5. セキュリティ上の課題

本章では、TLSを利用したアプリケーション、システムを開発する上でのセキュリティ上の課題について、脆弱性の生じる原因箇所、安全性に対する脅威と攻撃手法、また、脆弱性インシデントの管理に関してTLSを利用するアプリケーションやシステムの開発者として一般的に認識しておかなければならない観点について簡単にまとめます。

5.1 脆弱性の階層

「脆弱性」という言葉に厳密な定義があるわけではありませんが一般的には、ソフトウェア製品、システムの機能や性能を損なう原因となるセキュリティ上の問題個所を意味します。一言で「脆弱性」といってもその原因や対策はさまざまです。ここではまず、セキュリティシステムの階層ごとにその要因と性質についてみていきます。

1) 暗号アルゴリズム

セキュリティのもっとも基本的な部分を担うのは個々の暗号アルゴリズムです。セキュリティプロトコルに利用される暗号アルゴリズムはコンピュータセキュリティの歴史の中で数々の改善が重ねられ、今日利用されるアルゴリズムは細心の注意を払って設計、標準化されており、このレイヤーでアルゴリズムそのものに脆弱性が検出されることは極めて稀です。

しかし、そうした暗号アルゴリズムを使って最終的に安全なシステムを実現するためにはいくつかの点に注意しなければなりません。

例えば、第三章 暗号技術の中でも述べているように今日の暗号アルゴリズムのほとんどは鍵のランダム性に依存しています。鍵のランダム性が保証されない場合、その鍵長で保証されるだけの安全性がフルに実現されないことになります。鍵値を鍵長の自由度より高い確率で予測できるような場合、その分だけ実質的に短い鍵を使用しているのと等価となり、鍵値を容易に推測される原因になってしまいます。

また、デジタル技術の進歩、性能の向上は攻撃側を利する側面もあります。歴史上のある時点で特定の鍵長の暗号アルゴリズムが安全であったとしても、年月を経る間に攻撃側の技術とともに性能、攻撃能力も向上することを考慮しなければなりません。このような現象を暗号アルゴリズムの「危殆化」と呼びます。

第二章、第三章の中でも触れているように、プロトコル標準が取り入れている暗号アルゴリズムの種別や鍵長についてはプロトコルバージョンとともに見直されています。しかし、標準の見直しには長い時間がかかる場合も多く、危殆化のスピードのほうが標準の進化より早くなってしまうケースも多々あります。システムの設計、開発者はこうした課題を考慮して適宜、使用するアルゴリズムや鍵長を適切に取捨選択してく必要があります。

2) プロトコル仕様

イントロダクションでも紹介しているようにSSL、TLSはインタネットの発展とともにそのプロトコル仕様は大きく見直され、TLS1.3に至ってかなり成熟した標準仕様となっていると言えます。しかし、プロトコルバージョンの移行には長い時間を必要とすることが予想されます。しばらくの間はやむを得ない理由である程度のバージョンのプロトコルも維持する必要があるかもしれません。

古いバージョンにはさまざまな脆弱性リスクが知られています。また、直近のTLS1.2でも多くの改善が盛り込まれた一方で後方互換性のために古い仕様を捨てるとこができていません。そのため、TLS1.2では使用する暗号スイートやさまざまな機能を注意して選択した上で使用しないと思わぬセキュリティリスクを内包してしまうことになってしまうことに注意が必要です。

アルゴリズム種別		TLS1.2まで	TLS1.3
共通鍵	ブロック型	DES, TDES, Camellia	AES
	利用モード	ECB, CBC, CTR, CFB	GCM, CCM, CCM_8
	ストリーム型	RC4	Chacha20
ハッシュ	MD4, MD5, SHA-1	SHA-2/256, SHA-2/384	

暗号アルゴリズムの危殆化とプロトコル仕様

3) プロトコル実装

プロトコル実装の品質、安全性と脆弱性への対応は各実装ベンダーの努力で担保されることになります。例えば、組み込み向けTLSライブラリーベンダーwolfSSL社は、同社製品の品質保証について以下のような項目にわけて紹介しています。

- 基本的品質保証

  開発者の手元で行われるローカルテストから、Gitコミット時の検証、プルリクエスト時の自動レグレッションテスト、統合テストとピアレビューによるフィードバックなどのプロセスが規定されています。

- 品質管理の自動化

  Jenkinsをベースとしたオンサイトとクラウドのハイブリッド型のCIを規定しています。これにより、組み込み向け特有のハードウェアアーキテクチャに対応した品質保証の自動化が可能となっています。

- 暗号アルゴリズム、モジュール

  NISTによるFIPS140-2/3など第三者認証期間による認定について紹介しています。

- 相互運用性テスト

  多数の他社実装との相互運用性テストを実施しています。

- 単体テスト

  最初の単体テストは開発者のマシンで実行され、単体テストのテストカバレッジは毎週計測れされ開発者にフィードバックされます。

- APIの一貫性、後方互換性検証

  実装の変更、機能拡張がAPI仕様を変更していないこと、バージョン間の一貫性の確認は毎日の自動テストの一項目となっています。

- 統合テスト

  さまざまなアーキテクチャ、コンフィグレーションによるて質を実施します。

- 安全性の解析

  cppcheck, clang静的解析(scan-build), Facebook infer, valgrind 他さまざまな静的分析ツールによる定期的な分析。独自のwolfFjzzによるFuzzingテストは4兆個のPRNGシードを３ヶ月で検証します。

- 脆弱性管理

  脆弱性発見からアクションの起動、攻撃者に漏洩しない形での対応コードのレビュー、検証から正式な公開までのプロセスを規定します。

4) アプリケーション実装

- TLSバージョン

  できる限り最新のTLSバージョンのみを使用するようにすべきですが、当面はやむを得ずTLS1.2を使用するケースも想定されます。その場合には、セキュリティリスクを可能な限り下げるように使用するオプションを選択する必要があります。例えば、拡張マスターシークレットのような潜在的脆弱性リスクを回避するためのオプションは有効化し、再ネゴシエーションのようなリスクのあるものは無効化します。

- 暗号スイート

  TLS1.3では現時点でリスクの認識されている暗号スイートはすべて排除されているので、すべてのスイートを使用することができます。ただし、利用モードCCM、CCM_8のような利用モードは比較的低性能のMCU向けである点は認識しておく必要がります。

- 乱数シード

  多くのTLS実装では、実際に内部処理で使用する乱数は擬似乱数生成などを使って乱数シードで得られる乱数よりランダム性を高める工夫がされています。しかし、乱数シード値もその自由度を高めるよう配慮しておく必要があります。例えば、純粋なソフトウェアによる単純なシード生成ではシステム起動時には毎回同じ乱数値を生成することになってしまうことになります。

- 開発、テスト用オプション

  商用のTLSライブラリでは多くの場合、アプリケーション開発テスト時に使用できる便利なオプション機能を用意しています。前述の乱数シードについては、アプロケーション開発者が開発当初には特別なものを用意しなくても動作させることができるようなテスト用の乱数シード生成機能を用意しています。そのようなオプションを誤って運用時に実装してしまうと、思わぬ脆弱性の原因となってしまいます。

- エラーログ

  十分なテストをして出荷しても、運用時に予期しない異常が発生する場合があります。そのような場合に確実に原因が九名できるような動作状態のログ機能を実装することが推奨されます。

5) 運用

TSLを動作させるために、例えば、パスワードや公開鍵証明書のようにアプリケーション運用時に設定する部分もあります。そのような運用時の設定が安全性を損なわないような注意が必要です。

5.2 脅威と攻撃手法

5.2.1 ネットワーク上の基本的な脅威

ネットワーク通信では様々なセキュリティ上の脅威が想定されます。TLSプロトコルでは、基本的な脅威として主に、盗聴、なりすましおよび不正アクセス、改ざんの３つから情報、システムを守ることを目標としています。

1) 盗聴

ネットワークセキュリティの初期の基本的な目的はネットワーク上を流れる情報の秘匿にありました。TLSでは、この目的のためにネットワーク上を流れる情報に関しては暗号化し、正当な受信者だけが復号できるようなメカニズムを提供します。また、そのための鍵を安全に交換するための仕組みも含まれています。しかし、それだけで十分ではありません。

2) なりすまし、不正アクセス

TLSの利用シナリオにおいて、「なりすまし」はサーバ、クライアントの双方で想定されます。Webサーバとブラウザのような場合に不正なサーバが他のサーバになりすましてブラウザからの情報を盗み取るフィッシングはサーバなりすましの典型例です。反対にクライアント側が不正に他のクライアントになりすますことで、サーバ側の情報に不正にアクセスしたりサーバの動作を撹乱させたりするリスクも考えられます。

IoTデバイスや組み込み機器のようなROMベースの小規模なデバイスがクライアントとなる場合は、クライアント側には不正にアクセスするほどの価値ある情報が無いように感じるかもしれません。しかし、クライアント側への不正アクセスを許し、クライアントのなりすましを許してしまうことでサーバへの不正アクセスを許してしまったり、サーバの動作を撹乱されたりするリスクを招くことになります。

このように、なりすましや不正アクセスによって、ネットワーク上に出ない情報についても盗聴と等価な行為が可能となってしまいます。なりすまし、不正アクセスを阻止するためには、通信の相手方について、正当な相手であることを確認するピア認証が重要です。

3) 改ざん

ネットワークを介して通信する際には中間者攻撃によってデータが改ざんされるリスクもあります。受信者が正常にデータを復号できたとしても、そのままではもとのデータが復号できている保証はありません。TLSでは、改ざん検出のためにレコード毎のHMACベースのメッセージ認証を使用していました。しかし近年、これでは潜在的リスクが解消できないことがわかり、データの秘匿と改ざん検出を同時に行う認証付き暗号(AEAD: Authenticated Encryption with Associated Data)が採用されるようになってきました。TLS1.3の暗号スイートではAEAD型のみが採用されています。

5.2.2 完全前方秘匿性

近年、セキュリティに対する攻撃者のモチベーションは個人的興味から大規模な組織やその連携をベースに金銭的利益を目的とするものや国家的なものまで非常に多岐にわたるようになっています。2010年代中ごろには、組織的な非常に大規模、長期間にわたる漏洩があかるみにでる事件などもあり、それまでの暗号技術、セキュリティ技術の前提の見直しが迫られる事態にまでなっていました。また、セキュリティを必要とするIT機器についても、データセンターのような物理的、組織的に堅牢に守られた場所での運用や機器の注意深い廃棄を期待できるとは限らなくなってきています。安価で日常生活や通常の企業活動の場で広く使われる機器にも、セキュリティ上守るべき対象が拡散するようになっています。

例えば公開鍵暗号では、対をなすプライベート鍵が安全に守られていることが大前提ですが、大規模で巧妙な攻撃に対してはなんらかの原因でプライベート鍵が漏洩する可能性は否定できません。また、ネットワーク上のトラフィックを非常に長期間大量に蓄積しておくことも可能となってきています。そのような環境では、例え通信内容がその時点では守られていたとしても、プライベート鍵のような秘匿情報が漏洩した時点で過去に遡って多量の秘匿情報を解読するようなことも可能となってしまうかも知れません。そうした背景で、完全前方秘匿性(PFS: Perfect Forward Secrecy)の概念が提唱され、そような事態でも秘匿性を保証できる必要性が認識されるようになってきました。

TLSでは当初、鍵交換方式として公開鍵証明書の機能と組み合わせた静的RSA方式が広く使用されていました。この方式ではクライアントで生成したプレマスターシークレットをサーバから送られるRSA公開鍵で暗号化しサーバに送

ります。この際に使われるRSA公開鍵はサーバ証明書に格納されているものを利用します。この方式ではサーバ認証と鍵交換の情報を共有して行うことができるので効率の良い実現が可能である一方で、サーバ証明書を頻繁に更新することは現実的でないため、同じ公開鍵を長期間にわたって使い回すことになってしまいます。

このように静的RSAでは完全前方秘匿性を実現することが難しいため、近年はディフィーヘルマン(DH)をベースとした一時鍵(Ephemeral Key)方式に移行しつつありました。TLS1.3では静的RSAを完全に廃止し一時鍵ディフィーヘルマンのみが採用されています。

5.2.3 サイドチャネル攻撃

完全前方秘匿性の問題以外にも、純粋にアルゴリズム的な暗号理論の範囲を超える攻撃手法が多数知られています。その一つがサイドチャネル攻撃とよばれる一連の攻撃手法です。サイドチャネル攻撃では暗号処理を行っているコンピュータの物理的特性を外部から観測することで内部の情報を読み取ることを試みます。機器の内部で暗号処理を行う際の処理時間、消費電力の変化、外部に発生する電磁波、音、熱などの物理的変化を測定し解読のヒントとします。このような情報の正規の出入り口ではない「サイドチャネル」を利用することからサイドチャネル攻撃と呼ばれています。

通常、ハードウェア型の攻撃のうち攻撃対象を破壊しない攻撃をサイドチャネル攻撃に分類します。サイドチャネル攻撃の手法はこれからもさまざまなものが出現すると予想されますが、これまでに知られているものとしては次のようなものがあります。

1) タイミング攻撃

暗号処理をする機器の入力値を変えることによる処理時間の違いを計測して鍵情報などを推測する攻撃手法です。暗号処理の中でも特に公開鍵の処理 は単純に原理的なアルゴリズムを実現するだけだと処理時間が大幅に異なり比較的容易に鍵が推測できてしまうリスクが知られています。ソフトウェア的に注意深い実現をすることで処理時間を平準化することができることが知られており、多くの暗号ライブラリなどではそうした対策がとられています。しかし、ハードウェア的に詳細なタイミングが計測できる環境ではさらに精度の高い対策が必要です。

2) 故障利用攻撃

外部からの強い電磁ノイズなどによって故意に誤動作や故障を起こし、正常動作との差異を解析する攻撃です。

3) 電力解析攻撃

機器の消費電力、電流の変化を計測することで暗号鍵などクリティカルな情報を推測する攻撃です。

4) 電磁波解析攻撃

電力解析とは逆に、機器の発生する電磁波ノイズを解析することでクリティカルな情報を推測する攻撃です。

5) キャッシュ攻撃

処理内容によりキャッシュのヒット率などの変化を観測する攻撃です。マルチコアのサーバなどでは、同一チップ上のコアの動作状況を観測することが可能の場合があるので注意が必要です。処理前にあらかじめキャッシュ状況を揃えておくなどのソフトウェア的な対策もある程度可能です。

6) 音響解析攻撃

機器の発生する音響ノイズを解析すること処理内容を推測する攻撃です。

5.2.4 ハードウェア層の攻撃手法

サイドチャネル攻撃は、電気的特性などを使ってソフトウェアの動作状況を解析する攻撃手法ですが、ハードウェア的な手法によれば、さらに踏み込んだ解析が可能です。例えば、光学、レーザ顕微鏡などで直接ICチップ上の回路を読み取ることが可能です。また、回路の解析だけでなく、配線の切断、接続など改造、バス上のデータの読み取りなどの攻撃も可能です。あるいはICチップ上へのレーザ照射によってソフトウェアの動作を強制的に意図しない方向に分岐させ、データを読み取ることや、ソフトウェア的には隠蔽された情報を取得することも可能となります。

こうしたハードウェア層の攻撃に対しては、回路の配線を見えなくするためのシールド配線層を設ける、レーザ照射を検出するセンサの実装、チップ破壊の検出機構などの対策が知られています。また、そのような攻撃に耐える性質を「耐タンパー性」と呼びます。

5.3 鍵管理

公開鍵暗号によるディジタル署名では、署名鍵を外部に知られることなく第三者がその署名の正当性を検証することができます(3.6.5 デジタル署名参照)。この技術を利用して、物理的なハードウェアユニット上に署名鍵を安全に保存し署名検証機能を提供することでハードウェアレベルで安全なアイデンティティ管理を実現することができます。

このような鍵管理機能を物理的にも安全に実装した鍵管理のためのハードウェア的な装置をHSM: Hardware Security Management)と呼んでいます。従来HSMのはハードウェア的にも極めて堅牢な実装がされており高い耐たんパー性を実現し大規模なサーバシステムの一環として利用されてきました。近年ではこのような鍵管理機能を軽装にICチップ上に実現した「セキュアエレメント」と呼もばれる鍵管理チップも広く使用されるようになっています。そのようなデバイスはIoTデバイスのような軽量なデバイスのアイデンティティを安全に管理するためにも利用領域が広がっています。

このようなセキュアエレメントでは、公開鍵ペアは工場での製造工程で安全に管理され封印されるか、チップに鍵生成機能も提供されていて、鍵のライフサイクル全体でプライベート鍵は一切チップの外に出すことなしに機能を果たすことができるようになっています。また、こうした装置やチップは通常耐タンパー性も実現していて、ハードウェアレベルでの安全性も保証されています。

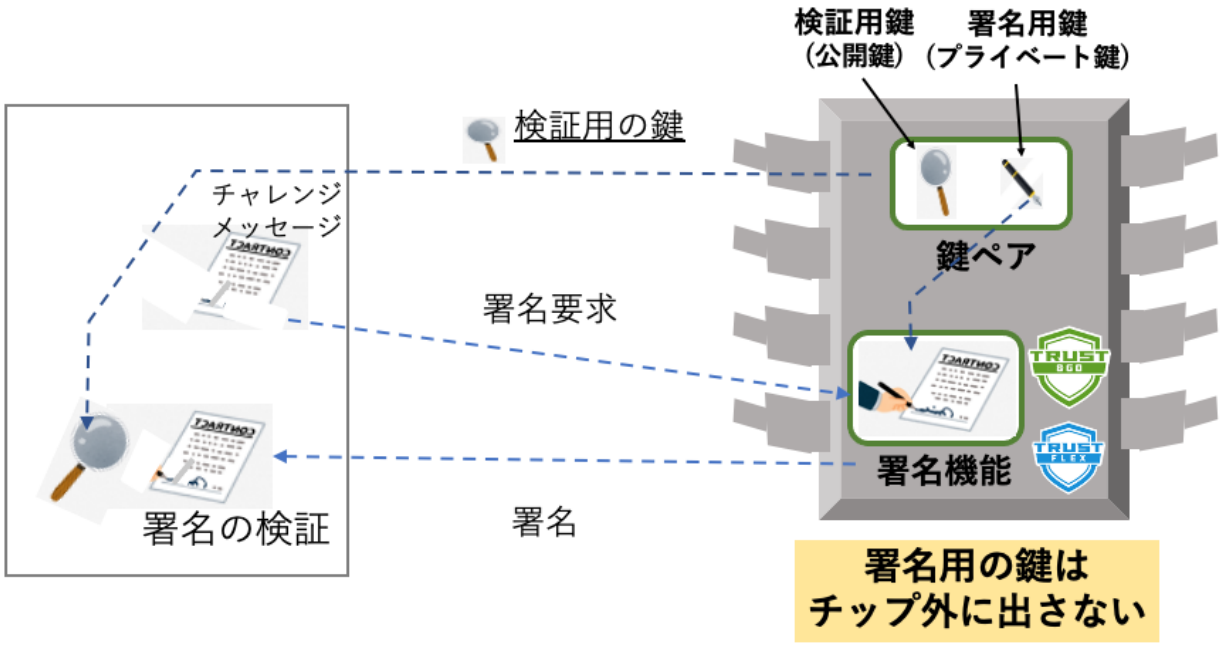

図　セキュアエレメントによる鍵管理の原理

図はセキュアエレメントによる鍵管理の原理を示します。セキュアエレメント内には署名鍵と検証鍵が保存されていて、署名鍵はチップ外から参照することはできないようになっています。署名検証をする場合は事前に該当チップから検証鍵を得ておき、適当なチャレンジメッセージをエレメントに送ります。エレメントは自分の署名鍵を使ってメッセージに対応する署名を生成しそれを返送します。受け取った側は用意してある検証鍵を使って送られてきた署名を検証します。このように署名鍵自身は一切外部に参照させること無しに自分が正しい署名者であることを証明することができます。

HSMアクセスのためのAPIは標準化も進められています。RSA社のPKCSの一環としてもPKCS #11が早い段階からの標準として普及しています(4.2 PKCS (Public-Key Cryptography Standards) 参照)。PKCS #11は比較的大掛かりなサーバ向けの鍵管理とそれに関連した暗号化処理など一連の処理に関するAPIを標準化しています。

一方、セキュアチップとその周辺サービスの標準化の例としてはTPM(Trusted Platform Module)が ISO/IEC 11889 があり, デジタル著作権管理(DRM: digital rights management)やWindowsのアクセス管理用などとして広く使われています。

これらの例では単純な鍵管理機能だけではなく、周辺のさまざまな機能、サービスが取り込まれています。一方で、IoTデバイスのように比較的小規模なデバイス向けに鍵管理だけに特化した軽装のチップも広く利用されています。

5.5 インシデント管理

システムにおいて脆弱性を撲滅することは永遠の課題です。現実的には、問題の発生や潜在的問題を発見した場合の速やかな対処ができる体制、影響を最小限に食い止める体制が大切です。

インタネットの世界では多くのケースはユーザは不特定多数です。脆弱性問題を発見した個人、機器やソフトウェアのベンダー必は必要とするユーザにその情報を確実に届ける必要がありますが、個別に対応していたのではそれは極

めて困難です。ユーザにとっても、使用しているたくさんの機器やソフトウェアが複雑に関係しあっているなかで必要な脆弱性情報が常に確実に把握しなければいけません。情報はできるだけ早く伝達しなければいけませんが、不確実な情報、不正確な情報はかえって影響を大きくしたり、問題を複雑にしてしまったりすることも考えられます。

米国では早い段階からそうした必要性が認識され、国土防衛政策の一環としてNVD(National Vulnerability Database)の運用が開始されました。NVDでは、発見された各脆弱性インシデントにはCVE(Common Vulnerabilities and Exposures)と呼ばれるIDを付与し、データベースとして誰でもアクセスできる環境で公開しています(https://nvd.nist.gov/)。

一方、コンピュータウィルスやウォームようなマルウェアの脅威についても早い段階から認識されていました。米カーネギーメロン大学は米国連邦政府の委託でCERT/CC(Computer Emergency Response Team Coordination Center)を立ち上げ、情報収集、解析、公開などの活動を行っています(後にCSIRT: Computer Security Incident Response Teamと改名)。現在では各国にNatioanl SCERTが組織されており、日本でもJPCERT/CC(一般社団法人JPCERTコーディネーションセンター)が活動を続けています。

IPA(独立行政法人 情報処理推進機構)とJPCERT/CCの共同でJVN (Japan Vulnerability Notes) が運用され、NVDの情報とともに日本国内の製品開発者の脆弱性情報を受付け、対応状況をとりまとめ、公開しています。