Boost.勉強会 #4 東京（2011-02-26）

C++プログラマの為のセキュリティ入門

この発表の内容について

- 具体的なコード寄りの話･･･ではなく、そもそもどういったことをやらなければならないのかと言った基本的かつ普遍的な事柄を中心とした内容になっています。

概要

- セキュリティって？
- さまざまな問題
- 歴史
- さまざまな技術
- Ｃ＋＋では
- ＪＩＳＥＣとＪＣＭＶＰ
- 参考情報

C＋＋プログラマの為のセキュリティ入門

セキュリティって？

セキュリティって？

- はじめに・・・

ツッコミ怖い！＞＜

セキュリティって？

- 後でもそのツッコミの怖さを紹介しますが、とにかくツッコミがいろんな意味で怖い世界です。
- ここ数年のトレンドはしっかり追跡してないんでその点についてはご容赦を。

セキュリティって？

- 多くの方がこの発表に求めているであろうもの：
  - 「こうしておけば大丈夫/安心です！」という類のもの
- そんなものはあり得ません！
- そんな幻想は今すぐ捨ててください！

セキュリティはトータル

- システム全体のセキュリティの強度はそのシステム全体の中でもっとも脆弱な部分の強度にほぼ等しい。
- ガチガチのセキュリティで固めたつもりでも１点の脆弱性で台無しになることもある。

コストパフォーマンスは重要

- 各種セキュリティの導入・運用には一般的にコストがかかるが、そのコストがセキュリティを破られ時の損害を上回るようでは意味がない。

コストパフォーマンスは重要

- 絶対に破られてはいけない。
  - 絶対になんてものがそもそも幻想。
  - そんなスタンスでは無尽蔵にコストがかかる。

- 万が一の損害からセキュリティにかけていい適切なコストを考える。
- 適切なレベルでの損害/リスクの最小化を考える。

コストパフォーマンスは重要

- 攻撃者視点に立っても、攻撃者がその攻撃を成功させたときに得られるメリットより、攻撃を成功させる為に必要なコストを上回らせることができれば一般的には十分。

リスクアセスメント

- 情報資産の棚卸
- CIAの観点からリスクを列挙
  - 機密性 Confidential
  - 完全性 Integrity
  - 可用性 Availability
- それぞれのリスクの対策に必要なコストと損害発生時の被害を天秤にかけ総合的に判断しどのような対策を施すのか決定する。
  - リスクとしては小さいが対策には非常にコストがかかる場合には「対応しない」という選択もあり。

そもそもなにを防ぐべきか？

- 意図していない/許していない権限の取得
- 意図していない/許していない処理の実行
- 情報の漏えい/盗聴/削除/改ざん
- DoS
- スパム/荒らし
- ハードウェアの損壊/盗難
  - 直接的にはソフトウェアの範疇ではないが、そういった事態に対し損害を最小化するソフトウェア設計になっていることが望ましい。

どう防げばいいのか？

- 適切且つ正確な権限管理
- 入力チェック
  - 自分のコードにとっては問題ないデータでもシステムの提供するAPIやシェル等(Webアプリではブラウザなど)がやられてしまうケースが存在する。
- 入出力の正確なエンコード/デコード/エスケープ/アンエスケープ
- 暗号
- 証明
- ケンジントンロック
- 法的圧力
- etc

公開されているロジックで

- ロジック自体が秘匿にされている暗号製品なんかもありますが、セキュリティ業界ではロジック自体を秘匿にするような暗号はよくないとされ実際一般的には脆弱な暗号であることが多いようです。
  - ロジックが公開されていれば、外部の第三者によるツッコミが入り洗練されます。

安全性の定義

- 各種情報セキュリティ関連の技術や製品には普通その技術や製品を使うことでなにがどう安全なのか明確な定義があります。
  - それが無いようなものの信頼性は推して知るべし。
- 多くの場合、それを正しく理解したうえで正しく利用しないと台無しになります。

安全性の定義

- ダメな例(錠前編)：
  - 錠前をつけても鍵をかけていない。
  - 錠前に鍵を刺しっぱなし。
  - 錠前をつけてる扉の耐久力が極度に脆弱。
  - 錠前の耐久力に対して保管物の価値が異常に高い。

暗号の安全性

- 情報理論的安全性
  - 情報理論的安全性を持つ暗号文はどんなに時間をかけても解くことは不可能。~~⇒強秘匿性~~
- 計算量的安全性
  - 計算量的安全性を持つ暗号文は一般的に現実的な時間の範囲では解けない。

暗号の安全性

- 情報理論的安全性のデメリット
  - 暗号文と同じ分量の鍵情報が必要になる。
    - この鍵情報は使い回しが効かない。(使い回すと情報理論的安全性は失われる。)
    - この鍵情報は素の真性乱数でなければならない。(疑似乱数などのアルゴリズムに頼った場合、情報理論的安全性は失われる。)

暗号の安全性

- 計算量的安全性のデメリット
  - 理屈的には時間をかければ解ける。
    - コンピュータの進化が速いこともこの問題に拍車をかける。
    - より高速に解く手法が発見される度に、脆弱になっていく。

事例を追いかけよう

- セキュリティ関連はこれだけやれば完璧・・・となかなか断言できるようなテーマではない。
- 仕事や趣味で手を出す分野のセキュリティインシデントをウォッチし続けよう。
  - 他山の石としよう。
  - クラッカーは飽きもせずあの手この手を試し続けるので既知の問題の対策を十分に施しているつもりでも、新しい攻撃手法により思わぬ穴を突かれることもある。

C＋＋プログラマの為のセキュリティ入門

さまざまな問題

さまざまな問題

- 通信の盗聴
- バッファーオーバーフロー
- 整数オーバーフロー
- エスケープ
- セッションハイジャック
- 辞書攻撃
- ファイルパス
- ファイルコンテンツ

C＋＋プログラマの為のセキュリティ入門

歴史

歴史

- DESの最強っぷりと衰退、そしてAESへ。
  - DESとは米国で公式連邦情報処理基準として採用された共通鍵のブロック暗号
  - DESの類似暗号の大量発生→DES以外全滅
    - DESの類似暗号には通じた攻撃も、DESだけには通じないという事態が多発。
  - 最後はDESも・・・
  - AESへの道のり
  - AES2

C＋＋プログラマの為のセキュリティ入門

さまざまな技術

さまざまな技術

- ハッシュ関数
  - 代表的なハッシュ関数アルゴリズム
    - MD5,SHA1,SHA2
      - MD5やSHA1は古いので改竄防止目的などではもう非推奨なので注意。

さまざまな技術

- ハッシュ関数
  - 一方向性関数
    - 一般的なハッシュ関数の特徴
      - 入力値によって出力値が変化する。
      - 入力値が同じならば同じ出力値が得られる。
      - 入力値が１ビットでも違えば出力値のビットパターン全体がごっそりと変化する。
      - 入力値は可変長で出力値は固定長。
    - セキュアなハッシュ関数の特徴
      - 出力値から部分的な入力値の予想すらも困難であること。
      - 特定の出力値が得られる入力値の計算が困難であること。

さまざまな技術

- ハッシュ関数
  - ハッシュ関数の用途
    - 入力値を保存/記録しないで、以前の入力値と同値であるかどうかを検証
    - 破損チェック
    - 改ざん防止
  - ハッシュ関数を使用する際、非公開のマジックワードを入力値に加えておくと
  - KDF拡張
    - 出力値=H(入力値) → 出力値=H(H(入力値)+シーケンス番号) の形でハッシュ値を引き延ばすことが可能。
      - シーケンスの部分は int 等の整数値を使用する。
      - 10GB以上の規模に引き延ばす場合は64bit整数を使用する。
    - 疑似乱数としても使用可能。
    - 値空間は元のハッシュ関数と同じ
  - 次世代ハッシュ関数は並列処理化が可能

さまざまな技術

- 暗号
  - 鍵暗号
    - 共通鍵暗号
    - 公開/秘密鍵暗号
    - ブロック暗号/ストリーム暗号

さまざまな技術

- 暗号
  - 秘密分割
    - XOR
  - 秘密分散
    - 多項式＋有限体
  - →秘密分割・秘密分散には一般的に情報理論的安全性と強秘匿性があります。

さまざまな技術

- 暗号
  - データ長やエントロピーの問題
    - 一般的に暗号化では平文のデータ長は暗号されない。データ長がバレるだけでも望ましくないケースもあるので注意が必要。
    - 例えばパスワードのようなデータは短いってのがばれるだけでも、ブルートフォースで簡単に敗れることがわかるのまずい。
      - そもそもパスワードなんかはハッシュ値で保存するべきですが。
    - 性別のようなbool値なデータは単独で同じ鍵を使ったブロック暗号で暗号化してもほとんど意味がない。
    - 対策：必要に応じて暗号化前にランダムな値で適当な長さにパディングする。

さまざまな技術

- 乱数
  - 真性乱数
    - ほぼ間違いなく偏りが現れる。
    - 一般的に疑似乱数に比べ遅い。
  - 疑似乱数
    - 質の良くない疑似乱数では、アルゴリズム特有の偏りが現れる。
    - シードが同じなら同じ出力が得られる。
      - ハッシュ関数とほぼ逆の特性となる。

さまざまな技術

- PKI
  - 詳細はこの発表では触れません自分で調べてください。

さまざまな技術

- PKI
  - ポイント
    - 実在証明でしかない。
      - 実在するその会社なりお店なりがブラックでないことまでは保証されない。
    - 盗聴/改竄/成りすましなどを防ぐことのできる正しいSSLの通信を利用する為には問題の無い証明書が必要。
      - 所謂オレオレ証明書では成りすましを防げない為、盗聴/改竄/成りすましを防げない。
    - 証明書の更新時には秘密鍵をちゃんと更新しよう！
      - その為の証明書の更新です。

さまざまな技術

- TPM (Trusted Platform Module)
  - ハードウェア耐タンパー性をもつセキュリティチップ
  - TPMは以下の機能を提供する。
    - RSA
      - 演算
      - 鍵生成
      - 鍵格納
    - SHA-1ハッシュ
      - ハッシュ値計算
      - ハッシュ値保管
    - 乱数生成
  - TPM1.2から以下の機能が追加された。
    - カウンタ
    - 単純増加カウンタ
    - ティックカウンタ
    - オーナー権委任 (パスワードは公開しない)
    - 不揮発性ストレージ保存機能
  - ※ http://ja.wikipedia.org/wiki/Trusted_Platform_Module より抜粋

C＋＋プログラマの為のセキュリティ入門

C＋＋では

C++では

- クライアントで動作するコードであれば
  - 全てクラック可能。
  - システム管理者とユーザーとゲストとリモートアクセスに対してどのようにあるべきなのか？
  - クラックにも難易度がある。
  - コードサイニング

C++では

- サーバーで動作するコード(外部との入出力を行うコード)であれば
  - 入力チェックを厳格に
  - 入出力のエンコード/デコード/エスケープ/アンエスケープを正しく
  - アクセス権を正しく設定・処理
  - 必要に応じてSSLを使用し安全に通信を行う。

C++では

- バッファーオーバーラン対策
  - →文字数上限チェック
    - マルチバイトまわりでチョンボしないこと。
    - サロゲートペアまわりでチョンボしないこと。
- 整数オーバーフロー対策→値域上限チェック
- 正しくエスケープ
  - どこかのフェーズでNUL文字となるコードを挿入された場合も考慮すること→これを気を付けていないとconst char * で処理を行った際にNUL文字以降の部分がエスケープ等の処理を免れることが発生してしまう。

C++では

- 不正な文字エンコードの検出
  - UTF-8 でのチェック逃れ
  - マルチバイト文字列での閉じ文字喰い
- ファイルパス
  - 相対パスのチェック
  - NGワードの除外(Windows での CON, AUX 等)
  - 偽バックスラッシュ
  - UNICODE制御文字
- ShiftとXORを使った文字列の難読化
  - NUL文字はNUL文字のままにできる。

C++では

- セキュアな乱数
  - Windows であれば CryptGenRandom() を使う。
  - Linux であれば /dev/random あるいは /dev/urandom を使う。

C++では

- ライブラリ
  - Crypto++
  - Crypto API
  - OpenSSL

C++では

- Windows の Crypto API を使用する上での注意。
  - CryptAcquireContext() を呼び出した際に環境によっては NTE_BAD_KEYSET エラー (GetLastError()の戻り値)になる場合がある。
  - その場合にのみ CRYPT_NEWKEYSET を指定して CryptAcquireContext() の最後の引数に加えて呼び出す必要がある。

C++では

```
//  Crypto Provider Handle の初期化
HCRYPTPROV handle;
if (!CryptAcquireContext(&handle, NULL, NULL, PROV_RSA_FULL, CRYPT_SILENT)) {
  if (GetLastError() == (DWORD)NTE_BAD_KEYSET) {
    if (!CryptAcquireContext(&handle, NULL, NULL, PROV_RSA_FULL, CRYPT_SILENT | CRYPT_NEWKEYSET)) {
      handle = NULL;
    }
  } else {
    handle = NULL;
  }
}
assert(NULL != handle);

//  Crypto Provider Handle を使った処理
...

//  Crypto Provider Handle の解放
CryptReleaseContext(handle, 0);
handle = NULL;
```

C++では

- Unix/Linux 系での注意。
  - /dev/random および /dev/urandom が無い環境もある。
    - 必要に応じてこれらの実装をインストールする必要があることをなんらかの形でユーザーに通知。
  - /dev/random の読み込むコードは十分なエントロピーが確保できるまで待たされる場合がある。
  - /dev/urandom は /dev/random のように待たされることはないが、セキュアな乱数であることの担保はない。

C＋＋プログラマの為のセキュリティ入門

ＪＩＳＥＣとＪＣＭＶＰ

ＪＩＳＥＣとＪＣＭＶＰ

- 本当に大丈夫なのか疑問な製品も多いし、残念ながら明らかにそりゃ駄目だろうって製品も多い。
- ほんとうにこの製品に情報をあずけて大丈夫なの？
  - →適切な対策が施されてることを知りたい。

ＪＩＳＥＣとＪＣＭＶＰ

- ＪＩＳＥＣ
  - IPAが運営しているITセキュリティ評価及び認証制度
  - CC
    - 国際的なセキュリティ評価基準
    - ISO/IEC 15408
  - CEM
    - 国際的なセキュリティ評価方法
    - ISO/IEC 18045
  - ST確認制度
    - 限定的な確認制度

ＪＩＳＥＣとＪＣＭＶＰ

- ＪＣＭＶＰ
  - IPAが運営している暗号モジュール試験及び認証制度

ＪＩＳＥＣとＪＣＭＶＰ

- 製品/サービスの売りになる
  - セキュリティ関連の為の開発工数の予算をゲットできる。
- 申請そのものの直接の費用は規模に応じて変わる。(審査を行う第三者機関の人件費分は最低でもかかるよね？ってあたりから想像してください。)

C＋＋プログラマの為のセキュリティ入門

参考情報

参考情報::IPA

- IPA
  - IPA セキュア・プログラミング講座
    http://www.ipa.go.jp/security/awareness/vendor/programmingv2/
  - JISEC
    http://www.ipa.go.jp/security/jisec/
  - JCMVP
    http://www.ipa.go.jp/security/jcmvp/
  - セミナー・イベント
    http://www.ipa.go.jp/security/seminar/seminar.html
- JPCERT/CC
  http://www.jpcert.or.jp/

参考情報::個人Webサイト

- セキュリティホールmemo
  http://www.st.ryukoku.ac.jp/~kjm/security/memo/
- 本当は怖い文字コードの話
  http://gihyo.jp/admin/serial/01/charcode
- それ Unicode で
  http://openmya.hacker.jp/hasegawa/public/20061209/momiji.html

参考情報::書籍

- 暗号技術大全
  ISBN-10: 4797319119
  ISBN-13: 978-4797319118

C＋＋プログラマの為のセキュリティ入門

質疑応答

C＋＋プログラマの為のセキュリティ入門

ご清聴ありがとうございました。