受動的攻撃の危険性と防衛対策

鈴　木　　斉

Abstract

The Internet is on the way to become a common infrastructure for everyone. But, it is hard to catch up the current situation for both applications and users. Many developers of applications, which depend on the Internet, like e-mail clients, did not consider about to protecting one's personal information. In this situation, scoundrels will attack people by means of to be famous, to show their skills, just for fun, and so on. There was no problem if users were doing a suitable defense countermeasure. But, now, network applications with poor consideration by their developers, users must consider about new style of attack "passive attack".

In this paper, the author classifies the types of attacks observed on the Internet, and explains the current security levels on some network client applications as a practice.

1．はじめに

この数年，世界的にインターネット利用人口が急激に増加の一途をたどっている[1][2]。インターネットの商用利用が活発になるにつれ，学術・研究目的での利用が主体であった従来とは異なり，ネットワークに接続する機器の比率も大学・企業内等で使用されていた共同利用端末の占める割合が減少し，代わりに個人利用や家庭内等の限られた少人数で利用されることを前提とした接続端末の比率が増加の傾向をたどっていることは疑う余地がない。

ネットワークや，コンピュータ利用人口が急激に増加してくると，システムの利用経験が豊富な利用者の指導の下，新規利用者に対する個別指導が可能であった従来とは異なり，システムの利用経験が豊富な者がほとんど存在しない状況で，新規利用者が利用し始めることも珍しくはない。このような状況下では，システムを安全に利用するための知識を十分に有していない利用者を対象とした攻撃が増加することになる[3]。この人材不足から発生する問題は，利用者側だけの問題にとどまるものではなく，サーバ等を管理する側にも当てはまる。管理者のなかでも十分な知識や経験を積まないままに手探りで作業を行っているものも少なくはない。このため，サーバアプリケーションがセキュリティ上，不適切に設定されたままの状態で放置されているサイトも多数存在する。また，設定は正しく設定されていたとしても，日々発表されるセキュリティ・ホール（設計上の問題や構造的欠陥）への対応が遅れ，攻撃を受けてしまうサイトも同様に多数存在する。現状では，サイト内で適切な利用者教育を期待することは難しく，利用者の危機管理意識がサイト内での教育により向上することを望むこともできない。利用者が使用するアプリケーションの設定もサーバアプリケーションの場合と同様に，相当数が危険な状態のまま放置されていることも予想される。さらに，このような利用環境では，システムの設計・開発者が予想すらしてこなかった様々な使用方法も発生する。仮に，想定されていない使用方法であっても，システムが適切に動作する限り問題はないと考えることもできる。しかし，コンピュータで扱う「情報」は姿形こそ存在しないものの，貴重な財産の一つと考えられている。貴重な財産を扱う以上，利用者が不適切にシステムを使用したとしても，少々のことではトラブルを発生させないような仕組み（fool proof）の提供が必要不可欠である。それが考慮されていないようなシステムは，完成度の低い未熟なシステムであるといえる。本来，トラブルを発生させてはいけない環境下であれば，紛れもなく欠陥システムであるといえる。

2．システム開発に対する期待と現状

コンピュータの処理能力は，年々格段に向上してきている。しかし，処理能力の高いシステムを要求することが，コストに反映されることは明白だ。発生するコストを低く抑えるためには，開発に使用するツールにかける予算を少なくし，短時間に，少人数で，効率良く開発することが期待される。Rapid Application Development(RAD)に代表される，これらの条件を満たすアプリケーション開発手法では，確かに高速なアプリケーション開発が可能である。しかし，RAD ツールで製作されたアプリケーションを検証すると，開発に使用したモジュールが確実に本来の仕様を満たしているか，他のモジュールと組み合わせて使用してもセキュリティ上の問題を発生させないか等の確認作業がほとんど行われないまま，製品化されているとしか考えられないレベルの品質のものが多数存在している。

本来，コストの低減と言えば，同じ品質，あるいは，より良い品質のものを，より廉価に提供するべきものであるといえる。しかし，現状では，多くのコンピュータアプリケーションでは，それが励行されておらず，コストを抑えるために，品質低下を招いている面が随所に見られる。開発時間を短縮するために，利用者の熟練度が高いことを仮定して，本来であれば，欠かすことができないはずの各種入力に対する検証用モジュールの作成が省略される等，利用者の目が届きにくい処理能力以外のコスト要因を軽視するケースが，従来のコンピュータシステム開発の現場では多数発生してきた。その結果，本来であれば，技術の進歩とともに，より完成度の高いシステムが実現されるべきであるが，不適切な使用に対する安全対策（fool proof)が欠落，あるいは，不足しているシステムが，現時点で稼動しているシステム中に多数存在しているといえる。

開発元・販売元等（以下，供給者）という，本来システムの脆弱性を検証する責務を有する立場の企業・団体以外であっても，既存システムの問題対

策の在り方そのものを，問題視している，有志による複数のユーザグループ[1)]等により，様々な側面から，既存のシステムが検証され，その過程で発見された不具合の修正・改善に反映させる努力が払われている。残念なことではあるが，従来はコンピュータソフトウェア開発による製造者への責任追及が厳しくなかったこともあり，セキュリティ上の不具合が発見されたとしても，人的資源の不足やコスト上の制約等の理由により，そのまま放置され続けてしまう事例が多数存在した。また，これらの不具合が修正されないばかりか，不具合そのものに関する情報を公開することさえ，公開することによるセキュリティ上のリスクが大きい等の理由を根拠として，供給者により強制的ににぎりつぶされてしまうことも少なくはなかった。

だが，インターネットの利用者人口が増加し，World Wide Web 等に見られる表現手段や，情報の連携機能等が発達した結果，現在では，一個人の立場であっても，強力に発言する環境を入手可能な時代へと突入した。この結果，利用者にとって重要な情報を隠蔽するような，供給者の態度が問題視されるようになってきている。依然として，積極的に自身の供給物に対するセキュリティ・ホール関連情報を公開する供給者数は少ない。しかし，供給しているアプリケーションに関するセキュリティ・ホールを誠実に公開し，対応をはかっている供給者も着実に増えてきている。これらの不具合に関する情報が，必要とする利用者に行き渡っているか否かという問題は存在するものの，事態は良い方向に進んでいることは間違いないだろう。

さらに，最近では[4](邦題: 伽藍とバザール）で説明されているように，オープンソース，オープンアーキテクチャ等と称し，アプリケーションの実装方法やソースコード等の情報をすべて公開し，外部からの問題点の指摘，修正を期待するスタイルの開発方法も一般に認知されつつある。しかし，現実に我々が利用可能な環境では，OS(オペレーティングシステム）を含め，実行形式のみが提供され，具体的な実装方式が利用者から隠蔽されていたり，

1）代表的なものにモデレーテッドなメーリングリスト BugTraq がある。

ライセンスによって，リバースエンジニアリング技術による解読を制限していたりするものが，大多数を占めているという状況に変更はない。

3．ネットワーク上の脅威とその対策

コンピュータを利用するまでもなく，通常の社会生活においてもセキュリティ・ホールを攻撃されることにより，個人のプライバシーに関する情報が流出する等，様々な問題が発生している。個人に対する，危機管理意識を高める啓蒙活動では，各人がセキュリティを確保するために，十分な知識を有していないことが最大の問題点であると指摘されることが多い。このことはコンピュータネットワークを利用している際にも当然当てはまる。しかし，コンピュータネットワーク上では，現実の社会と比較して，コミュニティーによる監視を受けにくいという特殊事情が存在していることを考慮する必要がある。これは，積極的かつ定期的にセキュリティに関する調査・監視を行わない限り，通常の行為と異常な行為とを区別することが非常に難しいことを意味している。また，コンピュータを使用したシステムを設計する際には，セキュリティ・ホールを完全に排除した完璧なシステムは存在しないと考えられている。これは，人間がミスをせず，大規模なソフトウェアを記述することが可能である事を立証できないためである。現時点で，安全と思われているシステムであっても，偶然，セキュリティ・ホールが現在まで発覚してこなかっただけであり，将来にわたっても安全が維持されることを保障するようなシステムは存在しない。さらに，インターネットの利用方法は常に進化しているため，現時点で想定されていない将来の使い方での安全性まで保証することは非現実的であるといえる。

3.1 コンピュータを利用する際に特有の問題

コンピュータを利用する際に特有の問題は，1）不適切な設定，2）設計・開発者による対応の不備，の2種類に大別することができる。

不適切な設定

不適切な設定で問題となる代表例としては，システムの設定・利用をパスワードで機能制限する際に，パスワードが特定の初期値で設定されていて，かつ，そのパスワードを変更することなく，運用開始ができるシステムや，空のパスワードといった不適切な設定を許可してしまうシステムが存在することが挙げられる。外部からシステムの設定を可能とする機能が存在することの危険性を，利用者が正しく理解している場合には，システムは適切なパスワードが設定され，さしあたっての問題は回避される。しかし，利用者のセキュリティに関する知識が不十分である場合には，システムによっては危険な状態のままで放置されることとなる。攻撃者にとって，初期値のままのパスワードや，空のパスワード等で放置してあるシステムを発見することは，単純かつ機械的な作業でしかないため，このような設定を許すシステムは安全なシステムとは言い難い。

設計・開発者による対応の不備

設計・開発者による対応の不備としては，装置や機能によっては，単体での防衛措置が不可能であるにもかかわらず，利用者への注意喚起が欠落しているシステム，設計・実装で安全面への配慮が不足，又は，欠落している等の構造的欠陥を抱えるシステム等がある。不適切な設定での場合とは異なり，設計・開発者による不備の場合では，利用者による対応だけでは完全な問題解決が不可能となる場合がある。システムのセキュリティ構造そのものに問題が存在する際には，通常は設計段階からやり直さない限り問題解決ができない。この問題点が発覚した際には，利用者は，供給者による正式な対応を待たなくとも，使用する機能を一部制限するか，システムの利用そのものを停止することで一時的に問題を回避できる。しかし，利用者がセキュリティ構造上の問題点が発覚していることを感知していない場合には，システムは危険な状態のまま放置されつづけることになる。このため，コンピュータシ

ステムの利用者は自身のセキュリティを確保するためには，正確な情報を迅速に入手し続ける必要がある。

3.2 機械的に実施される攻撃形態の分類

攻撃の種類を分類しようとすると，特定個人のプライバシーに関する情報を隠れて入手する場合など，攻撃後も極力，対象者に攻撃の事実を隠すタイプのものと，Web サーバの改竄等のように攻撃後は攻撃の成果を広く公開するタイプのものとに区別できる。前者のタイプの攻撃を行う際には，必要とする情報を管理するサーバ上での対象者のアカウント，若しくは，当該サーバ上の管理者のアカウントを不正に使用すること試みることになる。効率良く攻撃を実施するために，攻撃者は攻撃対象者の様々な行動を観察し，セキュリティの弱い部分を集中的に攻撃し，不正にサーバを使用するための権限を入手していく。一般的に個々の利用者がセキュリティ上の弱点になりやすく，特に個人のアカウント管理に関する意識が狙われることは想像に難しくない[2)]。システムの全利用者が適切なパスワードを設定することや，日々の記録から異常な行為の有無を検出すること等はセキュリティを高める上で非常に重要なことではあるが，以下では利用者に対するセキュリティ教育については論じないこととし，機械的に実施される攻撃の形態のみを分類する。

能動的攻撃

サーバのように常に外部からのリクエストを受け付けている場合には，攻撃者は攻撃対象を直接・積極的にサーバを攻撃できる。このような行為を指して「能動的攻撃」と分類をする。能動的攻撃の多くはサーバを対象とした無差別攻撃であり，次のような例が挙げられる。

2）肩越しにパスワードを盗み見るとか，管理者のふりをして対象者から直接パスワードを聞き出す等。

- セキュリティ・ホールを抱えたホストを機械的に探索し[3]，Webサーバ上のデータ改竄等を行う攻撃[4]
- 大量リクエストでサービスを妨害するDoS(Denial of Service)攻撃，及び，複数マシン利用でより攻撃的なDDoS(Distributed DoS)攻撃
  - UBE(Unsolicited Bulk Email)や，UCE(Unsolicited Commercial Email)を大量に処理させることで，MTA(Mail Transfer Agent)に通常の使用を行わせない攻撃[5]
  - Third-Party Mail Relay によるMTAの不正中継使用攻撃
- DNS(Domain Name Service)情報の改竄によるドメインの強奪攻撃

しかし，サーバで利用されているアプリケーションに対する脆弱性を検証するツールは既に多数存在し，サーバそのものの安全性まで含めて検証を行う商用サービスが提供可能な程に検証手段は確立されている。つまり，サーバの運営に関しては日々の管理と新規に発見されるセキュリティ・ホールの確認とを怠らない限り問題は簡単には発生しないと考えて差し支えない。

コンピュータ・ウイルス

不正なプログラムをクライアントマシンに侵入させる手段として，初期にはコンピュータ・ウイルス（以下，ウイルス）が使用された。ウイルスは感染する対象，発動方法で数種類に区別することも出来るが，一般的に，感染，潜伏，発動（自己増殖）というライフサイクルで活動を行う点で共通している。ウイルスといえども発動していな限りは，普通のプログラムやデータと

3）ネットワークの使用可能帯域によっては，探索だけでも十分に攻撃となりうる。

4）現在では，SMTP, POP3, WWW, DNS といったメジャーなサービスのみならず，rpc.statd 等の動作していることすら気づかれにくいものまで，全てのサービスが攻撃対象となっている。

5）送信時に差出人アドレスを詐称され，不必要なエラーメールを処理させられる攻撃には効果的な予防策は存在しない。

比べて特殊なことは何もない。しかし，時代とともにウイルスも変化し，現在，氾濫しているものは感染と同時に発動するタイプが主流を占めるようになっている。さらに，使用しているアプリケーションのデータ読み込み部分にセキュリティ・ホールが存在していると，データを読み込むだけでもウイルスに感染することもあるし，アプリケーション内部で使用している組み込みモジュールの初期設定が適切に設定されていない等の理由により，利用者が全く意図していないにもかかわらず，データを勝手に読み込み，かつ，ウイルスに感染させてしまうものも存在している[6)]。これら，ウイルスによる攻撃は攻撃者から見ると，全体的なくくりとしては，先の「能動的攻撃」のように積極的に攻撃しているとは言えないため，次に説明する受動的攻撃として分類される。

受動的攻撃

クライアントであっても，長時間ネットワークに接続し続けるものが多数存在するようになるにつれ，サーバと同様に，クライアントもネットワーク上で攻撃される事例が目立つようになってきた[1]。サーバが攻撃される際には，直接的，間接的といった違いはあるものの能動的に攻撃されることがほとんどである。クライアントも同様に能動的攻撃の対象となることがある。クライアントマシンといえども，最近のものは，高機能になっており，特定の情報を絶えず近隣のクライアントと連携してデータを更新することがあり，実装上の問題から，ここに脆弱性が生じることがある。この手の脆弱性が攻撃に使用されることを防ぐ目的として，ファイアウォールが導入されることが多いが，ファイアウォールを導入したとしても，クライアントのセキ

6）昨年までは，「メールを受け取った瞬間にウィルスに感染する」というのは，話そのものを広めることが目的の「デマウィルス」とされていた。しかし，現在では，そのような実際にウィルスも登場した。

ュリティそのものが向上しているわけではないことを留意する必要がある。

無差別攻撃を実施する際には攻撃者は能動的である必要はない。攻撃の引き金となる行為だけを攻撃対象の行動に委ねるだけであり，残りのプロセスは従来の攻撃方法と大差ない。この方式で問題視される点は，この方式では，ファイアウォールの内側に存在するマシンをも攻撃が可能な点である。この方法では，攻撃が攻撃対象の行動に依存する。このため攻撃者から見て「能動的」ではないという意味から「受動的攻撃」として分類される。受動的攻撃の多くはクライアントを対象とした無差別攻撃であり，次のような例が挙げられる。

- 利用者に適切な判断をさせないまま，ダイヤルアップ接続の設定を変更するソフトウェアをインストールさせ，意図しない国際電話等を強制的に掛けさせる
- 特定 Web サイトを訪問するだけで，Web 上のどのサイトを閲覧したかの情報や，クライアント上にあるファイル（EX. パスワードを保存しているファイル）等を盗む
- 特殊な機能を持つアプリケーションで電子メールを受信した際に，別のアプリケーションを無許可で実行し，ハードディスクの中身を強制的に消去する。

複合攻撃

受動的攻撃では，攻撃者は攻撃対象が接続して来るまで待つ必要がある。Web ブラウザを利用するのであれば，攻撃対象が特定の Web を閲覧してくるまで，攻撃は実施されないことになる。しかし，能動的攻撃と受動的攻撃とを組み合わせて効率的に攻撃を行う事例が既に多数存在し，以下のようなものが確認されている。

- 能動的攻撃でサーバを乗っ取り，受動的攻撃用プログラムの配置に利用
- DNS 情報を不正に改竄し，受動的攻撃用プログラムを準備した偽の

サイトに誘導[7)]

- メールを利用して受動的攻撃が成立した後，攻撃対象から入手した情報をもとに，攻撃対象の機能を利用して，さらなる，受動的攻撃を発生させるメールを送りつける

3.3 防衛対策

コンピュータシステムは，特定の個人・団体等への攻撃手段として，攻撃対象となることが考えられる。しかし，近年発覚した攻撃を基に考察すると，発覚する攻撃の多くは，個人情報の大量搾取，破壊活動等の単なる嫌がらせ，技術的能力誇示等の攻撃者の自己満足や，自己主張の手段等，不特定多数が対象となる攻撃であるといえる[1][3]。インターネットが成立した当初であれば，個々の機器は間欠接続でネットワークを構成していた。また，個々の機器にとっては接続の相手先も特定少数であったため，攻撃者が特定できないように隠れて攻撃することは難しかしく，結果として発覚した攻撃も，ごく少数であった。しかし，現在のネットワークは，論理的には不特定多数のコンピュータが相互に直接接続しているように構成され，サーバは常に外部からのリクエストを受け付けている。このため，サーバは24時間，常に攻撃の脅威にさらされている。また，自己主張が目的の攻撃では，話題性の面から，サーバを攻撃対象とするメリットが存在する。クライアントが攻撃を受ける理由も，サーバとほぼ同様な理由であると考えられる。しかし，現在のクライアントの台数は，サーバと比べ非常に大量な数が存在し，かつ，能力的にも従来の高性能マシンの能力をはるかにしのぐものであるため，DDoS攻撃での兵隊として使用する目的で攻撃を受けることもある。

上述の通り，ネットワークに接続している限り，何時，攻撃を受けたとし

7）SSL(Secure Socket Layer）で経路上の通信を暗号化し，流通する情報の安全性を高める方法すら無意味となる。

ても不思議ではない時代に突入している。[3]等の報告から推察されることとしては，[3]等の報告を基にして攻撃方法を考えつく模倣犯が多数存在していることが予想される。つまり，攻撃側も防御側も同一のリソースを参照していることになる[8]。このため，セキュリティ対策を正しく行うためには，少なくとも，攻撃側が入手するものと同等か，それ以上に適切なニュースソースを用意する必要がある。また，入手する情報が別なセキュリティ・ホールを発生させるためのデマかもしれないという問題も存在するため，信頼できる情報筋の確保，及び，一次情報の確認が重要となってくる[9]。

3.3.1 能動的攻撃に対する対策

先に述べた通り，能動的攻撃に分類される攻撃については，日々の状況を監視することで，通常の状態と異常な状態とを区別することは可能である。また，既知のセキュリティ・ホールが存在しているかを確認する分野は既に商用のサービスが提供可能なほどに成熟している。商用のサービスを受けられなくとも，利用者自身が実際に能動的攻撃を実施して脆弱性が存在しているか確認をしてみることもそれ程難しくはない。

3.3.2 受動的攻撃に対する対策

受動的攻撃に分類される攻撃は利用者の行動をきっかけとして攻撃が開始される。このため，攻撃の引き金となる行為さえ行わない限りは被害も防ぐことができる。つまり，適切な情報収集により当該機能を利用不可に設定，若しくは，使用を中止し，問題のあるシステムの早期修正を促す等の行動によって，多くの被害が回避可能である。

8）利用者による有効な対策は迅速な情報収集となる。

9）一次情報の存在が確認できない情報は他人に流さないことが賢明な措置である。

3.3.3 一般的な攻撃に対する対策

能動的・受動的攻撃に対する対策を実施することで，ほとんどの攻撃への対応が可能である。攻撃に対する情報収集が重要であることは当然であるが，デマに惑わされないためにも，情報の信頼性を検証する手段の確保，つまり，情報を信頼するための判断基準を確立する必要がある。

4．ネットワークアプリケーションの現状

現在，アプリケーション開発の現場ではRADツールに見られるように，高機能なモジュールを組み合わせて開発を行うことが多い。この組合せ方式による開発方式は，一見すると生産効率が向上するように見える。しかし，高機能なモジュールほど，存在する機能に関する詳細な説明が必要となるにもかかわらず，現状では説明は十分に用意されているとは言いがたい。このため，開発現場では実際の動作を調査しつつ，モジュールの組み込み使用が検討されている。また，システムとともに提供されているモジュールであっても，作りこみが甘いものが多く，標準システムで使用されているのだから，安心ということもないばかりか，OSの修正が入る際，突然に仕様が変更され，モジュールに渡す引数の個数さえ変わってしまうことすらある。つまり，RADツールによるアプリケーションの開発現場では，開発者は利用者が使用する環境での，確実な動作検証が行えないこととなり，RAD開発のアプリケーションは攻撃者の餌食となりやすい。また，高機能なモジュールは，攻撃用としても非常に利用価値の高いAPI(Application Programming Interface)を多く含んでいることもRAD開発によるアプリケーションが攻撃に利用されやすくなる側面を持つ。

利用者が個人のプライバシーをネットワーク上でも安全に保護しようとすると，利用者は各アプリケーションが個人のプライバシーを適切に保護できるように設計・実装されているか確認しておく必要がある。ソースコードが公開されているものであれば，ソースコードから様々な事実を確認すること

が可能である。しかし，ソースコードを読めない，又は，読まない利用者にとっては，ソースコードが添付されていようとも，添付されていないことと何ら変わりはないし，また，[4]で説明されているように，バザール方式でアプリケーションが開発されているとしても，大半のアプリケーションでは[4]中の説明に登場する具体例ほどに，良好な開発は行われていないというのが現実である。バザール方式では，ソースコードが広く公開されていても，セキュリティ面に興味のある人間がバザール（開発）に参加していない限り，セキュリティ面での向上は望めないということが，様々な具体例から確認されている。

ソースコードが添付されていないアプリケーションでは，本当にプライバシーを保護できているか確認することはほぼ無理であることは明白であろう。そこで，少なくとも攻撃者による悪意のあるプログラムを受け付けて，勝手に実行してしまうことがないかといった点を確認することとなる。最終的にはソースコードの有無にかかわらず，悪意のあるプログラムを勝手に受け付けて実行することがないことをアプリケーション毎に実際に使用して確認する以外に具体的に実装を検証する方法はないと考えるべきであろう。

4.1 ネットワークアプリケーションを実現するルール

インターネットは設計当初から，異種コンピュータシステム同士をネットワークで接続するように定められていた。このため，特定の機器が使用している内部表現を「標準」として，他のシステムが当該システムの表現形式に合わせる方式は採用していない。ネットワーク経由でデータを交換する際の通信規約[5]（プロトコル）を定め，すべてのシステムが，自身の内部表現で保持されているデータを規約に沿う形へと変換し，通信相手が，ネットワーク上で使用されている形式から，自身の内部形式へと変換する方式を採っている。このプロトコルは，RFC[6](Request For Comments)の一つとして世間に公開される。広範囲でのテストや有識者の意見をもとにIETF(Internet

Engineering Task Force)により標準化作業が実施され，最終的にIESG(Internet Engineering Steering Group)の承認を得て，標準プロトコルSTD(Standard)となる。

RFCとして公開されたものに対して得られた意見をもとに，改訂版のRFCが再度提出され，仕切り直しが行われる場合等もあり，標準となるまでには非常に長い時間が必要となる。例えば，ファイル転送に使用されるFTPの場合では，1985年に登場したRFC959で標準化作業がいったん終了し，STDでは9番となった。一方で電子メールの場合では1982年に登場したRFC821で転送方法が，その書式はRFC822でほぼ仕様が固まったものの，標準化作業に区切りが付いたのはFTPの標準化後で，STDでは10，11番として登録されている。

RFC文章では解釈する際に揺れが発生しないように，使用する表現方法から単語まで細心の注意を払い記述されている。それでも各アプリケーションでの実装を試して見るかぎりでは，誤解による揺れは存在するようだ。また，残念ながらRFCやSTDは各プロトコルの実装者に対し強制力が存在しないため，アプリケーションの供給者によっては積極的にRFCを解釈して忠実に実装を心がけようとはしていないのではと，疑わしいものも多数存在している。

RFC実装に見られる揺れ

RFCでは表現方法や使用する単語にも注意が払われていると述べたが，この書式もRFC2223として公開されている。これは，単純なテキストを表現する方法一つであっても，各種コンピュータの内部表現形式では行末の処理方法や，アルファベットの符号化方式等が統一されていないことに一因がある。プロトコル仕様に対する技術的要求レベルを示す用語としては，以下の5種類が用意され実装者に誤解が生じないようになっている。

(1) MUST/REQUIRED/SHALL (必須)

(2) MUST NOT/SHALL NOT (禁止)
(3) SHOULD/RECOMMENDED (推奨)
(4) SHOULD NOT/NOT RECOMMENDED (非推奨)
(5) MAY/OPTIONAL (オプション)

(3)は実装をすることが望ましいものの，あえて実装を取り止めることが可能であることを，(4)は実装を行わないことが望ましいものの，実装することを少なくとも禁止はしていないことを示している。しかし，先に述べたように，RFCそのものに強制力が存在しないため，必須事項であっても正常に実装されていないこともあれば，RFCを熱心に確認していないのか禁止事項を無視している実装が世に登場することもある。また，RFC文章の内部で使用される単語は極力，当該RFC内部で厳密に定義されているが，RFC 959 (FTP)でのLISTコマンドのように，肝心な表現方法が実装依存となってしまっている等，実装者泣かせな部分も存在する。

さらに，少し前までのRFCの場合では日本語等マルチバイト文字を使用する文化圏に対する考慮はもちろんのこと，ASCII文字のみで表現できない言語に対する配慮はほとんど存在しなかった。このため，日本語での電子メールの利用では，ASCII文字のみ定義されている所にエスケープ文字と呼ばれるものを含むものとして解釈する方式とし，メール本文では，日本語文字列をISO-2022-JPで符号化する方式（後にRFC1468として公開された）を使用する等，各地域毎に，オリジナルのRFCでの表現を拡大解釈する実装が行われてきた。このため，メール配送を行うシステムの実装方式によっては，このエスケープ文字を削除してしまい日本語を含む文章を正常に転送できないようなシステムもいまだに存在している。

4.2 ネットワークアプリケーションの脆弱性

本文ではネットワークを利用するアプリケーション中でも基本的な電子メールクライアントMUA(Mail User Agent)，及び， MTAについてセキ

ュリティ・ホールを発生させうる脆弱性について考察する。MUA でのセキュリティ対策といっても，標準的な電子メールを使用する際の約束上では，

1）受信者だけでは真の発信者を特定できない

2）本文は第3者に読み取り可能な形で配送される

言うなれば文面がすべて印刷された葉書のような，セキュリティなど一切考慮されていないシステムであるため，

1）闇雲に記述された内容を信用する

2）安易に自身が関係のない第3者に公開する予定のない情報を記入する

3）使用者が読み出したメールと連動させて外部アプリケーションを自動的に起動する設定にしている

等の行為を実施しない限り，問題が発生することなどありえないように見える。しかし，各種クライアントを検証した結果，多数のクライアントで以下に示す深刻かつ根本的な問題を抱えていることが確認された。

1）POP3/SMTP サーバからのエラーメッセージが異常に長い場合に，不正処理が発生し，最悪，悪意が存在する任意のプログラムを実行することも可能となる

2）Subject 行等で一行の長さが異常に長い場合に不正な処理が発生し，最悪，任意のプログラムを実行させることが可能となる

また，HTML 形式で届いたメールを HTML として解釈するクライアントの場合には，HTML を解釈するエンジンに依存する危険性も考慮する必要がある。クライアントによっては，明示的に HTML を解釈する機能を停止させることができないアプリケーションが存在し，記述内容に従って，後の攻撃に必要となるツールの入手が行われ，最終的には，利用者の意図に関係なく，その入手したアプリケーションが実行されてしまうことがある。

外部からソースコードなしに，アプリケーションを検査する場合であっても，クライアントが正しく仕様を満たしているか，また，規格外のデータを無理やり入力として使用した際に，どのような状態に陥るか等はそれなりに

検証することができる。規格外の非常に大きなデータをクライアント上の特定領域への入力とした際の動作を確認することで，実際にソースコードを確認するまでもなく，固定長の領域でデータを受け取ろうとしているか，とか，その固定長の領域内だけでデータを正常に受け取ることが，可能か不可能かさえも確認をしないまま，固定長領域にデータを転送している，といったことが判断できる。このように入力長のチェックを無視して，直接データを取り込むと，使用しているシステムによっては，最悪，任意のコードを実行させることが可能となってしまう。

一方，MTA での実装を確認してみると，接続する MTA や MUA 側が頑健な作りとなっていることを期待してか，規格外のデータが存在していたとしても，何ら加工もせず，MUA に規格外のデータを渡してしまっているものが存在した。

4.2.1 MUA の現状

異常な長さの入力を正常に処理しないと外部から簡単に任意のコードを実行させることができる問題が存在するため，各アプリケーション上でソースコードなしに確認が可能，かつ，検証が必要となる項目は，アプリケーション内部で使用している固定長領域の取扱い方だといえる。電子メールに関する RFC で基本となる RFC821，RFC822で定義されている項目内で，明示的にサイズが指示されているものは，以下の通りとなっている。

user 部（メールアドレス中で@より前の部分）：	64文字
domain部（メールアドレスの@より後の部分）：	64文字
メールアドレス[10]：	256文字
コマンドライン（行の終端CRLF[11]を含む）：	512文字

10）現在は DNS の MX レコードでの配送が主流となっているため，めったに使用されることはなくなっているが，仕様上ではメールの配送を特定のホストに指示することが可能なため，user 部＋ domain 部以上の領域を許容している

応答行（行終端のCRLFを含む）： 512文字
メールの各行（行終端のCRLFを含む）： 1000文字
配送先： 100個所

MUA の実装状況

最近の MUA は，非常に高機能となっているため，メールの送受信以外にも様々な機能を有している。今回はメールを受動的攻撃に対する健全性を検証することが目的なため，

- 受信するメール内に規格を無視した，異常な長さのデータを存在させた場合の動作状況
- 次節で検討する MTA での異常なデータの取扱いを検証する目的のためメールアドレス部等に規格外の文字[12]を入力した際の処理
- 各項目での異常な長さの入力を存在させた場合の処理

を検証することとし，検査対象には，長崎大学経済学部で現在よく使われているメールクライアントから3つを選択した。検査結果を表にまとめた結果を以下に記す。

MUA ＼ 検査項目	A	B	C	D	規格外の文字
AL-Mail32 Ver. 1.11	×	×	×	×	×（Shift JIS で出力）
EdMax Ver. 2.77	○	○	×	×	◎（出力しない）
Outlook Express Ver. 5.5	×	○	○	×	△（利用者に問う）

A：異常な長さのメールアドレス　B：異常な長さの応答行
C：異常な長さの本文中の一行　D：異常な長さの Subject 行
○：正常に処理が可能　×：異常動作を発生

表1　各 MUA での実装状況

11）MSDOS や Windows 等で使用されている行末形式
12）いわゆる全角文字のみで構成されるメールアドレス等

なお，メールの一行あたりの文字数を配送する際に1000文字以下に丸める処理を行うものはなかった。Subject 行で各 MUA ともに失敗するのは恐らく Windows 上で提供されているローレベルな API にそのままサブジェクト行のデータを渡していることが原因と思われる。

4.2.2 MTAの現状

MTA では，すべての MUA が規格を正しく守り規格に沿ったメールを MTA への入力としていると仮定すれば，規格通りの入力が行われたか確認する必要はないと考えることも可能だ。しかし，前節での結果を見る限りでは MUA 側では，真面目にフォーマットが規格どおりであるかチェック・修正していないものが多数存在すると考える必要がある。また，MUA の受信機能も数例検証しただけではあるが，異常なフォーマットのメールが来た時の防御体制も安全なものとは言い難い。現状では，サーバ側で極力，異常なフォーマットを検出し可能な範囲で修正する必要があるようだ。今回は，セキュリティ面で優れていると評判も高く，長崎大学経済学部で現在使用し

検査項目 / MTA	A	B	C	D	E	F
sendmail Ver. 8.10.1	◎	○	○	○	○	△
qmail Ver. 1.03	×	○	○	×	×	×

A：異常な長さのメールアドレス　　B：異常な長さのコマンドライン行
C：異常な長さの応答行　　D：異常な長さの本文中の一行
E：同時配送先数の取扱い　　F：規格外文字の取扱い
◎：エラーを出し，入力を受け付けない
○：正常に処理可能，かつ，他の実装に対し危険な入力を渡さない
×：正常に処理可能，しかし，他の実装に対し危険な入力を渡すことがある
△：ほとんどの規格外文字は受け付けないが，一部は受け付けてしまう

表2　各 MTA での実装状況

ている qmail[13]，及び，MTA のデファクト・スタンダードの地位を確立している sendmail[14]について検証することとした。

MUA での実装を検証した結果より規格外の文字を入力した場合について sendmail での実装をソースコードで確認してみた所，日本語の符号化方式で言う所の Shift JIS の場合のみエラーとして弾く処理が可能なようになっていた。しかし，大抵の場合ではそのまま出力されてしまう作りであった。

5．MUA/MTA の評価結果への考察

各アプリケーションの安全性を様々な団体が検証していることは前述のとおりである。しかし，これらの団体に利用者が使用しているすべてのアプリケーションの検証を望むことはできない。また，これらの団体が検証作業を行ったからといって，バザール方式にも似た検証方式では，確実にすべての検証項目を網羅しているのか，利用者が判断することもできない。MTA に対する検証システムは既に世の中に存在するが，現状では当該の MTA が堅牢に作成されているかの検証であって，他のサーバやクライアントに対する配慮が適切であるかどうかといった検証を行うツールは存在していない。また，MUA に対する既知のセキュリティ・ホールを検証するツールも存在はするが，容易に確認可能であるはずの，固定長領域に対する安全性を確認するツールは見当たらない。本来，これらの検証は開発の段階で既に終了しているはずのものではあるが，前述の検証結果を確認する限り利用者が検査をしておく価値は存在するといえる。こうした検証作業はシステマティックに実施することが可能であり，他の利用者の便宜をはかるためにも検証手法・結果は公開されるべきものであるといえる。また，最終的な総合判断を利用者個人で下すためにも，標準的な検証システムが必要となることは明白である。

13) http://cr.yp.to/qmail.html
14) http://www.sendmail.org/

6．終わりに

今回，長崎大学経済学部で使用している電子メール関連のツールに関して脆弱性を確認してみた。検証するMTAとして選択したsendmail等は実装のお手本となるようにかなり厳格に作られていることを確認することができた。しかし，所々で他のメール関連ツールとすり合わせるために厳しい検証があえて行われていないことも併せて確認することとなった。また，クライアント側のツールがなぜ，これほどまでに単純な検証作業で未然に防ぐことが可能となる問題点を抱え込んでいるのかについて検証をしていくと，開発現場ではAPI等の制限事項を確認することなしに，安易に使用しているのではないだろうかと疑ってしまう結果が得られた。本論文に見られる通り，ネットワーク上での利用者の安全性が適切に考慮されていない現状では，利用者は自身が被害者や間接的な加害者とならないためにも，各種アプリケーションを検証して使う必要があるといえ，関連する最新の情報を入手しておくことは，ネットワークを利用する者が最低限行うべき防衛策といえる。

また，現在，長崎大学経済学部で使用しているMTAは今回の結果を考慮して全角文字のShift JISで送信されたメールアドレスの変換機能やメール本文の異常な長さの一行等を安全な長さへと変換をしてから外部へと送信するように修正を加えておいたことを付け加えておく。

参考文献

[1] 日本インターネット協会，『インターネット白書2000』，株式会社インプレス，2000.
[2] Internet Software Consortium,“Internet Domain Survey”，http://www.isc.org/ds/，2000.
[3] コンピュータ緊急対応センター，『活動概要［2000年4月1日～2000年6月30日］』，http://www.jpcert.or.jp/pr/2000/pr000003.txt, 2000.
[4] Raymond, E. S., “The Cathedral and the Bazaar”，O'Reilly, 1999.
[5] 笠野 英松監修，『通信プロトコル事典』，株式会社アスキー，1996.
[6] 笠野 英松監修，『ポイント図解式インターネットRFC事典』，株式会社アスキー，1998.