

L-010

バイト値列の類似性に基づく亜種ウイルス検出 Detecting Subspecific Virus based on Similarity of Byte-Sequences

上園 智士¹, 小林 和朝², 高田 寛之²

Akihito Kamizono, Kazutomo Kobayashi, Hiroyuki Takada

1 はじめに

インターネットで蔓延するコンピュータウイルスやワームの被害を抑制するためには高度な検出手法が必要となる。これまで類似性に基づいたワームの検出手法として、ワームフローのペイロードに共通な文字列の集合をシグネチャとして用いる手法 [1] や、フローペイロード中のコードの出現確率を用いた検出手法等 [2][3] が提案されている。しかし今回我々は特に亜種ウイルスに注目してその検出手法を提案する。その理由はセキュリティベンダーの米サンベルトソフトウェアは2008年1月24日に2007年に出現したウイルスは過去最高の549万960種類だと発表し、この原因をサンベルトソフトウェアではオリジナルのウイルスをわずかに変えた亜種(変種)が多数作られるようになっているためと分析した [5]。549万960種の中にはこの亜種ウイルスが多数を占めていると言われていて、現代亜種ウイルスの存在がとても脅威であるからである。本研究では、亜種ウイルスは元々あるウイルスを改変して作るウイルスなので、同じシグニチャをもつことが予測され、2つのウイルスを比較し、共通のシグニチャを見つけ出し、それがウイルス全体のどれくらいの割合を占めるかを、そのウイルス同士の類似度とし亜種を検出する。検出を行う際に時間短縮のためにハッシュマップを用いて検出を行う。

2 バイト値列の類似性を用いた検出手法

ウイルスには特有のシグニチャをもつと可能性が高いという観点から、バイト値列の類似性に基づく亜種のウイルス検出について提唱する。今回の手法では2つのプログラムが全体のうちのどの程度共通するシグニチャを含んでいるかをもとに類似度を計り亜種ウイルスを検出する。ウイルスが特有のシグニチャをもつのであれば、そのシグニチャはある程度の長さをもつはずである。よってファイル同士の共通のシグニチャを見つけ出すためにリファレンスとなるファイルを150バイトごとにマッピングしてハッシュマップをつくる。ここでは簡単のためにファイル α をリファレンスとし、3バイトごとにマッピングした例を以下に示す。

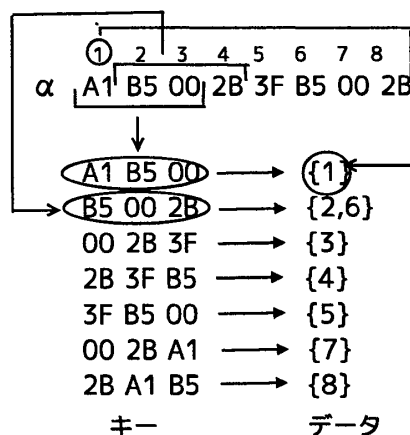


図 1: α を 3 バイトごとにマッピングしたハッシュマップ

¹長崎大学大学院生産科学研究科 電気情報工学専攻

²長崎大学工学部情報システム工学科

まず図1のようなハッシュマップを作る。ハッシュマップはプログラム α の1バイト目から3バイトずつをキーに渡してデータの部分にはそのキーが α の何バイト目にあたるかを格納する。3バイトとも同じキーの場合はデータを後ろに追加する。実際のプログラムでは3バイトではなく150バイトごとにマッピングしていく。

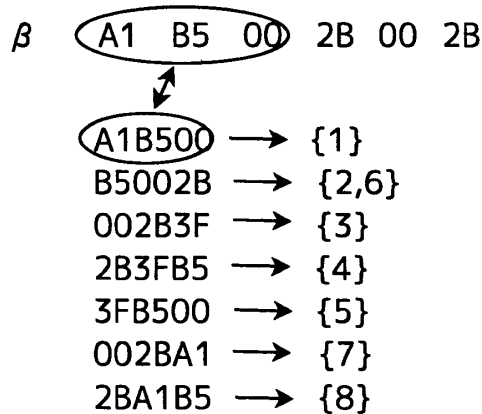


図2:検査したいファイル β をハッシュマップにかけキーを取得する

次に検査したいファイル β を3バイトごとにハッシュにかけてデータを取得する。図2ではA1B500からデータ1を取得できる。

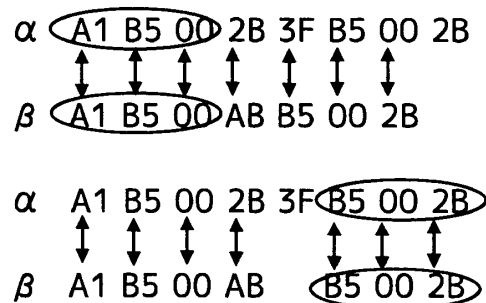


図3:ハッシュマップからデータを取得し2つのファイルのバイト値列を比較

データを取得すれば α と β の3バイトごとの一致箇所を特定できるので図3のようにその位置からバイトを比較していく。なお、一致箇所を特定し α と β を比較していく際に初めて α と β で異なったバイト値がでてくる場所で比較を止め、その位置を記憶しておく。詳しくは図4に示す。

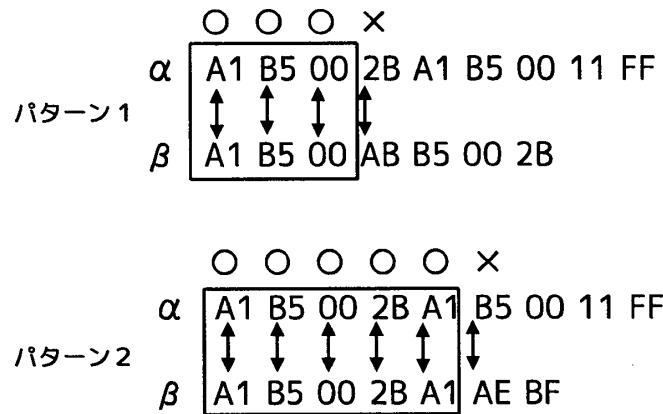


図4:最初に異なったバイト値がでてくる位置の特定

図4のパターン1の場合、ハッシュマップからA1B500の一致を見つけだし1バイトずつ比較していった際に4バイト目で初めて異なったバイト値がでてくる。同様にパターン2では6バイト目に初めて異なったバイト値がでてくる。これより図5の四角で囲んだ部分は一致していると断定できるので、次に β をハッシュマップにかける際は一致部分の次のバイトからでよい。つまりパターン1では4バイト目、パターン2では6バイト目からの3バイトをハッシュにかけていき、これを繰り返す。実際のプログラムでは3バイトではなく150バイトごとにマッピングしていく。但し、マッピングしていく際に150バイトの内135バイト以上が00の場合はマッピングしない。これはバイナリコードで書かれたファイルは00を非常に多く含んでいるため、誤検出と実行時間の削減のためである。

本論文では2つのプログラムのバイト値列の類似性指標として、次の δ を考える。

$$\delta = \sum_{i=1}^n (g_i \div d_1 \times 100) \quad (1)$$

n は β をハッシュマップにかけたときにハッシュマップからデータを取ってこれた回数(つまり150バイトごとの一致を見つけることができた回数)、 g_i はそのときの始めの150バイトを含めバイト値列が連続して完全に一致している数(図4の○の部分)である。 d_1 は2つのプログラムでファイルサイズが小さい方のファイルのサイズとする。 δ はファイルサイズの小さい方のファイルに対し、150バイト以上のバイト値が連続して一致している部分が、全体のどの程度の割合を占めているかを表したものである。

3 提案手法の評価

今回の実験にはインターネット上のサイトやP2PソフトWinnyから取得したウィルスを使用する。またFalse Positiveを検査する際にはLinuxのbinやusrbinの中にあるバイナリファイル(225個)やWindowsの実行ファイル(322個)を使用した。今回既知のウィルス(89個)と検出対象を比較し、その δ が閾値 t を越えた場合に検出対象を亜種ウィルスであると判定した。なお閾値は文献[4]で亜種だと保証されているものを検出できるぎりぎりの値0.72%とした。

表 1: 評価

Virus	TPR(%)	FPR(%)	WPN(個数)	LPN(個数)	NSVPN(個数)
Backdoor.Agobot.gen	100	0	0	0	9
Backdoor.Agobot.ok	100	0.18	1	0	0
Backdoor.Agobot.pp	100	0	0	0	0
Backdoor.Agobot.sy	100	0.18	1	0	1
Backdoor.Gobot.p	100	0.36	2	0	8
WORM_ANTINY.F	100	0	0	0	0
Win32.Pinom.c	100	0	0	0	25
Worm.Win32.Dedler.g	100	0.18	1	0	0

実験から表1のような結果が得られた。TPRは亜種ウィルスの検出率で、FPRはWPNとLPNの個数をたして全体のファイル数(547個)で割った誤検出率、WPNはWindowsの実行ファイルにおける誤検出を生んだファイルの個数、LPNはLinuxの実行ファイルにおける誤検出を生んだファイルの個数、NSVPNは亜種とは保証されていないウィルスを亜種だと検出したファイルの個数である。今回閾値は亜種だと保証されているものは検出できるぎりぎりの値を使用したのでTPRは100%となる。今回誤検出にひっかかったファイルは全てWindowsのファイルで、システム情報を知るためのプログラムやディレクトリを作るプログラムであった。NSVPNが示すように亜種だと保証されていないウィルスに関しても検出に多くひっかかることがわかった。

4 まとめ

Windowsの実行ファイルでひっかかったファイルはシステム情報を知るためのプログラムやディレクトリを作るプログラムであったため、ウィルスが感染活動を行う際にこれらのプログラムと似た動きをするのではないかと推測できる。亜種だと保証されていないウィルスも検出にひっかかるウィルスがあったことは、ウィルスが亜種でなくても共通のシグニチャを持つことを示し、この検出手法は亜種ウィルス検出に有効であるだけでなく、新種のウィルスも検出できる可能性があることを示唆している。

参考文献

- [1] S. Singth, C. Estan, G. Varghese and S. Savage, "Automated worm fingerprinting", In Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI), Dec, 2004
- [2] 鈴木洋平, 和泉勇治, 角田裕, 根元義章: パケットペイロードの類似性に基づくワーム検出に関する一検討, 電子情報通信学会通信ソサイエティ大会, B-7-25, 2007.
- [3] 鈴木洋平, 和泉勇治, 角田裕, 根元義章: フローペイロードのクラスタリングを用いた低負荷なワーム検出方式, CS2006-33, pp.67-72, 2006
- [4] トレンドマイクロ: ウィルスデータベース,
<http://www.trendmicro.co.jp/vinfo/virusencyclo/default.asp>
- [5] ITpro: セキュリティニュース, <http://itpro.nikkeibp.co.jp/article/NEWS/20080125/291973/>