Hot list strategyによる polymorphic viral binary code解析の高速化

安藤類央

独立行政法人　情報通信研究機構 情報通信セキュリティ研究センター
〒 184-8795　東京都小金井市貫井北町 4-2-1

あらまし　不正コード作成技術は洗練化しており、ソフトウェア暗号化や難読化が検出解析回避技術として適用されるようになった。本論文では ATP (automated theorem proving) の手法を用いて polymorphic viral binary code を解析し、検出技術の再現性や効率性を検討するための方法論を提案する。提案システムでは、等価代入 (equality substitution) などの手法を用いてバイナリコードから復号ルーチンの構造とパラメータを検出する。そして、look-ahead 型の計算戦略である hot list strategy を用いて、検出プロセスを高速化する方法を示した。評価実験では、レジスタの種類に応じた hot list を生成し、計算系のレジスタ (EAX,ECX,EBX,EDX) に焦点を当てるとより解析が高速化する結果を得た。

Faster parameter detection of polymorphic viral binary code using hot list strategy

Ruo Ando

National Institute of Information and Communication Technology, Tracable Network Group
4-2-1 Nukui-Kitamachi, Koganei,
Tokyo 184-8795 Japan
ruo@nict.go.jp

Abstract　Malicious mobile code has become more sophisticated. Software encryption and obfuscation are applied for evading signature maching based scanning. In this paper we propose a ATP (automated theorem proving) based analysis of polymorphic viral binary code. Structure and parameter are detected by theorem prover. In detection process, we apply a look-ahead heuristics called hot list strategy for faster equality substitution. In experiment, we discuss the effectiveness of this strategy by numerical output of theorem prover. It is shown that hot list group (EAX, ECX, EBX, EDX) reduces the number of generated clauses compared with hot list group (EDI, ESI, EBP, EDP).

1　はじめに

不正コード作成技術は洗練化しており、近年、ソフトウェア暗号化と難読化が、検出と解析を回避するために利用されている。解析と検出には、デバッガやメモリエディタを用いて、ソフトウェア内部に検査ポイントを設置し、ＡＰＩや復号関数のパラメータの検出を行うが、対象によって適切な手法は異なる。そのため、解析者によって、対象固有の手法（ノウハウ）がその場で編み出されるが、これらの手法に対し、再現性を検討し、分類や論拠を提供する方法論は確立されていない。本論文では、自動定理証明（Automated Theorem Proving）の技術を用いて、難読化された復号ルーチンを持つバイナリコード解析のプロセスを定式化する。また、Look-ahead 型の計算戦略の一つである Hot list strategy を用いて、解析にかかる計算コストを削減する方法を提案する。

2 Polymorphic viral code

シグニチャのマッチングを回避する不正コードは、polymorphic(多形）と metamorphic(変成）の２種類で呼ばれる。自らを暗号化するコードを polymorphic、難読化するコードを metamorphic（変成）を分類されるが、暗号化されたコードは、通常復号ルーチンも難読化されるため、polymorhic という用語は、metamorphic も含めて使われることが多い。感染動作は、複数の関数やＡＰＩで構成され、不正コードを作成する側は、これらを難読化して検出回避を試みる。表１と２は、Win32.Metaphor というウィルスにおいて、GetModuleHandleA という他のプロセスのハンドラを取得するＡＰＩ呼び出しが難読化された例である。次に、ポリモーフィックウィルスがもつ復号ルーチンのアセンブラコードの例を示した。復号関数を構成するパラメータは、ペイロード転送、復号鍵、分岐カウンタ、ループの始点アドレスの４つである。

```
set A address_of_payload
set B key
set C address_loop_start
set D counter

address_loop_start
   payload_transfer(A)
   decryptor(B)
   parload_transfer(A)
   branch(D)
   goto_start(C)
```

3 提案手法

3.1 構造の検出

図１に、提案手法を示した。提案システムでは、まず最初に、バイナリコードから、オペランドとオプコードを検出する。抽出したオペコードを、データ転送とその他の命令に分類する。一方で、オプランドをレジスタの組に変換しておき、データ転送命令の情報と組み合わせてパラメータ設定の検出を行う。この際に、ペイロード転送の命令群の抽出も可能である。その他のオプコードと、レジスタの情報から、分岐、ループ命令の検出を行う。本手法は、並列処理に適したものとなっている。

3.2 パラメータの検出

パラメータの検出には、等価代入（equality substitution）のの手法を用いる。等価代入には、デモジュレーションとパラモジュレーションの２つがあり、提案システムではこれらを併用する。

```
fact: f(g(x),x).
fact: equal(g(a),b).
conclusion f(b,a).

fact: equal(data_16e,514Bh).
fact: mov(reg(ah),const(data_16e),63,
time(1)).
conclusion :
mov(reg(ah),const(514Bh),63,time(1)).
```

上は、デモジュレーションの適用例を示したものである。equal 節はデモジュレータと呼ばれ、変数への定数の値の代入を処理する。

```
fact: mov(reg(ah),const(2Ch),162,time(1)).
fact: mov(reg(bx),reg(ah),300,time(1)).
fact: xor(reg(dx),reg(bx),431,time(1)).
/* decrypter */

-mov(reg(x),const(y),z,time(1)) |
x=const(y,z).
conclusion:
decrypter(reg(dx),key(const(2Ch,162),
431,time(1)).
```

上は、パラモジュレーションの適用例を示したものである。これらはデモジュレーションでは処理できない。デモジュレーションは効率性、パラモジュレーションは完全性を志向したものであり、提案システムではパラモジュレーションをパラメータ検出に不可欠であるが、同手法は通常は、制御が難しい場合が多い。

4 Hot list strategy

前節で述べたように、解析は構造とパラメータの検出の２つに分けられる。両者のうち、計算コストの大半を占めるのがパラメータの検出であり、この高速化が、解析器全体の性能を大きく左右する。本論文では、Hot list strategy と呼ばれる計算戦略を

1	mov dword_3, 6E72654Bh
2	mov dword_4, 32336C65h
3	mov dword_5, 0h
4	push offset dword_3
5	call ds:GetModuleHandleA

表 1: Assembly code of GetModuleHandleA API.

3	mov dword_1,0h
3	mov cdx,dword_1
3	mov dword_2,edx
3	mov edp,dword_2
2	mov edi,32336C65h
2	lea eax,[edi]
1	mov esi,0A624540h
1	or esi,4670214Bh
2	lea edi,[eax]
2	mov dword_4,cid
3	mov edx,ebp
3	mov dword_5,edx
1	mov dwrod_3,esi
4	mov edx,offset dword_3
4	push edx
5	mov dword_6,offset GetModuleHandleA
5	push dword_6
5	pop dwprd_7
5	mov edx,dword_7
5	call dword ptr ds:0[edx]

表 2: Obfuscated assembly code of GetModuleHandleA API

用いて、パラメータ検出の高速化を行う。Hot list strategy は、Wos によって提案されたもので、等価代入、特に paramodulation による推論の効率化に用いられるものである。

4.1 Moufang identity problem

Look-ahead 型の計算戦略である hot list strategy の有効性が示された問題に、Moufang identity problem がある。その内の 1 つに、

```
Moufang1: (x*y)*(z*x)=(x*(y*z))*x
Moufang2: (x*y)*z)*y=x*(y*(z*y))
Moufang3: x*(y*(x*z))=((x*y)*x)*z)
```

Moufang1 から Moufang3 を導出するのに、hot list strategy が有効であることが示されている。直観的にいうと、paramoudlation の探索の深さについて、right(left) solvable $x * rs(x, y) = y$ などの原則的な節の適用を優先させる方法であると想定される。

4.2 ホットリストの生成

提案システムでは、データ転送を追跡し、パラメータの検出を行う際に、どのレジスタに焦点を当てるかによって、計算コストが変化する。ここで、定理証明器にどのレジスタに焦点を当てさせるかを設定するために、Hot list を作成する。

```
# hot list group I :
calculation registers
list(hot).
ax=const(x,y). bx=const(x,y).
cx=const(x,y). dx=const(x,y).
end_of_list.

# hot list group II :
memory registers
list(hot).
di=const(x,y). si=const(x,y).
bi=const(x,y). bp=const(x,y).
end_of_list.
```

評価実験では、8 つのレジスタそれぞれについて hot list を作成し、計算レジスタと、メモリ操作レジスタの 2 つのグループに合わせて 2 つの hot list を作成した。

5 評価実験

評価実験では、2 節で述べた構成の polymorphic viral binary code を、生成し、構造とパラメータの

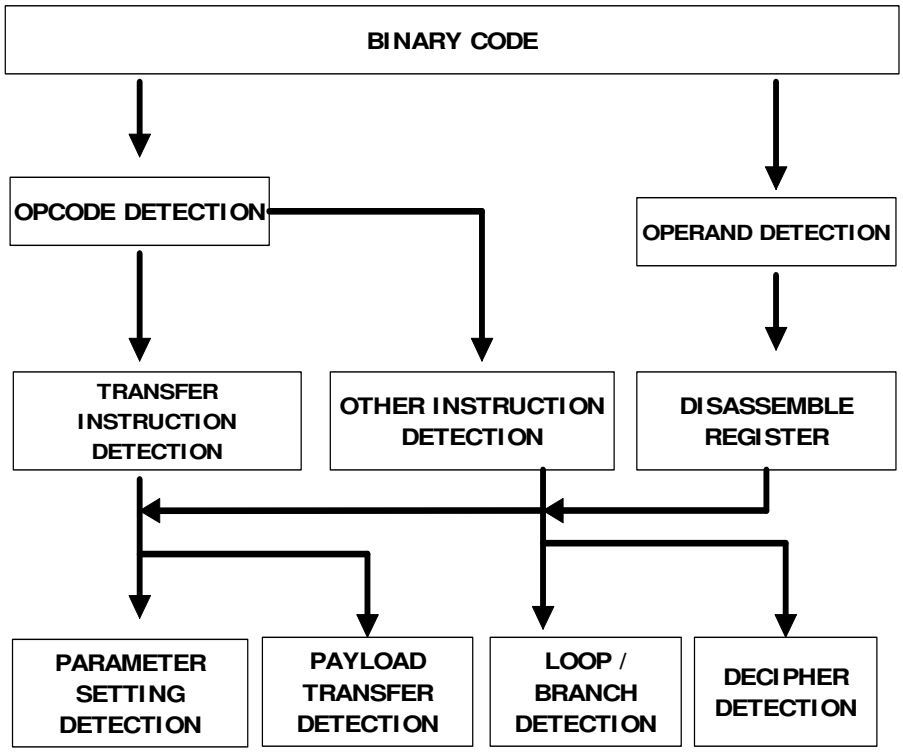

図 1: 提案解析システム

検出を行い、計算コストを測定した。生成したコードの種類は以下の４つである。

- タイプ１：データ転送に mov 命令を用いる。
- タイプ２:復号に間接アドレッシングを用いる。
- タイプ３：ペイロードの転送に xchg 命令を用いる。
- タイプ４：データ転送にスタック操作命令を用いる。

表３から６は、この４種類のコードの解析に、前節で述べた１０種類の hot list を適用した結果を示したものである。ポリモーフィックコード生成には乱数を用いる。このことから想定されるように、８種類の単一のレジスタについて生成した hot list の効果は、生成されたコードによって異なる。計算系のレジスタ (EAX,ECX,EBX,EDX) とメモリ操作系のレジスタ (ESI,EDI,EBP,EBI) の２つにグループによって作成した hot lists については、４種類のコードすべてにおいて、計算系のレジスタに焦点を当てた方が、より解析が高速化する結果を得た。

Type A (no weighting)		Type A (with weighting)	
HOT LIST	all clauses	HOT LIST	all clauses
no heat	915	no heat	707
EAX	677	EAX	677
EBX	670	EBX	602
ECX	799	ECX	541
EDX	756	EDX	540
EDI	1078	EDI	822
ESI	1055	ESI	801
EBI	1055	EBI	801
EBP	1055	EBP	801
Group I	468	Group I	366
Group II	1510	Group II	1206

表 3: Hot list strategies for Type A. Paramodulation for detecting parameters into register E* is speeded up by hot list. We set 10 hot lists for each register and two groups.

6 まとめと今後の課題

不正コード作成技術は洗練化しており、ソフトウェア暗号化と難読化が、検出解析の回避技術として適用されている。これらの手法に対して、再現性を与え、分類、効果の論拠を与える方法論は確立されていない。本論文では、自動定理証明（Automated

Type B (no weighting)		Type B (with weighting)	
HOT LIST	all clauses	HOT LIST	all clauses
no heat	1592	no heat	769
EAX	915	EAX	605
EBX	1561	EBX	494
ECX	497	ECX	490
EDX	519	EDX	593
EDI	1921	EDI	1164
ESI	1724	ESI	843
EBI	1724	EBI	685
EBP	1724	EBP	685
Group I	463	Group I	242
Group II	2422	Group II	1807

表 4: Hot list strategies for Type B.

Type C (no weighting)		Type C (with weighting)	
HOT LIST	all clauses	HOT LIST	all clauses
no heat	976	no heat	604
EAX	1018	EAX	605
EBX	720	EBX	494
ECX	946	ECX	490
EDX	976	EDX	593
EDI	1592	EDI	1164
ESI	1272	ESI	843
EBI	1114	EBI	685
EBP	1114	EBP	685
Group I	738	Group I	463
Group II	2284	Group II	1807

表 5: Hot list strategies for Type C.

Theorem Proving）の技術を用いて、難読化された復号ルーチンを持つバイナリコード解析のプロセスの定式化を行った。提案手法では、等価代入などの定理証明の手法を用いてバイナリコードから復号ルーチンの構造とパラメータを検出する。そして、検出プロセスにおいて、Look-ahead strategy のひとつである Hot list strategy を適用し、計算コストを削減する方法を提案した。評価実験では、レジスタに応じて１０の hot list を生成し、検出の過程で生成される節数を示した。これにより、生成されたコードに関しては、パラメータの検出についてメモリ操作系のレジスタより計算系のレジスタに焦点を当てると検出が高速化することが明らかになった。

Type D (no weighting)		Type D (with weighting)	
HOT LIST	all clauses	HOT LIST	all clauses
no heat	1877	no heat	801
EAX	1444	EAX	587
EBX	1675	EBX	587
ECX	870	ECX	599
EDX	1877	EDX	737
EDI	7406	EDI	1462
ESI	2028	ESI	876
EBI	2028	EBI	876
EBP	2028	EBP	876
Group I	563	Group I	259
Group II	8186	Group II	1891

表 6: Hot list strategies for Type D.

参考文献

[1] Computer viruses: from theory to applications. IRIS International series, Springer Verlag, ISBN 2-287-23939-1,2005.

[2] Diomidis Spinellis, "Reliable identification of bounded-length viruses is NP-complete", IEEE Transactions on Information Theory, 2000.

[3] Peter Szor and Peter Ferrie," Hunting for Metamorphic", Virus Bulletin Conference, 2001.

[4] Stephen Pearce, "Viral Polymorphism", paper submitted for GSEC version 1.4b, 2003.

[5] Michalis Polychronakis, Kostas G. Anagnostakis and Evangelos P. Markatos, "Network level polymorphic shellcode detection using emulation", Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), 2006.

[6] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, Randal E. Bryant, "Semantics-Aware Malware Detection", IEEE Security and Privacy, 2005.

[7] Mihai Christodorescu and Somesh Jha, "Static Analysis of Executables to Detect Malicious Patterns", USENIX Security Symposium, 2003.

[8] Hao Chen, Drew Dean, and David Wagner, "Model checking one million lines of C code", Annual Network and Distributed System Security Symposium (NDSS), 2004.

[9] O.Sheyner, J.Haines, S.Jha, R.Lippmann, and J. M. Wing, "Automated Generation and Anal-

ysis of Attack Graphs", IEEE Symposium on Security and Privacy , 2002.

[10] Arun Lakhotia, Moinuddin Mohammed, "Imposing Order on Program Statements to Assist Anti-Virus Scanners", Working Conference on Reverse Engineering, 2004.

[11] Matias Madou, Bertrand Anckaert, Patrick Moseley, Saumya Debray, Bjorn De Sutter, Koen De Bosschere, "Software Protection through Dynamic Code Mutation", Workshop on Information Security Applications, 2005.

[12] Larry Wos, George A. Robinson, Daniel F. Carson, Leon Shalla, "The Concept of Demodulation in Theorem Proving", Journal of Automated Reasoning, 1967

[13] W. McCune, "33 basic test problems: A practical evaluation of some paramodulation strategies", Preprint ANL/MCS-P618-1096, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1996

[14] Larry Wos, Gail W. Pieper, "The Hot List Strategy", Journal of Automated Reasoning, 1999

[15] Simulated Metamorphic Encryption Generator available at http://vx.netlux.org/vx.php?id=es06

[16] IDA Pro disassembler avaibale at http://www.datarescue.com/

[17] OTTER automated deduction system available at http://www.mcs.anl.gov/AR/otter/

[18] Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 2A: Insruction Set Reference A-M,2004.

[19] Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 2B: Insruction Set Reference N-Z,2004.