情報セキュリティ技術動向調査

タスクグループ報告書

(2011 年上期)

2011 年 9 月

独立行政法人 情報処理推進機構
セキュリティセンター

目次

序 2011 年上期の技術動向 - 今日のセキュリティエンジニアリングの話題

宮川 寧夫

1 背景

情報セキュリティ技術動向調査 TG（タスクグループ）[1]は、その第 6 回目の会合を 2011 年 6 月 24 日に開催した。情報セキュリティのエンジニアリングの分野において注目すべき動向を発表しあい、発表内容に基づいて討議した。

本報告書は、読者として IT 技術者を想定している。IT 技術者によるエンジニアリング活動の中で、注目すべき動向もしくは話題を各委員独自の視点から紹介・解説していただいて本書を取りまとめた。

2 目的

本書は、読者として主に情報処理技術者を想定している。情報技術の「アーリーアドプター（Early Adopters）」や「アーリーマジョリティ（Early Majority）」[2] と呼ばれるような先進的な技術に関心を寄せる読者に、そこで想定されている用途に関する情報を添えて提供する。これによって、各自もしくは各組織における情報セキュリティに関するエンジニアリングの課題の解決に資するものとしたい。

なお、先進的な技術を題材とするため特定の実装を紹介することがあるが、オープンな仕様が存在している場合、主にその仕様に基づいて解説するように留意している。

3 概況

今期の出来事として、楕円曲線暗号の署名方式 ECDSA における脆弱性が注目された。（1 章）

Linux カーネルに関しては、TPM（Trusted Platform Module）を用いた Linux Integrity Subsystem の目標等の大枠を説明し、現状の実装について整理する。（2 章）

IPv6 に関しては、イントラネットに配備する際の不正端末接続に関する問題と移行技術に起因する境界セキュリティの問題に対する標準化の場における議論を紹介する。（3 章）

インターネットインフラストラクチャについては、DANE（DNS-based Authentication of Named Entities）の話題を採りあげる。これは DNS と関連づけられた公開鍵の配送技術となる。（4 章）

今日の Web アプリケーションについてのセキュア・プログラミングに関して、Ajax を扱うブラウザのセキュリティ機能（5 章）と、IETF（Internet Engineering Task Force）における Web 2.0 セキュリティ関連の標準の策定状況（6 章）を採りあげる。今日の Web アプリケーションは、ブラウザ内で動作する Ajax を重視するようになってきていると共に、

複数の源泉からコンテンツをロードして「マッシュアップ」するようになってきている。音楽 DJ が複数の楽曲を組み合わせて新たな音楽を創造する過程になぞらえて、ソフトウェアを API 呼び出しによって組み合わせて構成することが「マッシュアップ」と呼ばれている。

クラウドコンピューティングに関するおける話題として、今期、発行された NIST 文書に見られる論点を紹介する。（7 章）そして、これまでアイデンティティ管理の文脈で普及してこなかったプロビジョニング API がクラウドコンピューティングの用途で有力ベンダーによって再度、策定されつつある。これは SCIM(Simple Cloud Identity Management)という仕様であるが、この動向について解説する。（8 章）

参照

[1] 情報セキュリティ技術動向調査 TG（タスクグループ）
http://www.ipa.go.jp/security/outline/committee/isec_tech1.html

[2] ジェフリー･ムーア（著）, 川又 政治（訳）,「ハイテク・マーケティング」『キャズム』, 翔泳社（2002）pp. 11-38

1. 楕円曲線暗号の署名方式 ECDSA における脆弱性の脅威

金岡 晃

1.1. 背景

現在利用されている公開鍵暗号のほとんどは RSA 暗号方式であると言っても過言ではないが、標準化がされている公開鍵暗号の方式は他にも存在する。特に楕円曲線暗号（ECC）は同レベルの強度を持つ RSA 暗号と比較して鍵長が短いことに加え処理の速さも特徴となっていることからポスト RSA 暗号として注目を浴びている。

本技術調査では 2010 年上期に「楕円曲線暗号の整備動向」を報告した[1]。ECC は Windows の CNG（Cryptography Next Generation）や、著名な暗号ライブラリ・SSL/TLS ミドルウェアである OpenSSL といった、広く使われるプラットフォームで採用がされている。また 7 月に正式公開された Java Platform SE（Standard Edition）7 では JCE（Java Cryptographic Extension）プロバイダで ECC に対応するなど利用プラットフォーム整備が進んでいる。

ECC は単一の暗号アルゴリズムではなく、楕円曲線上の離散対数問題を安全性の根拠とする暗号方式の総称である。そこには、楕円曲線上で DH（Diffie-Hellman）鍵共有を行う ECDH や楕円曲線上で DSA（Digital Signature Algorithm）を実現する ECDSA、楕円曲線上で MQV（Menezes–Qu–Vanstone）方式を実現する ECMQV などが含まれる。

2011 年 5 月に ECC の代表的電子署名方式である ECDSA について、OpenSSL の実装に脆弱性があることが報告された[2][3]。OpenSSL はさまざまなシステムで利用されているため脆弱性が発見されるとその影響が大きい。しかし 2011 年 7 月 29 日現在、この脆弱性対応は行われていない。

本調査ではこの脆弱性の概要と脆弱性を利用した攻撃の現実的な脅威について報告し、利用者が OpenSSL で ECC を利用するための判断材料を提供することを目的とする。

1.2. 脆弱性概略

ECDSA の署名生成では最初にランダムな値を生成する。これを nonce と呼ぶ。その nonce とプライベート鍵、署名対象データを用いて署名生成を行う。nonce が漏えいすると署名データから簡単な演算で容易にプライベート鍵の計算が可能になるため、nonce の保護もプライベート鍵の保護には重要となる。

ECDSA の署名計算時には楕円曲線上の点を整数倍する演算（スカラ倍算）が行われる。その演算方法は、利用される楕円曲線などの特徴を利用した個別の効率の良い演算方法が存在している。米国国立標準技術研究所（NIST）や IETF（Internet Engineering Task Force）、SECG (The Standards for Efficient Cryptography Group) で標準化されている楕円曲線パラメータは楕円曲線を構成する有限体の標数によりふたつに大別される。ひとつ

は標数に大きな素数を用いる素体上の楕円曲線と、もうひとつは標数に 2 を用いてその拡大体上で構成する楕円曲線である。OpenSSL では、この素体と標数 2 の拡大体でスカラ倍算の演算構成が異なっており、この脆弱性は 2 の拡大体を用いた曲線の演算実装に存在する（図 1)。

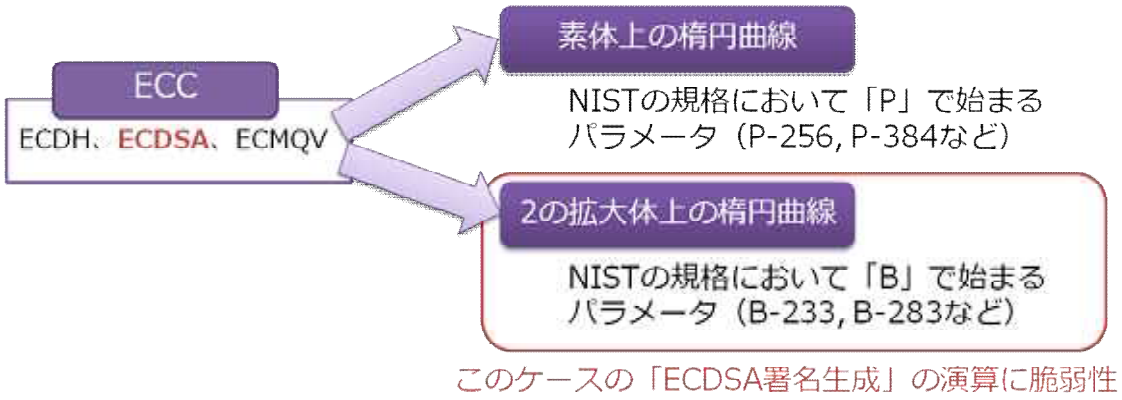

図 1

OpenSSL では 2 の拡大体を用いた曲線のスカラ倍算にモンゴメリ梯子算（Montgomery's Ladder）が採用されており、その実装では nonce データ上位何ビット目に初めて 1 が表れるかという情報をもとに演算が行なわれている部分が存在する。そこは、nonce の値に従ってわずかながら内部演算の回数が変わるようになっており、その結果演算処理時間が異なってくる。おそらく無駄な内部演算の回数を削減することで演算の効率を高めるための処理だと思われるが、この処理時間の異なりが脆弱性を生むことになっている。署名生成の演算処理を計測しておくことで、nonce 情報の一部を取得可能になり、nonce 値を取得可能なタイミング攻撃を可能にしてしまう脆弱性となる（図 2)。この脆弱性は 2011 年 5 月に Brumley と Tuveri によって指摘がされた[2]。

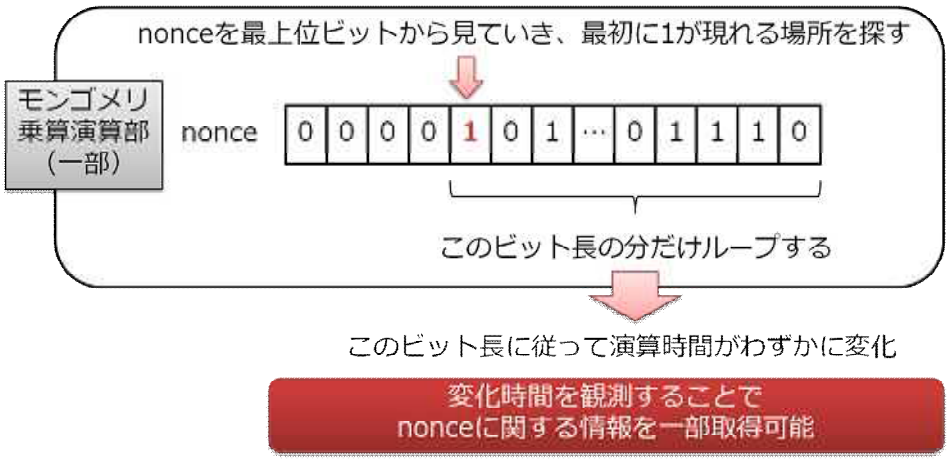

図 2

暗号演算等を行うソフトウェア・ハードウェアにおいてその動作を観測することで暗号で用いられる機密情報を得る攻撃はサイドチャネル攻撃と呼ばれ、タイミング攻撃はその攻撃の一種である。

1.3. 攻撃方法概略

攻撃は「署名データの収集」と「プライベート鍵の解析」のふたつから構成される。

「署名データの収集」では攻撃者は署名生成をする機器の署名データとその処理時間を観測する。そして署名データを多く集め、その中で署名時間が少ないものを一定数抽出する。「プライベート鍵の解析」では抽出された署名データを変換し、Lattice 攻撃と呼ばれる攻撃を実施する。

「署名データの収集」時に行なわれる処理時間の観測には、直接 CPU クロックを見て処理時間が観測可能な場合はその処理時間を測定する。

直接観測が不可能な場合には、SSL/TLS のクライアントとして SSL/TLS のハンドシェイクにおける CllientHello 送信開始から ServerKeyExchange 受領までの時間を計測する。ServerKeyExchange には署名生成がされたデータが入るために、nonce 値によって処理時間が変化することから間接的な計測が可能となる（図 3)。

一方「プライベート鍵の解析」は観測環境に依存せず攻撃者の任意の環境で実施が可能である。

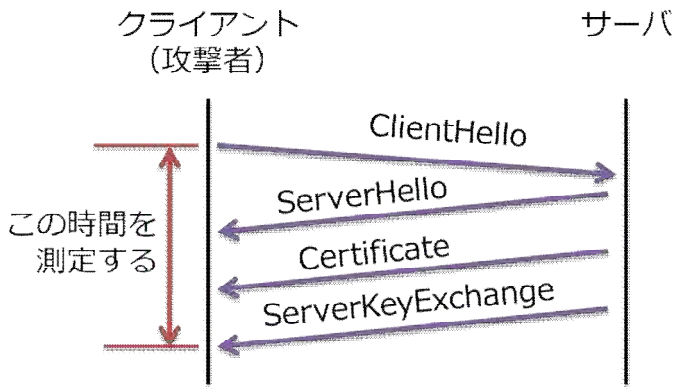

図 3

1.4. 攻撃実験環境

脆弱性発見者らは楕円曲線パラメータ B-163 を用いて攻撃実験を行った。「署名データの収集」は以下の 3 つのケースで実験が行なわれた。

(1) 署名生成機器のプロセスレベルまで観測可能な状態での直接観測
(2) 署名生成機器を SSL/TLS サーバ・クライアントとして間接観測
(3) 署名生成機器とスイッチで接続された機器による間接観測

脆弱性発見者らの実験結果では上記 2 の環境で 16384 個の署名データを集めるまで数分という結果が得られている。観測と署名データ収集後の Lattice 攻撃に関する詳細な攻撃時間の評価は行なわれていないが、あるケースでは数分で完了したものがあるとしている。

1.5. 現実的な脅威

脆弱性発見者らが論文[2] 上で示している攻撃の実験環境は、ネットワークを介しているとしているが 1 台スイッチを介して SSL/TLS ハンドシェイクをしているもので、リモート接続としては通信の距離が短く通信遅延が非常に少なく、攻撃者に有利な環境であると言える。しかしその環境においても既に通信遅延やその誤差による判定ミスがおこっている。つまり、攻撃者に有利なリモート接続環境であっても、nonce の処理時間の差が非常に小さいために通信の遅延や誤差によりその差が攻撃者から明確に判断することが難しくなっている。

より一般的な攻撃環境である、他のネットワークからの攻撃を想定すると、通信の影響による誤差はより高くなることは容易に考えられ、処理時間の差異は通信時間の遅延誤差に埋没してしまい攻撃が失敗する可能性が高いと考えられる。

また、本攻撃は 2 の拡大体上の曲線パラメータを選択したときの脅威であり、素体のパラメータを用いれば攻撃は回避可能である。さらに、Windows 環境では 2 の拡大体上の曲線パラメータは対応していない。

これらのことから、2 の拡大体上の曲線パラメータを選択したとしても攻撃成功率は低いと予想され、さらに ECC による暗号システム設計時や移行時に素体の曲線パラメータを選択すれば攻撃は回避可能であることから、現実的な脅威は低いと言える。

以上

参考文献

[1] 金岡 晃, "楕円曲線暗号の整備動向", 情報セキュリティ技術動向調査（2010 年上期）, 2010, <http://www.ipa.go.jp/security/fy22/reports/tech1-tg/a_01.html>

[2] B.B. Brumley, N. Tuveri, "Remote Timing Attacks are Still Practical", Cryptology ePrint Archive: Report 2011/232, 2011, <http://eprint.iacr.org/2011/232>

[3] Vulnerability Note VU#536044, 2011, <https://www.kb.cert.org/vuls/id/536044>

2. Linux Integrity Subsystem の現状

面 和毅

2.1. はじめに

2010 年下期から 2011 年上期にかけて、Trusted Keys や EVM など、Linux Integrity Subsystem に関わる実装が次々と LKLM（Linux Kernel メーリングリスト）及び LSM-ML（Linux Security Module メーリングリスト）に提出されている。その基礎は、TCG（Trusted Computing Group）により仕様が決められ、最近の PC に搭載されている TPM（Trusted Platform Module）と呼ばれるチップであり、これを用いた技術である Linux-IMA（Linux-Integrity Measurement Architecture）に関しては、2009 年上期の調査報告[1] にまとめた。

本稿では TPM をベースとした Linux Integrity Subsystem の目標などの大枠を説明し、現状の実装の整理を行う。

2.2. Linux Integrity Subsystem の目標

一般的な情報セキュリティの基本理念としては、いわゆる CIA の 3 大基本理念として

- 機密性（Confidential）
- 完全性（Integrity）
- 可用性（Availability）

がある。更に、これらに加えて

- 真正性（Authenticity）
- 責任追跡性（Accountability）
- 信頼性（Reliability）

もセキュリティの理念を説明する際に加えられることがある。

Linux Integrity Subsystem は、これらのセキュリティ理念のうち、「完全性」と「真正性」を満たすための実装となっている。その他の「機密性」等に関しては、強制アクセス制御を提供する SELinux や SMACK（「情報セキュリティ技術動向調査（2008 年上期）」[2] にて報告）など、別の Linux 上の実装で補完することになる。

これらの理念に対する脅威としては、以下のものが挙げられる。

- リモート攻撃
  - Virus やその他の悪意のあるソフトウェアによるファイルの変更
- ローカル攻撃
  - 悪意のあるシステムオペレータによる不正な変更

- **オフライン攻撃（ハードウェアやファイル単体などの盗難、CD/USB ドライブでのブートでの HDD 内部の不正な変更など）**

Linux Integrity Subsystem **は、これらの攻撃が行われた際に、攻撃が行われたことを検知するためのものである。そのため、TPM と連携し、通常と異なったシステム起動方法や、異なったハードウェア上でのデータの不正変更も検知する事が目標となっている。**

2.3. Linux Integrity Subsystem **の実装**

前述の目標を Linux **上で達成するために、**Linux Integrity Subsystem **では、**Kernel **内に以下のコンポーネントを実装する予定となっている。**

(1) IMA **（**2.6.30 **に取り込まれた）**
(2) Trusted and Encrypted Keys**（**2.6.38 **に取り込まれた）**
(3) IMA-Appraisal**（**LKML/LSM-ML **にて議論中）**
(4) IMA-Appraisal-Signature-Extension**（開発中）**
(5) IMA-Appraisal-Directory-Extension**（開発中）**
(6) EVM**（**LKML/LSM-ML **にて議論中）**

以下、それぞれのコンポーネントの役割を説明する。

(1) Linux IMA（Integrity Measurement Architecture）

TPM **とカーネル上に組み込まれたモジュールを利用して、実行ファイルを測定して記録し、実行ファイルが不正に改変されていないかをチェックするものである。この機能の詳細は、「情報セキュリティ技術動向調査（**2009 **年上期）」**[1] **で説明している。**

(2) Trusted and Encrypted Keys

"Trusted and Encrypted Keys"**は、**Linux Kernel 2.6.36 **から取り込まれている。この機構により、**TPM **の機能を用いて信頼できる対象鍵を作成する事ができるようになるため、後述する** EVM **のための鍵を作成する際に用いられる。**

Trusted Key：

TPM **に結びついた鍵である。**TPM **により生成され、**TPM **内で別途生成される** Storage Root Key **（**SRK**：**2048bit **の** RSA **鍵）により暗号化される（（**TPM **の用語では「**Sealing**」と呼ばれている）。更に** Trusted Key **は、**TPM **の** Platform Configuration Register**（**PCR**：特殊な更新操作でのみ書き換え可能な記憶領域）中にも保存され、**PCR **に保存された値が合致しないと復号化（**Unseal**）されない。**PCR **中には** BIOS **や** Boot Loader**、**OS **等の整合性情報も保存されており、**Trusted Key **がこれらと結びつけられることによって、**

これらの情報が合致しない限り鍵を復号化して使用できないシステムにすることができる。

Encrypted Key：

公開鍵暗号方式よりも高速な、AES 暗号化に使用される鍵であり、TPM とは結びついていない。鍵は Kernel の乱数プールから取り出される乱数から生成され、ユーザ空間に blob としてエクスポートされる際に、Trusted Key を用いて暗号化される。Trusted Key を用いて暗号化されているため、この鍵も整合性情報が合致しない限り使用することが不可能になる。

keyctl ユーティリティ：

Trusted Key と Encrypted Key を利用できるツールである。

(3) IMA-Appraisal

Linux-IMA の延長線上にあるものであり、ローカルファイルの整合性をチェックする機構である。IMA で確認された"正しい（good）"状態のハッシュ値を、security.ima というファイルの拡張属性に保存しておき、現在のファイルのハッシュ値が異なっているかを確認する。そして、ハッシュ値が異なっていた場合には、ファイルに対してのアクセスを拒否する。

security.ima にはハッシュ値のみが入るようになっている。この実装は簡単でかつ高速なため、ファイルの変更を検出するのに便利であるが、ハッシュ値は簡単に偽造できるため、強力な整合性を提供しているとは言えないものになる。

(4) IMA-Appraisal-Signature-Extension （ファイルの電子署名）

IMA-Appraisal-Signature-Extension は、IMA-Appraisal を「真正性」の方向に拡張したものになる。この実装は、security.ima にベンダーから提供された RSA 暗号化方式による電子署名を格納する。これを変更するためには特殊な権限が必要となるため、この電子署名を照合することにより、ファイルが変更されていないと考えられる。

また、この署名されたファイルは変更は許可されず、削除か、アップデートなどによる置換のみが認められる。これらの電子署名は、ベンダーの秘密鍵を入手する必要があるため、攻撃者による偽造は困難になっている。

(5) IMA-Appraisal-Directory-Extension （ローカルディレクトリの整合性）

IMA-Appraisal-Directory-Extension は、ファイル名の変更やオフラインによるリプレイ攻撃を防ぐための機構になる。

一般的に、IMA-Appraisal はファイルの整合性と真正性を検査するために、RSA 署名（変

更されないファイルの場合）やハッシュ（変更されるファイルの場合）を使用する。しかし、これだけでは未だファイルのメタデータを変更された場合等に対しての検出をすることができない。ファイルのメタデータとは、ファイル名や、inode に含まれる情報（所有者、グループ、モードなど）である。

例えば、簡単な例として、攻撃者が"rm"コマンドの名前を"ls"と変更するだけで、システムにダメージを与えてしまう事ができる。このような攻撃を検出するために、IMA-Appraisal-Directory-Extension ではファイルのメタデータのハッシュ値も作成して保存する。

(6) Extended Verification Module（EVM）（信頼できるメタデータ）

EVM は現在 LSM/LKML にて議論が行われている最中であり、オフライン攻撃から inode メタデータ（所有者、グループ、モード、セキュリティ拡張属性など）を保護することを目的とした機構である。（EVM は 8 月 17 日に、linux の security 関連の"Next"レポジトリに正式に追加された。最終的には linux-3.2 からカーネルに取り込まれる予定である。）

EVM は HMAC-SHA1 を用いて、各 inode 上の

- security.ima （IMA のハッシュ値や、ファイルの電子署名）
- security.selinux （SELinux のラベル／コンテキスト情報）
- security.smack （SMACK のラベル情報）
- security.capability （実行可能なファイル上のファイルケーパビリティラベル）

などのセキュリティ拡張属性を保護している。

これらのセキュリティ情報は極めて慎重に扱う必要があるため、EVM では前述の Trusted Key/Encrypted key を用いて、HMAC を作成する。これにより、セキュリティ拡張属性の情報が改竄された場合の検出を行うことができる。

2.4..期の総評

今期は OS、VM 双方に関して特に新しいセキュリティ技術は見られなかった。これは、昨今の IT 技術分野が、従来のサーバ／クライアント型のシステムから、クラウド／Web アプリベースのシステムへとシフトしてきたため、技術者がこれらクラウド／Web アプリの分野のセキュリティ技術の開発に注力しているためと思われる。しかし、これは、従来の OS、VM のセキュリティがすべて開発されたということは意味しておらず、クラウドサービスを提供する基盤として OS、VM の更なるセキュリティ強化は依然として望まれているため、今後新たなセキュリティ技術が開発される可能性は充分あると思われる。

以上

参考文献

[1] **面 和毅,**「Linux **カーネルにおけるセキュリティ強化機能**」
http://www.ipa.go.jp/security/fy21/reports/tech1-tg/a_02.html

[2] **面 和毅,**「Linux **用の新しいセキュア** OS：SMACK」
http://www.ipa.go.jp/security/fy20/reports/tech1-tg/1_05.html

3. IPv6 の配備におけるイントラネットセキュリティ

太田 耕平

3.1. はじめに

2011 年 2 月の IANA（Internet Assigned Numbers Authority）における IPv4 アドレスの枯渇[1]に続き、4 月には国内の割り当てを担っている JPNIC（Japan Network Information Center）でも新たな IPv4 アドレスの割り当てがなくなったことが周知され[2]、IPv6 の利用が喫緊の課題となった。現在各組織が留保しているアドレスがなくなり次第 IPv6 への移行が必要となる。一方で、近年では端末やサーバに加え、スマートフォンやタブレット端末でも既に IPv6 接続に対応しており、初期状態で利用可能となっている。結果として、これまで移行の努力が続けられてきたバックボーンだけではなく企業、家庭等あらゆる場面で急速に IPv6 の利用が始まっている。

IPv6 は、IPv4 ではアドレスが不足するという 1992 年の調査結果[3]をうけて検討が始まり、1995 年に最初の標準化文書[4]が発行されて以来、長きにわたって開発と普及の努力が続けられてきた。IPv6 は IPv4 での経験を踏まえ、アドレス空間の拡大の他にセキュリティの強化、プラグアンドプレイによる設定の自動化等を大きな課題として開発された。また同時にインターネットプロトコルの根幹となる部分の置き換えとなるため、移行のためのシナリオと技術にも大きな努力がなされてきた。具体的には IPv4 と IPv6 の両方に対応するデュアルスタック、点在する IPv6 ネットワークをつなぐトンネル技術、およびそれらを意識することなく利用可能とする自動化技術などである。これらは明示的な手間やコストをかけることなく IPv6 技術を段階的に浸透させる、という目的に沿ったものであり、今日までの普及に大きな役割を果たしているが、一方で IPv6 の利用を管理者が把握していない要因の一つともなっている。このことは本格的な利用が始まりつつある現在「リスクを適切に把握する」というセキュリティの根幹を脅かしている。

インターネット技術は、その普及につれてセキュリティに関する考え方が大きく変化した。当初はセキュリティに関する関心は希薄であったため、IAB（Internet Architecture Board）は 1998 年にインターネットにおけるセキュリティのあり方を議論した RFC（Request For Comments）を発行し[5]、IETF（Internet Engineering Task Force）でも RFC に義務付けられているセキュリティに関する考察の記載を強化するための RFC を発行した[6]。市場においてもファイアウォールの配備は常識となり、攻撃を検知する侵入検知システム等のネットワーク境界での保護技術の利用が進んでいる。IPv6 においても IP パケットの認証と暗号化を実現する IPsec がそのセキュリティ強化の大きな柱となっている。

しかし、これらのセキュリティ対策の主要な観点は対外接続からの攻撃に焦点をあてたものであり、イントラネット（＝内部ネットワーク）の保護については十分に検討されていない。現実的なセキュリティの脅威は外部からの攻撃よりも内部の問題に起因しており、近年の情報漏えい事故等の多くはイントラネットでの対策が不十分なことによるものである。イントラネットの保護には、外部との境界の他に、その反対側の境界=端末の接続についても併せて考える必要がある。本稿では

IPv6、特に端末接続の問題と移行技術に起因するセキュリティ上の課題、および標準化の場での議論を紹介し、これからのイントラネット管理で検討すべきポイントを述べる。

3.2. 外部接続からの保護

IPv6 を前提とするネットワーク保護については RFC 4864 [7]で議論され、IPv4 で一般化している NAT（Network Address Translation）技術の功罪をセキュリティ及び運用の両面から再検討している。IPv6 では十分なアドレス数を確保することで NAT を不要とし、インターネット本来の理念である End-to-End の接続性を取り戻すことが大きな目的のひとつとなっている。しかし、今日のセキュリティ、およびネットワーク運用環境を考慮すると NAT に「期待されている」セキュリティ上の役割、およびアドレスのポータビリティ等、結果的に恩恵をうけている効果についても無視することはできない。一方で NAT が「提供しているようにみえる」フィルタ機能は、技術的には十分ではないことが改めて示されており、IPv6 に限らず既存の IPv4 でも境界での明示的なフィルタリングは欠かせない要素であることが議論されている[8]。

一方で、外部との境界という観点では、近年広く使われるようになっているトンネリング技術が挙げられる。トンネリング技術は、パケットのカプセル化によってネットワーク上の 2 点間を論理的につなぐ技術であり、カプセル化によって本来ネットワーク境界で適用されるべき様々な検査が無効化される危険性が指摘されている[16]。IPv6 では、IPv4 ネットワーク中に点在する IPv6 ネットワークをつなぎ、スムーズな移行をサポートするための技術にトンネリングが多く用いられている。これらのトンネルは IPv6 へ移行するための障壁をできるだけ下げるために、自動で構築するものもあり、結果として管理者の認識していない（＝ネットワーク境界のフィルタで保護されていない）形の外部との接続となることがある。さらには IPv4 と IPv6 という独立した経路体系を自動的につなぐトンネル技術を悪用した攻撃の可能性も指摘されており[9]、移行技術によって新たに発生する脅威がある。トンネリング技術は原則として対外アクセスであり一般的なネットワーク境界と同等のセキュリティ対策の適用が求められる。

3.3. 内部接続からの保護

IPv6 の設計にあたっては、IPv4 では外付けの DHCP に頼らざるを得なかった端末のネットワーク接続をより簡素化、自動化するために MAC アドレスと IP アドレスを関連付けるプロトコルが全面的に再設計された。しかしその基本原理に変更はなく、以下のような IPv4 と同様の脆弱性を有している。

- なりすまし可能なアドレス解決
- なりすまし可能なネットワーク設定

(1) IPv6 でのアドレス解決

インターネットプロトコルでは、接続機器に固有の物理アドレスである MAC アドレスとインターネットアクセスのための論理プロトコルである IP アドレスを ARP（Address Resolution Protocol）によって動的に関連付ける（アドレス解決）ことで、任意の機器を任意のネットワークで利用することを可能としている。この柔軟性によって特定の IP アドレスを複数の機器でシェアする負荷分散、Proxy ARP 技術等の応用が可能となっているが、一方ではアドレスの所有者についての認証等の機能がないため、なりすましと区別がつかない。IPv6 では、ARP に相当する機能は NDP（Neighbor Discovery Protocol）として設計、実装されているが、そのメカニズムは本質的に同じである。IPv6 では IPv4 での ARP の問題を踏まえ、認証の概念を備えた SEND（SEcure Neighbor Discovery）[10]が標準化されている。しかし、正しいアドレスを確認できる、ということを基本としているため、不正利用を防ぐためには管理対象の全機器がこの技術に対応している必要がある。このことは認証システムの運用につきものの鍵管理、トラストチェーンの管理といった煩雑で普及の壁となる問題に直結する。そのため管理対象の数が非常に多くなりえる IPv6 では今後の普及の見通しもたっていない。

(2) IPv6 での基本設定

IPv4 ではネットワーク接続のための基本設定には DHCP（Dynamic Host Configuration Protocol）が使用され、事実上必須要素となっている。しかし ARP 同様 DHCP にも認証の概念がないため、不正な DHCP によってにせの設定を送り込むことができる。実際に多くのネットワークで勝手に運用される DHCP サーバによってネットワークの混乱が起こっている。IPv6 でもまったく同様のリスクがあるが、IPv6 では NDP によってローカルな通信のための設定は端末自身で解決され、外部アクセスについてもルータとの連携で設定されるため、基本的な接続に DHCP が不要であり、DHCP に依存することによるリスクは軽減されている。しかし、NDP の一部であるルータ設定のための RA（Router Advertisement）はアドレス解決と同様の認証の問題がありやはりなりすましの危険性がある[11]。

3.4. まとめと現状の対策

IPv6 では IPv4 での経験を踏まえ、インターネットの当初の理念である End-to-End の原則[12]を再び実現し、今後の発展を長く支えるべく設計されている。IPv6 の導入と利用は必須の課題であり、主に移行をスムーズにするために準備された各種の技術と、Dual スタック戦略によって実質的な導入は既に始まっている。ネットワークの管理者は特に以下の項目に留意して運用・管理計画を見直す必要がある。これらの各要素は[13]でも議論されている。

- 既存のネットワーク境界での IPv6 対応フィルタリング
- 移行（トンネリング）技術によって新たに構築されるネットワーク境界の管理とフィルタリング
- 不正な端末の接続および不正な RA に対応する NDP 対応フィルタリング

現状で、IPv6 の LAN に接続する際に考慮すべき標準化文書を表 1 に、ネットワークとして検討すべきフィルタ等のセキュリティ対策を図 1 にまとめる。

表 1：IPv6 によるネットワーク構築で考慮すべき標準

RFC	議論している内容
RFC 6105 [14] IPv6 Router Advertisement Guard	不正な RA を流通させないためのフィルタ
RFC 4864 [7] Local Network protection	境界での NAT に依存しないフィルタの必要性
RFC 6092 [15] Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service	IPv6 接続機器が備えるべきセキュリティ機能
RFC 6169 [16] Security Concerns with IP Tunneling	IPv6 への移行技術に広く用いられているトンネリング技術へのセキュリティ対策
RFC 6324 [17] Routing Loop Attack using IPv6 Automatic Tunnels: Problem Statement and Proposed Mitigations	IPv6 への移行技術に広く用いられている自動トンネル技術の問題と対策

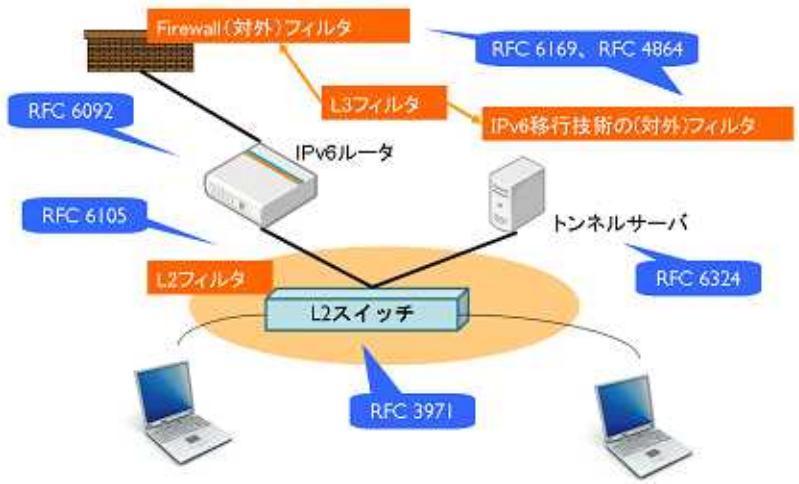

図 1：IPv6 LAN をとりまくセキュリティ対策の現状

上記のような対策のほかに、近年では、端末接続管理の必要性が浸透し、企業等のイントラネットでの普及が進んでいる。接続管理には、[18]でも紹介されている通り、市場にも複数の方式があるが、ARP および NDP を応用した通信遮断技術も実用的なソリューションのひとつとして利用されている[19]。そのため現実的な導入と接続の制御性を踏まえると、プロトコル内部のセキュリティ機構ではなく、802.1X 等の認証や、上記の応用技術のような外部から制御できる機構を利用することに大きな価値もある。

IPv6 ネットワークには大きな期待が寄せられているが、その導入にあたってはアクセス範囲を慎重に設計することが重要となる。IPv6 では豊富なアドレス空間を活用した Link-Local Address、ULA: Unique Local Address といったアドレス体系があり、それらを複数割り当てることができる。内部サーバや、クライアント、対外サービスサーバ、外部アクセスクライアントなど、役割に応じてわりあてるアドレスを使い分けることで、より確実で明確なアクセス制御を実現することが可能であり、IPv4 とは異なるより構造化されたフィルリング体系の構築も可能となるため、今後は IPv6 での適切なフィルタ設計、構築、運用に関するノウハウの構築が、イントラネット設計の重要な要素となる。

参考文献

[1] 社団法人日本ネットワークインフォメーションセンター,「IPv4 アドレスの在庫枯渇に関して」, 2011 年 2 月 3 日. http://www.nic.ad.jp/ja/ip/ipv4pool/

[2] APNIC における IPv4 アドレス在庫枯渇のお知らせおよび枯渇後の JPNIC におけるアドレス管理ポリシーのご案内, 15th, April, http://www.nic.ad.jp/ja/topics/2011/20110415-01.html

[3] P. Gross, P. Almquist, "IESG Deliberations on Routing and Addressing", RFC 1380, November 1992

[4] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture", RFC 1884, December 1995

[5] S. Bellovin, "Report of the IAB Security Architecture Workshop", RFC 2316, April 1998
（日本語訳：http://www.ipa.go.jp/security/rfc/RFC2316JA.html）

[6] E. Rescorla、B. Korver, "Guidelines for Writing RFC Text on Security Considerations", RFC 3552, July 2003

[7] G. Van de Velde, T. Hain, R. Droms, B. Carpenter, E. Klein, "Local Network Protection", RFC 4864, May 2007

[8] 太田 耕平, Glenn Mansfield, KEENI,「IPv6 ネットワークにおけるローカルネットワークの保護」, IPA 情報セキュリティ技術動向調査（2010 年上期）, 2010 年 9 月. http://www.ipa.go.jp/security/fy22/reports/tech1-tg/a_06.html

[9] 太田 耕平, Glenn Mansfield, KEENI,「トンネリング技術のセキュリティ上の弊害と IPv6 移行技術における経路ループの危険性」, IPA 情報セキュリティ技術動向調査（2010 年下期）, 2011 年 3 月. http://www.ipa.go.jp/security/fy22/reports/tech1-tg/b_09.html

[10] J. Arkko, Ed., J. Kempf, B. Zill, P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005

[11] T. Chown, S. Venaas, “Rogue IPv6 Router Advertisement Problem Statement”, RFC 6104, February 2011

[12] Aboba, B. and E. Davies, “Reflections on Internet Transparency”, RFC 4924, July 2007

[13] Enno Rey, Christopher Werny, “IPv6 Security in Enterprise LANs”, IPv6 Kongress, Frankfurt, 12-13 May 2011

[14] E. Levy-Abegnoli, G. Van de Velde, C. Popoviciu, J. Mohacsi, “IPv6 Router Advertisement Guard”, RFC 6105, February 2011

[15] J. Woodyatt, Ed., “Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service”, RFC 6092, January 2011

[16] S. Krishnan, D. Thaler, J. Hoagland, “Security Concerns with IP Tunneling”, RFC 6169, April 2011

[17] G. Nakibly, F. Templin, “Routing Loop Attack Using IPv6 Automatic Tunnels: Problem Statement and Proposed Mitigations”, RFC6324, August 2011

[18] **馬場 達也**, 「**検疫ネットワーク技術の標準化動向**」, IPA **情報セキュリティ技術動向調査**（2010 **年上期**）, 2008 **年** 9 **月**.
http://www.ipa.go.jp/security/fy20/reports/tech1-tg/1_03.html

[19] **太田 耕平**, 「**イントラネットセキュリティをめぐる技術動向**」, IPA **情報セキュリティ技術動向調査**（2010 **年上期**）, 2008 **年** 9 **月**.
http://www.ipa.go.jp/security/fy20/reports/tech1-tg/1_02.html

4. DNS を用いた公開鍵の配送技術 ー DANE

木村 泰司

4.1. はじめに

Web サービスにおける安全な通信を実現するために TLS（Transport Layer Security）プロトコル[1]では通信データの暗号化と通信相手の認証が行われている。本稿で取り上げる DANE（DNS-based Authentication of Named Entities）は、この TLS における通信相手の認証のために、認証局ではなく DNS を使う仕組みである。本稿では、2011 年度上半期のインターネットにおけるセキュリティ技術の動向として DANE を紹介する。

DANE は、2010 年 8 月頃、KIDNS（Keys in DNS）と呼ばれ、IETF（Internet Engineering Task Force）のメーリングリスト[1]で議論が始まった仕組みで、2010 年末にワーキンググループが設立され、プロトコルの策定に向けた議論が活発化した。DANE は、DNS を使って、電子証明書を配送したり電子証明書とサーバとの関連付けの正しさを確認したりできる仕組みで、電子証明書（以下、証明書と呼ぶ）を使ったサーバ認証のために使うことができる。DNS のゾーン管理者がこのリソースレコードを登録するため、原理的には、サーバのドメイン名の正しさを確認でき、認証局のような第三者証明機関を使う必要がない。

電子証明書利用して通信相手の認証を行う点はこれまでの TLS の使われ方と変わらないが、電子証明書や鍵の確認のための信頼構造は、認証局を使った仕組みと DANE では大きく異なる。DANE はまだ提案と議論が進められている段階ではあるが、認証の信頼構造という意味で大きな変化をもたらす可能性がある。本稿では背景や仕組みの概要を述べた上で、DANE がもたらす変化について考察する。

4.2. DANE の背景と現状

DANE の議論が開始された背景には、DNS ルートゾーンへの DNSSEC の導入が挙げられる。DNS ルートゾーンに DNSSEC が導入されトラストアンカーのデータ（KSK のハッシュ値）が IANA[2]の Web ページで公開されたのは 2010 年 7 月 15 日である[2]。このことで、IANA のトラストアンカーを利用することで、様々なドメイン名の DNSSEC のリソースレコードの正しさを検証できるような基点ができたことになる。この DNSSEC によるセキュアな情報伝達の仕組みを応用し、ドメイン名を識別子とする通信相手の認証手段として検討し始められたのが DANE である。2010 年 8 月にメーリングリスト Keyassure が設置された時に、現在の DANE WG のチェアである Warren Kumai 氏によって投稿された

[1] Keyassure ML (https://www.ietf.org/mailman/listinfo/keyassure)で議論が行われていたが、2011 年 7 月現在は DANE ML (https://www.ietf.org/mailman/listinfo/dane)に移行している。

[2] The Internet Assigned Numbers Authority (IANA), http://www.iana.org/

Problem Statement にこの背景をうかがうことができる[3]。その後 2010 年 12 月に DANE WG が設立され、2011 年 7 月現在は、この WG で以下の 2 つのドキュメントが議論されている。

- Use Cases and Requirements for DNS-based Authentication of Named Entities (DANE) [3]
  DANE を使った TLS サーバ認証とクライアント認証の考え方を整理するために、サーバやクライアント、サーバのアウトソース先や認証局といった要素を登場人物になぞられて説明した文書である。

- Using Secure DNS to Associate Certificates with Domain Names For TLS [4]
  ドメイン名に証明書を関連づけるリソースレコードである TLSA の仕様を記述した文書である。以前は TLS だけでなく S/MIME のためのリソースレコードも記述されていたが 09 版で削除された。

DANE WG の趣意書によると、2011 年後半までに現在議論されている TLS および DTLS と IPsec 向けの仕様が検討されることになっている[5]。

4.3. DANE で使われるリソースレコード

DANE では、ドメイン名にサーバ認証に使われる証明書や鍵の情報が関連づけられていることを示す"Certificate Association"が重要な役割を持っている。この関連づけは、DNS のゾーン管理者によってゾーンへのリソースレコードの登録を通じて行われ、特定のドメイン名への証明書などの関連づけが担保されることを意味している。この関連づけを担保するため、DANE では DNSSEC を利用することが実質的に前提となっている[4]。

関連づけできるものには、証明書のフィンガープリント、証明書自体、そして証明書に含まれる公開鍵があり、いずれも DNS サーバで TLSA と呼ばれるリソースレコード（以下、TLSA レコードと呼ぶ）に格納される。TLSA とは TLS Association の略で、TLS における Certificate Association である。図 2 に TLSA レコードの例を示す。

---

[3] Warren Kumai 氏の Keyassure ML への Problem Statement に関する投稿 http://www.ietf.org/mail-archive/web/keyassure/current/msg00000.html）

[4] draft-ietf-dane-protocol-09 では DNSSEC を使って保護される必要があるかも知れない（MAY）と記述されているが、DNSSEC 以外の保護方式は想定外とされており、実質的に DNSSEC が前提となっている。

(1) エンドエンティティ証明書のフィンガープリントのリソースレコード
_443._tcp.www.example.com. IN TLSA (
1 1 5c1502a6549c423be0a0aa9d9a16904de5ef0f5c98
c735fcca79f09230aa7141)

(2) 認証局証明書のリソースレコード
_443._tcp.www.example.com. IN TLSA (
2 0 308202c5308201ada00302010202090...)

図 2：TLSA レコードの例 [3]

(1)はエンドエンティティ証明書のフィンガープリントのリソースレコードである。図 2 の例では、www.example.com というドメイン名のホストに TCP の 443 番ポートに接続するために使われるリソースレコードが示されている。「エンドエンティティ」と記述されているが、ここでは TLS サーバを意味している。TLSA レコードのリソースデータは”5c150”で始まる文字列で、TLS サーバの証明書のハッシュ値（フィンガープリント）を示している。(1)の 2 行目の最初の「1」はエンドエンティティ、つまりサーバ証明書であることを示しており、次の「1」は証明書の SHA-256 のハッシュ値が入っていることを示している。

(2)は認証局証明書のリソースレコードである。(1)と同様に www.example.com というドメイン名のホストに TCP の 443 番ポートで接続するために使われる。(2)の 2 行目の最初の「2」は認証局証明書であることを示しており、次の「0」は”30820”で始まるリソースデータが、証明書のハッシュ値ではなく証明書そのものであることを示している。

TLSA リソースレコードとして登録できる値の種類とその使われ方は”Using Secure DNS to Associate Certificates with Domain Names For TLS”の”4. Semantics and Features of TLSA Certificate Types”に示されている[3]。

4.4. TLSA を使ったサーバ認証

DANE の TLSA を使ったサーバ証明書の検証、すなわちサーバ認証は次のようになると考えられる。はじめに https で www.example.com に接続しようとする TLS クライアントは、「_443._tcp.www.example.com」の TLSA を DNS で検索する。次に TLS で www.example.com に接続し TLS のプロトコルを使ってサーバ証明書を受け取る。(1)の場合には「www.example.com」のサーバ証明書の fingerprint と TLSA のリソースデータを比較し、一致している場合にサーバ証明書は有効であるとみなされる。つまり DNS でドメイン名に関連付けられた”Certificate Association”を使って、www.example.com と通信して得られたサーバ証明書の正しさを確認しているのである。

(2)の場合には、「www.example.com」のサーバ証明書と TLS のプロトコルを使って得られた証明書などを用いて証明書検証を行うが、その証明書チェインの中に(2)の TLSA の認

証局証明書と一致したものがあれば、サーバ証明書が有効であると判断される。ここでは認証局証明書を記述しているが、この認証局は DNS のゾーン管理者が立ち上げた認証局でよい。DNS を通じて適宜証明書を伝えることができるため、予めクライアント側に認証局証明書がインストールされている必要がない。

DANE では、ドメイン名の正しさを確認することを目的としたサーバ認証のために、第三者によって運用されている認証局が必要ないことがわかる。

4.5. 考察

DANE は TLS のサーバ認証において、第三者としての認証局を必要としない仕組みである。このことについて 2 点考察したい。

1 点目は、TLS における信頼構造についてである。従来の認証局を用いたサーバ認証と DANE によるサーバ認証とでは、認証するプログラムに必要となる信頼の先が異なる。この違いを図に示す。

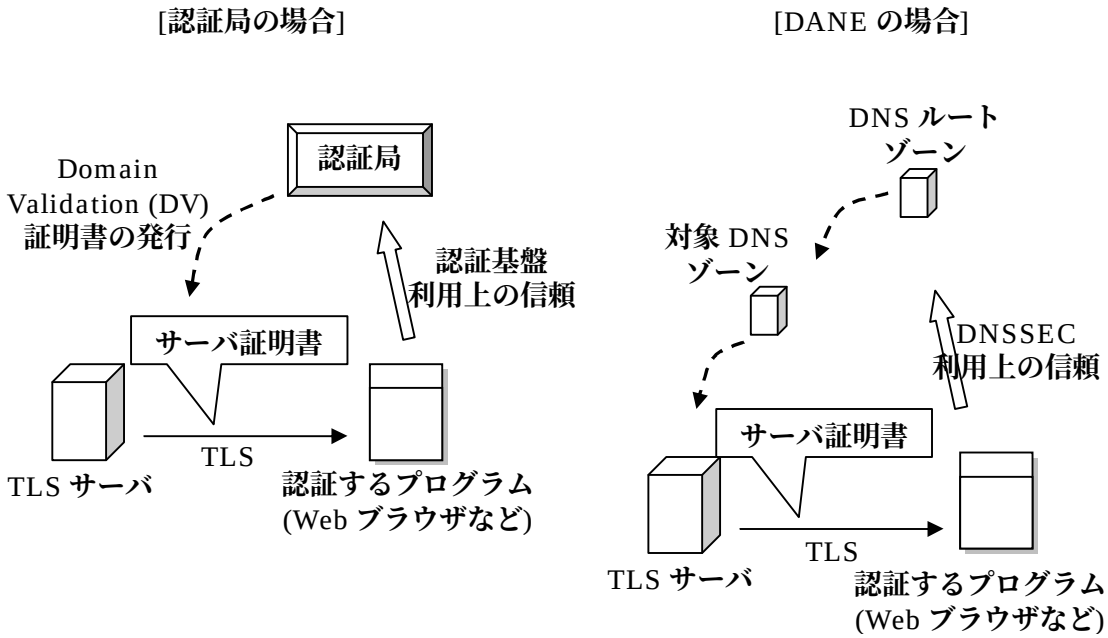

図 3：Domain Validation 証明書と DANE の信頼構造の違い

認証局によってドメイン名の正しさが確認され発行されたサーバ証明書は Domain Validation(DV)証明書と呼ばれる。DV 証明書を使ったサーバ認証を行うためには、Web ブラウザなどがサーバ認証を行う際、認証局を中心とした認証基盤に信頼をおくことになる。DANE の場合には、TLS サーバのドメイン名を登録しているゾーンの運用者がドメイン名とサーバ証明書との関連付けを行うため、ドメイン名の正しさを担保する構造が DNS に直結している。DNS でドメイン名と TLSA を正しく調べることができれば、正しいサー

バであると見なされるのである。信頼構造としては、DNS ルートゾーンが単一の信頼のポイントになると考えられる一方で、認証の対象となる TLS サーバが登録された DNS ゾーンに至るまでの一連の DNSSEC の運用が正しく行われている必要があり、信頼性を担保する仕組みとしては構造が複雑になる可能性がある。

2 点目は、Extended Validation 証明書（EV 証明書）のようなサーバとドメイン名以外の情報の関連付けである。例えば https を使って銀行や政府の Web サーバに接続する際、確認したいことがドメイン名の正しさだけでなく、むしろ Web サーバの運用組織の方が重要であることがある。DANE はドメイン名の正しさを担保するが、ドメイン名が割り当てられた組織やその組織がサーバを運用していることを担保する仕組みではない。

今後、Web サーバの認証に期待されること、すなわちドメイン名の正しさなのか運用組織の違いなのかによって DANE が使われる場面が違ってくると考えられる。また、DANE の実装や普及によって DNSSEC がより厳格に運用される必要が出てくる可能性があり、DNS サーバの運用者に求められる業務が変わってくる可能性がある。DNSSEC の運用には鍵の管理や導入のステップを検討する必要があるが[6][7]、DANE では、更にリソースレコードの登録を正しく行うための業務が加わることになる。

以上

参照文献

[1] “The Transport Layer Security (TLS) Protocol Version 1.2”, T. Dierks, E. Rescorla, August 2008, RFC5246, http://www.ietf.org/rfc/rfc5246.txt

[2] “Status Update, 2010-07-16”, ICANN, VeriSIgn, Jul 2010, http://www.root-dnssec.org/2010/07/16/status-update-2010-07-16/

[3] “Use Cases and Requirements for DNS-based Authentication of Named Entities (DANE)”, R. Barnes, http://tools.ietf.org/html/draft-ietf-dane-use-cases

[4] “Using Secure DNS to Associate Certificates with Domain Names For TLS”, P. Hoffman, J. Schlyter, http://tools.ietf.org/html/draft-ietf-dane-protocol

[5] “DNS-based Authentication of Named Entities (dane) - Charter”, IETF, http://datatracker.ietf.org/wg/dane/charter/

[6] “DNSSEC 導入に当たって”, DNSSEC Japan, Nov 2010, http://dnssec.jp/wp-content/uploads/2010/11/20101122-techwg-DNSSEC-deployment.pdf

[7] “DNS サーバ DNSSEC 導入鍵管理チェックリスト”, DNSSEC Japan, Apr 2011, http://dnssec.jp/wp-content/uploads/2011/04/20110316-dnssec-techwg-keymgmt-checklist.pdf

5. Ajax ブラウザセキュリティ - 主戦場を DOM に移した XSS

長谷川 武

5.1. 概要

この何年かで Web アプリケーションは、サーバ上のみでプログラムが動作するものから、多くの JavaScript コードが Web ブラウザで動作し、背後でサーバと連携して、リッチかつスムーズな操作感を提供するものへと進化してきた。このことは、Web ブラウザ上に悪意のスクリプトを送り込む「スクリプト注入（XSS）」攻撃を、より防止しづらく、悪用の懸念がより大きなものに変化させた。本稿は、Web アプリケーションに Ajax 等の進化をもたらした技術について振り返るとともに、スクリプト注入対策を考慮すべき新たな文脈について論じるものである。

5.2. Ajax の登場と進化

Ajax（Asynchronous JavaScript and XML）[1]は、必要となるデータを非同期で Web サーバから取り寄せつつ、Web ブラウザの画面上でなめらかな操作性を提供する形態の JavaScript プログラムのことである。かつてのブラウザにおいてはこのようなことはできなかったが、ある時点から Web ブラウザが装備する JavaScript 向け API が進化し、可能になった。この、JavaScript の中からの Web サーバとの間の非同期の通信を可能にした API は、XMLHttpRequest という組み込みオブジェクトである（文書上においては XHR と略されることもある）。最初 Internet Explorer に登場し、やがて他のブラウザにも広まっていった。現在 XMLHttpRequest オブジェクトには「レベル 2」と呼ばれる仕様のものが登場し、さらなる進化を続けている。

図 1 に、主なブラウザが XMLHttpRequest をサポートした年次を示す。

図 1：JavaScript 向けに非同期通信 API が登場

5.3. Ajax 以前 - スムーズでない操作性

Ajax の登場以前の Web アプリケーションは、ユーザにとって、必ずしもスムーズとは言

えない操作性のものだった。

ブラウザの画面でボタンやリンクをクリックすると、Web サーバから応答が返されて画面が再描画されるまで、ユーザは待たされる。表示される情報の変化がごく小さいものであっても、画面（もしくはひとつのフレーム）全体の書き換えが必要だった。プログラマにとっても自由度が低かった。ブラウザ上で JavaScript プログラムを動作させることはできたが、JavaScript プログラムから Web サーバへリクエストを発信すると、その時点で JavaScript プログラムは終了してしまう。呼び出した新しい Web コンテンツに場所を明け渡さなくてはならなかったのである。

5.4. Ajax の始まり

上記の不便は、JavaScript に非同期通信を許す XMLHttpRequest オブジェクトの登場によって（すべてではないが大幅に）解消された。JavaScript プログラムを終わらせなくても処理の途中で Web サーバへデータを要求できるようになったことの恩恵は大きい。DOM（Document Object Model）の API を使って画面の一部分のみ表示内容を変更するテクニックが以前から使えたが、これを Web サーバとの非同期通信と組み合わせられるからである。画面の背後で必要なデータのみをサーバから受け取り、画面の再描画を最小限の範囲で行うことによって、なめらかな操作性が実現する。

5.5. 「同一源泉」の制約

当初の XMLHttpRequest には、セキュリティへの配慮から、"Same Origin Policy"（同一源泉ポリシ）と呼ばれる制約が課されていた。XMLHttpRequest オブジェクトを使ってリクエストを送信できる先は、現在の Web コンテンツの出身の Web サーバに限る、というものである。

5.6. XML / JSON

Ajax は、その名の由来が示すとおり、非同期で Web サーバから受け取るレスポンスのデータフォーマットに XML を用いることが当初想定されていた。しかし、XML にはデータの記述に比較的多くの文字数を要し、Web サーバとのデータの送受信には XML のもつ高度な記述能力は必ずしも必要とされない。最近では XML に代わり、文法がより単純で記述のための文字数も少なくて済む、JSON（JavaScript Object Notation）と呼ばれるフォーマットが使われるようになってきた。

その名のとおり JSON は、JavaScript と縁が深い。JSON は、JavaScript のソースコード内でオブジェクトの値を表す構文を、プログラム外でデータ交換するためのフォーマットに転用したものである。

JavaScript の規格である ECMAScript edition 5（2009 年 12 月）においては、JSON 文字列データと JavaScript オブジェクトとの間の相互変換の API が用意された。

・JSON で扱えるデータ型と値の例

- ・数値　1701　-2.78　3.6e-12
- ・文字列　"hello"　"\""
- ・真偽値　true　false
- ・ナル　null
- ・配列　["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
- ・ハッシュ　{"name": "浦島太郎", "age": 256, "phone": null, "real": false}

・JSON によるデータ構造記述例

```
{"books": [
    {"subject": "Rama II", "authors": ["Clarke", "Lee"], "issued": "1 Nov 1990"},
    {"subject": "Orion", "authors": ["Kirk", "Scott"], "issued": "1 Apr 2264"},
    {"subject": "Android",
        "authors": ["Data", "Laforge"], "issued": "stardate 47232.1"}]}
```

5.7. マッシュアップ時代の訪れ

最近は、既存の Web API（Web サービス）を積極的に利用して作られる「マッシュアップ」タイプの Web アプリケーションが増えてきた。マッシュアップタイプの Web アプリケーションとは、外見は一貫性のあるデザインをもち一枚岩のように見せつつ、内部は、異なる主催者が提供する Web API（Web サービス）を（場合によっては複数）組み合わせて作られるものを指す。この方法をとることによって、既存の Web サイトにさらなる付加価値を付ける別のサイトを短期間で立ち上げられる等のメリットがある。

5.8. クロスドメイン呼出しの必要性と危険な JSONP

Ajax の形態でマッシュアップタイプの Web アプリケーションを構築しようとする際に邪魔になるのが XMLHttpRequest の"Same Origin Policy" だ。従来の XMLHttpRequest においては、HTTP リクエストを送ることが可能な先は、自分の「出身地」の Web サーバのみに限られるからである。現在いくつかのブラウザにおいては、XMLHttpRequest レベル 2 がサポートされ、「出身地」とは異なるドメインの Web サーバへもリクエストを送れるようになった（クロスドメイン呼び出し）[2]。しかし、2009 年になるまでそれはできなかった。そこで、一部のプログラマたちは XMLHttpRequest を使わずして非同期通信を行う手法を見つけ、利用してきた経緯がある。そのひとつが JSONP（JSON with Padding）[3] と呼ばれる手法だ。JSONP によるクロスドメインの非同期通信は次のように行われる。

(1) <script src="*uri*">タグを DOM へ動的に追加して、他ドメインの Web サーバへスクリプトのコンテンツを要求する形の Web API 呼び出しを行う。このときの *uri* には、当該 API に引き渡す必要のあるパラメータを含めておく

(2) その API が返すレスポンスは、次のような関数呼び出しの形のスクリプト文字列

になるようにする。

callback_func("*JSON 形式データ*")

(3) 呼び出し側に *callback_func*()関数の実体を用意しておき、Web サーバから返された"*JSON 形式データ*"を処理するようにする。なお、*callback_func* の関数名は、(1)の段階で例えば、?callback=*callback_func* のようなパラメータで API 側へ受け渡すことが多い

レスポンスに JSON 形式データのみならず、それを処理する関数呼び出しも「追加の詰めもの」（padding）として含める形をとることから JSONP の呼び名ができた。この方法は任意のサーバとのやりとりができて便利である反面、相手が悪意あるサーバの場合、ブラウザを侵害するスクリプトが送られてきて実行されるという、大きなリスクを抱えている。

5.9. XHR レベル 2

2009 年になって、Firefox, Google Chrome, Safari, IE 8 等のブラウザが相次いで、XMLHttpRequest レベル 2 をサポートし始めた（IE 8 においては XDomainRequest というオブジェクト）。XMLHttpRequest レベル 2 においてはクロスドメイン呼び出しが解禁された。ただし、無差別にクロスドメイン呼び出しを許すのは危険であるので、同時に、呼び出しの許可や制限を行うための Access-Control-Allow-Origin 等のレスポンスヘッダが導入された。

例えば、次のようなレスポンスヘッダを Web サーバが返してくる場合、

Access-Control-Allow-Origin: *呼び出し側ドメイン*

ここに示された *呼び出し側ドメイン* に該当する「出身地」のコンテンツからは当該サーバへクロスドメイン呼び出しが許され、他は許されない。悪意あるマッシュアップサイトが善意の Web API サイトを悪用してユーザに詐欺をはたらく、といったケースはこの制約である程度防げると期待できる。お気づきのとおり、これらのレスポンスヘッダは Web API 側が「呼び出し元」に対して制約をかけるものである。呼び出し側が「呼び出せる先」の範囲を制約するような仕組みはいまのところ用意されていない。呼び出し側が何らかの干渉を受け、悪意の Web API を誤って呼び出すよう仕向けられた場合には、被害が生じるおそれがある。

5.10. 新しいタイプのスクリプト注入（XSS）攻撃

Ajax アプリケーションにおいては当然のことながら JavaScript が多用される。ユーザに提供する機能が高度になるにつれ、使われる JavaScript コードも大規模かつ複雑になってきている。そのような Web コンテンツの中へひとたび「悪意のスクリプト」が紛れ込めば、機密情報の漏えい、重要なデータの改ざん、業務妨害をはじめとする、望ましくない事態の発生が危惧される。しかも、悪意のスクリプトがコンテンツに紛れ込む手口は従来のス

クリプト注入（XSS）攻撃とは事情が違ってきている。Ajax アプリケーションに悪意のスクリプトが侵入する攻撃は、概ね次のいずれかの形で起こる。

(1) 信頼できない第三者スクリプトのロード。第三者の Web サイトから取り込んだ JavaScript プログラムに悪意のスクリプトが潜んでいる
(2) JSONP によるスクリプト侵入。JSONP を用いた第三者の Web API 呼び出しへの応答として悪意のスクリプトが返される
(3) 従来型のスクリプト注入（XSS）。サーバ側プログラムの不備によって、Web ページ内に悪意のスクリプトが侵入する
(4) DOM ベースのスクリプト注入（XSS）[4]。ブラウザ上の JavaScript プログラムが、DOM 上の危険な入力地点（ソース）から得たデータに含まれる攻撃パターンをそのままにして危険な出力地点（シンク）へ送り出すことで、Web ページ内に悪意のスクリプトが侵入する

5.11. 従来型のスクリプト注入（XSS）攻撃

従来型のスクリプト注入攻撃は、HTML のタグの中に<script>...</script>のようなスクリプトタグを紛れ込ませて悪意のスクリプトを実行させるものだった。

図 2 に従来型のスクリプト注入（XSS）攻撃の図式を示す。ここでは、サーバ側プログラムの「ソース」から入ってくる攻撃パターンを見過ごして「シンク」へそのまま出力することで攻撃が成立する。したがって対策は、「シンク」に危険な内容が出力されないよう、サーバ側プログラムの中で特殊記号を無害化する等、比較的単純なもので済んでいた。

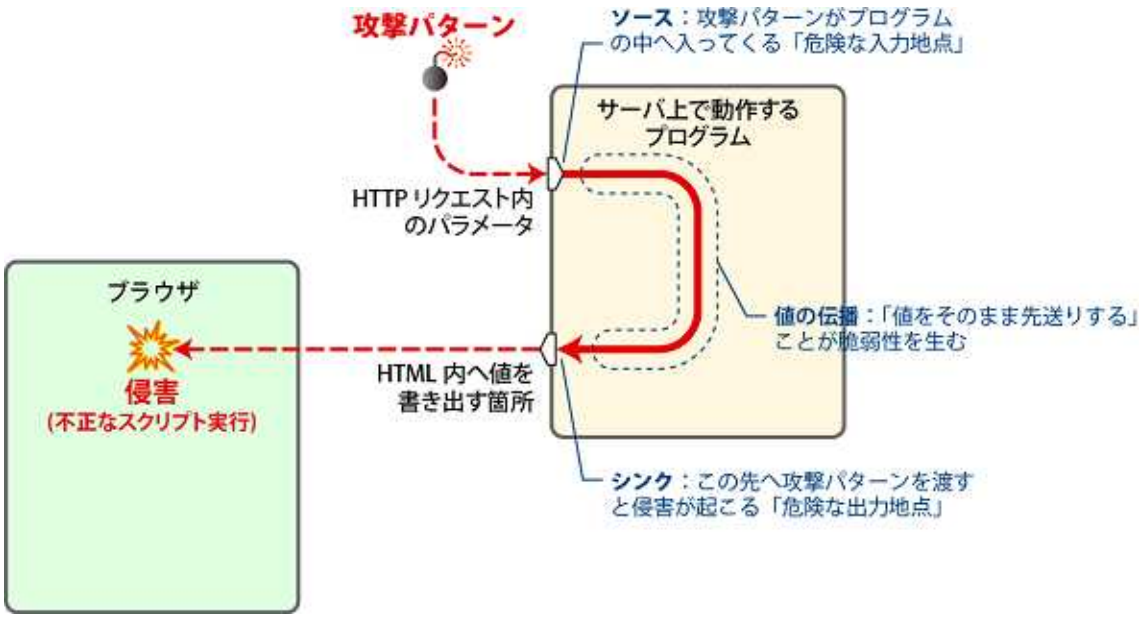

図 2：従来型のスクリプト注入（XSS）攻撃

5.12. DOM を舞台とするスクリプト注入（XSS）攻撃

DOM を舞台とするスクリプト注入（XSS）攻撃においてはだいぶ事情が違ってきている。Ajax タイプのプログラムの動作は、ブラウザ上で動く JavaScript によって動的に DOM が書き換えられつつ進行する。そのため、Web ページ内への不正なスクリプトの侵入は、ブラウザ上の JavaScript プログラムが、DOM 上の危険な入力地点（ソース）から得たデータに含まれる攻撃パターンをそのままにして危険な出力地点（シンク）へ送り出すことで、Web ページ内に悪意のスクリプトが侵入する形をとる[4]。

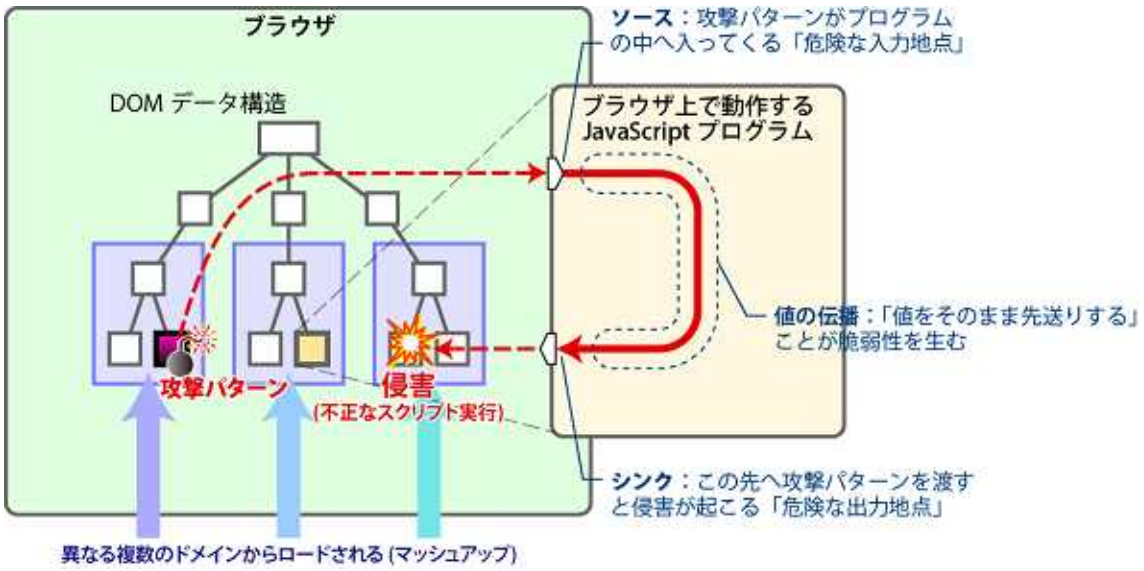

図 3： DOM を舞台とするスクリプト注入（XSS）攻撃

マッシュアップ・アプリケーション（他者が用意した Web API を積極的に利用するスタイルのアプリケーション）のばあい、第三者の Web サービスからデータやスクリプトを受け取る場面で、予期せぬ攻撃や脆弱性を呼び込むおそれがある。

5.13. DOM 上の警戒すべき入力地点（ソース）

DOM ベースのスクリプト注入（XSS）に関して警戒すべき入力地点（ソース）には、例えば、次のものが考えられる。

(1) コンテンツをロードした際の URI の構成要素
- document.URL
- document.location
- location.pathname
- location.search
- location.hash

(2) コンテンツの URI が書かれていたリンク元ページの URI

・document.referrer

(3) Cookie

・document.cookie

(4) **ウィンドウ間/フレーム間の値受渡しに使われる変数**

・window.name

・postMessage(*arg*) **受信側の関数引数** f(*arg*) **の** *arg*.data

5.14. DOM **上の警戒すべき出力地点（シンク）**

DOM **ベースのスクリプト注入**（XSS）**に関して警戒すべき出力地点（シンク）には、例えば、次のものが考えられる。**

(1) **信頼できない第三者スクリプトを含むおそれのあるコンテンツのロード**

・*iframe*.src = *html-uri*

・*script*.src = *script-uri*

(2) **文字列がスクリプトとして解釈実行される文脈への値の送り込み**

・*script*.innerHTML = *code-string*

・eval(*code-string*)

・setTimeout(*code-string*, *msec*)

・window.execScript(*code-string*)　　// Internet Explorer **固有**

(3) <script>**タグが効力をもつ文脈への値の出力**

・document.write(*text*)

5.15. DOM XSS **対策**

DOM **ベースのスクリプト注入**（XSS）**対策としてクライアント側で動作するプログラムの中で次の事項を行う。**

(1) **十分信頼できる** Web **サイト以外から** JavaScript **ファイルをロードしない。**

(2) **十分信頼できる** Web **サービス以外を** JSONP **で呼び出さない。**

(3) JSON **形式文字列データを** JavaScript **オブジェクトに復元するには、**eval()**関数ではなく** JSON.parse()**メソッドを使う。**

(4) eval()**関数をはじめとする、文字列がスクリプトとして解釈実行される文脈へ送り出す値については次のようにする。**

a) **原則として、警戒すべき入力地点（ソース)から得た値を渡さない。**

b) **他に方法がなく、警戒すべき入力地点（ソース)から得た値を渡さざるを得ない場合、実行されることになるスクリプト文字列をチェックし、望ましい振る舞いをすることが確認できたもののみを渡す。**

(5) document.write()**に渡す値は、常に定番のサニタイズ処理を施してから引数へ渡す**

a) < → <

b) > → >
c) " → "
d) ' → '
e) & → &

(6) 警戒すべき入力地点（ソース）から得た値はどれも、その項目の所定の仕様を満たしているか厳しくチェックする。例えば次のような観点でチェックする。
a) 文字種
b) 桁数の範囲
c) 数値の範囲
d) 特定のリストに含まれる　等

上記に加えて従来型のスクリプト注入対策も、もちろん、必要である。

(7) サーバ側プログラムにスクリプト注入（XSS）対策を施す

5.16. ブラウザにおける XSS 対策

ブラウザベンダによるスクリプト注入（XSS）対策も進められている。ここでは主なものをふたつ紹介する。

(1) X-Content-Security-Policy
そのひとつは、Firefox 4 以降に導入された、Content Security Policy と呼ばれる対策機能である。X-Content-Security-Policy レスポンスヘッダによって活性化される。

(2) X-XSS-Protection（XSS Filter）
もうひとつは、Internet Explorer 8 以降に導入された、XSS Filter と呼ばれる対策機能である。X-XSS-Protection レスポンスヘッダによって活性化される。

5.17. X-Content-Security-Policy

Content Security Policy は、Firefox 4 以降（厳密には HTML レンダリングエンジン Gecko 2.0 以降を使用するブラウザ）が備えるスクリプト注入（XSS）対策機能である[5][6]。

これは、ブラウザがどこからスクリプトをロードするかについて、整然と制約を設けることによって、信頼できないサイトからのスクリプトの混入を防ごうというものである。この対策機能は、

X-Content-Security-Policy: ***仕様***

の形のレスポンスヘッダを Firefox が受信すると活性化される。Web コンテンツ内のスクリプトの配置の自由度は減るが、どのスクリプトが実行されどれが禁止されるかを客観的に把握しつつ対策を行えるのが特長である。

この***仕様***には複雑な指定が可能であるが、最も単純なものは、例えば、

X-Content-Security-Policy: allow 'self'

のような指定である。これによって、ブラウザにおけるスクリプトの実行が次のように制約されるようになる：

・HTML の途中に次のような形で記述する、いわゆる「インライン」のスクリプトは実行が禁止される：

<script>*スクリプトソースコード*</script>...

・動作させたいスクリプトは原則として、次の形で独立した別のコンテンツとして呼び込まなくてはならない：

<script src="*スクリプト uri*"></script>

この*スクリプト uri* として許されるのは、この Web ページの源泉と同じサーバに限られる。

・文字列をスクリプトのソースコードとして解釈実行する eval()等の関数の実行が禁止される。

なお、X-Content-Security-Policy レスポンスヘッダは、多くのパラメータもっていて、よりきめ細かな指定をすることができる。

X-Content-Security-Policy: allow *ホスト*...
[; options {inline-script | eval-script}...]
[; img-src *ホスト*...][; media-src *ホスト*...]
[; srcipt-src *ホスト*...][; object-src *ホスト*...]
[; frame-src *ホスト*...][; font-src *ホスト*...]
[; xhr-src *ホスト*...][; frame-ancesors *ホスト*...]
[; style-src *ホスト*...]
[; report-uri *uri*][; policy-uri *uri*]

・ホスト名を具体的に羅列することによって、Web ページの源泉以外のサーバからのスクリプトのロードを許可/限定できる：

script-src *ホスト*...

・スクリプトのロード元を許可するのと同様の要領で、画像等各種リソースのロード元のホストを明示できる：

img-src *ホスト*...; media-src *ホスト*...; style-src *ホスト*...; ...

・Web ページの源泉のホストにからの追加リソースのロードを禁止することもできる

allow 'none' [*別のホスト*...]

・せっかくのインラインスクリプトや eval ()の禁止機能であるが、その禁止を解除することもできる。

options inline-script eval-script

・現在はまだ報告される項目がごく少ないようであるが、指定されたホストにエラー報告を行うような機能も備わる模様である：

report-uri *uri*

5.18. X-XSS-Protection（XSS Filter）

XSS Filter は、Internet Explorer 8（以下 IE 8）以降が備えているスクリプト注入（XSS）対策機能である[7][8]。類似の対策機能は、Google Chrome や Safari の新しいバージョンにも備わっている。

これは、HTTP リクエストに含まれていたスクリプト注入（XSS）攻撃パターンがサーバからそのまま HTTP レスポンスに出力されてきたら、そのスクリプトを動作させないというものである。この対策機能は、例えば、

X-XSS-Protection: 1

のようなレスポンスヘッダを IE 8（およびそれ以降のバージョン）等が受信すると活性化される。

この XSS Filter が HTTP レスポンスに対して防御効果を発揮する条件はつぎのようなものである：

(1) 機能が活性化する条件

次のいずれかの指定/設定によって、XSS Filter 機能がオンになる

・HTTP レスポンスに次のいずれかのレスポンスヘッダが存在する

X-XSS-Protection: 1　または

X-XSS-Protection: 1; mode=block

・HTTP レスポンスに X-XSS-Protection レスポンスヘッダが存在せず、「インターネットオプション」の「XSS フィルターを有効にする」項目が「有効」に設定されている

操作要領：

コントロールパネル > インターネットオプション >
セキュリティ > 該当するゾーン > レベルのカスタマイズ >
XSS フィルターを有効にする > 有効にする

これは IE 8（およびそれ以降のバージョン）についてのみ言えることで、Google Chrome と Safari には上記のようなデフォルト設定の切り替え機能はない。Google Chrome と Safari の新しいバージョンにおける XSS Filter 機能のデフォルトは常に「オン」になっている

HTTP レスポンスに X-XSS-Protection: 0 レスポンスヘッダがあると、「インターネットオプション」の「XSS フィルターを有効にする」項目よりも優先される。このヘッダがあると、XSS Filter 機能はオフになる

(2) 検出/防御してくれるもの

次の両方が成り立つと判定したとき、XSS Filter は防御効果を発揮する

・HTTP レスポンスの元となる HTTP リクエストに、スクリプト注入（XSS）攻撃パターンと見なされるものが含まれていて、それと同じものが HTTP レスポンスにも出現している
・HTTP レスポンスの元となる HTTP リクエストを発生させたアクションやリンクは、HTTP レスポンスを返したものとは異なるサイトから来たものである
・同一のサーバでも、ホスト名による参照と IP アドレスによる参照は異なるサイトとみなされる

例えば、次のようなリクエストデータ、

```
<script>alert(1)</script>
```

を HTTP レスポンス内にエコーさせると、ブラウザの HTML ソース上にはそのまま置かれるが、注入されたスクリプトは実行されない。

また、次のようなリクエストデータ、

```
<img src="x" onerror="alert(1)">
```

を HTTP レスポンス内にエコーさせると、次の、

```
<img src="x" #nerror="alert#1#">
```

のように書き換えられ、HTML ソースの記述上も注入されたスクリプトが無効にされる。

次のレスポンスヘッダ、

```
X-XSS-Protection: 1; mode=block
```

が指定されたときに、攻撃パターンがあると判定されると、IE 8（及びそれ以降）においては、HTTP レスポンスのコンテンツ全体が

```
#
```

ただひとつに置き換えられる。Google Chrome と Safari においては、空白のページが表示される。

注意すべき点として、この XSS Filter では、次のものを阻止できないことがあげられる。

・document.write()によるページ内への<script>...</script>タグの挿入
・DOM を通じた<script>タグのスクリプト内容の書き換え（Google Chrome や Safari において）
・JSONP 方式によるスクリプトの無防備な取り込み　等

DOM ベースの XSS への対策としては課題を残している。

5.19. まとめ

Ajax の進歩とマッシュアップ開発スタイルの発展に伴い、Web アプリケーションのブラウザ側のプログラミングは複雑になった。Web コンテンツの中への悪意のスクリプトの侵入の手口も多様化している。かつては、予定外のスクリプトが入り込まないよう、Web ペー

ジの保護につとめればよかった。しかし今や、クライアント側の、DOM から値を取り出す箇所、DOM の一部を書き換える箇所のそれぞれにおける警戒と対策が必要である。攻防は、いわば「DOM がかたちづくるジャングル」の中のゲリラ戦に移ってきたといってよい。

その一方、ブラウザのスクリプト注入（XSS）対策機能も進歩してきている。Firefox 4[9]以降がもつ Content Security Policy は、インラインスクリプトや他ホストからのスクリプトのロードを抑制でき、DOM ベースのスクリプト注入（XSS）に効果を発揮する。

Internet Explorer 8 以降[10]（および Google Chrome[11]、Safari[12]の新しいバージョン）がもつ XSS Filter は、DOM ベースのスクリプト注入（XSS）対策には不足があるものの、有用な対策機能のひとつである。

以上

参考 Web

Ajax

[1] "Ajax - Wikipedia"
http://ja.wikipedia.org/wiki/Ajax

[2] "Cross-Origin Resource Sharing - Wikipedia, the free encyclopedia"
http://en.wikipedia.org/wiki/Cross-Origin_Resource_Sharing

[3] "JSONP - Wikipedia“
http://ja.wikipedia.org/wiki/JSONP

DOM ベースのスクリプト注入（XSS）

[4] "DOM Xss Identification and Exploitation“
http://dominator.googlecode.com/files/DOMXss_Identification_and_exploitation.pdf

Content Security Policy - Firefox（Gecko 2.0）の XSS 対策機能

[5] "Introducing Content Security Policy - MDC Docs"
https://developer.mozilla.org/en/Introducing_Content_Security_Policy

[6] "Using Content Security Policy - MDC Docs"
https://developer.mozilla.org/en/Security/CSP/Using_Content_Security_Policy

XSS Filter - IE 8 以降、および Google Chrome、Safari の XSS 対策機能

[7] "IE8 Security Part IV: The XSS Filter"
http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

[8] "IE8 XSS filter: what does it really do?"
http://stackoverflow.com/questions/2051632/ie8-xss-filter-what-does-it-really-do

ブラウザのバージョン履歴

[9] "Old Version of Firefox 1.0 Download - OldApps.com"
http://www.oldapps.com/firefox.php?old_firefox=48

[10] "History of Internet Explorer - Wikipedia, the free encyclopedia"
http://en.wikipedia.org/wiki/History_of_Internet_Explorer

[11] "Old Version of Google Chrome Download - OldApps.com"
http://www.oldapps.com/google_chrome.php

[12] "Safari version history - Wikipedia, the free encyclopedia"
http://en.wikipedia.org/wiki/Safari_version_history

6. IETF における Web 2.0 セキュリティ（Websec WG，JWT 等）

宮川 寧夫

6.1. はじめに

Web 2.0 との潮流の中で、Web アプリケーションを開発する際に、いわゆる「マッシュアップ」が卑近に行われるようになってきている。「マッシュアップ」という用語は、音楽 DJ が複数の楽曲を組み合わせて新たな音楽を創造する過程になぞらえて、ソフトウェアを API 呼び出しによって組み合わせて構成することを表現するようになっている。このような「マッシュアップ」を行う際のセキュリティに関する論点を扱う WG が、インターネットの標準化団体：IETF（Internet Engineering Task-Force）において発足しており、Websec WG という。

また、アイデンティティ管理技術を構成する専用のプロトコル（OAuth 2.0 等）において、時流を反映して JavaScript で書く JWT（Json Web Token）を用いるクレデンシャル仕様が規定されるようになってきている。

本稿においては、今期、IETF における、これらの動向を解説する。

6.2. Websec WG

(1) 概況

Websec ワーキンググループ（以下、WG）は、第 78 回 IETF（Maastricht）において開催された HASMAT（HTTP Application Security Minus Authentication and Transport）BoF（Bird of Fether）が盛況であったことを受けて 2010 年 10 月に設置された新しい WG である。このように「Authentication（本人認証）」の論点は、当初から対象外とされていた。ただし、「HASMAT」という名前は、HASMAT（Hazardous Materials Response Team）と同様に発音されていたが、Web セキュリティとは異なる種類の畏れの念を起こさせることもあり、一般に伝え易く、かつ、他で使われていない名称として「Websec」に改名された。当初、Websec WG のチェアは、Tobias Gondrom 氏のみであった。Gondrom 氏は、かつて LTANS（Long-Term Archive and Notary Services）WG のチェアを勤めていた。ちなみに LTANS WG は、今年の 7 月をもって正式に終結した。当初から共同チェアも募集されていたが、今年 4 月から Alexey Melnikov 氏が共同チェアとなった。

この WG は、W3C と連携しており、Web アプリケーションセキュリティに関するプロトコルについての論点整理と、その中から選択された案件についての標準化を行う[1]。まず、論点整理は、下記の I-D（Internetr-Draft）に書かれている。これは、Informational RFC として策定中である。

- “HTTP Application Security Problem Statement and Requirements”

一方、具体的に選択された標準化案件は、下記の 3 件であった。これらすべては、Standards Track RFC として策定中である。

- Web コンテンツについての源泉概念（Web Origin Concept）
- 厳密なトランスポートセキュリティ（Strict Transport Security）
- メディア種別の盗聴（Media Type Sniffing）

(2) **論点整理：**”Web Security Framework: Problem Statement and Requirements”

今年 3 月に発行された 00 版の I-D の目次構成は、下記のとおりであった。9 月に発行された 01 版[2]においても目次構成は変わっていない。

目次

まだ、詳細は書き込まれていないが、8 章の見だしに見られるように、現存している複数の対策技術仕様のポリシーを合体する（Coalesce）アプローチが行われている。01 版において対策技術仕様としては下記のものが掲げられている。

- CORS（Cross-Origin Resource Sharing）
- XDomainRequest
- toStaticHtml
- innerSafeHtml
- X-Frame-Options
- CSP（Content Security Policy）frame-ancestors

(3) 選択された標準化案件

選択された標準化案件について、該当 Web ページの不備によって複数の I-D の最新版がリンクされていない状態になってしまっている。また、紛らわしいことに同一名称の文書が複数、存在することになってしまった。それらの所在を明確にすると共に概説する。

Web コンテンツについての源泉概念（Web Origin Concept）

同一源泉ポリシー（Same origin policy）の標準化は、源泉の判定ルールを規定するものであるが、これは W3C の HTML5 に由来する。この案件については、紛らわしいことに、ふたつの I-D 文書が並存していた。

- draft-abarth-origin-
  こちらは、informational RFC とするとされたが、結局、更新されなかった。
- draft-ietf-websec-origin-
  こちらが websec WG によるものとされた。02 版[3] が 6 月に発行され、その後も更新され続けている。

厳密なトランスポートセキュリティ（Strict Transport Security）

STS（Strict Transport Security：厳密なトランスポートセキュリティ）も@ W3C に由来する案件である。この案件は、途中でファイル名の番号部分以外も変更されているので、追跡する際には注意を要する。

- draft-hodges-strict-transport-sec-
  02 版の後、下記のようなファイル名に変更された。
- draft-ietf-websec-strict-transport-sec-
  00 版から再開し、01 版[4] が 3 月に発行され、更新され続けている。

メディア種別の嗅ぎ分け（Media Type Sniffing）

これは、HTTP レスポンスのメディア種別について、セキュリティと互換性の両方のバランスを図りつつ実質的に判定するアルゴリズムを規定する案件である。この案件についても、途中でファイル名の番号部分以外も変更された。

- draft-abarth-mime-sniff-
  06 版まで更新された後、下記のようなファイル名に変更された。
- draft-ietf-websec-mime-sniff-
  上記の 06 版と、この 00 版は、同一の内容のものである。03 版[5] が 5 月に発行され、更新され続けている。

HTTP Header Frame Options 等

CSRF（Cross Site Request Forgery：リクエスト強要）や Clickjacking に対する対策仕様として、サーバからブラウザにヘッダー中に渡されるポリシーに基づく制御方法の規定についても I-D が草稿され始めて、3 月には 01 版[6] が発行された。

- “HTTP Header Frame Options”
  draft-gondrom-frame-options

このような制御に用いるヘッダー項目の仕様には同様ながら別の仕様も策定されつつあり、代表的なものとして、W3C には下記のふたつの案件がある。技術仕様の状況を整理する必要性が認識されつつある。

- “The From-Origin Header” or “Cross-Origin Resource Embedding Exclusion”
  http://dvcs.w3.org/hg/from-origin/raw-file/tip/Overview.html
- “Content Security Policy”

CSP（Content Security Policy）は、Gecko 2.0 系のブラウザによる実装が先行してきたが、W3C においても検討されている。

6.3. JSON をクレデンシャル仕様に使う JWT（JSON Web Token）

(1) JWT（JSON Web Token）とは

JSON は、JavaScript Object Notation のアクロニムであり、データ記述言語である。2006 年 7 月に RFC 4627 [7]として規定されている。JSON 仕様は、YAML（YAML Ain't a Markup Language：JavaScript（ECMAScript）におけるオブジェクト表記法）の部分でもある。

今期、アイデンティティ管理において、この JSON 規格を用いたクレデンシャル仕様を規定する動向があり、このようなクレデンシャルは JWT（JSON Web Token）と呼ばれている。

- “JSON Web Token (JWT)”[8]

JWT[8] は、符号化に base64url を使い、ディジタル署名が付されたり、オプションとして暗号化したりするものである。

最近、この JWT が用いられるプロトコル仕様が策定されるようになった。代表的な例として、OAuth 2.0 の Bearer token 仕様のひとつとして策定されつつあることが挙げることができる[9]。

base64url とは、base64 に基づく符号化であり、RFC 4648, “The Base16, Base32, and Base64 Data Encodings”（の 5 章）中に規定されている。

変更点は、通常の base64 が`+'と`/'を使用しているところに、それぞれ`-'と`_'を使用する点のみである。これによって URL の記述が攻略されて意味が変わることがないように設計されている。

例：
base64 の場合
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

base64url の場合
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_

また、JWT においてパディングを行うことは禁じられている。

(2) JSON を使う長所

JSON によってアイデンティティ情報のようなデータを運ぶことの長所として、下記の事項が挙げられる。

- XML を書く際のようにタグを覚えることが不要であるので簡単に書ける。
- 文字列を規則的に符号化することによってシリアライズされたひとかたまりのデータとなる。
- 特殊文字としてエスケープ処理する必要がある文字が少なく限定的である（「"」のみ）ので失敗が少なくなることを期待できる。
- ディジタル署名を付すことや暗号化することに問題が無いため、アイデンティティ情報を運ぶのに向いている。

このような長所は、来たる HTML5 の世代においても存続する見通しである。

(3) JWT（JSON Web Token）仕様

JWT を規定する I-D の 04 版[8] の目次構成は、下記のとおり。

目次

9. Unsigned JWTs
10. IANA Considerations
11. Security Considerations (To Be Determined)
12. Open Issues and Things To Be Done (To Be Determined)
13. References

まだ、書かれていない章が多く、11 章の「セキュリティについての考慮事項」についても埋まっていない。

OpenID の後継仕様や OAuth 2.0 においても、JWT は採用される。OpenID は、シングルサインオン用プロトコルであり、現行仕様は 2.0 であるが、次期仕様として OpenID Artifact Binding/Connect が策定中であり、この中で採用される。一方、OAuth 2.0 は、代理アクセス認可用プロトコルであるが、複数の「grant type（トークン公布種別）」というシステム構成パターンに対応するように仕様が規定されている。その「grant type」に、”JSON Web Token (JWT) Bearer Profile for OAuth 2.0” [9]という仕様が追加される。

IETF にも WOES（Web Object Encryption and Signing）BOF のメーリングリストが設置され、検討が進められている。

(4) 他の JSON Web 仕様

OpenID コミュニティには他のも暗号技術の「JSON Web *」機能として、JWS（JSON Web Signature）と JWE（JSON Web Encryption）があり、暗号技術以外の仕様として、SWD（Simple Web Discovery）も検討されている[10]。

以上

参考文献

[1] Websec WG 憲章（charter）
http://tools.ietf.org/wg/websec/charters

[2] ”Web Security Framework: Problem Statement and Requirements” 00 版（2011 年 3 月 7 日）
http://tools.ietf.org/html/draft-hodges-websec-framework-reqs-00

[3] “Web Origin Concept” 02 版（2011 年 6 月 24 日）
http://tools.ietf.org/html/draft-ietf-websec-origin-02

[4] “Strict Transport Security” 01 版（2011 年 3 月 14 日）
http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-01

[5] “Media Type Sniffing” 03 版（2011 年 5 月 7 日）
http://tools.ietf.org/html/draft-ietf-websec-mime-sniff-03

[6] “HTTP Header Frame Options” 01 版（2011 年 3 月 15 日）
http://tools.ietf.org/html/draft-gondrom-frame-options-01

[7] RFC 4627, “The application/json Media Type for JavaScript Object Notation (JSON)”（2006 年 7 月）
http://tools.ietf.org/html/rfc4627

[8] “JSON Web Token (JWT)”04 版（2011 年 3 月 30 日）
http://tools.ietf.org/html/draft-jones-json-web-token-04

[9] ”JSON Web Token (JWT) Bearer Profile for OAuth 2.0” 00 版（2011 年 3 月 28 日）
http://tools.ietf.org/html/draft-jones-oauth-jwt-bearer-00

[10] Michael B. Jones, “The Emerging JSON-Based Identity Protocol Suite” （2011 年 4 月 27 日）
http://self-issued.info/papers/The_Emerging_JSON-Based_Identity_Protocols.pdf

7. クラウドコンピューティングセキュリティ - NIST ガイドラインより

馬場 達也

7.1. はじめに

現在、多くの企業がクラウドを利用するようになってきたが、セキュリティについては十分理解されているとは言い難い状況にある。本報告では、米国標準技術研究所（NIST：National Institute of Standards and Technology）が 2011 年上半期に発行したドキュメントから、クラウドのセキュリティに関連した部分をまとめたものを報告する。

7.2. 調査対象文献

NIST が 2011 年上半期に発行した下記のふたつのドキュメントから、クラウドのセキュリティに関連した部分を調査した。

- Draft Special Publication 800-144, "Guidelines on Security and Privacy in Public Cloud Computing"（2011 年 1 月）
- Special Publication 800-146, "DRAFT Cloud Computing Synopsis and Recommendations"（2011 年 5 月）

7.3. NIST の定義するクラウドコンピューティング

クラウドコンピューティングは、ユーザがインターネット（クラウド）の向こう側からサービスを受け、利用料金を払う形態である。2006 年 8 月 9 日、米 Google の CEO であるエリック・シュミット氏が、米国カリフォルニア州サンノゼ市で開催された”Search Engine Strategies Conference”の中で「クラウドコンピューティング」と表現したのが始まりと言われている。

NIST では、NIST Draft Special Publication 800-145 "NIST Cloud Computing Definition"の中で、クラウドコンピューティングの特徴とともに、サービスモデルとデプロイメントモデルのふたつの側面から以下のように分類している。

(1) クラウドコンピューティングの特徴

- On-demand self-service
  - 人間の介在なしに、自動的に必要なリソースを必要な時に割り当てることができる
- Broad network access
  - 携帯、ノート PC、PDA などのさまざまな端末からネットワークを介してアクセスできる
- Resource pooling
  - ストレージ、CPU、メモリ、ネットワーク帯域、仮想マシンなどのリソースをロケー

ションに縛られずに、利用者の要求に応じてダイナミックに割り当てられる

- Rapid elasticity
  - 能力（リソース）を、必要な時に、必要な量を、素早く、柔軟に提供できる
- Measured Service
  - リソースの使用量は、監視・制御・報告される

(2) サービスモデル

- SaaS（Cloud Software as a Service）
  - クラウドインフラ上で動作する、サービス提供者が提供するアプリケーションを利用者が利用する
- PaaS（Cloud Platform as a Service）
  - クラウド事業者によってサポートされたプログラミング言語やツールを使用して利用者が開発したアプリケーションをクラウドインフラ上に配置できる
- IaaS（Cloud Infrastructure as a Service）
  - 利用者が OS やアプリケーションを動作させるための CPU やストレージ、ネットワーク等の基本的なコンピューティングリソースを提供する

(3) デプロイメントモデル

- Private Cloud
  - クラウドインフラは、単一組織のみのために運用される。利用組織か、第三者が管理し、オンプレミスまたはオフプレミスで配置される。
- Community Cloud
  - クラウドインフラは、複数組織で共有される。利用組織か、第三者が管理し、オンプレミスまたはオフプレミスで配置される。
- Public Cloud
  - クラウドインフラは一般に公開、または、大企業グループ向けに提供され、クラウド提供者が管理する。
- Hybrid Cloud
  - クラウドインフラは、上記の 2 つ以上の組み合わせで提供される。

7.4. パブリッククラウド環境におけるセキュリティ/プライバシーの問題と対策

NIST の DRAFT SP 800-144 "Guidelines on Security and Privacy in Public Cloud Computing"では、パブリッククラウド環境におけるセキュリティやプライバシーの問題と対策について、表 1 のようにまとめている。

表 1：セキュリティ/プライバシーの問題と対策（出典：NIST DRAFT SP 800-144）

問題となるエリア	対策
ガバナンス	管理規程を、設計、実装、試験、監視だけでなく、クラウドでのアプリケーション開発や、クラウドでのサービス提供におけるポリシー、手順、標準にまで拡張する
コンプライアンス（データロケーション、法律と規制、E ディスカバリ）	データのロケーションやプライバシー管理/セキュリティ管理などに関する、クラウドにインパクトを与える可能性のある法律や規制を理解しておく
トラスト（内部犯行、データ所有権、複合サービス、見える化、リスク管理）	クラウド提供者が行うセキュリティおよびプライバシーの制御やプロセスをいつでも確認できる仕組みを契約に入れ込む
アーキテクチャ（攻撃対象領域、仮想ネットワーク保護、補助データ、クライアント側保護、サーバ側保護）	クラウド提供者がサービスを提供するために使用する技術を理解しておく
アイデンティティとアクセス管理（認証、アクセス制御）	認証、承認、その他のアイデンティティ管理機能およびアクセス管理機能を安全に保つために適切な安全装置が存在することを確認する
ソフトウェア隔離（ハイパーバイザの複雑性、攻撃手法）	クラウド提供者が使用する仮想化およびその他のソフトウェア分離方法を理解し、リスクを評価する
データ保護（データ隔離、データのサニタイジング）	クラウド提供者のデータ管理手法が適したものであることを評価する
可用性（一時的な停止、永久的な停止、サービス不能攻撃、価値への集中）	災害時に、重要な運用を即座に再開し、タイムリーでよく練られた方法ですべての運用を再開することができることを確認する
インシデントレスポンス	組織にとって必要なインシデントレスポンスのために、契約約款と手順を理解し、交渉する

7.5. クラウドコンピューティングのセキュリティ

NIST SP 800-146 "DRAFT Cloud Computing Synopsis and Recommendations" では、以下のセキュリティの要件について整理されている。

(1) 意図しないデータ漏えいのリスク（Risk of Unintended Data Disclosure）
(2) データプライバシー（Data Privacy）
(3) システムインテグリティ（System Integrity）
(4) マルチテナンシー（Multi-tenancy）
(5) ブラウザ（Browsers）
(6) 信頼に対するハードウェアサポート（Hardware Support for Trust）
(7) 鍵管理（Key Management）

以降、この 7 つの要件について概要を紹介する。

【意図しないデータ漏えいのリスク（Risk of Unintended Data Disclosure)】

- 機密扱いではない政府のシステムは、しばしば、下記のような機密情報を、機密でない公開情報と同じように処理してしまう。このため、機密データを適切な保護なしにクラウドに保管することがないように注意する必要がある。
  - PERSONALLY IDENTIFIABLE INFORMATION （PII）
  - FOR OFFICIAL USE ONLY （FOUO）
  - 知的財産情報

【データプライバシー（Data Privacy)】

- 以下の理由から、クラウドコンピューティング環境では、プライバシーの保護はさらに複雑化している。
  - クラウドは分散している
  - クラウド環境では、どこにデータが保管されているのか、誰がアクセスしたのか、誰がアクセスできるのか、ということに対する利用者の意識が低い

【システムインテグリティ（System Integrity)】

- クラウド提供者は、クラウドの機能の破壊や妨害から保護しなければならない。
- クラウドのインテグリティを維持するために、下記のステークホルダ間でアクセス権を分けることが必要である。
  - サービス利用者
  - サービス提供者
  - 管理者

- 利用者にはクラウドの仕組みが分からないため、クラウド上で動作しているアプリケーションのインテグリティを確認することが難しいという問題がある。

【マルチテナンシー（Multi-tenancy）】

- クラウド提供者が使用する共有の仕組みでは、利用者を隔離するために複雑なユーティリティを使用するため、隔離が失敗するリスクが存在する。
- 論理隔離が物理隔離の適切な代替となるかどうかは長い間の研究課題となっている。この課題は、データをクラウド上に送る前に暗号化することによってある程度解決することができる。
- データ処理を行うクラウドにおいては、利用者は、処理するデータの種類を制限するか、VM 毎に借りるのではなく、コンピュータシステム全体を借りるような、特別な隔離の仕組みを提供する契約を行うことが考えられる。

【ブラウザ（Browsers）】

- 多くのクラウドアプリケーションは、利用者側のブラウザを GUI として利用する。しかし、ブラウザは複雑であるため、セキュリティ上の欠陥を持ち、不正アクセスに脆弱である。
- アプリケーション提供者は、ブラウザの種類やバージョン毎に動作検証を行わなければならない。
- 利用者が管理している端末やブラウザはセキュリティ管理がきちんと行われていない可能性があり、最新でない可能性がある。もし、利用者のブラウザに侵入されると、クラウドに委ねられているすべてのリソースはリスクにさらされる。このため、ブラウザがクラウドへのアクセスポイントとなる場合には、ブラウザがセキュリティ侵害されていないことが重要となる。
- 信頼性を確保するために、アプリケーションゲートウェイやファイアウォール越しにクラウドにアクセスしたり、ブラウザの種類やプラグインを制限したり、ブラウザが最新であることを確認したり、ブラウザ経由でアクセスするシステムをロックダウンしたりする方法があるが、コストや機能性や使い勝手の低下などの問題が生じる。

【信頼に対するハードウェアサポート（Hardware Support for Trust）】

- TPM（Trusted Platform Module）は、システム起動時に生成されたチェックサムのセットを保管することで、システムが本当に正しいコンポーネントから起動されたことを証明するが、仮想マシンが別の物理サーバに移動した場合は、TPM の信頼の連鎖が壊れてしまう。
- TPM の仮想化や、異動後の VM が別のハードウェア上で信頼を再構築する仕組みが研究されているが、この問題は未解決のままである。

【鍵管理（Key Management)】

- 利用者の鍵を適切に保護するためには、クラウド提供者の協力が必要となる。
- クラウド環境では、専用のハードウェアとは異なり、メモリバッファをゼロにしても以下の場合は鍵を削除できない可能性がある。クラウドの内部で暗号を安全に使用する方法に関する問題は、未解決のままである。
  - ハイパーバイザが、メモリの一貫性を保とうとしている場合
  - VM が、リカバリのためにスナップショットを取っている場合
  - VM が、別のハードウェアへのマイグレーションのためにシリアライズされている場合

7.6. まとめと今後の展開

実際のビジネスでは問題が表面化していないが、クラウドコンピューティングを企業がビジネス利用するためには、セキュリティに関する潜在的な問題が多く存在する可能性がある。今回紹介した NIST の他にも、CSA（Cloud Security Alliance）の「Guidance for Critical Areas of Focus in Cloud Computing」など、他の組織もクラウドのセキュリティについて問題を指摘している。

クラウド事業者は、NIST や CSA のガイダンスを参考にセキュリティに関する取り組みを行うことが期待される。

以上

8. SCIM（Simple Cloud Identity Management）

工藤 達雄

8.1. はじめに

本報告では 2011 年上半期のアイデンティティ管理技術に関する動向として、アイデンティティ・プロビジョニング・インタフェースの標準化を目指す SCIM（Simple Cloud Identity Management）の概要を紹介する。

8.2. アイデンティティ・プロビジョニング・インタフェースの標準化と SPML の失敗

ユーザが情報システムを利用するためには、先だってそのシステムへのログインに必要なユーザ・アカウントが存在し、かつ必要なアクセス権限が設定されている必要がある。そしてユーザの立場や役割の変化（例：企業内における社員の異動や昇進、コンシューマー向けサービスにおける普通会員から特別会員へのアップグレード）に応じ適切にアクセス権限を付与・剥奪し、さらに利用資格を喪失した際（例：退職や休職、退会、利用料未払い）には、ユーザ・アカウントの無効化や削除を行わなくてはならない。このような、ユーザ・アカウントとアクセス権限の継続的な管理を、アイデンティティ・プロビジョニングと呼ぶ。

複数のシステム間をまたがって行われるアイデンティティ・プロビジョニング（例：人事アプリケーション・システムから社内ディレクトリ・システムへ、契約者管理データベースから Web サイトの会員データベースへ）では、ほとんどの場合、システム間のインタフェースとして各サービス独自のプロトコルやフォーマットが利用されている。たとえばディレクトリ・システムであれば LDAP、データベースであれば各製品固有の通信プロトコルであり、場合によっては CSV ファイルによってアカウント情報の更新を行うことも少なくない。

これらのサービスごとに違うインタフェースを統一するべく、過去、OASIS のサービス・プロビジョニング技術委員会 [1] によって策定されたのが SPML（Service Provisioning Markup Language）[2] である。SPML はアカウント管理に特化したものではなく、より汎用的な、XML によってサービス・プロビジョニング情報を交換するためのフレームワークであり、2003 年にバージョン 1.0 が、2006 年に同 2.0 が OASIS によって承認されている。

SPML は従前に存在した 3 つのプロビジョニング製品ベンダーの外部インタフェースを統一した仕様であり、これに代わる他の仕様は存在しなかったことから、この分野では SPML が主流になるとみられていた。しかし今日現在、SPML の普及は進んでいない。この背景には、仕様が汎用的であるがゆえに複雑であることと、SPML に対応する製品・サービスが少ないことが挙げられる。

8.3. SCIM

(1) 概要

SPML は広まらなかったものの、アイデンティティ・プロビジョニング・インタフェースの標準化のニーズは消えたわけではなく、むしろ以前より高まっている。最も大きな要因は、「クラウド・サービス」の利用拡大である。

CSP（Cloud Service Provider：クラウド・サービス事業者）の多くはユーザ管理機能を Web API 化し、ECS（Enterprise Cloud Subscriber：ユーザ企業などといった、サービスを利用する組織）に提供しているが、その API 仕様は CSP ごとにまちまちであり、互換性がない。そのため ECS が自社 ID 管理システムからクラウド・サービスへのプロビジョニングを行うためには、たとえばユーザの追加・削除といった単純な操作であっても、CSP ごとに異なる API に対応しなくてはならない。

このような状況に対し、クラウド・サービスのアイデンティティ・プロビジョニング・インタフェースの統一を目指す動きとして登場したのが SCIM（Simple Cloud Identity Management）[3] である。SCIM の特徴は下記の通り。

- RESTful な Web API（SPML は SOAP）
- CRUD（Create, Read, Update, Delete）に特化（SPML はその他の操作を多数定義）
- シンプルで拡張しやすいユーザ・スキーマ（SPML は DSML 由来のため拡張が面倒）

このように、クラウド・サービスにフォーカスしたシンプルな仕様を目指している SCIM だが、SPML と異なるのは、この仕様策定の方針だけではない。もう一つの大きな違いは、SPML のときには BMC や Sun などのプロビジョニング製品ベンダが中心となっていたが、SCIM では、Google や Salesforce.com、Cisco（WebEx）、VMware といった有力な CSP が積極的に仕様策定の議論に関与している点である。これは SCIM の利用可能な局面が多くなることを意味しており、実際に、企業とクラウド・サービスを連携させる製品・サービスのベンダーや、他の中小 CSP も、SCIM コミュニティに続々と参加している。

SCIM 仕様の策定は、クラウドベースのアプリケーションやサービスのユーザ情報の管理を容易にすることを目的に進められている。また既存のスキーマや導入パターン、認証・認可モデルを活用し、仕様自身はシンプルに留める方針である。そのため仕様としては、「プロトコル」、「スキーマ」および「バインディング」の 3 種類のドキュメントにまとめられている。

(2) ユースケース

SCIM コミュニティが作成した SCIM シナリオ [4] では、どのような用途に SCIM プロトコルを適用するかが紹介されている。まずどのアクター（関与するエンティティ）間に

て SCIM を利用するかについては、ECS と CSP の間と、ある CSP と別の CSP の間のフローが挙げられている。前者は典型的には企業内の ID 管理システムからクラウド・サービスへのプロビジョニングであり、後者は例えばクラウド・サービスのマーケットプレイスから、そこに出店する別のクラウド・サービスへのプロビジョニングである。またフローには、プッシュ（例：ユーザ生成を他所に反映）／プル（例：他所からの変更通知を受け取り、ユーザの最新の属性情報を取得）のふたつの向きがあることが示されている。

そして、SCIM プロトコルを開始する起点が「トリガー」である。同文書では 5 種類のトリガーを規定している。

- 生成（Create）：新入社員の登録や、サービス利用契約の締結に基づく、CSP 上のアイデンティティ・リソース（アカウント情報）の生成

- 変更（Update）：ユーザのパスワード初期化、あるいはサービス利用契約のプラン変更などに基づく、CSP 上のアイデンティティ・リソースの更新

- 削除（Delete）：社員の休職・退職や、サービス利用契約の終了に基づく、CSP 上のアイデンティティ・リソースの削除や無効化

- 同期（Sync）：社員の属性情報の変更に基づく、アクター間での変更内容の伝播

- シングル・サインオン（SSO）：CSU（クラウド・サービス・ユーザ： ECS に所属するユーザ）が CSP にアクセスを試みたタイミングで、シングル・サインオンと同時に、アイデンティティ・リソースの生成や変更を通知

ECS が CSP にアイデンティティ・リソースの生成をリクエストし、そしてその CSP が別の CSP に対してさらにアイデンティティ・リソースの生成をリクエストするユースケースを下図に示す（図 1）。

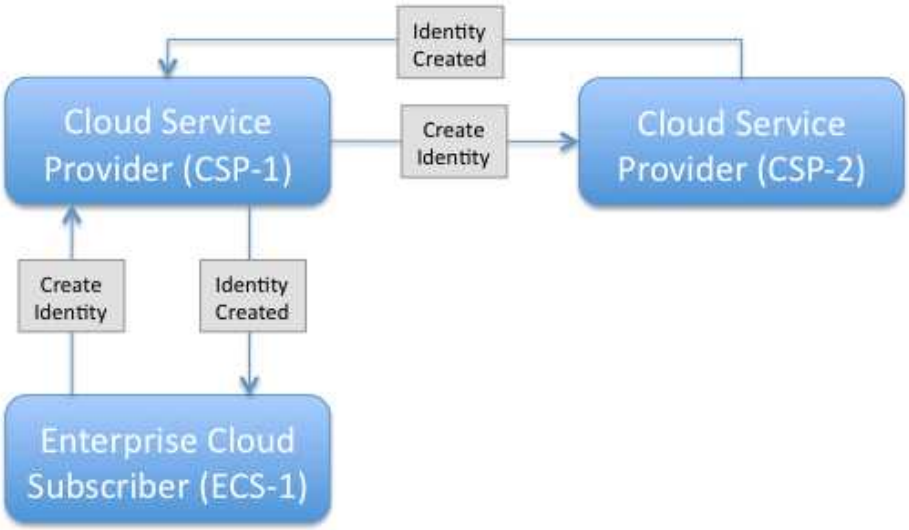

図 1：ECS を起点とする複数 CSP へのアイデンティティ・リソース生成のユースケース
（出典： http://www.simplecloud.info/specs/draft-scim-scenarios-04.html）

(3) SCIM プロトコル

前述のトリガーに基づく処理を規定しているのが、SCIM プロトコル [5] である。

SCIM プロトコルは RESTful API を指向している。クライアント（Web サイトやアプリケーション）はサービス・プロバイダ（SCIM プロトコルを受けつける Web サイト）に対し、下記の HTTP メソッドを利用してリソース（ユーザ、グループ、パスワード）操作のリクエストを発行する。そしてサービス・プロバイダからクライアントへのレスポンスは、HTTP ステータス・コードと、処理結果を含むボディから構成される。

- GET：リソース取得（全体/部分）
- POST：新規リソース生成
- PUT：リソースの変更（指定した内容で置き換え）
- PATCH：リソースの変更（部分更新）、パスワード変更
- DELETE：リソース削除

API エンドポイントは各リソースを単位として、下記のように定義されている（表 1)。

表 1: 定義済みのエンドポイント

Resource	Endpoint	Operations	Description
User	/User	GET, POST, PUT, PATCH, DELETE	Read/Modify Users
User Query/Listing	/Users	GET	Retrieve User(s) via ad hoc queries
Group	/Group	GET, POST, PUT, PATCH, DELETE	Read/Modify Groups
User Query/Listing	/Groups	GET	Retrieve Group(s) via ad hoc queries
Change User Password	/User/{userId}/password	PATCH	Change a User's password
User Schema	/Schema	GET	Retrieve a specified User schema
User Schemas	/Schemas	GET	Retrieve all Service Provider supported schemas

（出典：http://www.simplecloud.info/specs/draft-scim-rest-api-01.html#endpoint-summary）

ユーザ情報生成のリクエスト・レスポンスの例を以下に示す。リクエストはユーザ情報を POST することによって行われる。そしてレスポンスでは、当該リソースの場所が Location ヘッダとして、リソースの表現がボディとして返却される。

(リクエスト)

```
POST /User HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas":["urn:scim:schemas:core:1.0"],
  "userName":"bjensen",
  "externalId":"bjensen",
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  }
}
```

(レスポンス)

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/v1/User/uid=bjensen,dc=example,dc=com
ETag: "e180ee84f0671b1"

{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id":"uid=bjensen,dc=example,dc=com",
  "meta":{
    "created":"2011-08-01T21:32:44.882Z",
    "lastModified":"2011-08-01T21:32:44.882Z"
  },
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  },
  "userName":"bjensen"
}
```

なお API アクセスに関する認証・認可の方法は、SCIM の仕様上では定義・指定しないが、OAuth 2.0 [6] の利用を推奨している。

(4) スキーマ

コア・スキーマ [7] では、ユーザ/グループを表現する最小限のスキーマと、スキーマの拡張モデルを定義している。

SCIM のスキーマ定義は、既存のクラウド・サービス事業者の API、Portable Contacts [8]、LDAP などを参考にしている。現在定義されているスキーマは下記の通り。

- ユーザ（urn:scim:schemas:core:user:1.0)
- エンタープライズ・ユーザ（urn:scim:schemas:extension:user:enterprise:1.0)
- グループ（urn:scim:schemas:core:group:1.0)

スキーマの拡張モデルとしては、LDAP の ObjectClass の考え方を援用している。ただしシンプルさを維持するため、LDAP と異なりスキーマの継承はない。
またスキーマの表現方法としては、JSON 形式ならびに XML 形式の両方が可能となっている。それぞれの例を以下に示す。

（JSON 形式）


```
{
    "schemas":["urn:scim:schemas:core:user:1.0",
 "urn:scim:schemas:extension:user:enterprise:1.0"],
    "id": "005D0000001Az1u",
    "externalId": "701984",
    "userName": "bjensen@example.com",
    "name": {
          "formatted": "Ms. Barbara J Jensen III",
          "familyName": "Jensen",
          "givenName": "Barbara",
          "middleName": "Jane",
          "honorificPrefix": "Ms.",
          "honorificSuffix": "III"
    },
    "displayName": "Babs Jensen",
    "nickName": "Babs",
    "profileUrl": "https://login.example.com/bjensen",
    "emails": [
          {
            "value": "bjensen@example.com",
            "type": "work",
            "primary": true
          },
          {
            "value": "babs@jensen.org",
            "type": "home"
          }
    ],
(略)
    "meta": {
          "created": "2010-01-23T04:56:22Z",
          "lastModified": "2011-05-13T04:42:34Z"
    }
}
```

（XML 形式）

```
<SCIM xmlns="urn:scim:schemas:core:user:1.0"
xmlns:enterprise="urn:scim:schemas:extension:user:enterprise:1.0">
    <id>005D0000001Az1u</id>
    <externalId>701984</externalId>
    <userName>bjensen@example.com</userName>
    <name>
        <formatted>Ms. Babs J Jensen III</formatted>
        <familyName>Jensen</familyName>
        <givenName>Barbara</givenName>
        <middleName>Jane</middleName>
        <honorificPrefix>Ms.</honorificPrefix>
        <honorificSuffix>III</honorificSuffix>
    </name>
    <displayName>Babs Jensen</displayName>
    <nickName>Babs</nickName>
    <profileUrl>https://login.example.com/bjensen</profileUrl>
    <emails>
        <email type="work" primary="true">bjensen@example.com</email>
        <email type="home">babs@jensen.com</email>
    </emails>
(略)
    <meta>
        <created>2010-01-23T04:56:22Z</created>
        <lastModified>2011-05-13T04:42:34Z</lastModified>
    </meta>
</SCIM>
```

(5) バインディング

SCIM のスキーマやプロトコルを別のプロトコル仕様と組み合わせて利用するための規定を、バインディングと呼ぶ。

現在、SCIM メッセージを SAML SSO（SAML Web SSO プロファイル）に載せるためのバインディングとして SAML 2.0 Binding for SCIM [9] が定義されている。これは、SSO と同時にアカウントを作成したり、属性情報を更新する、いわゆる「Just-in-time プロビジョニング」（ユーザが利用する前に事前設定するのではなく、ユーザの初回利用と同時にアカウント生成やアクセス権限設定を実施）を目的としている。同バインディングで

は、下記の方法を定義している。

- SCIM のユーザ属性から SAML 属性へのマッピング
- SAML SSO アサーションに SCIM のユーザ属性を記述する方法
- SAML AttributeQuery を用いた SCIM ユーザ属性の取得
- SAML メタデータによる、サポートする SCIM 属性の公告

8.4. SCIM の今後

今後のスケジュールとしては、下記のように予定されている。

- 2011 年 10 月：IIW（Internet Identity Workshop）にてデモンストレーション
- 同月末：バージョン 1.0 Draft 1 公開
- 11 月 30 日：相互運用性テスト開催
- その後、バージョン 1.0 を確定し、IETF（Inernet Engineering Task Force）などの標準化機関に策定を移管する予定

8.5. まとめ

アイデンティティ・プロビジョニング API の標準化の必要性は、従来より業界内で認識されており、SPML が有望な仕様とされていたが、広く普及するまでに至らなかった。これに対し、有力クラウド・サービス・プロバイダを中心とするメンバーが、各ベンダー独自のプロビジョニング API の共通化を目指し、SCIM を策定する動きに出ている。

今後、早い段階での仕様確定と実サービスへの適用が行われれば、他のクラウド・サービス・プロバイダやアイデンティティ・プロビジョニング製品ベンダーの採用が進むものと考えられる。

以上

参考文献

[1] OASIS Provisioning Services TC
http://www.oasis-open.org/committees/provision/

[2] Technical Work Produced by the Committee
http://www.oasis-open.org/committees/provision/#technical

[3] Simple Cloud Identity Management
http://www.simplecloud.info/

[4] SCIM scenarios
http://www.simplecloud.info/specs/draft-scim-scenarios-04.html

[5] DRAFT: SCIM PROTOCOL
http://www.simplecloud.info/specs/draft-scim-rest-api-01.html

[6] OAuth 2.0 - OAuth
http://oauth.net/2/

[7] Draft: Simple Cloud Identity Management: Core Schema 1.0 - draft 1
http://www.simplecloud.info/specs/draft-scim-core-schema-01.html

[8] Portable Contacts
http://portablecontacts.net/

[9] SAML 2.0 Binding for SCIM
http://www.simplecloud.info/specs/draft-scim-saml2-binding-02.html