INSTITUTE FOR SECURITY AND OPEN METHODOLOGIES

SPSMM

セキュアプログラミング標準化手法マニュアル

Created by Victor A. Rodriguez

CURRENT VERSION:	v0.5.1
NOTES:	Japanese Translation by Ikeda Masakazu mikeda<at>attglobal.net
CHANGES:	-
DATE OF CURRENT VERSION:	May 2002
DATE OF ORIGINAL VERSION:	-

セキュアプログラミング標準化手法マニュアル Victor A. Rodriguez (Bit-Man) 著

インターフェースが暴露されたとき、（プログラミング、ユーザーなどが）利用又は悪用することができる。（コインのどちらにいるかによるが）悪用されることは避けるべきである。インターフェースがきれいでかつ有用かつ悪用されないコンポーネントをプログラマやシステム設計者、チームリーダーなどと言った人が望んでいる。そのため、プログラミングサイドからセキュアなプログラミング手法を標準化することを計画した。

インターネットアプリケーションはより複雑になり、コード生成はWizardやプログラム環境、フレームワーク等を使用することで自動化されてきているが、別の問題が発生している。セキュアなプログラミングテクニックは多くの型や多くの言語に依存するものである。プログラミング言語、アプリケーション環境および開発ツールによらず単一の方法でこれに対処することである。

オープンスタンダードを作成しようとするアイデイアはすべての人に恩恵をもたらすだけでなく、クラッカーにもプロセスの中身を知らせることになる。しかしながら、手法に従った場合、弱点は既に解決されているだろう。別の視点では、より多くのプログラマがこの文書を読み、理解することで、安全なアプリケーション開発を行うための対策を講じることができるし、この手法が進歩することになる。

謝辞

多くの感銘を受けた「Secure Programing for Linux and Unix」[1]を書いたDavid A. Wheelerに賛辞を送りたい。また、「SECPROG@SevurityFocus.com working document」[9]、セキュアプログラミングについての一種の「My First Dictionary」を記述した多くの人に感謝する。

このマニュアルに寄付してくれた人々はこのドキュメントの一番上にリストした。みんながどんなことに寄与してくれたかは認識してません。偏見を防ぐためにこのようにしてある。ただし、このマニュアルの翻訳においてのみ翻訳者はドキュメントを最新に保つのみならずその言語でのＦＡＱや提案やコメントを適切なときにオリジナルの文書のために英語に翻訳するため翻訳者と翻訳された文書の接点として書き留められる。

用語

- DoS（Denial of Service）：システムへのアクセスを行わせないようにする攻撃である。システムにあるデータには影響を与えない。
- out of the box：標準、工場出荷時のインストール状態である。
- super user：システムで制限の無い特別なユーザーである（rootやadministratorとして一般には言われている）。

対象者

プログラミング言語、アプリケーション環境および開発ツールに依存しないあらゆるITプロジェクトにおけるガイドラインとセキュアなプログラミング手法を確立したいプログラマのためにこのマニュアルは記述されている。

このマニュアルは特定の開発ツールを使用する上での正しい方法や、特定の言語のプログラム手法でのテストを行っていない。

よりよいプログラム開発で有用なことやインターネットでパフォーマンスを要求されると言った対立する事象を実装するのにこのマニュアルは開発者にとって有用となるだろう。

適用範囲

手動、自動にかかわらずセキュリティ検査を実施したときにインターフェースの使用方法がセキュリティ要件を満たしていることと、不正利用されないようになっていることが、セキュアプログラミング手法の基準を作る最終目的である。別の目的はプログラミング言語、アプリケーション、開発ツールによらない全てのセキュリティ検査において中心的な役割を果たす教育方法を作ることである。

例外

このマニュアルでは以下のものは除外する。

- セキュリティ管理を&#© 2000 - 20037;目的とするプログラム
- 管理者権限で起動する必要があるプログラム

攻撃を受ける環境で実行する準備

この準備は容易な作業ではないし、既に別の方法、その視点をカバーするいくつかの方法、別の出発点等で行われてきた。私たちは、この仕事での一つの方法を述べる。

Secure Programing for Linux and Unix HOWTO（セクション2.2 Security principles）[1]を見るとセキュアプログラミングの達成すべきゴールとして機密性、完全性、

可用性であるということを解釈するかもしれない。それは正しいことであり、特定の環境に応じて変更されたり、異なる可能性もあるが、よい答えであり、一般的な答えである。ここで、これらをどのようにして行うかと言うことからはじめる。我々が強調したいことは本来ならば決まらないが、手法やプロシジャ開発の基盤を提供するオープンな手法である。

システム・タスクは、適切に一致しておりテストすることができる機能的な必要条件を実行することである。より詳細なアプローチでは、コーディングは（サブルーチンやプロシジャレベルのプログラミングといった）アトミックスケールにおけるこの目的を達するために行う。この機能要件にはシステムの可用性、つまり使用可能な日（営業日、7日等）、使用可能な時間帯（AM9:00からPM5:00まで等）、システムが使用できない時間（月に2時間まで等）、サービスレベル許諾（SLA）と呼ばれるものを満たす状態にしておくことは含まれない。

この点においてインターネットのような危険な環境で実行する際には、SLAレベルを満たすこと（Integrity)、情報漏洩が不可能であること（security）、または適切な所有者に情報を提供すること（confidentiality）が特に重要である。

一度これら及びその否定的な効果について知ることは解決するシステマティックなアプローチを採用する方法に大きな効果がある。まず最初にこれらの問題を生じさせる要因を見てみる。

行っては行けないこと：

「行ってはいけないこと」について考えることは非常に困難である。正確に知らないことを守ることは非常に大変な作業となる（これの非常によい例が1999年11月のCrypto-gram ニュースレター[2]にある）。これは細菌からの防御によく似ている。もしそれらが存在することや細菌に感染しないようにする手段が分らなければそれらを防ぐことは出来ない。

複雑さ：

これによって、人間の最も古く、困難な問題の一つがもたらされる。それはコミュニケーションと呼ばれる。また、複雑さがシステムに加わることで、全体のタスクが異なる構成員またはチームが所属するサブタスクに分割される場合、目標に到達するためにそれらの間でコミュニケーションが重要となってくる。存在するチーム間でより多くのコミュニケーションリンクが作られるだろうし、知っているように、システムの複雑さが増すと指数関数的に増加する（「銀の弾&#© 2000 -
20034;がない」[3]ことはソフトウェアを根本的に難しく見せる）。これにより、「見るべきことがあまりにも多すぎる」症候群によって隠された問題が存在してしまう。

方法論

この論文は「The Open-Source Security Testing Methodology Manual（略してOSSTMM）」の一部なので、the Parameters and Tasks oriented to Secure Programmingと若干目的が異なるが、同じ手法を使用している。OSSTMMでは、私たちは、欠陥があることを発見した。また、今、私たちは、コード化する際に、正確に、欠陥がどこにあるのか、欠陥を修正し、それを回避する方法を知りたい。

このセクションでは第一、第二の目的（発見と修正）について述べます。回避方法については「Secure Programming reinforcing」セクション（執筆中ですが）で述べる。

入力データ

インターフェースはいくつかの実装技術に依存していること、完全な問題解決が得るために実装上のセキュリティの脆弱性を修正すべきだということを注意しておかなければならない。テスト段階でプログラムのインターフェース実装を修正するだけでは十分ではない。

留意しておくべきことは入力データはユーザーまたは他のプログラムから受け取るだけでなく、システムから受け取ったり（日付や環境変数等）、同一のシステムの他のソフトウエアから受け取ったりすることである。要するに、データは全て解析されたプログラム・システムに与えられるということである。

入力データの制限確認

一度入力データが正しいパターンとマッチしたからといって、すべてが正しいとは限らない。例えば、アカウントＩＤが数値の場合、すべての組み合わせが有効であるとは限らないし、すべての桁が利用可能だとも限らない。これらの制約は変数の実世界的な意味とは関係がないが、セキュリティ上の問題である変数の長さ(実装技術（言語、OS等）に強く依存し、その振る舞い（一つの一般的なテストは、高々数バイトであると思われるパラメータを持つ関数にキロバイトオーダーのパラメータで関数を呼び出すことである）に注意する事を言っている）には関係することに注意すべきである。

予期された結果：

- システムが期待しない動きをすることと、正しい入力

実行すべきタスク：
システムモニタリング

- ＣＰＵ（使用率、時間、ピーク等）
- メモリ（実メモリ、仮想メモリ、キャッシュ等）

- I/O（ディスクスペース、バーストアクセス等）
- ネットワーク（パケットの取りこぼし、生成等）

コードの対策

- 入力とその許可を扱うコードの特定
- コードの修正

このトピックでは、返されるデータではなくシステムの振る舞いが重要なので、analysisとcode
workingは一種の芸術である。これには使用している技術、つまりメモリ管理、プロセス / スレッドの生成、ロック、IPC等といった言語の実装基盤となっているものの幅広い知識が必要である。

このテストで発見される例の一つが「付録A」の「Format String
バグ」である。データが正しくチェックされることで制限され、整えられていれば、その影響は最小限にすることができる。

プロセス

一度外部からの保護がなされたなら、システムコードと内部プロセスを保護しなければならない。

プロセスをどのように保護するかを分析することは基本的にシステムに使用されるすべてのリソースを保護すること（メモリ、ディスク、ネットワーク等をそれ自身や他のものによる誤用から防ぐこと）である。

リソース消費

すべてのプログラムはそれ自身の目的（計算にCPU時間を使用したり、内部記憶やキャッシュにメモリを使用したり、様々な処理にファイルを使用する等）のためにリソースを消費する。ある条件下でシステムに負荷をかけたりプログラムを終了させるようなプログラムを作ることができる。これらはサービス不能、または一般にその頭文字をとってDoSと呼ばれる。

一般にこれは容認されたデータのみ処理されるが、拒否されたものでも作用することで正しいプラクティスであり、（dirty testとして知られる）後者の変形である。

予期された結果

- リソース消費と入力データと状態の使用法（implemented state machineを参照）。

実行すべきタスク
システムモニタリング:個々のリソースクラス（CPU、メモリなど）

- 適切なリソース確保
- 適切なリソースの使用法
- 適切なリソースの解放
- デッドロックモニタリング／分析

コードの対策

- DoSの原因となるコードの特定
- コードの修正

そのようなDoSの条件を回避するにはリソースの使用量を制限し、指定されたレベルまでリソースを使用した場合にリソース使用の要求を拒否することである。

このテストで発見される例の一つが「付録A」の「resource retention」である。

入力データのパターン確認

入力データが正しいまたは正しくない様々なデータパターンをテストする。これは利用可能なリソース（時間、予算）と人に依存するので異なった画面、スクリプト、URL、インターフェースタイプ等の中に分離されるかもしれない。ユーザーに提供されているものだけでなく、すべての利用可能なインターフェース（GUI、API等）をテストすることが望ましい。

この点においてテスターは正しいデータパターンに一致しないだけでなく、プログラムに基づく（OS、言語、データベース等）いくつかの隠されたり不明瞭なアーキテクチャの要素を利用しようとするデータセットを入念に作る方法に注目すべきである。

予期された結果

- 許可されて拒否されるデータ・パターンのリスト

実行されるべきタスク
データの拒否・許可

- 拒否されるべき許可されたデータのチェック（false negative）
- 許可されるべき拒否されたデータのチェック（false positive）

- 許可されるべき許可されたデータのチェック
- 拒否されるべき拒否されたデータのチェック

コードの対策

- 入力とその許可を扱うコードの特定
- コードの修正

コードの修正では、これについての重要な部分は発見された正しくないデータを拒否するコードを付け加えるだけでなく、それからより一般的なパターンを想定することである。この点では、正しくないデータのフィルタリングと拒否するのに使用される方法を議論すべきだ。選ばれた方法は付録Cの「Input Data」で説明されるような環境に依存する。しかしここではフィルタリング（許可・拒否する方法）について議論する。

以下の二つの方法がある。

- 許可するパターン：入力データは許可されるパターンでチェックされ、新しい脆弱点が発見されるとそれが制限される。
- 拒否されるパターン：入力データは拒否されるパターンでチェックされ、新しいパターンが発見されると（拒否されるパターンを）追加する。

あなたが注意を払えば、各方法は欠点を持っている。第二のものは新しく発見されたパターンを加える事にとって簡単ではあるが、各拒否するパターンに対する追加のテストが必要である。第一のものは許可するパターンを減らすことになるので、誤りが無く包括的である。しかし、よく考え、テストする必要がある許可すべきパターンを拒否しないように注意しなければならない。

このテストで見つかったバグの典型的な例は、「付録A」の「Remote compromise」で見つけることができる。

リソースの使用予測

しばしば、システムの次の動きを予測する手助けとなるので、使用方法が推測可能であること（使用パターンをもたせること）はあまりよくない。攻撃者はシステムのふりをすることで、これ（同じネットワークであると答え、システムをだましたり、次に使用するファイルを知り、重要なファイルを削除するようリダイレクトしたり、セッションＩＤを推測する等）を利用する。

予期された結果

- リソース使用の追跡

実行すべきタスク：

システムモニタリング：すべてのリソースクラス（CPU、メモリ等）に対するもの
リソース分析：リソース使用のパターンを分析する

コードの対策

- DoSの原因となるコードの特定
- コードの修正

攻撃の予防

得たデータを処理している場合、利用可能なデータは安全であると考える傾向がある。しかし．．．指定された環境下でプログラムが起動していないなら、実際のデータはセキュリティコントロールをパスしたのだろうか？攻撃者がセキュリティチェックを回避する処理を見つけた場合どうするのか？

いくつかのプログラムはそれらが適切な環境で動いているか、かつ／またはコードが修正されていないかどうかを制御することでコードのセキュリティをチェックする。

予期された結果

- 不適切な実行環境での詳細なシステムレスポンス

実行すべきタスク
システムの実行

- 標準化されたもの以外のエントリー・ポイントの使用
- コードの追加、修正
- 実行条件の修正

コードの対策

- DoSの原因となるコードの特定
- コードの修正

思いつくもっとも一般的な修正は入力データをチェックする、かつ／または安全にする似たようなコードを追加することである。しかしこれは入力チェックモジュールに予約されたタ

スクでありこれらは複製されてはならない。ここで実行されたタスクは異なるタスク（データの正常チェックではなく異常な実行状態の検知）である。

出力

すべての入力をフィルタリングし、リソースの使用量について警告を出すようにした。しかしそれだけでは十分ではない。攻撃者に何が成功の鍵を伝えるのか？時々このように徹底的ではない、しかしいくつかの手がかりを与えることで情報すべてを与えることになる。

何かが注目されるであろうし、それらがかすかな情報（予期しない（偽の）アウトプット）が出力されるので、いくつかのアウトプットエラーがプロセスエラーとして見なすことができることである。ここで記述しようとしていることはHTML hidden
タグや内部構造情報を出力するデフォルトエラー、スタイル上の問題といった通常のアウトプットチャンネルを通る情報である。

データの除去レベル

この概念は、Miltary使用においてかなり明らかである。簡潔に言うと各オブジェクト（情報資産や人等）につけたいくつかのカテゴリ（極秘、マル秘、公開、等）がある。そして人は同じまたは自分より低いレベルの資産にアクセスできる（例えば、「マル秘」レベルのアクセス権を持つ人は「マル秘」や「公開」レベルの資源にはアクセスできるが「極秘」レベルのものはアクセスできない）。これとは別に、資産はいくつかのものから成り立っている場合、最大のセキュリティ分類と同じセキュリティ分類を持つ（「マル秘」と「公開」データが文書にある場合セキュリティ分類は「マル秘」になる。これに「極秘」情報が加わると「極秘」となる）。同じ事がコミュニケーションチャンネルでも当てはまる。「極秘」文書を「マル秘」チャンネルで送ることはできない（チャンネルには送信されたドキュメントが許可できる以上の保護メカニズムがある）。

同じコンセプトがシステムの出力にも適用されるべきである。

予期された結果

- 各出力オブジェクト（データ、人、チャネル等）のセキュリティレベル

実行すべきタスク

- 出力するコードの特定
- 意図された出力の&#© 2000 - 20037;分類
- 出力チャネルの分類
- 出力データの分類

コードの対策

- データのセキュリティレベルによる分類
- コードの修正

FIX ME：UMLが使用するケースとダイアグラムに似たものを考える。

時々、あなたは多くの情報を与えたいかもしれない、そしてこれは見てきたように必ずしもよいことだとは限らない。また、許可されていない人に手がかりとなる情報（ファイルパス等）がOSの出すエラーに含まれている。

明快な例がWebサーバーである。デフォルト設定のMS IIS（Internet Information Server）は「ページがなかった」というエラーメッセージと（ファイルが存在しなかった場合）ファイルシステムの仮想ファイルパスまたは（一般に公開していないにも関わらず内部データ構造またはデバッグ情報を含んでいるかもしれない）エラーの原因となるスクリプトの出力を表示する。IISの対抗物（Apache）はエラーを起こした事を示すメッセージを表示し、管理者に知らされerror_logに情報が格納される。

Appendix A - 最も一般的な脆弱性

このセクションでは障害が記述される。しばしば詳細に、また小さなアイデアが示されているし、障害について論じる幅広い研究や論文が示される場合もある。

このAppendixでは障害についての完璧なリストを作成する代わりに、最も一般的で代表的なものや、情報、ツール、関係のある見つかった結果にある共通のバックグラウンドをあげた。

Stack smashing

恐らくsmashがなにであるか、コンピューター用語でstackがなにであるか知っているだろう。「buffer overflow」という言葉がたぶんよく知られているだろうが、両方とも同じ問題を指している。

すでにこれについてよい論文（[5]、[6]）があるので簡単に記述することにする。

高級言語で２つの変数を定義し、それら二つがソースコードで隣接する場合、それら二つが隣接したメモリに格納される可能性が高くなる。それらのうちの一つに制限を越えて書き込んだ場合、もう一方のメモリを書き換えることになる。最初の変数が３０バイト確保されていたとすると、第３１バイト目にアクセスすると２つ目の変数の第１バイト目にアクセスすることになる。

これらはほとんど役に立たないように見えるが、私たちは少し研究した。高級言語を使用しているとほとんどの場合、ローカル変数を使用するサブルーチン（関数、プロシジャー等とよく呼ばれる）を使用する。コンパイラがスタックと呼ばれるメモリの一部が使用される。そしてそこにサブルーチンが終わったときに実行されるリターンアドレスと呼ばれる次の命令のアドレスが格納される。さらにローカル変数もそこに格納される。ここでこれらのローカル変数にアクセスすればその範囲を超えることができリターンアドレスにアクセスできるのでこれを用いてexploitを作成できる。リターンアドレスを魔法で知ることはできないが、新しいコードを挿入し、それをリターンアドレスで指すことができればプログラムを開発でき、あなたのために動かすことができる。

リターンアドレスをいくつかの方法でチェックしたり、それに従って作用したり（プログラムを終了したり破壊されたリターンアドレスを回復しようとしたりすることができる）するようなトリックを用いることでこれはさけることができる。また、このチェックは破られるかもしれない、しかしこれらの問題が起こるいくつかの小さな可能性がある。このトピックのよい記事は[7]で読める。

Format String bug

いろいろな関数があり、あるものは他のものより幾分柔軟である。しかし、柔軟さにはそのコストと制限が存在する、そのようなものの一つがセキュリティである。パラメータの一つがフォーマット、続く引数が関数に渡される順番を含む文字列があるような関数の共通のセット（printf系関数と呼ばれる）がCにある。例として関数を呼び出すと：

printf("Hello, %s. You have visited us %u times", name, counter);

渡された引数は3つ（文字列、変数名、カウンター）であり、第1引数（フォーマット文字列と呼ばれる）から、1つの文字列(%s)と1つの正整数(%u)というようにそれらの長さ及び型の情報を読み出す。

ここでは、この「フォーマット文字列」はアプリケーションにハードコーディングされている。フォーマット文字列のコントロールが外部に与えられることについてはどうなのか？？例として次のコードの一部を見てみる。

```
void flex_printf( char *format_string name,unsigned int counter  ) {
     printf( format_string, name, counter );
}
```

ここでformat_stringは証明されない/信頼されなかったユーザによって入力された文字列である。これは%sや%uの代わりにメモリ情報を取得するのに便利な方法（例えば、カウンターの中にHEXを入れ、文字列へのポインタまたはchar
*をフォーマット文字列で宣言すること）に変更するような文字列操作を利用して、いくつかの内部情報を公開したり、データ収集に有効であったり、（stack

smashingのような）exploitに後で使用されたりすることができる。完全な記述は（例「in the wild」）reference[10]を見ること。

Remote compromise

これは2つの一般的な脆弱性（フィルタリングがない、もしくは不完全である）と過剰な特権レベルをもったプログラムが動いていること）が合わさったExploitの説明である。

参考資料[11]のアドバイザリーから抽出された「Brief Summary」より引用：

「MS
Jetデータベースエンジン（Accessデータベースで使用されている）はユーザーがＳＱＬ文(それらはユーザーがコマンドラインのNTコマンドを実行することを可能にするかもしれない)にVBAを埋め込むことを可能にしている。これとＯＤＢＣコマンドがsystem_local権限で実行できるIISの欠点と組み合わせることで攻撃者はリモートからシステムを完全に掌握することができる。他のWebサーバーも影響を受けるかもしれない。多くのMS
Jetエンジンは影響を受けるが、特権を持たせることはないかもしれない。」

基本的にVBA（Visual basic for
Applications）コマンドをＳＱＬ文に埋め込むことができ、シェル呼び出しを利用してシステムコマンドを実行するように、ODBCから（適切にフィルタリングしていない）それらを実行できる。もし効率的な方法でタスクが最低限の権限で実行できるユーザーで起動していた場合この問題は極小化される。この場合はそうではなかった。

それ以上は説明されないだろう。引用されたアドバイザリーは「詳しく」読むに値する良著である。

リソース維持

TCPコネクションを初期化する際に以下の3つのステップを行って接続を確立する。

- リモートの端末にコネクション要求を行う。
- リモートの端末は要求を認める（又はコネクションを拒絶する）返答を行う。
- 要求にこたえる。

その単純なメカニズムに関して内部のリソースについての議論を避けることはメカニズムの制御を欠く事になる。あなたのコンピューターがクラッシュするリモートコンピューターに接続要求をする時、リモートの端末はACKを送信する、しかしあなたのコンピューターが受け付けないとリモートコンピューターはタイムアウトになるまで待ち、コネクションを閉じ

る。TCP/IPは信頼できないハイレイテンシー環境で動くようにデザインされているのでコネクションが閉じられる（又は失敗した）と考えられるまで待つ。大量のコネクションがこの状態にあると開放できるリモートコンピューターのリソース（CPUとメモリ）をこれが無駄にする。

とても良く出来た報告書はCERTにある（資料[12]を参照のこと）。

Appendix B - ツール

ツールと言うと何が思い浮かぶか？もし答えがsecure programingを強いることに使うプログラムであるならほぼ正しいが、使用することだけを意味しない。自家製のものが用意されているかもしれない、あるいはいくつかのプログラムが自身の要求に適合されているだろう。

すべてのログ

これらの問題をすべてソートすることで一つのことが明らかになるに違いないし、またすべての試みに関わらずプログラムは完璧ではないし、不備を見つけられるし、見つかるだろう。これらのホールを見つけさせる適切なツールはデバッガーやダンプアナライザとして使用される。しかし、時々これらのツールは有用ではないし、どこから探索を始めればいいのかわからない場合は特にそうである。

「ヘンゼルとグレーテル」の童話を知っているか？彼らは道に沿ってパンを落としていったが、後で無駄になった。我々も同じトリックを使ったり、得られた最も重要なデータを利用して、重要なコードの一部を追跡したりする。そして、攻撃（あるいは単純な共通の失敗）を用いてプログラムを解析することができる。

FIX ME：許可することとしないこと（DoS攻撃、特殊文字、パスワード等）

使用する準備

これはセキュアプログラミングを行うのに有用な（好ましいものはオープンソースのものである）ツールの単純なリストである。

名称：Flawfinder
　要約：C/C++でかかれたソースコードのセキュリティ上の問題点を探し出すプログラム
ライセンス：General Public License (GPL).
Link: http://www.dwheeler.com/flawfinder/

名称：ITS4
要約：統計的にC/C++のソースの潜在的なセキュリティの脆弱性を探し出す単純なツール

ライセンス： ITS4 NON-COMMERCIAL LICENSE for source code
Link: http://www.cigital.com/its4

名称：LCLint
要約：静的にCプログラムをチェックするツール
ライセンス：General Public License (GPL).
Link: http://lclint.cs.virginia.edu/

名称： StackGuard
要約：StackGuardは「stack smashing」攻撃を防御するコードを生成するコンパイラである。stack smashing攻撃は攻撃の最も一般的な手法である。
ライセンス： StackGuard is Free Software: it is an enhancement to GCC, and is distributed under the GPL in both source and binary forms
Link: http://www.immunix.org/stackguard.html

名称：FormatGuard
要約：FormatGuard はC/C++プログラムで、可変変数を使用するprintf系の関数でよく使用される攻撃であるフォーマットバグを防御する。
ライセンス：FormatGuard is Free Software: it is an enhancement to GCC, and is distributed under the GPL in both source and binary forms
Link: http://www.immunix.org/formatguard.html

名称：RSX
要約：RSXは実行できないスタック、ショート/ロングヒープ領域を実装するためにLinuxのバイナリをリマッピングするRuntime addressSpace eXtenderである。マップされたデータエリアのコード実行を防ぐ一般的なバッファーオーバーフロー問題の対策となる。
ライセンス: Proprietary License with Source
Link: http://freshmeat.net/projects/rsx

名称：PageExec
要約：IA-32プロセッサーのための実行可能でないページを実装する。
ライセンス： ???
Link: http://pageexec.virtualave.net/

名称：Libsafe
要約：バッファオーバーフローとフォーマットストリングバグを防ぐツール。脆弱であると

知られているライブラリー関数への関数呼び出しをすべて入れ替えるライブラリーとしてパッケージにされている。
ライセンス：Libsafe version 2.0 source code licensed under the GNU Lesser General Public License.
Link： http://www.avayalabs.com/project/libsafe/index.html

Appendix C - 脆弱性

海岸の近くで航海する際に、昔風に航海したならば、もっと詳細な地図をトレースする必要がある。それは、総括的な様相から特別のものまで行う私の個人的好みである。したがって、この方法でスタートする事にする。

入力データ

プログラムあるいは情報操作に関して基本から始める。情報操作の典型的な概要はデータを得、処理（変換）し、結果として提供すること（入力ー処理ー出力）である。

大したものではないが、不調および（システムに依存する）副作用を防ぐには、適切なデータを得るためにフィルタリングしなければならない。

口座番号を入力しなければならなかったり、アカウント・バランスを見せる必要があるような銀行業務システムを考えてみると、入力（口座番号）は（３つの数字と５個の文字、８個の数字であるような）適切なフォーマットであるべきである。そしてそれは検証（テンプレートと照合）され、効果的で（適切なフォーマットでなかったり、口座が存在しなかった場合に拒絶され）なければならない。

この例においてフォーマットと効果的にする方法はかなり単純である。つまり、存在しない口座のアカウント・バランスは提供できないが、記述されたアイテムのような機能的な問題でない（それらがシエルコマンドのように現れる際にさらによく見られる（修正：それらを加える））より微妙な問題がある。このルールを作れる：使用されるとは限らないものはすべて回避されるかもしれないが。

これを効果的にする多くの手法があり、それらは固有のシステム要件に依存する。

- 異常終了：これは最も顕著なものである。そのため無効なデータが検知される場合、システムが（不正なデータフォーマットを見つけるといった）それを攻撃と解釈すると終了する。
- 拒絶：無効なデータが検知される場合正しいフォーマットを求め、新しい入力を待ち受ける。これは攻撃者にとって有用であり、有効なデータとなるまで何度でも入力することができる。

- 使用可能：データの悪い部分は削除され、適切なフォーマットのデータだけが使用される。代表的なものとして、emailアドレス、URLおよび郵便のアドレスのような可変長データに使用される。

セキュアプログラミングに対する適応性がある言語の一つがperlである。「汚染モード」と呼ばれるモードで実行することができる。そしてそれは本質的に実際のプログラムの外から得られたデータを汚い、または「汚染された」ものとして扱い、きれいにするまで使用できない様にするモードである。通常、この汚染されたデータをきれいにするものは効果的にインプリメントされたフィルタリングが使用される。そして汚染されたデータがきれいにされずに使用された場合、Perlプログラムは中断する（より深く理解するために[4]を参照すること）。

FIX ME：他の言語で同じようなものがある場合ここでいったことが当てはめられる。

プロセッシング（パンドラの箱）

時々プログラムの外から得られるデータに大変気をつけるが、（私はそれを言うことを嫌う）それだけでは十分ではない。我々は完璧ではなく、ミスを犯すので、プログラムとシステムを我々から守らなければならない。どんなプログラムにも原因となる多くの要因があり（これはソフトウエアの信頼性の分析において真実なので、プログラムの振る舞いは統計を利用して研究されている）、それらのすべてを直ちに気をつけることができないということを覚えておく必要がある。そのためバグが混入する可能性は実際には高い（プログラムを時間をかけてテストすることでこの可能性は低くなる）。

これはよくあるソフトウエアの開発現場では珍しくないテストの方法、動作環境等である。そしてそれらは実地（それが最終的な環境で動くこと）での条件をシミュレートしたり、動かなかったり、エラーを起こしたり、行いことならすべてを可能にするような保護された環境でそれを実行することである。最終的な環境でインストールされたときに同じことが行われる。実際に動くまで注意して実行し、最初の週では監視することが重大な業務となる。

C++,Java,Perl,Phytonといった最近のほとんどの言語はプログラマ保護のメカニズムを持っている。基本的にそれらは保護された環境でいくつかの部分のコードを実行し、プログラムによって生成されるかもしれないどんなエラー（あるいは例外）もトラップする。

例えば、動かなくなった部分を他の環境から分離し、エラー処理コードから処理を戻すことができる。:

```
try() {
      Code_under_suspicion();
  } catch (Exception e) {
      System.println "Code_under_suspicion(),Exception generated : " + e;
  }
```

さて、プログラムが動かなくなったとしても、システムが動き続けなければならない、医学、航空学、ロケット工学といったクリティカルなプログラムを開発する場合を考えてみよう。フライト中、目的の空港へのルートをたどるプログラムのコンソールにこのメッセージがでた場合を想像してみる。

"Exception thrown at 0x3F745DE9. The system is going down"
（0x3F745DE9での例外エラー。システムは落ちています）

従って、これらのプログラムはこれに気をつけて作られている。特殊な条件下（問題があり、ハードウエアとソフトウエア等を調べるコードを分離することを言おう）でもプログラムが動き続けるだろう。

Javaに適用された例外処理についてよく知られた修正を見るにはリンクセクションにある[8]を見てください。

時々、この方法はそれを行うのに良い選択ではないが、すべてのプログラムを防ぐよりは簡単である。それらは余分なプログラミング時間や、リコンパイル、防御を活性化、あるいは非活性化するために必要なインストールを必要としない。これが生じる場合、ダンプアナライザー（導入後評価分析）またはプログラムをラップするデバッガーのようにかなり標準的な一連のツールを使用することができる。エラーが起こったときには、デバッガによってトラップされ管理される。これらは単にスタックを追跡結果や、変数の内容を表示したり、システムから抜けたりする事ができる。また、プログラム解析、レジューム、リスタート、中断、ダンプ等を行う一種のShellをプログラマやオペレーターに提供する。

出力データ

副作用について述べるまえに、出力は他のプロセスへの入力になることに気付いていただろうか。最悪なことに....くずについてはどうだろうか?

まず最初に、入力データにあるのと同じ問題が出力データを受け取り、処理するプロセスに発生する。プログラム/システムがフィルタとして働くのでプロセスはそれほど不完全ではないが、プログラムが（もちろん目的ではないが）ダメージを与えたり、正しい結果を間違えたりするような悪いことをする事ができる。

データ処理プロセスと出力プロセスの一部としてのファイル処理を例に取ってみる。なにも間違っているようには見えないし、すべて仕様書に従っており、ゲームのルールに従っている。本当にそうだろうか? アウトプットデータの固定ファイル名を作成する場合に単純にoutput.dataとつけてみよう:

filehandle = open("output.data", w);
write(filehandle, data);
close(filehandle);

もちろんファイルが存在しているかをチェックし、存在していた場合オペレータ/ユーザーにファイルを消すか聞き、バックアップし、あらゆる種類のことを確実に加えるだろう。しかし...万が一根本的なレイヤーが危険にさらされ、考えている場所に書いていない場合、どうするのか。ファイルに対する「クリーンなパス」があるだろうか？

FIX ME：前のトピックにもっと関係する。

さて、ご存じのように、のぞき見するのに重要な情報源の一つは（結局シュレッダーが存在するので）ゴミ箱の収集と分析である。この最も一般的な例は「あなたが行ったことに対する警告やエラーメッセージは何か？」という問いに答えることである。確かにいくつかのWebサーバーではエラー（ページが見つからない、cgiに失敗した、クライアントにエラーデータが送られた）が発生した場合このようなメッセージがでるものがある：

The requested CGI program in '/home/httpd/cgi-bin/script' failed to execute. The next lines contain the error:

Couldn't connect to database 'Customers' in server 'internal_db'.
Send a message to hostmaster@this_domain.com

これらのメッセージにはなんと多くの情報源があるのだろう！！エラーログや管理者/オペレーターにしか読めないような場所にこれは格納されるべきである。結局これはほとんどのユーザーにとって無意味だが、クラッカーにとっては有用な情報源となりはしないだろうか？また、この話の悲しむべき部分はほとんどのWebサーバーがデフォルトでは安全でないように設定されていることである。

Appendix D - Links

[1] **Title:** Secure Programming for Linux and Unix HOWTO
Author: David A. Wheeler
Location: http://www.dwheeler.com/secure-programs

[2] **Title:** Why Computers are Insecure
Author: Bruce Shneier
Location: http://www.counterpane.com/crypto-gram-9911.html#WhyComputersareInsecure

[3] **Title:** No silver bullet
Author: Frederick P. Brooks, Jr.
Location: http://www.undergrad.math.uwaterloo.ca/~cs212/resource/Articles/SilverBullet.html

[4] **Title:** Perl security
Location: http://www.perl.com/pub/doc/manual/html/pod/perlsec.html

[5] **Title:** Smashing The Stack For Fun And Profit
Author: Aleph One
Location: http://www.phrack.com/search.phtml?view&article=p49-14

[6] **Title:** Avoiding security holes when developing an application
Author: Frederic Raynal, Christophe Blaess, Christophe Grenier
Location: http://www.linuxfocus.org/English/March2001/article183.meta.shtml

[7] **Title:** Bypassing StackGuard and StackShield
Author: Bulba and Kil3r
Location: http://www.phrack.com/search.phtml?view&article=p56-5

[8] **Title:** What's an Exception and Why Do I Care?
Author: Sun Microsystems
Location: http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html

[9] **Title:** SECPROG@SecurityFocus.com working document
Author: Secure Programming list contributors
Location: http://www.securityfocus.com/forums/secprog/secure-programming.html

[10] **Title:** Analysis of format string bugs
Author: Andreas Thuemmel
Location: http://www.securityfocus.com:80/data/library/format-bug-analysis.pdf

[11] **Title:** NT ODBC Remote Compromise
Author: Matthew Astley & Rain Forest Puppy
Location: http://www.securityfocus.com/archive/1/13882

[12] **Title:** CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks
Author: CERT
Location: http://www.cert.org/advisories/CA-1996-21.html

Open Methodology License (OML)

PREAMBLE

A methodology is a tool that details WHO, WHAT, WHICH, and WHEN. A methodology is intellectual capital that is often protected strongly by commercial institutions. Open methodologies are community activities which bring all ideas into one documented piece of intellectual property
which is freely available to everyone.

With respect the GNU General Public License (GPL), this license is similar with the exception for the right for software developers to include the open methodologies which are under this license in commercial software. This makes this license incompatible with the GPL.

The main concern this license covers for open methodology developers is that they will receive proper credit for contribution and development as well as reserving the right to allow only free publication and distribution where the open methodology is not used in any commercially printed material of which any monies are derived from whether in publication or distribution.
Special considerations to the Free Software Foundation and the GNU General Public License for legal concepts and wording.

TERMS AND CONDITIONS

1. The license applies to any methodology or other intellectual tool (i.e. matrix, checklist, etc.) which contains a notice placed by the copyright holder saying it is protected under the terms of this Open Methodology License.

2. The Methodology refers to any such methodology or intellectual tool or any such work based on the Methodology. A "work based on the Methodology" means either the Methodology or any derivative work by copyright law which applies to a work containing the Methodology or a portion of it, either verbatim or with modifications and/or translated
into another language.

3. All persons may copy and distribute verbatim copies of the Methodology as are received, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and creator or creators of the Methodology; keep intact all the notices that refer to this License and to the absence of any warranty; give any other recipients of the Methodology a copy of this License along with the Methodology, and the location as to where they can receive an original copy of the Methodology from the copyright holder.

4. No persons may sell this Methodology, charge for the distribution of this Methodology, or any medium of which this Methodology is apart of without explicit consent from the copyright holder.

5. All persons may include this Methodology in part or in whole in commercial service offerings, private or internal (non-commercial) use, or for educational purposes without explicit consent from the copyright holder providing the service offerings or personal or internal use comply to points 3 and 4 of this License.

6. No persons may modify or change this Methodology for republication without explicit consent from the copyright holder.

7. All persons may utilize the Methodology or any portion of it to create or enhance commercial or free software, and copy and distribute such software under any terms, provided that they also meet all of these
conditions:

a) Points 3, 4, 5, and 6 of this License are strictly adhered to.

b) Any reduction to or incomplete usage of the Methodology in the software must strictly and explicitly state what parts of the Methodology were utilized in the software and which parts were not.

c) When the software is run, all software using the Methodology must either cause the software, when started running, to print or display an announcement of use of the Methodology including an appropriate copyright notice and a notice of warranty how to view a copy of this License or make clear provisions in another form such as in documentation or delivered open source code.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on any person (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If said person cannot satisfy simultaneously his obligations under this License and any other pertinent obligations, then as a consequence said person may not use, copy, modify, or distribute the Methodology at all. If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

9. If the distribution and/or use of the Methodology is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Institute for Security and Open Methodologies may publish revised and/or new versions of the Open Methodology License. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
NO WARRANTY

11. BECAUSE THE METHODOLOGY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE METHODOLOGY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE METHODOLOGY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE IN USE OF THE METHODOLOGY IS WITH YOU. SHOULD THE METHODOLOGY PROVE INCOMPLETE OR INCOMPATIBLE YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY USE AND/OR REDISTRIBUTE THE METHODOLOGY UNMODIFIED AS PERMITTED HEREIN, BE LIABLE TO ANY PERSONS FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE METHODOLOGY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY ANY PESONS OR THIRD PARTIES OR A FAILURE OF THE METHODOLOGY TO OPERATE WITH ANY OTHER METHODOLOGIES), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.