ドメイン認証をメールの受信側に普及させるための プログラムの設計と実装

山本和彦
IIJ イノベーションインスティテュート

迷惑メール対策の次の段階として、ドメイン認証技術を受信側へ普及させる取り組みが続けられている。しかし、正当なメールを拒否する可能性があるため普及は進んでいない。本論文では、IP 方式のドメイン認証と署名方式のドメイン認証を組み合わせて、互いの欠点を補うという Wong 氏のアイディアを実現する方法について述べる。作成したポリシー記述言語を用いれば、すべてのドメイン認証方式で不当とされた場合にのみ、メールの受信を拒否するといった規則を記述できる。作成したプログラムは、記述されたポリシーに従い Milter プロトコルを通じて MTA の動作を操作する。複数のテストを通じてこのプログラムの安定性を確認し、フリーソフトウェアとして公開した。

Design and Implementation to Deploy Domain Authentication Technologies in Receiver Sides

Kazuhiko YAMAMOTO
IIJ Innovation Institute Inc.

As the next step of anti-spam activities, promotion to deploy domain authentication technologies in receiver sides is going on. However, such technologies are not widely accepted yet since legitimate e-mail messages might be rejected. This paper describes a way to implement Wong's idea that combining IP-base and digital-siginature-base domain authentication would solve the problem. With our implemented language, we can specify a policy to reject e-mail messages only when all results of the domain authentications are failure. Our server implementation controls an MTA through the Milter protocol according to the policy language. We have confirmed stability of our implementation through multiple tests and released it as free software.

1 はじめに

迷惑メールという深刻な問題を解決するために、IETF[1] が対策プロトコルを標準化したり、ISP や ASP が対策技術を導入したりといった努力が続けられている。日本では JEAG(Japan Email Anti-Abuse Group)[2]が積極的な啓発活動を継続しており、その勧告では以下の対策技術を順に普及させるべきであると述べている。

1. OP25B、SMTP AUTH[1]、投函ポート [2]
2. IP 方式のドメイン認証 (SPF[3]、SenderID[4])
3. 署名方式のドメイン認証 (DomainKeys[5]、DKIM[6])
4. ドメインのレピュテーション

日本では ISP や ASP などの努力により、OP25B、SMTP AUTH、および投函ポートを普及させることに成功した。また、WIDE プロジェクトが実施するドメイン認証の普及率の調査[3]から、IP 方式のドメイン認証が送信側に普及したことも分かる。次の段階へ進むためには、IP 方式のドメイン認証を受信側に普及させること、署名方式のドメイン認証を送受信側両方に普及させることが必要である。

しかし、正当なメールを不当だと判断してしまう可能性があるために、ドメイン認証を受信側へ普及させるのは難しいと言われている。たとえば、メールを転送すると始点 IP アドレスが変わるために、IP 方式のドメイン認証による検証では、正当なメールも不正だと判断されてしまう。また、メーリングリストに投稿されたメールは、メールサーバーによって件名などが変更されるために、署名方式のドメイン認証による検証では、同じ問題が発生する。

本論文では、ドメイン認証を受信側へ普及させるための方法として、IP 方式のドメイン認証と署名方式の署名認証の欠点を互いに補う手法について議論する。受信側のポリシーを記述する言語を定義して、Milter

[1] http://www.ietf.org/
[2] http://jeag.jp/
[3] http://member.wide.ad.jp/wg/antispam/stats/index.html.ja

プログラムとして実装し、実環境で動作させて安定性を確かめるとともに、簡単なログ解析を試みる。

2 問題解決のアイディア

2005年、SPFの設計者であるWong氏が、迷惑メール対策カンファレンスのために来日した。その際、著者はWong氏と上記の問題について議論したところ、「IP方式のドメイン認証と署名方式のドメイン認証を組み合わせて使うとよいのでは」というアイディアをいただいた。

IP方式のドメイン認証は、転送メールを誤判定する反面、メーリングリストのメールに対しては正しく判定する。逆に、署名方式のドメイン認証は、メーリングリストのメールを誤判定し、転送メールは正しく判定する。そこで、両方あるいは一方のドメイン認証でドメインが正当であると判定されれば、そのドメインは正当であると判断する。

2005年当時は、特許の問題やプロトコル策定の混乱から、どのドメイン認証技術が自由に利用できるのか分からなかった。しかしながら現在では、SPF、SenderID、DomainKeys、そしてDKIMを利用できることが分かっている。そこで、これらのドメイン認証の内、少なくとも一つでドメインが正当であると判定されれば、ドメインは正当であると判断する。

IP方式のドメイン認証では、転送元が転送したという情報を加えることで転送問題を解決できるが、ドメイン認証を導入する受信側から見ると、それは受け身の方法であるし、すべての転送元が情報を付加するとは限らない。Wong氏のアイディアのように、受信側で採用できる対策技術も必要だろう。そこで本論文では、このアイディアを実現する方法について議論する。

3 ポリシー記述言語の設計

上記のアイディアを実現するため、そして柔軟性を持たせるために、受信側のポリシーを記述するための言語を設計した。また、実装では特定の受信MTAを改良するのではなく、MTAの振る舞いを変更できるプロトコルであるMilter[4]を利用することにした。ポリシー記述言語は、Milterと相性がよくなるような設計を心がけた。

[4]https://www.milter.org/

具体的な例を挙げると、SMTP MAIL FROMの時点でSPFの認証結果は判定できる。このときMTAに受信するよう通知すれば、たとえばSMTP DATAでDKIMの署名を検証する必要はない。このように、Milterプロトコルの対話的な特性を活かし、できるだけ不要な処理を省けるようにした。

3.1 ブロックと制御コマンド

Milterプロトコルと相性がよくなるように、言語の骨格として、`connect`、`mail_from`、`header`、`body`というブロックを用意した。`connect`はSMTPコネクションが張られた際の動作を記述するブロック名である。`header`と`body`は、SMTP DATAに対応する。また、MTAの制御コマンドとして、Milterプロトコルで定められている`accept`、`discard`、`hold`、`reject`、`continue`を用意した。それぞれ、受信、受信するが破棄、受信を一時停止、受信拒否、次のブロックの評価に進むである。

SMTPコネクションを受け取った際に、すべてのメールを受信するポリシーは以下のように記述できる。

```
connect {
    accept;
}
```

3.2 条件式

条件は、制御コマンドの後に“:”を付け、「変数名“==”値」か「変数名“!=”値」で記述する。変数名は“#”で始まる。右辺の値はカンマで区切って列挙する。ドメイン認証の値のリテラルは文献[7]に従って書く。以下は、SPFの認証結果が`softfail`か`hardfail`であれば、受信を拒否する設定である。

```
mail_from {
    reject: #spf == softfail, hardfail;
    continue;
}
```

例のように、右辺が複数の値がカンマで区切られて列挙されている場合、“==”はどれか一つに一致すれば真、“!=”はどれにも一致しなければ真を意味する。この例で、条件が真とならなければ、次の制御コマンド`continue`が実行され、次の評価対象は`header`ブロックとなる。

ブロックの最後には、条件の付かない制御コマンドを記述する必要がある。continue であっても、省略はできない。すべてを明示的に記述すべきだという考えから、そう設計した。

3.3 IP アドレス

ポリシーを記述する際に、IP アドレスやドメイン名も参照したい場合がある。そこで、SMTP コネクションの始点 IP アドレスを表す変数を #ip とし、IPv4 アドレスと IPv6 アドレスの値はそのまま記述できるようにした。以下は、ローカルホストからの送信メールに対して、ドメイン認証を省略するための設定例である。

```
connect {
    accept: #ip == 127.0.0.1, ::1;
    continue;
}
```

3.4 ドメイン名

次にドメインであるが、値はダブルクオートで囲んで記述する。問題となるのは、メールのドメインとは何かである。SMTP MAIL FROM に記述されているドメイン名であろうか？あるいは、From: フィールドに書かれているドメイン名であろうか？この言語では、以下のように複数の変数を用意することで、この問題を解決した。

- #mail_from – SMTP MAIL FROM に指定されたドメイン名
- #from – ヘッダーの From: に指定されたドメイン名
- #pra – ヘッダーから PRA[8] で決定されたフィールドに指定されたドメイン名
- #dkim_from – DKIM の d パラメーターで指定されたドメイン名
- #domainkeys_from – DomainKeys の d パラメーターで指定されたドメイン名

3.5 論理積

各制御コマンドのつらなりは、条件の論理和であると解釈できる。これまでの文法で、条件を記述するのに欠けているのは論理積であるから、論理積演算子 && を導入した。これを用いて、DKIM 署名を検証せずに、DKIM-Signature: フィールドの存在を確認するだけで、迷惑メールだと判定するポリシーの例を示す。

```
body {
    reject: #mail_from == "example.com"
         && #sig_dkim == No;
    continue;
}
```

上記の例にある #sig_dkim は、DKIM-Signature: フィールドの存在を表す変数である。Yes が存在を表し、No が存在しないことを表す。よって、この例は SMTP MAIL FROM のドメインが “example.com” で、かつ DKIM-Signature: フィールドが存在しなければ、メールの受信を拒否する。

なお、body ブロックの最後で continue を命じた場合、MTA はデフォルトの動作に従う。Milter プロトコルでは、デフォルトの動作を「受信」と定めている。

3.6 変数の型

これまでの説明から分かるように、変数は型を持つ。型の種類は、IP アドレス、ドメイン名、ドメイン認証の結果、およびフィールドの有無である。表 1 に利用できる変数すべてを型で分類して示す。

それぞれの変数は、値が設定されるブロックが決まっている。たとえば、#spf は mail_from ブロックで設定され、これ以降のブロックで利用できる。connect では利用できない。変数が設定されるブロックを表 2 示す。

3.7 文法のクラス

設計した言語の文法は文脈自由文法であり、文脈自由文法の中で最も小さなクラスである LL(1) に属す。サーバーの動作を記述する言語は、サーバー管理者が理解しやすい文法であることが重要であり、本論文で提案するポリシー記述言語はこの要請を満たすといえる。

一方、字句解析ではバックトラックが必要となる。その理由は、IPv4 アドレスと IPv6 アドレスに対して固有の引用記号を与えなかったために、先頭の数字が重複するからである。

表 1: 変数の型

型	変数
IP アドレス	#ip
ドメイン名	#mail_from, #from, #pra, #dkim_from, #domainkeys_from
ドメイン認証	#spf, #sender_id, #dkim, #domainkeys
フィールドの有無	#sig_dkim, #sig_domainkeys

表 2: 変数がセットされるブロック

ブロック	変数
connect	#ip
mail_from	#spf, #mail_from
header	#from, #pra, #sender_id, #dkim_from, #domainkeys_from, #sig_dkim, #sig_domainkeys
body	#dkim, #domainkeys

4 Milter プログラムの実装

ポリシー記述言語と MTA から Milter プロトコルを通じて受け取る情報を元に、MTA の挙動を指示する Milter プログラムを実装した。このサーバーの名前は、Receiver Policy Framework (RPF) とした。すべて純粋関数型言語 Haskell [10] で記述しており、できる限り部品化を心がけた。RPF プログラム本体は、設定ファイルの解析、Milter プロトコルの取り扱い、ポリシー記述言語の解釈、そしてログの記録といった機能のみを持ち、その他の機能については以下に示す独自に開発したライブラリーを利用する (図 1)。

- ドメイン認証用のライブラリー `domain-auth`
- DNS 問い合わせライブラリー `dns`
- IP アドレスの経路表ライブラリー `iproute`
- 簡易のパーサーコンビネーター・ライブラリー `appar`
- prefork を用いたサーバーライブラリー `c10k`

この章では、RPF の実装する際に工夫した点について述べる。

4.1 ポリシー記述言語の実装

Haskell を採択した理由は、Parsec[9] というパーサーコンビネーターが提供されており、ポリシー記述言語を実装するのが簡単だからである。Parsec では、小さなパーサーを作成し、それらを組み合わせることで、より大きなパーサーを作成できる。また、Parsec を使って文法規則を記述したプログラムは、Haskell そのものである[5]ことも特筆すべきである。Parsec で記述したプログラムは、Haskell コンパイラーでコンパイルできるので、第 6 章で述べるように、たくさんの誤りを発見できる。

Parsec は、Haskell の高階関数を利用して、字句解析器とパーサーを合成できるので、ポリシー記述言語の実装では字句と構文を同時に解析する。また Parsec では、LL(1) と LL(∞) を try コンビネーターで切り替える。3.7 節で述べたように、IP アドレスの判定にはバックトラックが必要であるから、この解析のみに try コンビネーターを用いた。

作成したパーサーでは、ポリシー記述言語の型について厳密に検査する。また、変数が利用できないブロックに記述されたエラーも検出する。

4.2 Milter プロトコル

Haskell で Milter プログラムを書く場合には 2 つの選択肢がある。1 つは、C 言語の libmilter に対し Foreign Function Interface の FFI[10] を使ってフック関数を提供すること。もう 1 つは、Milter プロトコル自体を Haskell で実装することである。前者の方法では、2 つの言語のデーター構造の違いを吸収することなどが煩わしいと予想されたので、後者を採用した。

ネットワークプロトコルの典型的な実装方法には、イベント駆動方式とスレッド方式がある。イベント駆動方式はプログラムの見通しが悪くなる欠点があるこ

[5]たとえば Yacc は、Yacc の文法を C のプログラムへ変換する。たとえば、C コンパイラーの機能が改善されたとしても、Yacc の文法の検証が高度となる訳ではない。

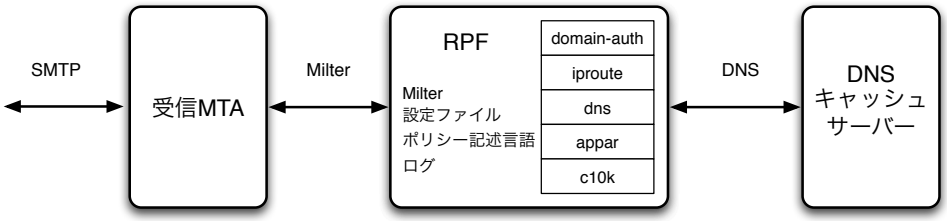

図 1: RPF の構成

と、Haskell の代表的なコンパイラーである GHC には、軽量なユーザースレッドが完備されていることから、スレッド方式を採用することにした。

GHC では、イベントを処理する IO マネージャーというユーザースレッドが、各ユーザースレッドの入出力を処理する。その処理には古典的なシステムコールである select が利用されている。

select では、扱えるファイル/ソケット数に制約がある。また、カーネル空間で判別できている情報をユーザー空間でも再探索しなければならず、扱うファイル/ソケット数が増えると処理が重くなる。ファイル/ソケット数の制約の問題は fork で解決できるが、それではユーザースレッドの機能を活かせない。

そこで、prefork [11] という方法を採用し、ライブラリー `c10k` として部品化した。すなわち、accept システムコールの前に、複数のプロセスを fork することで、1 つの TCP ポートを共有する。その TCP ポートに対するコネクションの振り分けは、カーネルが担当する。文献 [11] で prefork の問題として指摘されている thundering herd は、最近の Linux などでは解決されている。また、プロセスが受け取るソケット数には上限を設けて、上限に達するとプロセスはコネクションを受け付けないようにした。

ファイル/ソケット数の制約の問題を根本的に解決するとともに、処理速度の問題を解決する方法については、第 10 章で触れる。

4.3 ドメイン認証ライブラリー

SPF の宣言には include や redirect という機能があるため、SPF レコードの問い合わせがループすることがある。ループを断ち切る一般的な方法の一つは、なんらかの上限を設けることである。たとえば、MTA では Received: フィールドの数に上限を設けて、ループを検知する。

ドメイン認証ライブラリー `domain-auth` の当初の SPF 実装では、SPF レコードを問い合わせて文法を解析することと、文法に従ってドメインを認証することを同時に実現していた。このコードは SPF レコードの検索数の上限も判定しており、複雑で見通しが悪かった。そこで、遅延評価を利用して、2 つの機能を別々の関数に分離した。

Haskell の型 IO a は、命令ではなく命令書である。SPF レコードを問い合わせて文法を解析する関数では、include や redirect が指示されている場合、「SPF レコードを問い合わせて文法を解析しろ」という命令書を発行する動作を繰り返す。SPF レコードの検索数の上限は考慮しない。

文法に従ってドメインを認証する関数では、SPF レコードの検索数の上限を考慮ながら、命令書を実行し、得た文法からドメインを認証して、結果が出た時点で実行を止める。

2 つの関数は遅延評価で結び付けられており、SPF レコードの無駄な検索は発生しない。このように 1 つの関数が 1 つの役割を果たすので、プログラムの機能を損なわずに、コードの見通しを改善できた。

遅延評価がないか、あるいは入出力が命令書の生成とその実行に分かれていないプログラミング言語では、このような機能分離は極めて困難である。

なお、実装した SPF の認証プログラムでは、SPF レコードの検索数の上限に加えて、IP アドレスのマスク長の制限や “`+all`” を不正として取り扱うかといったことも指示できる。

4.4 DNS 問い合わせライブラリー

ドメイン認証情報を得るために、ユーザースレッドごとに DNS の問い合わせが発生する。このため、DNS の問い合わせライブラリーは、スレッドセーフである必要がある。

Haskell の純粋な関数はスレッドセーフであるが、IO a という型を返す関数は必ずしもスレッドセーフとはならず、他のプログラミング言語と同様に注意を払う必要がある。たとえば、共有するソケットから UDP パケットを送信すると、あるスレッドの問い合わせに対する返答が他のスレッドに戻ってしまうことが考えられる。

5.2 節で述べる理由により、DNS ライブラリー `dns` を独自に実装した。Haskell で、DNS の問い合わせを実装する際、スレッドセーフとならないのは、ソケットを共有した場合のみである。このライブラリーでは、スレッドごとにソケットを開くことで、スレッドセーフを実現した。カーネルは、ソケットごとに固有の始点ポートを割り当てるので、上記の問題は発生しない。

5 導入を容易にするための工夫

5.1 インストールの簡便化

RPF が各 OS のパッケージ管理システムで提供されるようになるまでは、ユーザー自身が RPF をコンパイルする必要がある。実装したすべてのパッケージは HackageDB[6] に登録しており、他の言語のライブラリーは必要としないために、`cabal` コマンド[7] を使えば RPF を自動的にインストールできる。

5.2 依存ライブラリーの除去

RPF の初期のバージョンでは、DNS の問い合わせライブラリーとして、HackageDB に登録されている ADNS を使っていた。このライブラリーは、C 言語で書かれた GNU adns へのバインディングである。`cabal` コマンドでは、GNU adns を自動的にはインストールできないという問題があった。(また、ADNS ライブラリーは GHC のスレッド化ランタイムを要求し、GHC 6.10 のスレッド化ランタイムはデーモンにすると IO マネージャーが止まるという問題もあった。) そこで、ADNS への依存をなくすために、前述のように DNS の問い合わせライブラリー `dns` を実装した。

Haskell 環境のインストールを簡便にする Haskell Platform[8]では、Parsec 2 を提供する。このバージョンには、入力に ByteString[12] が使えない問題がある。最新の Parsec 3 では、この問題が解決されているが、Haskell Platform 環境にインストールすると不整合を起こす。そのため、入力に String や ByteString を許すパーサーコンビネーター `appar` を独自に開発した。ポリシー記述言語以外でパーサーが必要な部分では、このパーサーコンビネーターを利用している。

ポリシー記述言語の実装には、字句解析コンビネーターなど、Parsec は提供しているが、`appar` が提供していない機能が必要である。入力は String であるので、Parsec 2 を利用した。

5.3 2 つの補助モード

RPF を運用に導入しやすくするようにデバッグモードとログオンリーモードを提供している。デバッグモードでは、プログラムがデーモンにならずに端末に残り、ログを端末に出力する。MTA の動作は変更しないので、MTA はメールを必ず受信する。ログオンリーモードでは、デーモンとなり、ログを syslog 経由で記録するのみで、MTA の動作は変更しない。もちろん、通常のモードでもログを syslog 経由で記録する。ログの例を簡略化して図 2 に示す。line の後の数値は、ポリシーを記述したファイルの中で、合致したルールがある行の番号である。

6 テスト

Haskell は、極めて強く型付けされたプログラミング言語である。Haskell の型には、関数の仕様、関数の簡易なドキュメント、コンパイルによる仕様の検証という意味がある。これはコンパイル時の型検査がテストの役割を果たすことを意味しており、Haskell での開発はテストを書かなくとも、多くの機能がテストされることになる。

もちろん、型検査をすり抜けるバグも存在する。そのため、必要だと思われる部分には、HUnit[9] を用いてユニットテストのテストケースを書いた。また、IP アドレスのフィルターを実現する iproute パッケージには、満たすべき性質を記述し、QuickCheck[13] でテストケースを自動生成している。

プログラム全体は、著者のドメインである Mew.org で実際に動かしてテストしており、安定して動作することを確認している。より正確なテストのためには、

[6] http://hackage.haskell.org/
[7] http://www.haskell.org/cabal/
[8] http://hackage.haskell.org/platform/
[9] http://hunit.sourceforge.net/

```
Jul 25 04:03:01 [/NOTICE] 127.0.0.1 Accepted in line 3
Jul 25 04:49:17 [/NOTICE] 202.0.2.1 Rejected in line 44
```

図 2: ログの例

ユーザーがメールの転送先に指定しているドメインなど、さまざまな環境でテストする必要がある。

7 ログ解析

メールとドメイン認証の傾向を見るために、Mew.org で動かしている RPF が記録したログの内、2010 年 7 月 21 日から 27 日までの一週間のログを集計した。外部からの TCP コネクションで成立した数は 486,405、その内 SMTP セッションが成立した数は 10,332 であった。10,332 通のメールに対し合致したルールの内訳を表 3 に示す。行の順番には意味があり、上の行に該当したメールは、下の行に該当することはない。

SPF の成功率が高いのは当然として、SPF のドメイン認証で不正とされ Sender ID のドメイン認証で正当とされたメールが存在することは、注目に値する。また、特定のフィールドを検査することにも意味があることが分かる。DKIM や DomainKeys のみでドメイン認証が正当とされたメールが存在しない理由は、DKIM や DomainKeys 署名のついたメールの送信ドメインすべてが、SPF/SenderID に対応していたためである。また、Mew.org へメールを転送するユーザーは存在しないため、IP 方式のドメイン認証で不正とされ、署名方式のドメイン認証で正当とされることは起こりにくい。

櫻庭氏が、ある会社のメールを本研究とは独立に調査した結果[10]、SPF のドメイン認証で不正とされ Sender ID のドメイン認証で正当とされたメールの多くは、エラーメールとメールマガジンであると分かった。また、SPF レコードが宣言されていないが、署名が付いているメールは存在し、その多くは SPF レコードが宣言されていないドメインから、署名方式のドメイン認証に対応しているドメインで管理されているメーリングリストへ投稿されたメールであった。

著者のフィールドテストでは、ルールに合致しなかったメールを受け取るようにしている。このメールを受け取り拒否にするかは、RPF を導入した管理者の判断にゆだねられる。SPF/SenderID の普及率の高さから考えると、ルールに合致しなかったメールは積極的に拒否してもよいかもしれない。

8 成果物の公開

RPF は、BSD 改変ライセンスの下、オープンソースとして HackageDB に登録しており、誰でも自由に利用できる。マニュアルは RPF のホームページ[11]で公開している。

9 関連ソフトウェア

RPF と似た機能を提供する Milter プログラムに milter-greylist がある。このプログラムは、行指向のポリシー記述言語を提供する。MTA を制御できるタイミングは SMTP RCPT TO と SMTP DATA である。元々グレイリスティングの機能に特化していたが、最近では機能拡張され、コンパイルオプションを指定すれば SPF と DKIM を利用できる。SenderID と DomainKeys には対応していない。

10 今後の課題

select システムコールの問題は、GHC 6.14 に epoll および kqueue API を利用した IO マネージャーが採用される予定であるので、自ずと解決されると思われる [14]。GHC 6.14 のリリース後は、大量の SMTP コネクションに対する負荷検査を実施したい。

機能強化の項目としては、文献 [7] のフィールドを残せるようにすること、`rcpt_to` ブロックと受信者の変数を設けること、DKIM Author Domain Signing Practices (ADSP) に対応すること、ドメイン名のレピュテーションに対応することなどが挙げられる。ユーザーからの要望に応じて、対処していきたい。

[10] 個人的に教えていただいた。

[11] http://www.mew.org/ kazu/proj/rpf/

表 3: 受け取ったメールがどのルールに合致したか

ルール	メールの数	比率
`#spf == pass`	1,015	9.8 %
`#sender_id == pass`	47	0.5 %
`#mail_from == "yahoo.com" && #sig_domainkeys == No`	144	1.4 %
`#dkim == pass`	0	0 %
`#domainkeys == pass`	0	0 %
ルールに合致しない	9,126	88.3 %
合計	10,332	100.0 %

11 おわりに

メールの受信側で、IP 方式のドメイン認証と署名方式の署名認証の欠点を互いに補完する Milter プログラム RPF を実装した。ポリシーは、文脈自由文法の言語を用いて記述できる。実装には純粋関数型言語 Haskell を用い、型検査、ユニットテスト、フィールドテストを通じて安定性を高めた。RPF や関連ライブラリーは、BSD ライセンスの元で公開している。

謝辞

本論文のアイディアを与えて下さった Meng Wong 氏に深謝します。Milter プロトコルについて教えていただいた小林直さん、須藤功平さん、Haskell について教えていただいた木戸崇裕さん、山下伸夫さん、メールの解析をしていただいた櫻庭秀次さん、そして RPF の初期バージョンを検証していただいた橋本祐作さんに感謝します。

神明達哉さん、そして上記の須藤さん、木戸さん、山下さん、櫻庭さん、そして橋本さんは、論文の草稿を読んで貴重な意見を下さいました。心よりお礼申し上げます。

参考文献

[1] R. Siemborski and A. Melnikov, “SMTP Service Extension for Authentication”, RFC4954, 2007 年.

[2] R. Gellens and J. Klensin, “Message Submission for Mail”, RFC4409, 2006 年.

[3] M. Wong and W. Schlitt, “Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1”, RFC4408, 2006 年.

[4] J. Lyon and M. Wong, “Sender ID: Authenticating E-Mail”, RFC4406. 2006 年.

[5] M. Delany, “Domain-Based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys)”, RFC 4870, 2007 年.

[6] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton and M. Thomas, “DomainKeys Identified Mail (DKIM) Signatures”, RFC 4871, 2007 年.

[7] M. Kucherawy, “Message Header Field for Indicating Message Authentication Status”, RFC5451, 2009 年.

[8] J. Lyon. “Purported Responsible Address in E-Mail Messages”, RFC4407 2006 年.

[9] Daan Leijen and Erik Meijer, “Parsec: A practical parser library”, Electronic Notes in Theoretical Computer Science 41 No. 1, 2001 年.

[10] Simon Marlow at el, “Haskell 2010 Language Report”, 2010 年.

[11] W. Richard Stevens, Bill Fenner and Andrew M. Rudoff, “UNIX Network Programming”, 2004 年.

[12] Duncan Coutts, Don Stewart and Roman Leshchinskiy, “Rewriting Haskell Strings”, Practical Aspects of Declarative Languages 8th International Symposium, PADL 2007, 2007 年.

[13] Koen Claessen and John Hughes, “QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs”, In Proc. of International Conference on Functional Programming (ICFP), ACM SIGPLAN, 2000 年.

[14] Bryan O’Sullivan and Johan Tibell, ”Scalable I/O Event Handling for GHC”, in Proceedings of Haskell Symposium 2010, 2010 年. (掲載予定)