システムコール引数の暗号化によるコード注入攻撃の防止

大山 恵弘　　米澤 明憲
東京大学 大学院情報理工学系研究科 コンピュータ科学専攻
{oyama,yonezawa}@yl.is.s.u-tokyo.ac.jp

1　背景: コード注入攻撃

本研究が扱うテーマは、計算機を乗っ取るタイプのウィルスや攻撃コードの多くが実行する、コード注入攻撃と呼ばれる攻撃の防止である。コード注入攻撃は、プログラム中のバッファなどのメモリ領域に悪意のコードを注入し、さらに、バッファをオーバフローさせることなどによりプログラムの制御をそのコードに移動させる。注入されたコードは資源を不正に操作したり外部プログラムを起動するなどの形で計算機システムに被害を与える。たとえば、UNIX 上のプログラムに注入したコードに `execve("/bin/sh", ...)` というシステムコールを呼び出させれば、攻撃者はその計算機上でシェルを利用できる。Linux/x86 用の注入されるコードの例を以下に示す。

```
unsigned char exploit[]=
...
/* システムコール番号をセット */
"\xb8\x1b\x11\x11\x11" /* movl $0x1111111b, %eax */
"\x2d\x10\x11\x11\x11" /* subl $0x11111110, %eax */
/* システムコール引数をセット */
"\x8b\x5d\x08" /* movl 0x8(%ebp), %ebx */
"\x8b\x4d\x0c" /* movl 0xc(%ebp), %ecx */
"\x8b\x55\x10" /* movl 0x10(%ebp), %edx */
/* ソフトウェア割り込み */
"\xcd\x80"; /* int $0x80 */
```

このコードは `execve` のシステムコール番号である 11 をレジスタ eax にセットし、システムコール引数を ebx、ecx、edx にセットした後、システムコール呼び出しを示すソフトウェア割り込みを発生させている。

2　提案手法

本研究では、システムコール番号およびシステムコール引数を暗号化・ランダム化することによりコード注入攻撃を防止する手法を提案する。提案手法に基づくシステムは、カーネルモジュールと動的リンクされる標準ライブラリの改造版の二つからなる。改造標準ライブラリは、カーネルに制御を移す直前（すなわちソフトウェア割り込みを発生させる直前）にシステムコールの番号と引数を暗号化する。カーネルモジュールはシステムコールをフックし、改造標準ライブラリとの間で共有している鍵でシステムコールの番号と引数を復号する。復号した結果の番号を用いて元々のシステムコール表を引き、システムコールハンドラ関数を呼び出す。暗号化には RC4 などのストリーム暗号を利用する[1]。暗号化の鍵はプロセスごとに異なるものが生成される。

本手法を利用すると、システムコール呼び出し時（ソフトウェア割り込み時）にレジスタおよびメモリが持つ値と、カーネルに渡されるシステムコール番号および引数との間の対応付けが、システムコール呼び出しごとに変わる。その結果、攻撃者が望みのシステムコール呼び出しを実行することが困難になる。攻撃者が `execve("/bin/sh", ...)` を実行しようとしても、復号の結果 `execve` のシステムコール番号になる値や `"/bin/sh"` へのポインタになる値を得ることが難しい。

暗号化の鍵はプロセスの fork 時にカーネルモジュールによって作成される。鍵のデータは、改造標準ライブラリが静的に確保している領域に格納する。ユーザプログラムが `fork` システムコールを呼び出すと、改造標準ライブラリを経由し、カーネルに対しては、

```
pid_t cfork(char *buf);
```

という特殊システムコールが呼び出される。改造標準ライブラリは鍵を格納するための領域へのポインタを `buf` として `cfork` に与える。`cfork` システムコールは `fork` システムコールと動作は同じであるが、鍵を生成して、引数に受け取った領域に書き込む点のみが異な

[1] 提案手法では、システムコールの「正しい」番号と引数を予測しにくくできれば十分なので、暗号には大きな強度は必要ない。

る。カーネルモジュールは、プロセス番号と暗号化の鍵との対応付けを表で管理している。

3 関連研究

文献 [2] は、ランダム性の導入により攻撃を防ぐための多くの手法を提案している。その中には、システムコール表を乱数でかき混ぜて、システムコール番号とシステムコールハンドラ関数の対応付けを変える手法がある。カーネルの再コンパイルによってランダム化されたシステムコール表を実現し、プログラムをそれに合わせてバイナリ変換している。全プロセスが同じ対応付けを共有し、それは再コンパイル以外のタイミングでは変化しない。よって、悪意の者は、正しいプログラムを解析することにより、ランダム化されたシステムコール表の内容を予測できる。本研究の手法では、プロセスごとに、また、呼び出しごとに対応付けが変わるので、攻撃がより困難になる。

StackGuard [4] では、スタック上の制御情報を乱数と XOR する形で保存することにより、攻撃者が望みの制御情報をスタック上に作ることを難しくする。本研究はスタック上の情報ではなくシステムコールの番号と引数を対象としている。

文献 [1, 3, 5] の研究は命令セットをランダム化して攻撃を防ぐ手法を提案している。これらの手法ではプログラムが仮想機械上で実行されるため、本研究の手法よりもオーバヘッドが大きい。

4 議論

提案手法には以下の制限がある。

- 注入されたコードが標準ライブラリに制御を移すような攻撃（いわゆる return-into-libc 攻撃など）を防止することはできない。本研究の対象は、注入されたコードが自らソフトウェア割り込みを発生させてシステムコールを呼び出すような攻撃である。
- 改造されていない標準ライブラリを静的にリンクしたプログラムの実行では、システムコールの番号と引数は暗号化されない。
- 攻撃者が鍵を盗むことは（面倒ではあるが）可能である。バッファオーバフローなどの形で制御を乗っ取った時点で、攻撃者はプログラムの全メモリイメージを読む権限を得る。改造標準ライブラリが鍵を格納するために確保している領域を探し出し、そこを読み出せば、鍵を得られる。
- システムコールフックを利用するシステム（デバッガ、トレーサなど）の動作が異常になる。

今後、システムの実装を完了させ、暗号化のオーバヘッドを測定する予定である。また、現実の攻撃コードを収集し、提案手法による防御が成功する攻撃コードの割合についても調査したい。

参考文献

[1] E. G. Barrantes, D. H. Ackley, S. Forrest, T. S. Palmer, D. Stefanović, and D. D. Zovi. Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pp. 281–289, Washington D.C., October 2003.

[2] M. Chew and D. Song. Mitigating Buffer Overflows by Operating System Randomization. Technical Report CMU-CS-02-197, Carnegie Mellon University, December 2002.

[3] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pp. 272–280, Washington D.C., October 2003.

[4] P. Wagle and C. Cowan. StackGuard: Simple Stack Smash Protection for GCC. In *Proceedings of the GCC Developers Summit*, pp. 243–255, Ottawa, Canada, May 2003.

[5] 大澤. 動的命令カスタマイズに基づくセキュリティバグに対する適応的保護機構. 第 3 回コンピュータセキュリティシンポジウム 2000 (CSS 2000), pp. 243–248, 東京, October 2003.