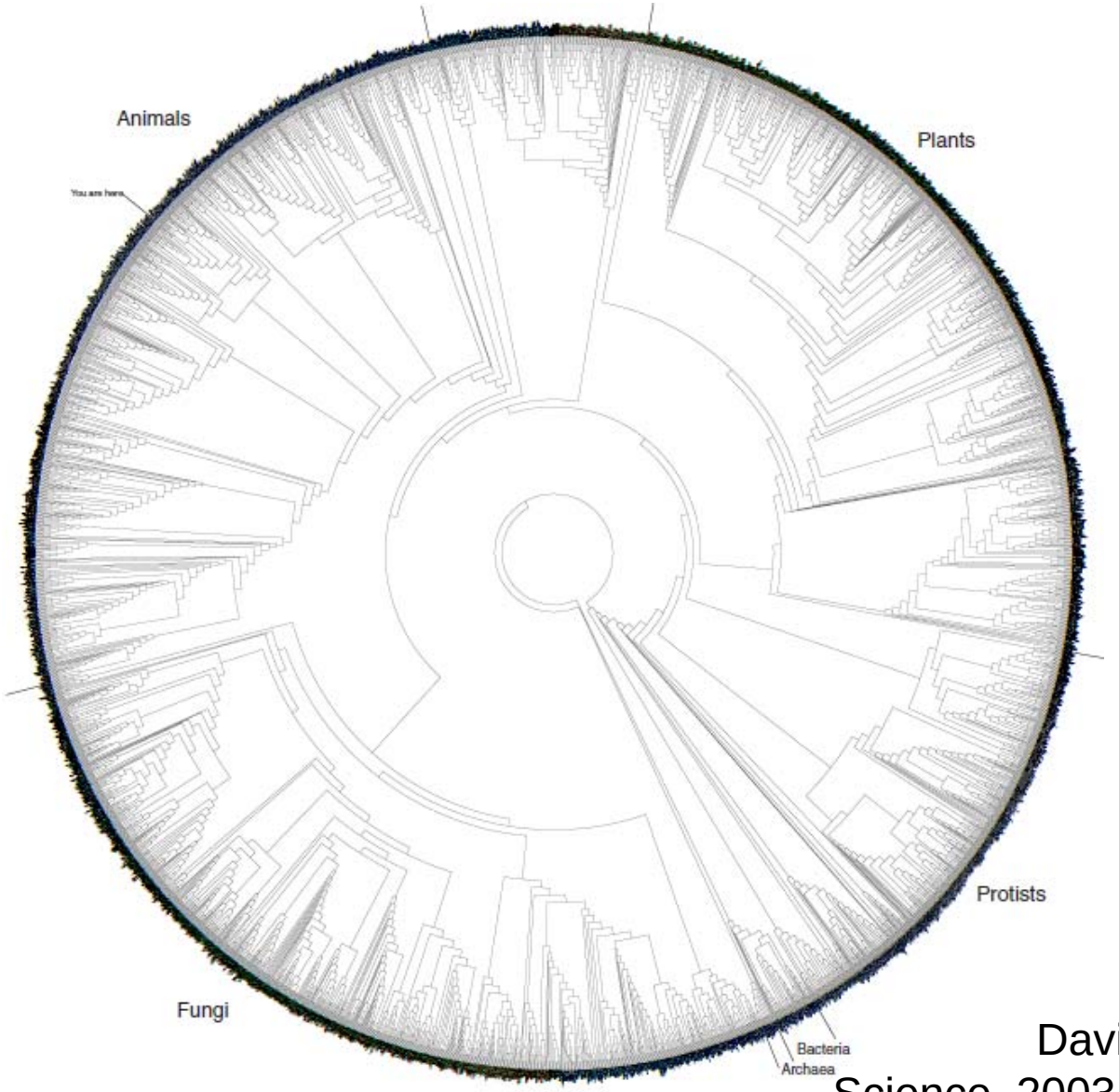
NTT
NTT Information Sharing Platform Laboratories
NTT 情報流通プラットフォーム研究所

機械語命令列の類似性に基づく自動マルウェア分類システム

2009年10月28日

岩村　誠[*1*2], 伊藤　光恭[*1], 村岡　洋一[*2]
*1 NTT情報流通プラットフォーム研究所
*2 早稲田大学

Tree of Life



David M. Hillis, et al.
Science, 2003, 300:1692-1697

発表の流れ

- 研究の背景，動機
- 自動マルウェア分類システムの全体像
- プログラムコードに基づくマルウェアの分類
- LCSと，ビットベクトル化と，縮約命令と，
- 分類結果
  - CCC DATAset 2009
  - 我々のハニーポットで収集した最近のマルウェア
  - 京都大学で収集されたマルウェア
- まとめ

研究の背景，動機

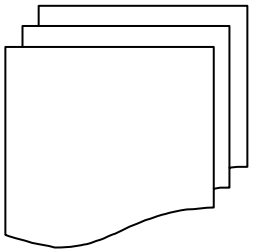
- 近年，マルウェアの種類数が急増している
  - 何が流行っているのか，何が廃れたのか，も見えにくい．
  - アンチウィルスソフトの命名もあやふやに？？
    - 命名することより，検知・駆除することが重要なんでしょうけど．
- マルウェアの全容を解明し（続け）たい！
  - まずは，収集したマルウェアの傾向（流行り廃り）を掴む．
  - できれば，
    【全てのマルウェアを静的解析済み】と同じ状態にしたい．
- マルウェアの分類技術が有効

マルウェアの分類技術

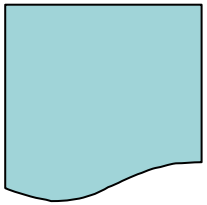
- やり方は大雑把に二通り.
  - –挙動に基づく分類
    通信ログとか，システムコールの呼び出し履歴とか，
    【長所】監視の仕組みさえ構築すれば，後はマルウェアを動かすだけ.
    【短所】動作しない箇所の特徴は出てこない.（時限式だったり，コマンド待ちだったり）
  - –プログラムコードに基づく分類
    【長所】潜在する機能も踏まえて分類可能
    【短所】アンパックや逆アセンブルが必要．ものによっては，コールツリーやIATを再構築したり.

マルウェア自動分類の戦略

マルウェア



オリジナルコードを含む
メモリダンプ



逆アセンブル結果

動的生成コード抽出用
カーネルドライバ

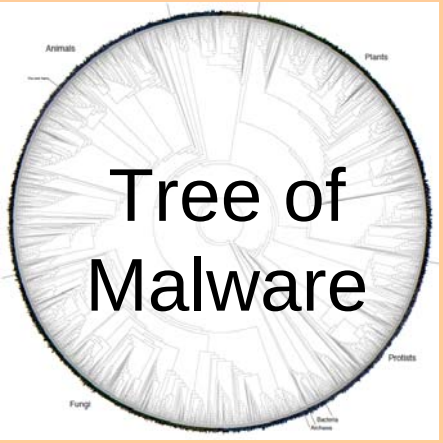
HMMに基づく
確率的逆アセンブル手法

もごもご

類似度行列

	A	B	C
A	-	-	-
B	0.9	-	-
C	0.5	0.7	-

階層的
クラスタリング



従来研究：プログラムコードに基づくマルウェアの分類

- プログラムコードの何に着目するかによって色々
  1. Nグラム，Nパーム
     連続するN個の命令（正確にはオペコード）に着目し，各々何回出現したかをマルウェアの特徴ベクトルとして分類．速い．
  2. コール・ツリー
     コール・ツリーの類似度を算出する．木構造の類似度を算出するのは大変なので，葉となるWin32 API呼び出し部分の情報を使って，計算量を減らす．

従来研究:どの方法がいいの？

- 速い
  - –速いに越したことはない.
- 適切に分類可能
  - –Nグラムが似ている→ふーん.
  - –コール・ツリーが似ている→あぁ，そー.

適切って何だ？

分類の目的によって良し悪しは変わる.

- 全体の傾向を早く掴みたい.
- コンパイラによる差異を吸収したい.

目的に立ち返る

- 「できれば，【全てのマルウェアを静的解析済み】と同じ状態にしたい．」
  - 2つのマルウェアのコール・ツリーが似ている
    → 片方だけ解析すればよいの？
      - コール・ツリーは似ているけど，関数の中身は違う場合．
      - 結局，両方のアセンブリを読まなきゃいけない・・・orz

- 命令単位で類似度を求めたい！
  - 命令列がほぼ一緒
    → 片方だけ解析すれば，両方解析したようなもの．

LCSについて

- Longest Common Subsequence(**最長一致部分列**)**の略**

- {A,B,C}**と**{A,D,C}**の**LCS**は**{A,C}
  - **連続している必要は無く，間に挿入・削除が入ってよい．**
  - **要するに**diff**の逆．**

動的計画法を使ったLCSの求め方

- {a,b,c,d}と{b,a,c,d}のLCSを求める場合

	ϕ	a	b	c	d
ϕ	0	0	0	0	0
b	0	0	1	1	1
a	0	1	1	1	1
c	0	1	1	2	2
d	0	1	1	2	3

LCSの長さ

（動的計画法）
小さな問題から大きな問題を解いていく方法.

$S = s_1^m, T = t_1^n$のLCS長を$L(s_1^m, t_1^n)$とすると次式が成り立つ.

$$L(s_1^i, t_1^j) = \begin{cases} i = 0 \ or \ j = 0 \rightarrow 0 \\ s_i = t_j \rightarrow L(s_1^{i-1}, t_1^{j-1}) + 1 \\ s_i \neq t_j \rightarrow \max\left\{L(s_1^{i-1}, t_1^j), L(s_1^i, t_1^{j-1})\right\} \end{cases}$$

時間はどれくらいかかる？

- 計算量，メモリ使用量
  - 2つのマルウェアの命令数をそれぞれM,N（M<N）とすると，
    - 動的計画法を使えば，計算量はO(MN)
    - Hirschbergらの方法を使えば，メモリ使用量はO(M)

- 1マルウェアあたり多いと10万命令くらい．
  - 素直にdiffをとろうとすると，1ペアで最大5分くらいかかる．（Intel Core2Quad 2.6GHz，1コア）．平均でも1分くらい．
  - 3000検体の分類には

$${}_{3000}C_2 \times 1\text{分} \fallingdotseq 8.5\text{年}！？$$

分類結果は
MWS2017で！？

NTT

世の中には賢い人がいるもので

ビットベクトル化

- 単純なDPベースのLCS算出アルゴリズムを演算ワード長倍高速化する方法.

	a	b	c	d
b	0	1	1	1
a	1	1	1	1
c	1	1	2	2
d	1	1	2	3

L

	a	b	c	d
b	1	0	0	0
a	0	1	1	1
c	1	1	0	0
d	1	1	1	0

V

	a	b	c	d
b	0	1	0	0
a	1	0	0	0
c	0	0	1	0
d	0	0	0	1

M

	a	b	c	d
b	1	0	1	1
a	0	1	1	1
c	1	1	0	1
d	1	1	1	0

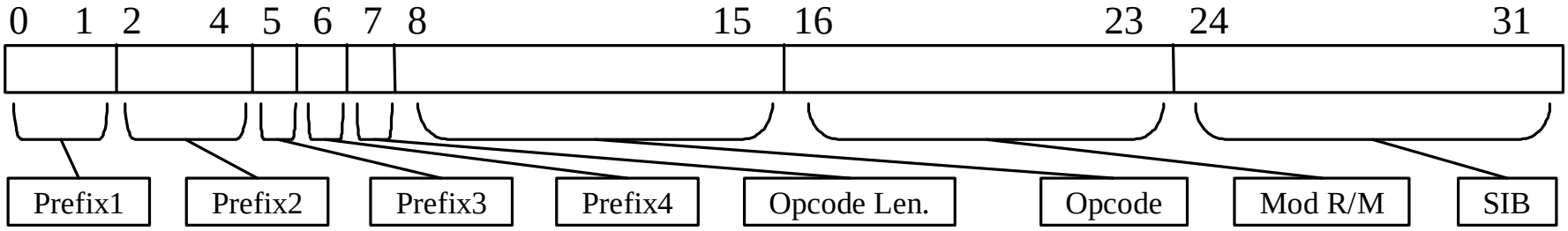
M’

$$V[j] = \begin{cases} 1111, if \;\; j = 0 \\ (V[j-1] + (V[j-1] \,\&\, M[y_j])) \,|\, (V[j-1] \,\&\, M'[y_j]), if \;\; 1 \le j \le n \end{cases}$$

ここで出てくるキャリーがL[m,j]-L[m,j-1]になる.

縮約命令

- 縮約命令
  - １機械語命令のオペランド以外の情報を32ビットで表現．
  - オペコードは勿論，オペランドの型情報や命令プレフィックスの情報も含まれる．
- 効果
  - 縮約命令の種類数＜機械語命令の種類数
    ビットベクトル化の際のメモリ使用量を減らせる．
  - 分岐命令と分岐先の間で命令が挿入・削除されても，類似度への影響は挿入・削除分だけで済む．
  - 絶対アドレスを意味するオペランドがあった場合，コードセグメントのリロケーションによりオペランドが変化するが，この影響も無くなる．



ビットベクトル化の効果

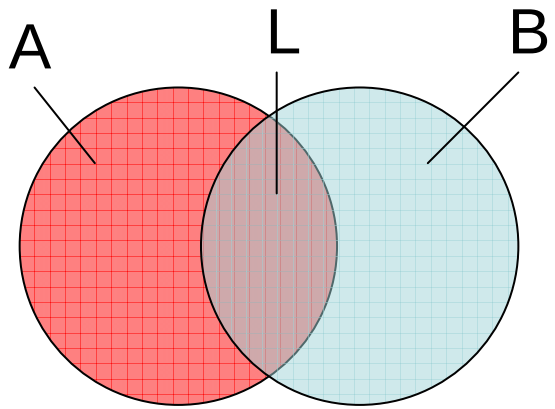
- SSE2を使うと・・・
  - and, andnot, orが128ビットでいける.
  - addは64ビット. ただしキャリーフラグがないため, 実質63ビット.
  - それでも(メモリアクセスを除けば)10クロック程度で126ビット(マス)の計算をこなせるようになった.
  - 単純なCの実装と比較すると, 100倍超の高速化！

- Core2Quad Dual(2.6GHz)フル回転, 約4日で $_{3000}C_2$ のペアのLCSを算出が可能.

類似度の定義

- マルウェアの逆アセンブル結果から縮約命令列を生成し，2つの縮約命令列A,BのLCSをLとしたとき，類似度Sを以下のように定義する．
  - A,Bを集合，LをAとBの積集合とみなしたときのJaccard係数に相当する．

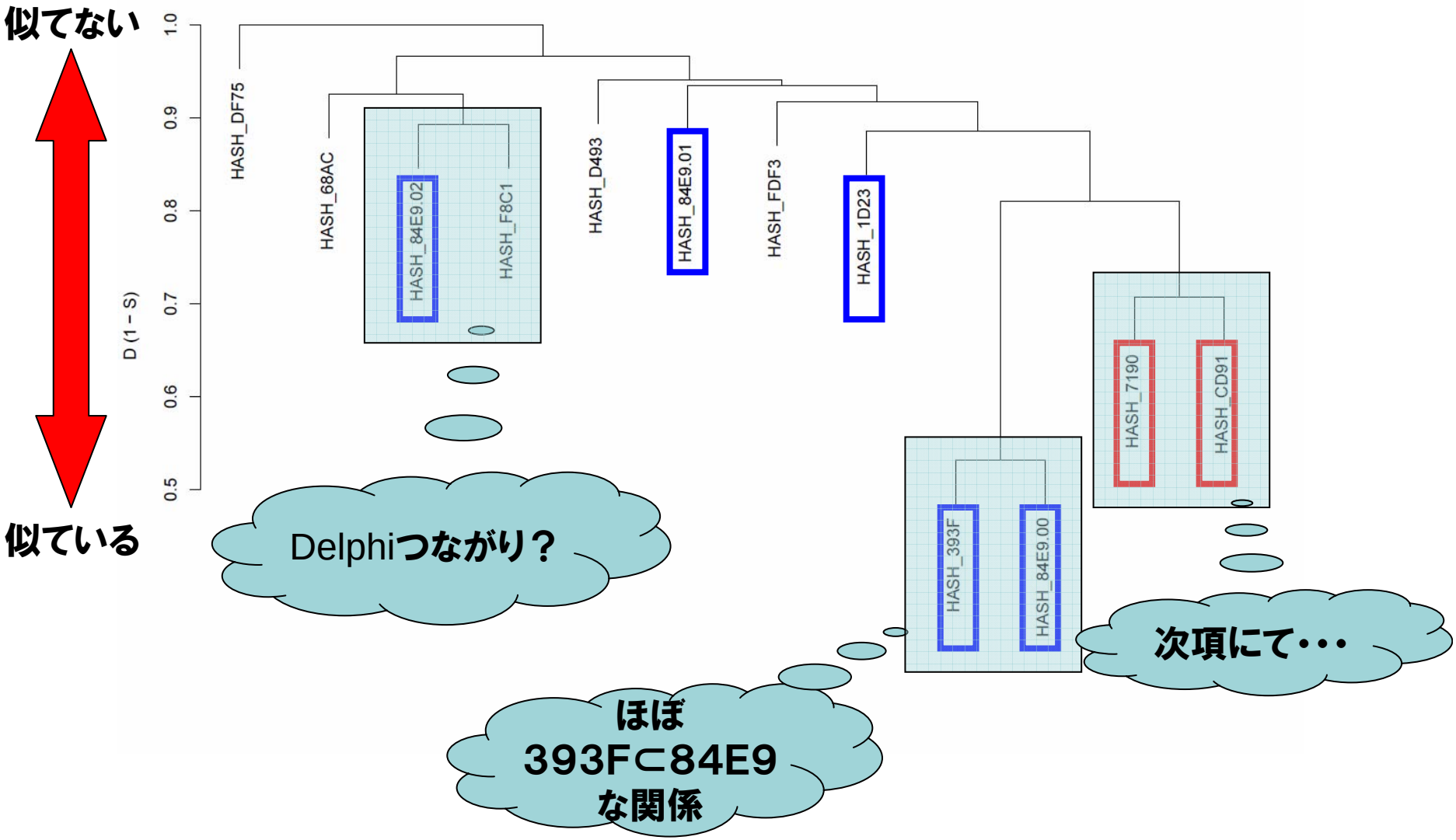
$$S = \frac{|L|}{|A| + |B| - |L|}$$



NTT

MWS2017を待たずして，
いよいよ分類結果です

CCC DATAset 2009マルウェア検体の分類結果



NTT CD91より7190の方が機能が豊富に見えるが・・・

7190(IRCを使うボットっぽい)

```
push    offset CriticalSection
call    ds:RtlEnterCriticalSection
push    dword ptr [ebp+74h+netshort]
push    dword ptr [ebp+74h+in.S_un] ; in
call    edi ; inet_ntoa
push    eax
lea     eax, [ebp+74h+var_2F4]
push    offset aScanIpSPortDIs ; "[SCAN]: IP: %s, Port %d is open."
push    eax
call    ebx ; wsprintfA
add     esp, 10h
cmp     [ebp+74h+var_1C], 0
jnz     short loc_4013F1
cmp     [ebp+74h+String], 0
push    1               ; int
push    [ebp+74h+isNotice] ; isNotice
lea     eax, [ebp+74h+var_2F4]
push    eax             ; int
lea     eax, [ebp+74h+String]
jnz     short loc_4013E5
lea     eax, [ebp+74h+var_148]

                        ; CODE XREF: StartAddress+115↑j
push    eax             ; lpString
push    [ebp+74h+s]     ; s
call    sub_4018CB
add     esp, 14h

                        ; CODE XREF: StartAddress+100↑j
push    offset CriticalSection
call    ds:RtlLeaveCriticalSection
```

CD91(ワームっぽい，autorun.inf関連も)

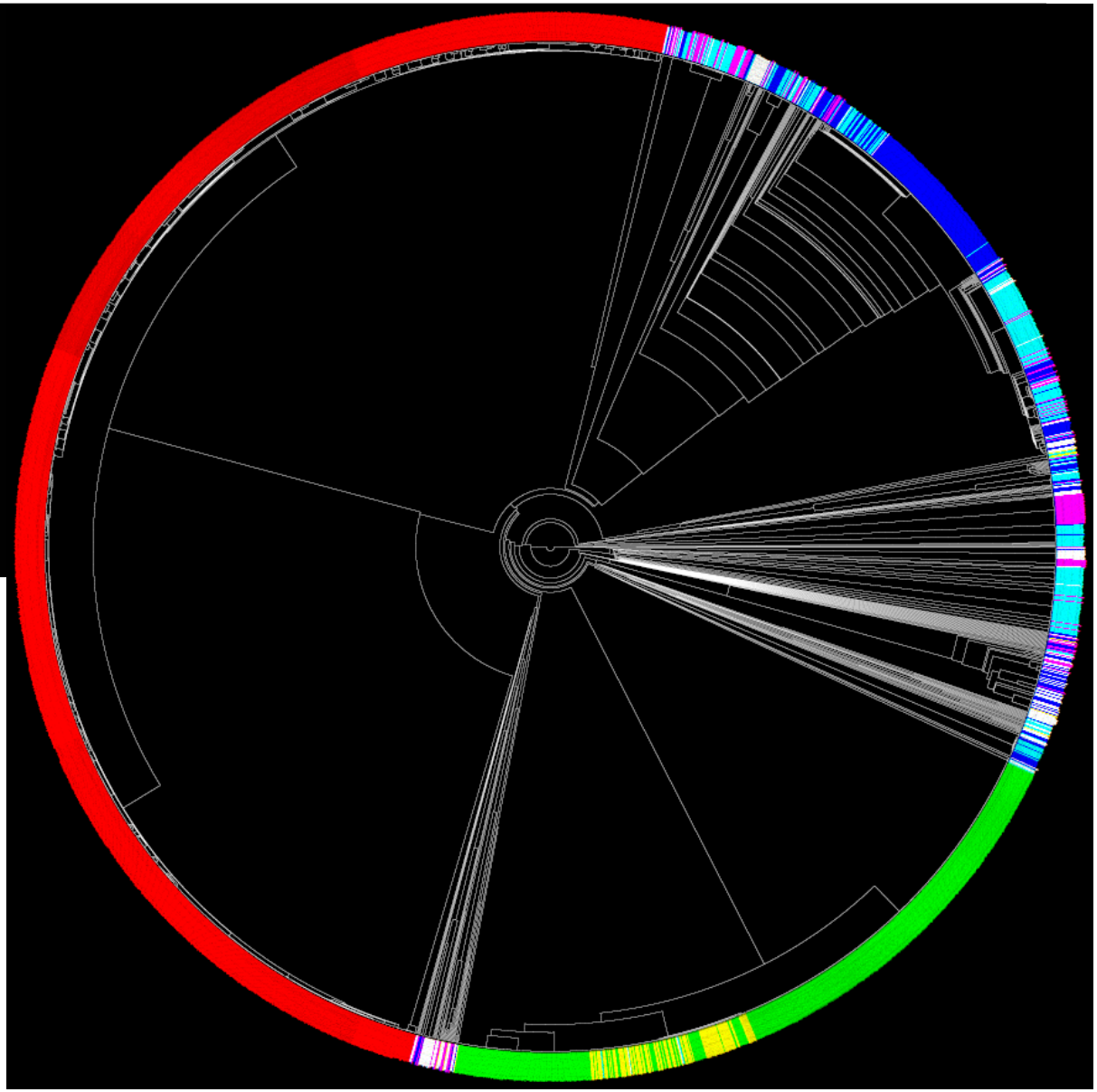
```
push    dword ptr [ebp+74h+netshort] ; netshort
push    dword ptr [ebp+74h+in.S_un] ; int
call    sub_4010F7
add     esp, 2Ch
cmp     eax, 1
jnz     loc_40130A
cmp     [ebp+74h+var_24], 0FFFFFFFFh
jnz     short loc_401270
mov     esi, offset CriticalSection
push    esi
call    ds:RtlEnterCriticalSection
push    esi
call    ds:RtlLeaveCriticalSection
jmp     loc_40130A
-----------------------------------------------

                        ; CODE XREF: StartAddres
push    dword ptr [ebp+74h+in.S_un] ; in
call    edi ; inet_ntoa
push    eax
lea     eax, [ebp+74h+var_224]
push    eax
```

CD91検体では，明らかにクリティカルセクション関連の処理が不要．
元となったマルウェアの残骸か．
汎用ボットから
感染に特化したワームへ？

次，3000検体です！

ある環境で収集したマルウェア

Downadup
Rahack
IRCBot
Virut
Spybot
Backdoor.Trojan

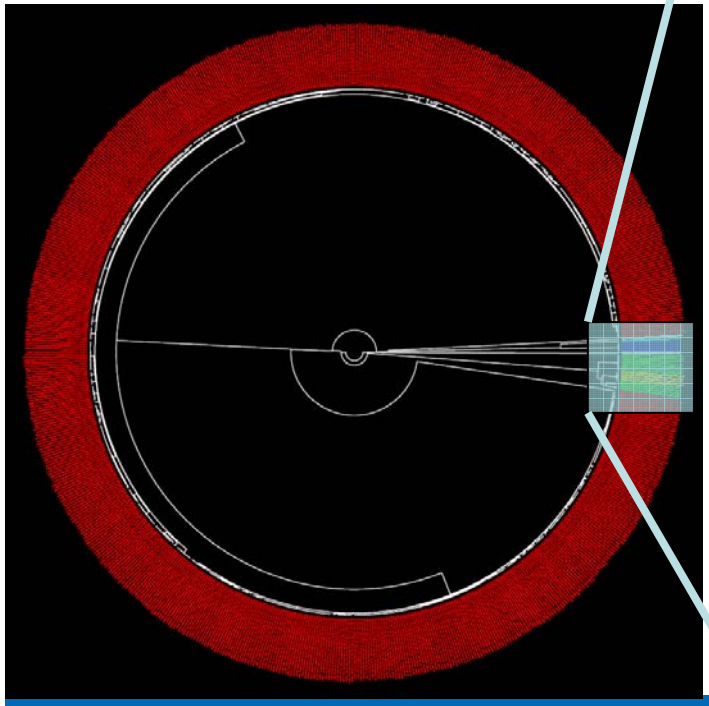


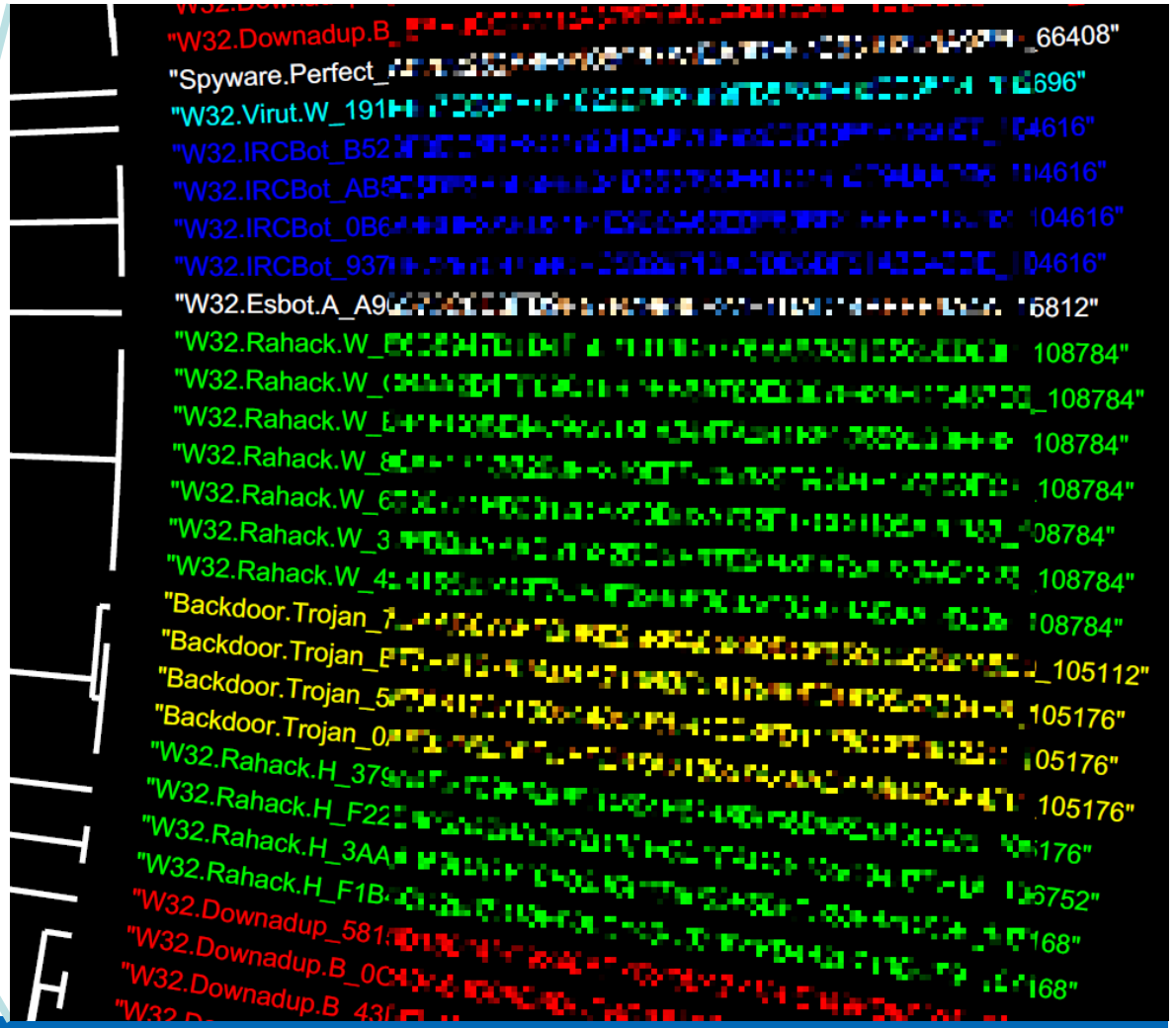
静的解析してみると，
主なクラスタは
- Downadup
- Rahack
- IRCBot
- **その他**

Virut**が入り混じっているのは**
宿主で分類したため．

NTT京都大学で収集されたマルウェア702種類の分類結果

- 我々が開発しているハニーポットで収集．シェルコードによる1次検体のみ．
  - 680種類（96.9%）がDownadup
    - うちAが1種類，BおよびB++が679種類
  - Rahack（Allaple）が15種類．
    アンチウィルスソフトの
    検出名が少し揺れている？





まとめと今後の課題

- アンパック・逆アセンブル・類似度算出・クラスタリングの全過程を自動化するシステムを提案
  - 縮約命令の導入
  - SSE2を使ってLCS算出アルゴリズムをビットベクトル化
- 約3000検体のマルウェアを分類してみた
  - (類似度の閾値にもよるが)かなり少ないクラスタ数.
    10個くらい解析すれば，大勢は掴めそう.
  - 差分もわりときちんと見える
    - レジスタのカラーリング変更まで見える
- 約700検体@京都はもっとすごい
  - 類似度80%で考えるとクラスタ数はわずか7つ.
- 今後の課題は
  - アンパック・逆アセンブル精度の向上
  - コンパイラ(オプション)変更に伴う変化への対応方法を検討