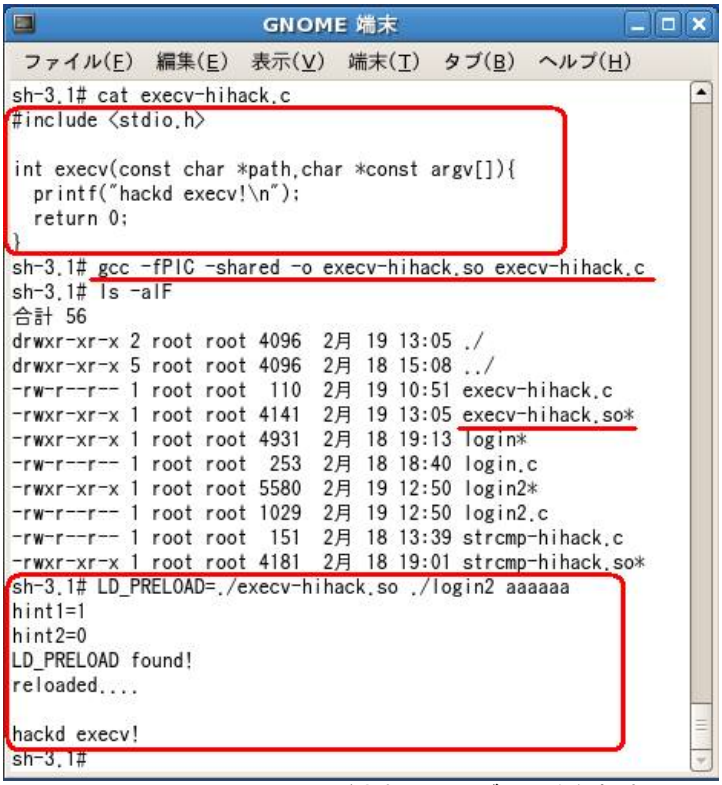

図 3.3-4 : 図 3.3-3 では LD_PRELOAD でロードされたライブラリを無視することができたが、
リロードするために呼び出した execv()関数が置き換えられて、
プログラムが乗っ取られているのが確認できる

3.4. LD_PRELOAD 攻撃の対策のまとめ

もう一度まとめてみる。
今回は、起動した最初の段階で環境変数「LD_PRELOAD」を検索するようにプログラムを修正することを対策の一つとして取り上げた。
しかしながら、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は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、

- 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)
- プログラムが起動した最初の処理として、環境変数「LD_PRELOAD」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ライブラリの置換方法である LD_PRELOAD 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。

3.5. LD_PRELOAD 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環境変数「LD_PRELOAD」が使用されることがある(「5 参考 12」)。
libsafe もセキュリティ対策として必要な場合は、環境変数「LD_PRELOAD」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。
(「3.3LD_PRELOAD 攻撃の対策」では、環境変数「LD_PRELOAD」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. CREDENTIALS
http://www.linux.or.jp/JM/html/LDP_man-pages/man7/credentials.7.html

6. 履歴

- 2009 年 02 月 20 日 : ver1.0 最初の公開

7. 最新版の公開 URL

http://www.icto.jp/security_report/index.html

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