

オープンソース・ソフトウェアの セキュリティ確保に関する調査報告書

第 部

セキュアな実行コードの生成・実行環境技術の利用ガイド

- Stack Smashing Protector・Libsafe -



情報処理振興事業協会

セキュリティセンター

目 次

1. はじめに.....	1
2. どのツールを利用するか?.....	1
3. STACK SMASHING PROTECTOR.....	2
3.1. 概要.....	2
3.2. SSP のインストール.....	2
3.2.1. SSP のインストール環境.....	2
3.2.2. SSP 対応の egcs 構築.....	3
3.3. SSP による標準 C ライブラリの再構築.....	7
3.3.1. glibc パッケージの入手・インストール.....	7
3.3.2. glibc の spec ファイルへのパッチ適用.....	7
3.3.3. glibc の再構築とインストール.....	8
3.4. SSP によるカーネルの再構築.....	9
3.4.1. カーネルパッケージの入手・インストール.....	9
3.4.2. カーネルの spec ファイルへのパッチ適用.....	11
3.4.3. カーネルの再構築とインストール.....	12
3.5. その他のパッケージの再構築.....	13
4. LIBSAFE.....	14
4.1. 概要.....	14
4.2. LIBSAFE のインストール.....	14
4.2.1. Libsafe のインストール環境.....	14
4.2.2. RPM でのインストール方法.....	15
4.2.3. ソースコードからのインストール方法.....	16
4.3. LIBSAFE の初期設定.....	17
4.3.1. システム全体への適用.....	17
4.3.2. 特定のプログラムへの適用.....	17
4.4. LIBSAFE の動作確認.....	18
4.4.1. バッファオーバーフロー攻撃検出の確認.....	18
4.4.2. フォーマットストリングバグ攻撃検出の確認.....	19
5. まとめ.....	21

参考文献..... 22

1. はじめに

本利用ガイドでは、Stack Smashing Protector と Libsafe という 2 つのツールの利用に関して記述する。適用する対象および、利用者のスキル・状況に応じて使い分けをすることが望まれる。

2. どのツールを利用するか？

セキュアな実行コードの生成・実行環境という観点からセキュリティ対策を実施する際に、どの技術を利用するかは大きな問題である。それぞれの技術に対応したツールがカバーする範囲を正確に把握した上で、適切にツールを適用しなければならない。また利用者の状況（例えば、管理者権限を持っていない、既存の稼動中のマシンに適用したい等）を考慮しなければならない。

本利用ガイドでは、セキュアな実行コードの生成・実行環境技術として、Stack Smashing Protector と Libsafe という 2 つのツールを取り上げ、そのインストール方法から利用方法まで説明する。（これら 2 つのツールを選択した理由については、第部の報告書を参照してほしい。）この 2 つのツールは、カバーする脆弱性の範囲が、我々が評価した中では広く、パフォーマンスも良いため、実際の環境に適用可能であろうと判断した。しかしそれぞれのツールは、保護手法が大きく異なる。例えば、Stack Smashing Protector はコンパイラを拡張するものであり、ソフトウェアを保護するためには、拡張したコンパイラでソフトウェアを再コンパイルする必要がある。それに対して、Libsafe は標準 C ライブラリを拡張したもので、プログラムの実行時に、脆弱性を含む C の関数を安全な関数で動的に置き換える。このようなツールの特性を考慮すると、どちらのツールを適用するか判断が重要であると考え。状況によっては、Stack Smashing Protector のような技術は適用できない。

そのような状況を考慮して、どちらのツールを利用すればいいのかについての意思決定を行うための簡易的なチャートを用意したので参考にしてほしい。

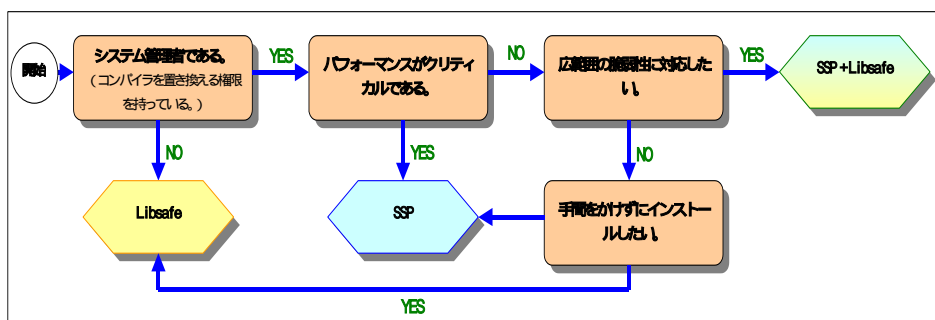


図1. 利用技術選択チャート

3. Stack Smashing Protector

3.1. 概要

Stack Smashing Protector (SSP) はスタックに対するバッファオーバーフロー攻撃を検出・阻止するために、コンパイラにより生成される中間コードを拡張するようにコンパイラを拡張したものである。プログラムのソースコードに変更を加える必要はなく、コンパイル時に意識する必要もない。また実行コードに直接バッファオーバーフロー攻撃の検出コードを挿入するのではなく、中間言語として検出のためのコードを生成するため、さまざまな環境で SSP の技術を利用することが可能である。SSP によって生成されたプログラムは、スタックにおけるリターンアドレスへの不正な書き込みや変数・引数の不正な変更を検出することが可能である。

3.2. SSP のインストール

ここでは、SSP のインストール方法について解説する。SSP は gcc のパッチとして提供されるため、ここでは、gcc に SSP のパッチを適用し、gcc を再構築する方法について解説する。

3.2.1. SSP のインストール環境

本利用ガイドでは、SSP をインストールするにあたって以下の環境を想定する。

- Red Hat Linux 6.2 日本語版¹
- egcs-1.1.2-30p (gcc パッケージ)
- Intel アーキテクチャ

SSP では、現在 egcs-1.1.2-30p 対応のパッチが提供されている。FreeBSD へも対応しているが、ここでは、Red Hat をベースとする²。

¹ Red Hat Linux に関しては 6.2 以降のバージョンでも適用可能であるが、egcs は同一でなければならない。

² 報告書にて Red Hat をベースとした評価実験を行ったため、同様の環境を想定した。FreeBSD のインストールに関しては[1]を参照のこと。

3.2.2. SSP 対応の egcs 構築

SSP のコードは、egcs のパッチとして提供される。そのため、SSP のパッチを egcs に適用し、egcs を再構築する必要がある。egcs を再構築する手順は以下の通りである。

1. egcs パッケージのインストール
2. SSP の入手
3. egcs パッケージ定義ファイル(SPEC ファイル)へのパッチ適用
4. パッケージとして egcs を再構築
5. 再構築した egcs パッケージのインストール
6. 動作確認テスト

1) egcs パッケージのインストール

egcs のソースコードパッケージを以下のサイトから入手する。

```
ftp://ftp.redhat.com/pub/redhat/linux/6.2/ja/os/i386/SRPMS/egcs-1.1.2-30.src.rpm
```

入手したパッケージをインストールする。

```
# rpm -ivh egcs-1.1.2-30.src.rpm
```

通常、以下のディレクトリに egcs のソースコードと SPEC ファイルが格納される。

```
# ls /usr/src/redhat/SOURCES
egcs-1.1.2-addressof.patch  egcs-1.1.2-integrate.patch
egcs-1.1.2-asm.patch       egcs-1.1.2-linux.patch
egcs-1.1.2-cpu.patch       egcs-1.1.2-strlen.patch
egcs-1.1.2-davem.patch     egcs-1.1.2-warn.patch
egcs-1.1.2-expr.patch      egcs-1.1.2-tar.bz2
egcs-1.1.2-fold.patch      egcs-libstdc++-compat.tar.gz
egcs-1.1.2-gcse.patch

# ls /usr/src/redhat/SPECS
egcs.spec
```

2) SSP の入手

SSP 用の SPEC ファイルのパッチを以下のサイトから入手する。

```
http://www.tri.ibm.com/projects/security/ssp/redhat62/egcs.spec.patch
```

入手したパッチファイルを以下のディレクトリに移動する。

```
# mv egcs.spec.patch /usr/src/redhat/SPECS
```

SSP のパッチファイルを以下のサイトから入手する。

```
http://www.tri.ibm.com/projects/security/ssp/redhat62/egcs-1.1.2-protector.patch
```

入手したパッチファイルを以下のディレクトリに移動する。

```
# mv egcs-1.1.2-protector.patch /usr/src/redhat/SOURCES
```

3) egcs パッケージ定義ファイル(SPEC ファイル)へのパッチ適用

/usr/src/redhat/SPECS ディレクトリ下に置いたパッチファイルを、以下のよう
にして、egcs パッケージの定義ファイルに適用する。

```
# cd /usr/src/redhat/SPECS  
# patch -p0 < egcs.spec.patch
```

4) パッケージとして egcs を再構築

パッケージとして egcs を構築する。パッケージを作成するために、以下のよ
うにコマンドを実行する。

```
# rpm -bb --buildpolicy redhat /usr/src/redhat/SPECS/egcs.spec
```

インストールが成功すると以下のようにパッケージが作成される。

```
# ls /usr/src/redhat/RPMS/i386
cpp-1.1.2-30p.i386.rpm      egcs-g77-1.1.2-30p.i386.rpm
egcs-1.1.2-30p.i386.rpm    egcs-objc-1.1.2-30p.i386.rpm
egcs-c++-1.1.2-30p.i386.rpm  libstdc++-2.9.0-30p.i386.rpm
```

5) 再構築した egcs パッケージのインストール

再構築した egcs のパッケージを以下のようにしてインストールする。

```
# rpm -Uvh --force /usr/src/redhat/RPMS/i386/egcs-1.1.2-30p.i386.rpm
```

6) 動作確認テスト

SSP で拡張された gcc の動作確認テストを行う。以下のテストプログラム (test.c) を用いる。

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>

char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";

char large_string[128];
int contains_null_bytes(caddr_t addrp) {
    uint addr = (uint) addrp;
    return !(addr & 0xff &&
             addr & 0xff00 &&
             addr & 0xff0000 &&
             addr & 0xff000000);
}
```

```
void foo() {
    char buffer[96], *p;
    int i;
    long *long_ptr = (long *) large_string;

    printf("Press any key to continue...");
    getchar();

    for (p=buffer; contains_null_bytes(p); p++);
    if (contains_null_bytes(p)) {
        printf("We can't find an acceptable address that doesn't contain¥n");
        printf("a zero byte. Giving up.¥n");
        exit(-1);
    }

    for (i = 0; i < 32; i++)
        *(long_ptr + i) = (int) p;
    for (i = 0; i < sizeof(shellcode)-1; i++) {
        large_string[i] = shellcode[i];
    }
    strcpy(p, large_string);
    return;
}
int main(int ac, char *av[]) {
    foo();
    return 0;
}
```

以下のようにして、コンパイルし、実行する。

```
% gcc -o test test.c
% ./test
```

実行結果として、以下のようにログファイルにアラートが残されていれば、正常に動作しているといえる。

```
% tail -n 1 /var/log/message
Jan 8 14:57:46 hostname test: stack smashing attack in function foo
```

3.3. SSP による標準 C ライブラリの再構築

SSP を導入しても、既存の標準 C ライブラリは SSP で拡張された gcc でコンパイルされていないので安全ではない。C ライブラリ内にエラーが存在するケースもある。そこで、標準 C ライブラリである glibc を再構築し、インストールする必要がある。ここでは glibc の 2.1.3 というバージョンを対象にインストールの方法を説明する。

3.3.1. glibc パッケージの入手・インストール

以下のサイトより glibc のソースコードパッケージを入手する。

```
ftp://ftp.redhat.com/pub/redhat/linux/6.2/ja/os/i386/SRPMS/glibc-2.1.3-15.src.rpm
```

ダウンロードしたパッケージをインストールする。

```
# rpm -ivh glibc-2.1.3-15.src.rpm
```

以下のようなディレクトリ下にファイルがインストールされる。

```
# ls /usr/src/redhat/SOURCES
glibc-2.1.3.tar.gz

# ls /usr/src/redhat/SPECS
glibc-2.1.spec
```

3.3.2. glibc の spec ファイルへのパッチ適用

SSP 用の glibc の SPEC ファイルのパッチを以下のサイトから入手する。

```
http://www.tri.ibm.com/projects/security/ssp/redhat62/glibc-2.1.spec.patch
```

入手したパッチファイルを以下のディレクトリに移動する。

```
# mv glibc-2.1.spec.patch /usr/src/redhat/SPECS
```

/usr/src/redhat/SPECS ディレクトリ下に置いたパッチファイルを、以下のようにして、**glibc** パッケージの定義ファイルに適用する。

```
# cd /usr/src/redhat/SPECS
# patch -p0 < glibc-2.1.spec.patch
```

glibc 用の SSP のパッチファイルを以下のサイトから入手する。

```
http://www.tri.ibm.com/projects/security/ssp/redhat62/glibc-2.1.3-propolice.patch
```

入手したパッチファイルを以下のディレクトリに移動する。

```
# mv glibc-2.1.3-propolice.patch /usr/src/redhat/SOURCES
```

3.3.3. **glibc** の再構築とインストール

パッケージとして **glibc** を構築する。パッケージを作成するために、以下のよう
にコマンドを実行する。

```
# rpm -bb --target i686-redhat-linux /usr/src/Redhat/SPECS/glibc-2.1.spec
```

パッケージの作成が成功すると以下のようにパッケージが作成される。

```
# ls /usr/src/redhat/RPMS/i386
glibc-2.1.3-15p.i386.rpm      glibc-profile-2.1.3-15p.i386.rpm
glibc-devel-2.1.3-15p.i386.rpm  nscd-2.1.3-15p.i386.rpm
```

再構築した **glibc** のパッケージを以下のようにしてインストールする。

```
# rpm -Uvh --force /usr/src/redhat/RPMS/i386/ glibc-2.1.3-15p.i386.rpm
```

3.4. SSP によるカーネルの再構築

SSP を導入しても、カーネルは SSP で拡張された gcc でコンパイルされていないので安全とはいえない。カーネル内にエラーが存在するケースもある。そこで、カーネルを再構築しインストールする必要がある。ここでは Linux カーネルの 2.1.3 というバージョンを対象にインストールの方法を説明する。

3.4.1. カーネルパッケージの入手・インストール

以下のサイトよりカーネルのソースコードパッケージを入手する。

```
ftp://ftp.redhat.com/pub/redhat/linux/6.2/ja/os/i386/SRPMS/kernel-2.2.14-5.0.src.rpm
```

ダウンロードしたパッケージをインストールする。

```
# rpm -ivh kernel-2.2.14-5.0.src.rpm
```

以下のようなディレクトリ下にファイルがインストールされる。

```
# ls /usr/src/redhat/SOURCES
README.kernel-sources          linux-2.2.14-ide-cd-shutup.patch
ibcs-2.1-981105.tar.gz         linux-2.2.14-ide-probe.patch
ibcs-2.1-locking.patch         linux-2.2.14-iobuffix.patch
ibcs-2.1-rh.patch              linux-2.2.14-ipvs-template.patch
installkernel                  linux-2.2.14-joyfix.patch
ipvs-0.9.7-2.2.13.patch        linux-2.2.14-lfs-headers.patch
kernel-2.2-BuildASM.sh         linux-2.2.14-lfs.patch
kernel-2.2.14-alpha-BOOT.config linux-2.2.14-loop.patch
kernel-2.2.14-alpha-smp.config linux-2.2.14-lucent-hang.patch
kernel-2.2.14-alpha.config     linux-2.2.14-mediaGX.patch
kernel-2.2.14-i386-BOOT.config linux-2.2.14-megaraid.patch
kernel-2.2.14-i386-smp.config  linux-2.2.14-moremaestro.patch
kernel-2.2.14-i386.config      linux-2.2.14-msdos-fixup.patch
kernel-2.2.14-i586-smp.config  linux-2.2.14-nautilus-srm.patch
kernel-2.2.14-i586.config      linux-2.2.14-network-fixes.patch
kernel-2.2.14-i686-smp.config  linux-2.2.14-newagpdist.patch
```

kernel-2.2.14-i686.config	linux-2.2.14-nfs-fix.patch
kernel-2.2.14-propolice.patch	linux-2.2.14-nfsattack2.patch
kernel-2.2.14-sparc-BOOT.config	linux-2.2.14-nobfddep.patch
kernel-2.2.14-sparc-smp.config	linux-2.2.14-oom-hang.patch
kernel-2.2.14-sparc.config	linux-2.2.14-plip-fix.patch
kernel-2.2.14-sparc64-BOOT.config	linux-2.2.14-psi-update.patch
kernel-2.2.14-sparc64-smp.config	linux-2.2.14-rpc.patch
kernel-2.2.14-sparc64.config	linux-2.2.14-scsi-blacklist.patch
ksymoops-0.7c.tar.gz	linux-2.2.14-scsi-devs.patch
linux-2.2.12-3c90x.patch	linux-2.2.14-security-a1.patch
linux-2.2.12-PIII-xor.patch	linux-2.2.14-security-a2.patch
linux-2.2.12-PIII.patch	linux-2.2.14-security-a3.patch
linux-2.2.12-bigmem-initrd.patch	linux-2.2.14-security-a4.patch
linux-2.2.12-bigmem-raw.patch	linux-2.2.14-shmem-overwrite.patch
linux-2.2.12-cpq-mdh.patch	linux-2.2.14-sigio.patch
linux-2.2.12-ipvsvfix.patch	linux-2.2.14-sigkill.patch
linux-2.2.12-limits.patch	linux-2.2.14-sk98-fix.patch
linux-2.2.12-peerbus.patch	linux-2.2.14-sound-update.patch
linux-2.2.12-symversion.patch	linux-2.2.14-sparc-config.patch
linux-2.2.13-IOAPIC.patch	linux-2.2.14-sparc-cpu-bug.patch
linux-2.2.13-aic7xxx-5.1.22.patch	linux-2.2.14-sparc-cpu-bug2.patch
linux-2.2.13-aic7xxx-5.1.23.patch	linux-2.2.14-sparc-cpu-bug3.patch
linux-2.2.13-aic7xxx-5.1.24.patch	linux-2.2.14-sparc-fixes.patch
linux-2.2.13-aic7xxx-5.1.25.patch	linux-2.2.14-sparc-lockd.patch
linux-2.2.13-aic7xxx-5.1.26.patch	linux-2.2.14-sparc-mmap.patch
linux-2.2.13-aic7xxx-5.1.27.patch	linux-2.2.14-sparc-nfs.patch
linux-2.2.13-alphamsnd.patch	linux-2.2.14-sparc-raid.patch
linux-2.2.13-bigmem-dcache.patch	linux-2.2.14-sparc-syscall.patch
linux-2.2.13-bigmem-no-lfs.patch	linux-2.2.14-sparcacenic.patch
linux-2.2.13-bigmem.patch	linux-2.2.14-sparcswift.patch
linux-2.2.13-smart2-1.0.6.patch	linux-2.2.14-sunpartshaddap.patch
linux-2.2.14-82596-crash.patch	linux-2.2.14-sunqe.patch
linux-2.2.14-MegaRAID.patch	linux-2.2.14-timersync.patch
linux-2.2.14-acenic041.patch	linux-2.2.14.tar.gz
linux-2.2.14-agphjlfixes.patch	linux-autoconf.h

```

linux-2.2.14-aic7xxx-5.1.28.patch  linux-modversions.h
linux-2.2.14-alpha-exception.patch linux-version.h
linux-2.2.14-alpha-ramdisk.patch  module-info
linux-2.2.14-alphasym.patch
linux-2.2.14-blkdev.patch          pcmcia-cs-2.8.8-network.script
linux-2.2.14-bonding.patch         pcmcia-cs-3.1.3-3com.patch
linux-2.2.14-cyclades-smp.patch   pcmcia-cs-3.1.4-xircom.patch
linux-2.2.14-duh.patch            pcmcia-cs-3.1.8-config.patch
linux-2.2.14-eeepro100.patch      pcmcia-cs-3.1.8-script.patch
linux-2.2.14-eeepro7.patch        pcmcia-cs-3.1.8.tar.gz
linux-2.2.14-elf-loader.patch     raid-2.2.14-B1.gz
linux-2.2.14-emu10k1.patch        raw-2.2.13-rh61.diff
linux-2.2.14-fb-modules.patch     rhkmvtag.c
linux-2.2.14-i386-asm.patch

# ls /usr/src/redhat/SPECS
kernel-2.2.14.spec

```

3.4.2. カーネルの spec ファイルへのパッチ適用

SSP 用のカーネルの SPEC ファイルのパッチを以下のサイトから入手する。

```
http://www.trl.ibm.com/projects/security/ssp/redhat62/kernel-2.2.14.spec.patch
```

入手したパッチファイルを以下のディレクトリに移動する。

```
# mv kernel-2.2.14.spec.patch /usr/src/redhat/SPECS
```

/usr/src/redhat/SPECS ディレクトリ下に置いたパッチファイルを、以下のよう
にして、カーネルパッケージの定義ファイルに適用する。

```
# cd /usr/src/redhat/SPECS  
# patch -p0 < kernel-2.2.14.spec.patch
```

カーネル用の SSP のパッチファイルを以下のサイトから入手する。

```
http://www.tri.ibm.com/projects/security/ssp/redhat62/kernel-2.2.14-propolice.patch
```

入手したパッチファイルを以下のディレクトリに移動する。

```
# mv kernel-2.2.14-propolice.patch /usr/src/redhat/SOURCES
```

3.4.3. カーネルの再構築とインストール

パッケージとしてカーネルを構築する。パッケージを作成するために、以下
のようにコマンドを実行する。

```
# rpm -bb --target i686-redhat-linux /usr/src/Redhat/SPECS/kernel-2.2.14.spec
```

パッケージの作成が成功すると以下のようにパッケージが作成される。

```
# ls /usr/src/redhat/RPMS/i386  
kernel-headers-2.2.14-5.0p.i386.rpm  
kernel-ibcs-2.2.14-5.0p.i386.rpm  
kernel-pcmcia-cs-2.2.14-5.0p.i386.rpm  
kernel-smp-2.2.14-5.0p.i386.rpm  
kernel-source-2.2.14-5.0p.i386.rpm  
kernel-2.2.14-5.0p.i386.rpm      kernel-utils-2.2.14-5.0p.i386.rpm  
kernel-BOOT-2.2.14-5.0p.i386.rpm  kernel-doc-2.2.14-5.0p.i386.rpm
```

構築したカーネルパッケージを以下のようにしてインストールする。

```
# rpm -Uvh --force /usr/src/redhat/RPMS/i386/ kernel-2.2.14-5.0p.i386.rpm
```

3.5. その他のパッケージの再構築

その他のパッケージに関しては、以下のコマンドでパッケージを再構築する。

```
# rpm --rebuild --buildpolicy redhat src_package_name
```



【 注意 】

以下のパッケージについては SSP で拡張された gcc で再構築しないように注意する。これらは、SSP で再構築すると問題のあるパッケージである。

- mkisofs-1.8-2.src.rpm
- popt-1.4-1.src.rpm
- usernet-1.0.9-2.src.rpm
- xpilot-4.1.0-1.src.rpm

4. Libsafe

4.1. 概要

Libsafe はバッファオーバーフロー攻撃を検出し、エラーを防ぐツールである。従来の方法と比較して、ソースコードを要求せず、すでにコンパイルされたプログラムに適用できる点が異なる。基本的にはバッファオーバーフロー攻撃に対してプロセスを安全に保護することができる。こうしたセキュリティツールは他にあまりなく非常に有用なツールである。Libsafe では、脆弱性があると知られている C 言語の標準ライブラリ関数の呼び出しを横取りし、安全な関数と置き換える。横取りされる関数の代用としての関数はオリジナルの機能を有しているが、バッファオーバーフローを制御する機能を併せ持つ。バッファオーバーフロー攻撃のほか、一部のフォーマット・ストリング・バグ攻撃にも対応可能である。

4.2. Libsafe のインストール

ここでは、Libsafe のインストール方法について解説する。Libsafe は動的にロード可能なライブラリとして提供される。そのため、SSP のようにパッチの形式ではなく、独立したライブラリの形式で提供される。

4.2.1. Libsafe のインストール環境

本利用ガイドでは、Libsafe をインストールするにあたって以下の環境を想定する。

- ・ Red Hat Linux 6.2 日本語版³
- ・ Intel アーキテクチャ

Libsafe の最新バージョンは 2.0-16 である。

³ 6.2 以上のバージョンであればインストール可能である。

4.2.2. RPM でのインストール方法

Libsafe では RPM が提供されている。ここでは、バイナリーパッケージでのインストール方法について説明する。

Libsafe のバイナリーパッケージを以下のサイトから入手する。

```
http://www.research.avayalabs.com/project/libsafe/src/libsafe-2.0-16.i386.rpm
```

入手したパッケージをインストールする。

```
# rpm -ivh libsafe-2.0-16.i386.rpm
```

通常、以下のようなライブラリとドキュメント、ツールがインストールされる。

```
/lib/libsafe.so.2.0.16  
/usr/doc/libsafe-2.0  
/usr/doc/libsafe-2.0/COPYING  
/usr/doc/libsafe-2.0/ChangeLog  
/usr/doc/libsafe-2.0/EMAIL_NOTIFICATION  
/usr/doc/libsafe-2.0/INSTALL  
/usr/doc/libsafe-2.0/LIBPRELUDE  
/usr/doc/libsafe-2.0/README  
/usr/doc/libsafe-2.0/doc/*  
/usr/doc/libsafe-2.0/exploits/*  
/usr/doc/libsafe-2.0/tools/*
```

`/lib` ディレクトリ以下に格納されるのが Libsafe の本体である。
`/usr/doc/libsafe-2.0/exploit` 以下にはテスト用のプログラムがインストールされる。
`/usr/doc/libsafe-2.0/tools` 以下には Libsafe を制御するためのユーティリティがインストールされる。



【 注意 】

RPM でバイナリーパッケージをインストールした場合は、インストール直後から Libsafe が利用可能となる。システム全体に対して、Libsafe を適用したい

場合は、デフォルトのままで良いが、もし特定のアプリケーションのみに Libsafe を適用したい場合は、以下のようなコマンドで、Libsafe をシステム全体に提供することを解除する必要がある。

```
# /usr/doc/libsafe-2.0/tools/libsafe-install.sh -r
```

4.2.3. ソースコードからのインストール方法

Libsafe ではソースコードが提供されている。ここでは、ソースコードからのインストール方法について説明する。オプションを変更してアクションをカスタマイズしたい場合は、ソースコードからコンパイルする必要がある。

Libsafe のソースコードを以下のサイトから入手する。

```
http://www.research.avayalabs.com/project/libsafe/src/libsafe-2.0-16.tgz
```

以下のように、入手したソースコードを展開する。

```
% tar -zxvf libsafe-2.0-16.tgz
```

以下のように、コンパイルし、インストールする。

```
% cd libsafe-2.0-16
% make
% su root
# make install
```

インストール時に以下のように、Libsafe をシステム全体で利用するかどうか確認してくるので、システム全体で利用するのであれば、[y]を、そうでなければ[n]を選択する。

```
Type y for installing libsafe system wide?[default n]
```

通常、Libsafe の本体は/lib 以下に格納される。その他に man がインストールされるが、RPM でインストールした場合のように exploits や tools はインストールされない。

4.3. Libsafe の初期設定

Libsafe はシステム全体に適用することも可能であるし、個別のアプリケーションにのみ適用することも可能である。ここではそれぞれの方法について説明する。

4.3.1. システム全体への適用

システム全体に対して適用する場合は、`libsafe-install.sh` コマンドを利用する。RPM でのインストールの場合は、`/usr/doc/libsafe-2.0/tools` ディレクトリ以下にコマンドが存在する。ソースコードからインストールした場合には、ソースコードディレクトリ以下の `tools` ディレクトリ以下に存在する。

```
# libsafe-install.sh -i
```

システム全体に適用したい場合は、`-i` オプションを利用する。逆に適用をはずしたい場合は、`-r` オプションを利用する。

4.3.2. 特定のプログラムへの適用

特定のプログラムに適用したい場合は、以下のように環境変数 `LD_PRELOAD` の設定を行う。

```
sh の場合
% LD_PRELOAD=/lib/libsafe.so.2
% export LD_PRELOAD

csh の場合
% setenv LD_PRELOAD /lib/libsafe.so.2
```

上記のように `LD_PRELOAD` 環境変数が設定された環境下で実行されたプログラムは、Libsafe が適用される。

【 注意 】

特定のプログラムに適用する場合、`setuid` されたプログラムでは、`LD_PRELOAD` 環境変数を無視する。そのため、`setuid` されたプログラムに適用したい場合には、必ずシステム全体に `Libsafe` を適用する必要がある。

`Libsafe` では `-fomit-frame-pointer` を指定してコンパイルされたプログラムには適用することができないので注意する必要がある。

4.4. Libsafe の動作確認

`Libsafe` には、動作を確認するためのツールが提供されている。これらのツールを利用して、動作の確認をおこなうことができる。

4.4.1. バッファオーバーフロー攻撃検出の確認

ここでは、`Libsafe` が対応する攻撃の一つである、バッファオーバーフロー攻撃の検出について確認する。

`Libsafe` を有効にして、以下のツールを実行する。本ツールは、RPM の場合は、`/usr/doc/libsafe-2.0/exploits` ディレクトリ以下に、ソースコードからのインストールの場合は、ソースコードディレクトリ以下の `exploits` ディレクトリ下にある。

```
% t1
```

このツールは `strcpy` を使ってバッファをオーバーフローさせ、`sh` を起動するコードを不正に実行させるものである。

実行後に以下のようなメッセージが表示されていれば、`Libsafe` が正常にバッファオーバーフローを検出したことが確認できる。

```

Telnet 10.01.10.145
[root@rdlinux exploits]# ./t1
This program tries to use strcpy() to overflow the buffer.
If you get a /bin/sh prompt, then the exploit has worked.
Press any key to continue...
Libsafe version 2.0.16
Detected an attempt to write across stack boundary.
Terminating /usr/doc/libsafe-2.0/exploits/t1.
  uid=0  euid=0  pid=19253
Call stack:
  0x40017a91 /lib/libsafe.so.2.0.16
  0x40017b8a /lib/libsafe.so.2.0.16
  0x80485bc  /usr/doc/libsafe-2.0/exploits/t1
  0x80485d3  /usr/doc/libsafe-2.0/exploits/t1
  0x400389c6 /lib/libc-2.1.3.so
Overflow caused by strcpy()
Killed
[root@rdlinux exploits]#

```

4.4.2. フォーマット・ストリング・バグ攻撃検出の確認

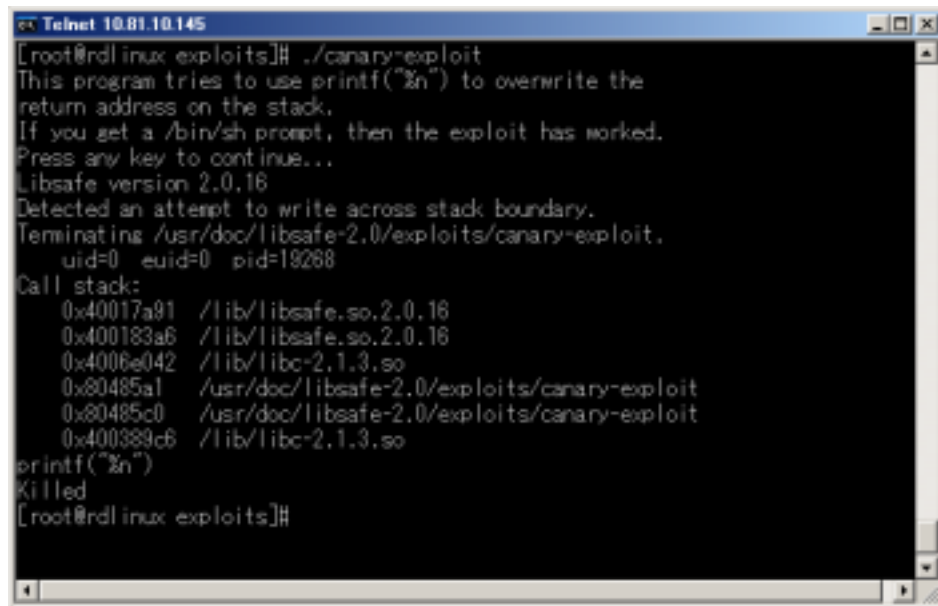
ここでは、Libsafe が対応する攻撃の一つである、フォーマット・ストリング・バグ攻撃の検出について確認する。

Libsafe を有効にして、以下のツールを実行する。本ツールは、RPM の場合は、`/usr/doc/libsafe-2.0/exploits` ディレクトリ以下に、ソースコードからのインストールの場合は、ソースコードディレクトリ以下の `exploits` ディレクトリ下にある。

```
% canary-exploit
```

このツールは `fprintf` を使ってバッファをオーバーフローさせ、`sh` を起動するコードを不正に実行させるものである。

実行後に以下のようなメッセージが表示されていれば、Libsafe が正常にフォーマット・ストリング・バグ攻撃を検出したことが確認できる。



```
cx Telnet 10.81.10.145
[root@rdlinux exploits]# ./canary-exploit
This program tries to use printf("%n") to overwrite the
return address on the stack.
If you get a /bin/sh prompt, then the exploit has worked.
Press any key to continue...
Libsafe version 2.0.16
Detected an attempt to write across stack boundary.
Terminating /usr/doc/libsafe-2.0/exploits/canary-exploit.
  uid=0  euid=0  pid=19268
Call stack:
  0x40017a91  /lib/libsafe.so.2.0.16
  0x400183a6  /lib/libsafe.so.2.0.16
  0x4006e042  /lib/libc-2.1.3.so
  0x80485a1   /usr/doc/libsafe-2.0/exploits/canary-exploit
  0x80485c0   /usr/doc/libsafe-2.0/exploits/canary-exploit
  0x400389c6  /lib/libc-2.1.3.so
printf("%n")
Killed
[root@rdlinux exploits]#
```

5. まとめ

本利用ガイドでは、Stack Smashing Protector と Libsafe の2つのツールを取り上げ、そのインストール方法・利用方法について解説した。またチャートを用いて、こういった状況下でどちらのツールを利用すればいいかについて示した。

これらのツールを利用するにあたっての注意は、これらのツールが完全にすべての脆弱性をカバーできるわけではないということである。攻撃手法の多様化によっても当然これらのツールの効果は低下する。またこれらのツールは攻撃を防御するが、プログラムは停止してしまうため、DoS 攻撃などに対しては無力である。そのため、ソースコードレベルで脆弱性をなくしていくことが非常に重要となる。利用者の立場からは難しい問題であるが、極力ソースコードレベルでの検査を実施し、ソースコードレベルでのリスクの把握をしていただきたい。

利用状況を十分考慮し、これらのツールで少しでもシステムの安全性を高めて行っていただきたい。

参考文献

- [1] How to build RedHat Linux with stack protection,
<http://www.trl.ibm.com/projects/security/ssp/buildredhat.html>
- [2] Manpage of Libsafe,
<http://www.research.avayalabs.com/project/libsafe/doc/libsafe.8.html>
- [3] 若島茂紀: セキュリティ実験室「プログラムバグを利用したクラッキング行為を防ぐ」,
LinuxWORLD, 2002 Jun.