URL 正規表現自動生成による悪性通信検知手法に関する一考察

芹田 進† 藤井 康広† 角田 朋‡
吉竹 利織‡ 大鳥 朋哉‡ 木城 武康‡ 寺田 真敏†

†(株)日立製作所 研究開発グループ システムイノベーションセンタ
185-8601 東京都国分寺市東恋ヶ窪一丁目 280 番地
{susumu.serita.wb, yasuhiro.fujii.sj, masato.terada.rd}@hitachi.com

‡(株)日立システムズ
141-8672 品川区大崎 1-2-1 大崎フロントタワー
{tomo.kakuta.sv, michiori.yoshitake.od, tomoya.ohtori.za,
takeyasu.kishiro.uy}@hitachi-systems.com

あらまし マルウェア感染検知のため悪性のWebサイトをブラックリストとして登録する対策が広く使われる。しかし，ドメイン名やIPアドレスをブラックリストに利用する従来の方法では，既知の悪性のWebサイトは検知できるが，Webサイトが変更された場合は検知できない。一方，悪性のWebサイトのURLのパス部分には共通する文字列パタンがあることが知られているが，人手による共通パタンの抽出は経験と時間を要する。本研究ではパス文字列の構造の類似性によりURLをクラスタリングし，その結果に基づいて正規表現を自動生成する手法を提案する。この方法により通常のブラックリストでは検知できないWebサイトを検知可能な正規表現を効率的に生成できると期待される。

Automatic Generation of URL Regular Expression for Detecting Malicious Traffic

Susumu Serita† Yasuhiro Fujii† Tomo Kakuta‡
Yoshitake Michiori‡ Ohtori Tomoya‡ Takeyasu Kishiro‡ Masato Terada†

†Hitachi, Ltd. Research & Development Group
Center for Technology Innovation– Systems Engineering
1-280, Higashi-Koigakubo, Kokubunji-shi, Tokyo, 185-8601, JAPAN
{susumu.serita.wb, yasuhiro.fujii.sj, masato.terada.rd}@hitachi.com

‡Hitachi Systems, Ltd.
1-2-1, Osaki, Shinagawa-ku, Tokyo, 141-8672, JAPAN
{tomo.kakuta.sv, michiori.yoshitake.od, tomoya.ohtori.za,
takeyasu.kishiro.uy}@hitachi-systems.com

Abstract Blacklisting malicious Web sites is widely used to detect malware infections. However, conventional method using domain names and IP addresses does not deal with

moved Web sites by attackers. Although it is known that common string patterns appear in URL's path used by malicious Web site, extracting such a pattern needs high skills and time-consuming task. In this study, we propose automatic generation for regular expressions through URL clustering based on structural similarity in URL's path strings. This method is expected to create regular expressions efficiently to detect malicious Web sites that conventional blacklist cannot detect.

はじめに

サイバー攻撃の手法が巧妙化し，あらゆる攻撃を完全に防ぐことは難しくなってきている。そのため，たとえ攻撃を受けたとしても，攻撃を早期に発見して被害を最小限に抑えることが求められる。マルウェア感染を早期に検知するため，ブラックリストを用いた対策が広く使われる。ブラックリストは，攻撃に利用される悪性サイトの URL やドメイン名などをリスト化したものである。

Web プロキシのフィルタ機能を使うことで，ブラックリストへのアクセスを禁止でき，攻撃の進行を止めることができる。また，ブラックリストをアクセスログ解析に利用することで，マルウェア感染の疑いのある端末を特定できる[3, 4]。しかし，攻撃者が悪性サイトを変更した場合，攻撃の検知が困難になるという問題がある。

一方で，悪性の Web サイトの URL のパスやクエリ部分には共通する文字列パタンがあることが知られている[2]。たとえば，FRRI Data Set 2013 の中に「{数値 3 桁}/{数値 3 桁}.html」という文字列パタンをパスに持つ悪性 URL が複数存在することが報告されている。

悪性サイトの URL に共通する文字列パタンを正規表現で表し，プロキシフィルタやアクセスログ解析に利用することで，通常のブラックリストでは検知できない攻撃を検知できる。しかし，人手による共通パタンの抽出は経験と時間を要する。

それに対し，悪性サイトの URL のサンプル集合から，共通する部分文字列を抽出し，正規表現を自動生成する技術が知られている[1]。しかし，従来技術は，同一ドメインの URL サンプルから正規表現を生成するため，複数のドメインに URL が分散し，同一ドメインの URL サンプルが少ない場合，正規表現生成が困難である。また，サンプルに共通する部分文字列を連結して正規表現化するため，上記の例のように，文字の種類や長さといった構造に関する共通性を抽出することは困難である。

本研究では，URL の構造の類似性により URL をクラスタリングし，その結果に基づいて正規表現を自動生成する手法を提案する。この方法により，通常のブラックリストでは検知できない悪性サイトを検知可能な正規表現を効率的に生成できることが期待される。

本論文は以下のように構成される。まず第 1 章で従来技術を説明し、その問題点を明らかにする。次に第 2 章でパス・クエリを用いた正規表現生成方法を提案する。第 3 章で，提案方式の有効性評価のために行った実験の結果を報告する。最後に第 4 章でまとめと今後の課題を述べる。

1　従来技術と問題点

本章では，従来技術を紹介し，その問題について考察する。

1.1　URL 用語の定義

本論文で用いる URL に関する用語を図 1 に定義する。URL はスキーム，FQDN，パス，クエリから構成される。さらにクエリは，キーとバリューの組から構成される。クエリを持たない URL も存在する。本論文では，このうちパスとクエリ部分（あわせてパス・クエリと呼ぶ）を用いた検知を目的とする。ドメイン，FQDN に基づく検知については[5]が取り組んでいる。

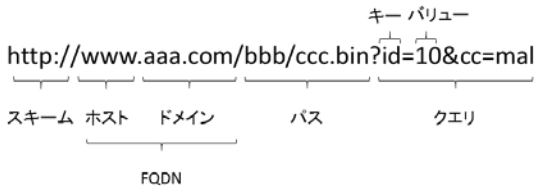

図 1 URL の構成要素の定義

1.2 URL の一部によるブラックリスト

URL をブラックリストに利用する場合，構成要素のどの部分を使うかによって，得られる効果が異なる。

(1)URL 完全一致

FQDN，パス，クエリ含めて完全に一致した場合に悪性通信と判定する。誤検知が少ないが，URL がわずかでも変化すると検知できない。

(2)FQDN，ドメイン

URL のうち，FQDN あるいはドメインをブラックリストに利用する。悪性サイトが属するサーバ単位，あるいはドメイン単位で検知できるため，既知の悪性サイト，悪性ドメインをもれなく検出できる。しかし，攻撃者が悪性サイトを変更した場合は，検知できない。

(3)パス・クエリ

URL のうち，パスとクエリをあわせた部分をブラックリストとして利用する。攻撃に使うサイトが変更されても，パス・クエリ部分が共通している可能性がある。その場合，未知の悪性サイトも検知できる。しかし，パスあるいはクエリの一部が変更された場合検知はできない。

1.3 URL の正規表現によるブラックリスト

1.2 で挙げた方法は，いずれも URL の構成要素をそのままブラックリストとして利用する。そのため，要素が完全に一致する場合は検知できるが，要素がわずかでも変化すると検知できない。

Xie らは，ボットを検知するために，スパムメールに含まれる URL のサンプルから共通するパタンを抽出し，正規表現を自動生成する手法を提案している[1]。以下でその生成方法を説明する。なお，Xie らの手法は，スパムメールから悪性 URL を判定する処理も含むが，本研究の目的である正規表現生成処理のみを示す。

正規表現生成方式

Step1（ドメインによる分割）

悪性 URL の集合を同一のドメインを持つ URL グループに分割する。

Step2（正規表現生成）

まず，URL グループごとに正規表現を生成する。URL グループに属するパス・クエリの集合から頻出する部分文字列を抽出する。そして，頻出キーワードを連結して正規表現を生成する(図 2)。

次に，別のドメインにおいてパス・クエリ部分が同一の正規表現が生成された場合，それらを集約する(図 3)。

図 2 URL グループからの正規表現生成

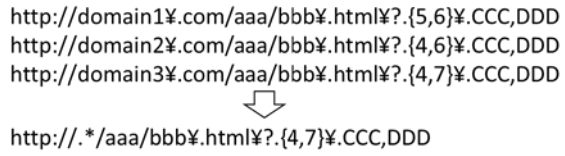

図 3 異なるドメインの正規表現集約

Step3（汎用性評価）

生成された正規表現は，サンプル URL 以外の悪性 URL も検知できるが，汎用性が大きすぎると悪性ではない URL を誤検知してしまう。極端な例では，http://.*はすべての http スキーマを持つ URL を検知対象にしてしまう。

[1]では，正規表現の汎用性をエントロピーの

差分 $d(e)=B(u)-B_e(u)$を用いて評価する。ここで，$B(u)$，$B_e(u)$は正規表現 e にマッチする文字列 u をエンコードするのに必要なビット数の期待値を表す。ただし，$B(u)$は正規表現を使わずにエンコードする場合，$B_e(u)$は正規表現を使ってエンコードする場合を表す。
正規表現の汎用性が大きくなると $B_e(u)$は $B(u)$に近づき，$d(u)$は小さくなる。
$d(u)$に下限値を設け，汎用性が大きすぎる正規表現を除外する。

課題

上記の方法により，悪性 URL のサンプル集合から，正規表現を生成することができるが，下記の課題がある。

1. 正規表現を生成するためには，同一ドメインに属する URL をある一定量取得できることを前提とする。そのため，複数のドメインに分散するURL集合から類似するパス・クエリを発見し，正規表現を生成することは困難である。
2. 頻出する部分文字列を連結して正規表現を生成する。言い換えると部分文字列の一致性を元に正規表現を生成している。そのため，「{数値 3 桁}/{数値 3 桁}.html」のような文字種別とその長さといった構造の類似性に基づく正規表現は生成できない。

2 提案方式

本章では，URL 正規表現の自動生成方式について説明する。

2.1 解決方針

前節で考察した課題を解決するため，本論文では，下記の方針を採用する。

1. サンプル URL をドメインに基づいて分割するのではなく，パス・クエリの類似性に基づいて分割する。類似性の定義は 2.4.1 で詳細を説明する。
2. 部分文字列の一致性に加えて，文字種別やその長さといった構造を考慮した類似度を定義し，サンプル URL を分割する。

2.2 処理フロー

提案方式の処理ステップを以下に示す。

Step1(サンプル URL 読み込み)

正規表現を生成するもととなるサンプルの集合を読み込む。URL の入手ソースとして，マルウェア動的解析結果，あるいはマルウェア情報を提供するウェブサイトなどがある[6,7]。一般に，入手ソースの情報には URL 以外の情報も含まれている。それらの情報を除外し，悪性 URL のみを抽出する。

Step2(パス・クエリの抽象化)

パス・クエリの文字列の一部を文字列の種類で置き換える。この処理を抽象化と呼ぶ。文字列の種類は，たとえば，16 進数，整数などがある。抽象化の方法については 2.3 で詳細を説明する。

Step3(クラスタリング・正規表現化)

抽象化したパス・クエリを文字列と構造の類似度に基づいてクラスタに分割する。同一クラスタに属するパス・クエリから共通する文字列パタンを抽出し，正規表現の形式に出力する。正規表現化の詳細は，2.4 で説明する。

2.3 パス・クエリの抽象化

抽象化の目的は，パス・クエリを構成する要素の構造的な類似性を考慮するためである。抽象化の手法は，[8]が用いている手法を利用する。

図 4 に抽象化の例を示す。文字列の種類を判定し，<文字列種類　文字列長>で置き換える。

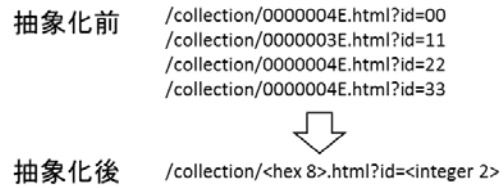

図 4 パス・クエリの抽象化の例

表 1 抽象化対象の文字列種類

文字列種類	例
BASE64	VGhpcyBpcyBhbiBlbmNvZG VkIHN0cmluZw==
URL	http://aaac.com/bbb.html
整数	10
Boolean	True/False
16 進法	09fe12eac

抽象化は，スラッシュ(/)，ピリオド(.)などの記号で分割された文字列，クエリのバリュー単位で行う。表 1 に提案手法で利用した文字列種類を示す。

2.4 クラスタリング・正規表現化

クラスタリング・正規表現化のプロセスを図 5 に示す。

まず，抽象化パス・クエリを 2.4.1 で説明する方法でクラスタリングする(B)。次に，クラスタごとに，2.4.2 の方法でひとつの正規表現を生成する(C)。そして，生成された正規表現から評価指標を算出する(D)。評価指標があらかじめ設定した条件を満たす場合は，生成した正規表現を最終的な出力とする。条件を満たさない場合は，クラスタリングの閾値を変えて再度クラスタリングする。あらかじめ指定した閾値をすべて試した段階で処理が終了する。

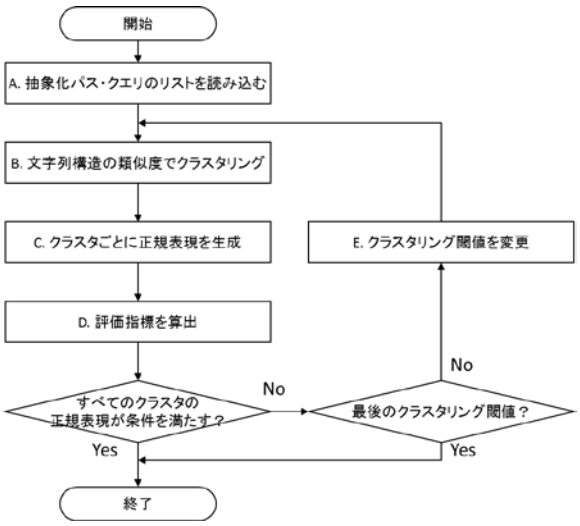

図 5 クラスタリング・正規表現化のフロー

2.4.1 クラスタリング

クラスタリングを行うためには，2つのパス・クエリの間に，類似度を定義する必要がある。本論文では，類似度のかわりに 0 から 1 の値をとる距離を定義する。類似度＝(1 - 距離)により，距離と類似度を変換できる。パスとクエリそれぞれに距離を定義し，その平均値をパス・クエリの距離とする。

a. パス間の距離

パスは通常，ディレクトリ階層を持つ。抽象化パス・クエリのディレクトリは，もとの文字列(例:handlers)か文字列種類を表す抽象化表現(例:<integer 2>)を含む。距離を計算する要素のパタンに応じて下記のように距離を定義する。

パタン 1. 両方とも文字列

文字列間の編集距離を合計文字列長で割った値を利用する。編集距離はある文字列から別の文字列を得るのに必要な編集操作の最小回数で定義され，動的計画法により効率的に計算できる[9, 10]。

パタン 2. 両方と抽象化文字列

文字列種類が異なる場合，距離 1 とする。文字列種類が同じ場合は，文字列長の差を最大文字列数で割った値を距離とする。図 6 の例では，<integer 4>, <integer 5>の距離は，|4-5|/5 = 0.2 と計算される。

パタン 3. 片方のみ抽象化文字列

文字列種別に関わらず，距離 1 とする。

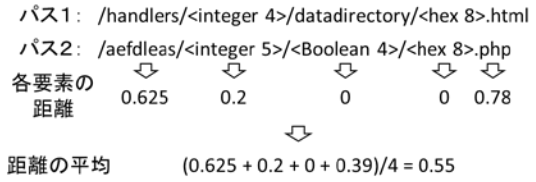

図 6 パス間の距離の例

1つのデイレクト階層に，ピリオド(.)などで区切られた要素が存在する場合は，平均値をディレクトリ間の距離とする。そして，各ディレクトリの平均値をパスの距離として利用する。

b. クエリ間の距離

クエリは複数のキーとバリューの組から構成される。キーの集合に対する Jaccard 距離をクエリの距離として利用する。Jaccard 距離は，二つの集合 A と B に対し，次式で定義される。

$$\frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (1)$$

c. パス・クエリ間の距離

パス・クエリ間の距離は，クエリ部分の有無に応じて下記のようにパタン分けされる。

パタン 1. 両方ともクエリ有り

パス間とクエリ間の距離の平均値を用いる。

パタン 2. 両方ともクエリなし

パス間の距離のみを用いる。

パタン 3. 片方だけクエリあり

距離大として，処理対象から外す。

d. 階層的クラスタリング

クラスタリング手法として，階層的クラスタリングを利用する。図 7 に閾値とクラスタの関係の例を示す。各要素は抽象化パス・クエリであるパス・クエリ間の距離に基づいて各要素を連結していくことで，デンドログラムが生成される。距離に閾値を指定することで，クラスタが決定される。閾値を小さくすると，生成されるクラスタ数は多くなる。

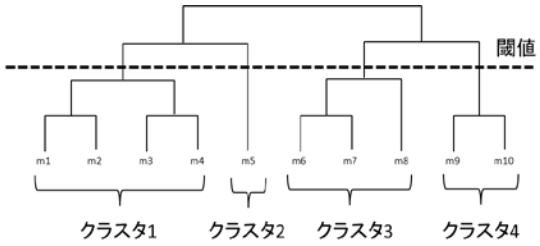

図 7 階層的クラスタリングの例

2.4.2 正規表現化

クラスタに属する抽象化パス・クエリの集合からパス，クエリ部分それぞれの正規表現を下記の方法で生成する。生成したパスとクエリの正規表現を連結し，最終的な正規表現とする。

a. パスの正規表現化

ディレクトリ階層ごとに正規表現を生成する。ディレクトリを構成する要素（トークンと呼ぶ）が文字列か抽象化文字列かで下記のようにパタン分けされる。

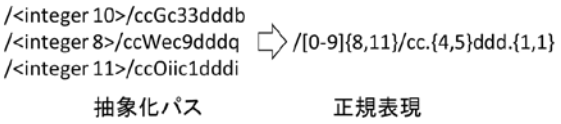

図 8 パスの正規表現化の例

パタン 1. すべてのトークンが抽象化文字列

文字列種類(図 8 の例では整数)に応じた正規表現を生成。出現するトークンの文字列長の最小，最大を算出し，正規表現の文字種別部分と連結する。

パタン 2. 1 つ以上のトークンが文字列

抽象化文字列を含む場合は，抽象化する前の文字列に戻しておく。トークン集合からよく一致する部分文字列を抽出するため，Smith-Waterman アルゴリズムを利用する[11]。図 8 の例では，{cc, ddd}が抽出されている。一致する部分文字列以外の文字列は任意の文字列を表現する正規表現で置き換える。

b. クエリの正規表現化

クエリを構成するキー，バリューの集合のうち，クラスタ内のすべてのサンプルに共通するキーを抽出する。図 9 の例では，{id, cc}が共通するキーである。キーに対応するバリューをパスのトークンと同様に正規表現化する。共通でないキーは任意の文字列を表す正規表現(.*)に変換し，連結する。

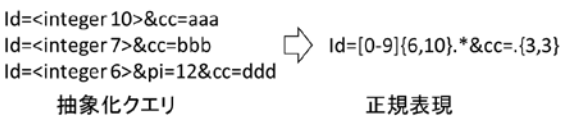

図 9 クエリの正規表現化の例

2.4.3　正規表現の評価指標

生成された正規表現を下記の 2 つの指標に基づいて評価する。

1. FQDN 参照数

生成した正規表現の元になるサンプル URL のうち異なる FQDN 数を表す。この値が大きいほど，正規表現で表される URL の共通パタンがより多くの悪性サイトで利用されていることを意味する。そのため，危険性が高いと考えられる。

2.正規表現の汎用性

1.3 で示した[1]の方法と同様，エントロピーの差分 $d(e)$を用いて正規表現の汎用性を評価する。この値が大きいほど，正規表現は元になるサンプル URL に近いと考えられる。よって，未知の URL が正規表現にマッチした場合，悪性である可能性が高いと考えられる。

3　検証実験

提案手法により有効な正規表現が生成できるかを検証するため，提案方式を実装し，FFRI Dataset 2013, 2014 を使って正規表現生成を行った。生成した正規表現の数および処理時間を表 2 に示す。FQDN 数 2 以上の正規表現の中には，[2]で報告されている「{数値 3 桁}/{数値 3 桁}.htm」の文字列パタンを表す正規表現が生成されていることを確認した。

表 2 正規表現生成数

データセット	2013 年	2014 年
サンプル URL	1,628 件	46,794 件
ユニーク URL	628 件	2,775 件
正規表現 (FQDN 参照数 1 以上)	15 件	110 件
正規表現 (FQDN 参照数 2 以上)	5 件	40 件
処理時間	5 秒	242 秒

4　まとめと今後の課題

本論文では，悪性 URL のサンプルから正規表現を自動生成する従来方式の課題を考察し，その解決策を提案した。

従来の正規表現生成方式は，同一ドメインに属する URL から正規表現を生成するために，複数のドメインに分散するサンプル URL から正規表現を生成することは困難であった。また，部分文字列の一致性を元に正規表現を生成するため，構造の類似性に基づく正規表現は生成できない。

それに対し，ドメインではなくパス・クエリに着目し，部分文字列の一致性および文字種別やその長さといった構造の類似性による分割を行い，正規表現を生成する方式を提案した。FFRI Data Set に対し，提案方式を適用し，実際に正規表現が生成されることを確認した。

今後は，検知率，誤検知率を詳細に評価するとともに，方式の改良を行っていく。

参考文献

[1] Y. Xie, et al., “Spamming Botnets: Signatures and Characteristics”, ACM SIGCOMM, 2008.

[2]畑田 他, “サンドボックス解析結果に基づく URL ブラックリスト生成についての一検討”, IWSEC, 2013.

[3] 榊原 他, “ログ分析によるサイバー攻撃の検知について”, SCIS, 2014.

[4] 角田 他, ” グレーリストを用いたホワイトリスト/ブラックリストの自動生成によるマルウェア感染検知方法の検討”, CSS, 2014.

[5] S. Yadav, et al., "Detecting Algorithmically Generated Malicious Domain Names," in Proceedings of the Internet Measurement Conference (IMC), 2010.

[6] Malware Domain List, 2015, http://www.malwaredomainlist.com

[7] CyberCrime Tracker, 2015, http://cybercrime-tracker.net/
[8]T. Nelms, et al., “ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates”, USENIX Security Symposium, 2013.
[9]E. W. Myers, “An O(ND) difference algorithm and its variations”, Algorithmica 1 ,pp.251-266. 1986.
[10] S. Wu, et al., “An O(NP) sequence comparison algorithm.” Information Processing Letters 35, pp. 317–323, 1990.
[11] T. F. Smith, et al., "Identification of Common Molecular Subsequences", *Journal of Molecular Biology* 147, 195–197, 1981.