



@PacSec 2013

Fighting advanced malware using machine learning

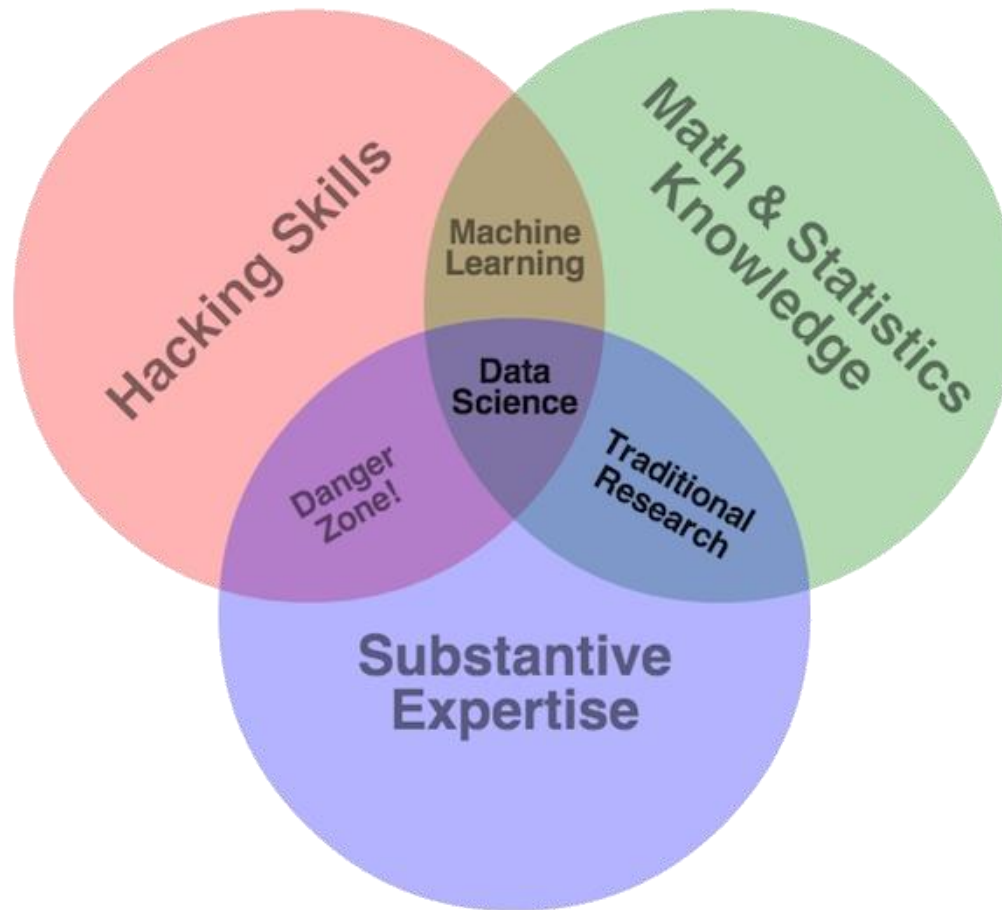
村上純一
新技術開発部長

FFRI, Inc.
<http://www.ffri.jp>

自己紹介

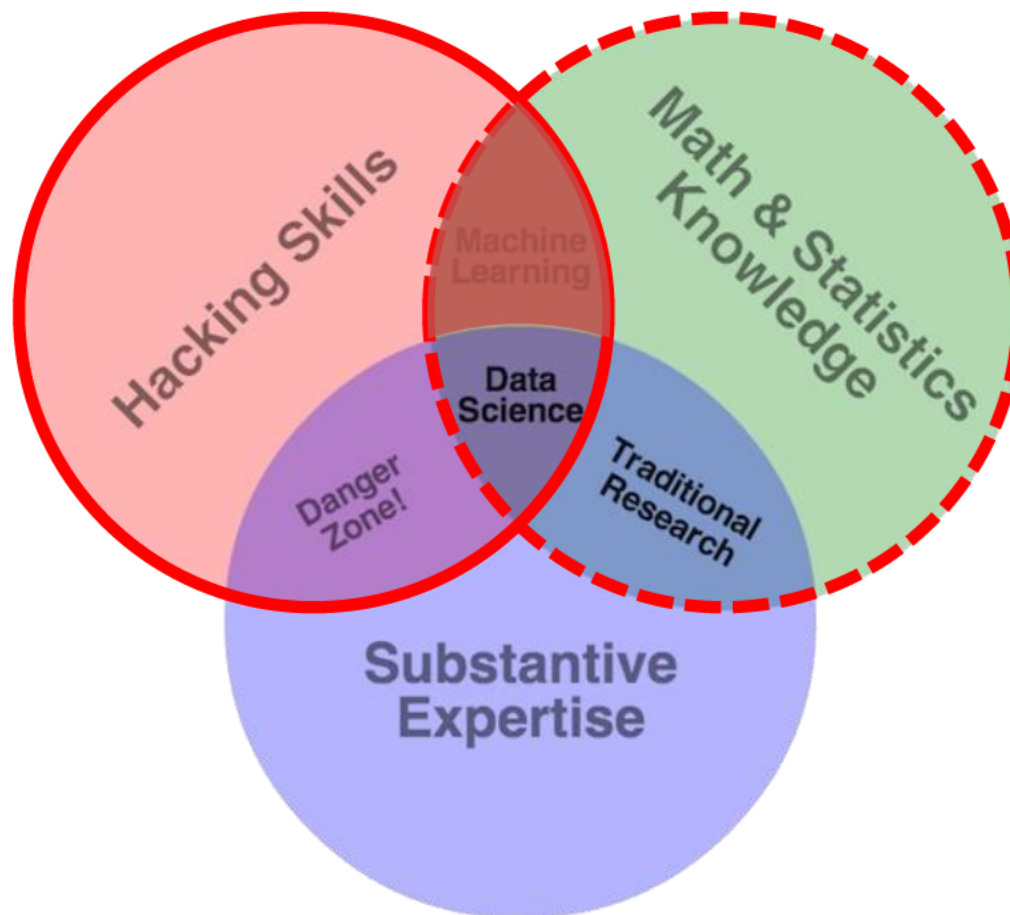
- セキュリティ研究者@FFRI
 - マルウェア解析、脆弱性解析
 - Windows, Linuxでのデバイスドライバ開発等
- 各種テクニカルカンファレンスにて講演
 - BlackHat USA/JP, RSA, PacSec, AVAR, etc.
- 機械学習の専門家ではない
 - 機械学習の*ユーザー*

データサイエンスの定義(情報セキュリティ)



<http://www.niemanlab.org/images/drew-conway-data-science-venn-diagram.jpg>

データサイエンスの定義(情報セキュリティ)



<http://www.niemanlab.org/images/drew-conway-data-science-venn-diagram.jpg>

アジェンダ

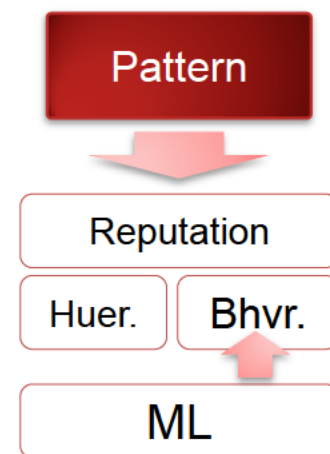
- 背景
- 検出アプローチ
- コンピュータ vs. 人間
- リアルタイム検知への応用
- まとめ

背景 – マルウェアおよびその検出技術

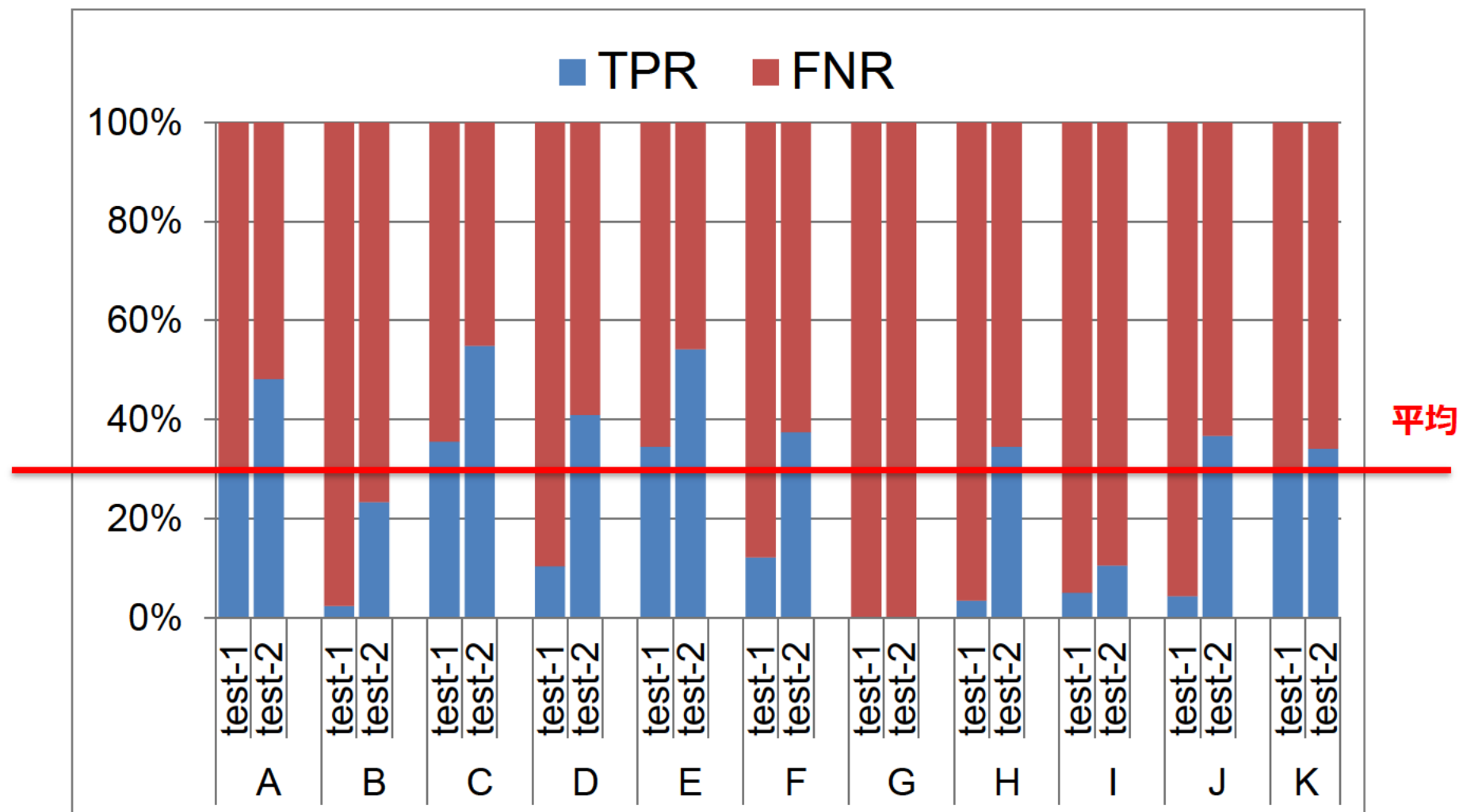


パターンマッチングの限界

- Metascanを利用して11社の製品の検出力を評価
- 最新のマルウェアを利用
- 異なるソース・期間から2つのデータセットを用意
 - test-1: 1,000 検体
 - test-2: 900 検体

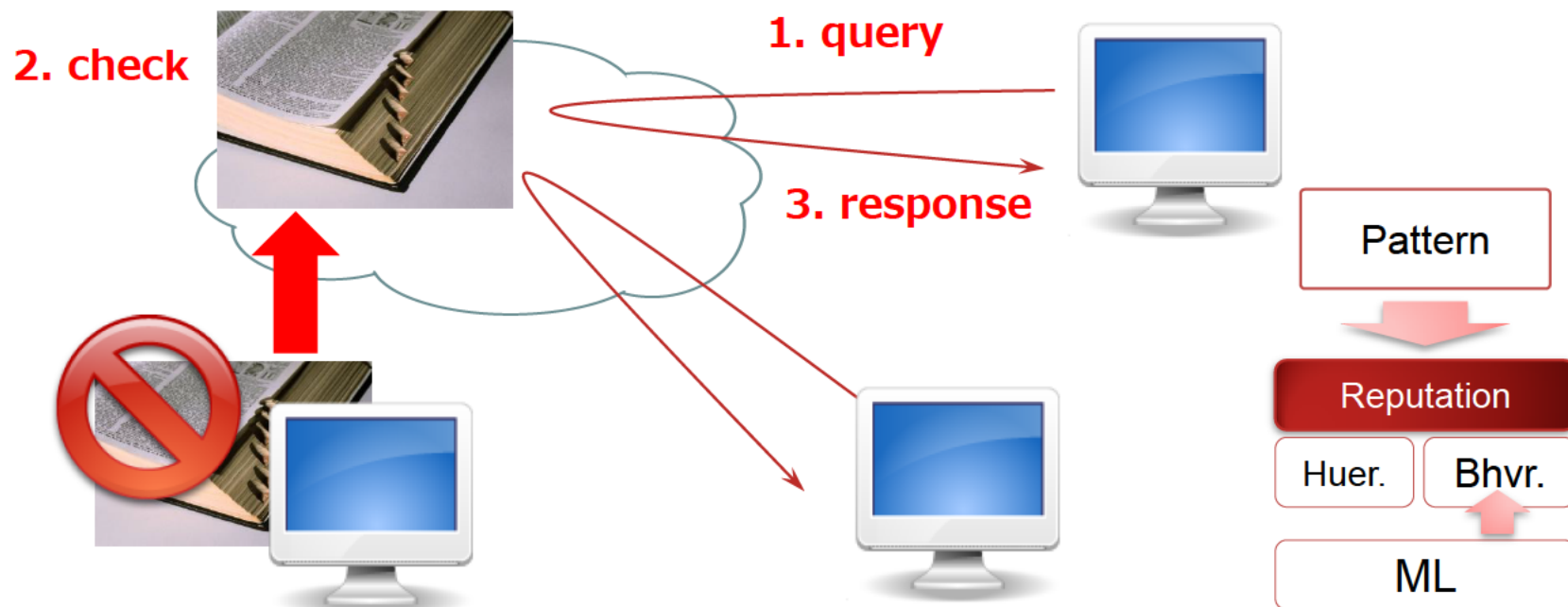


パターンマッチングの限界



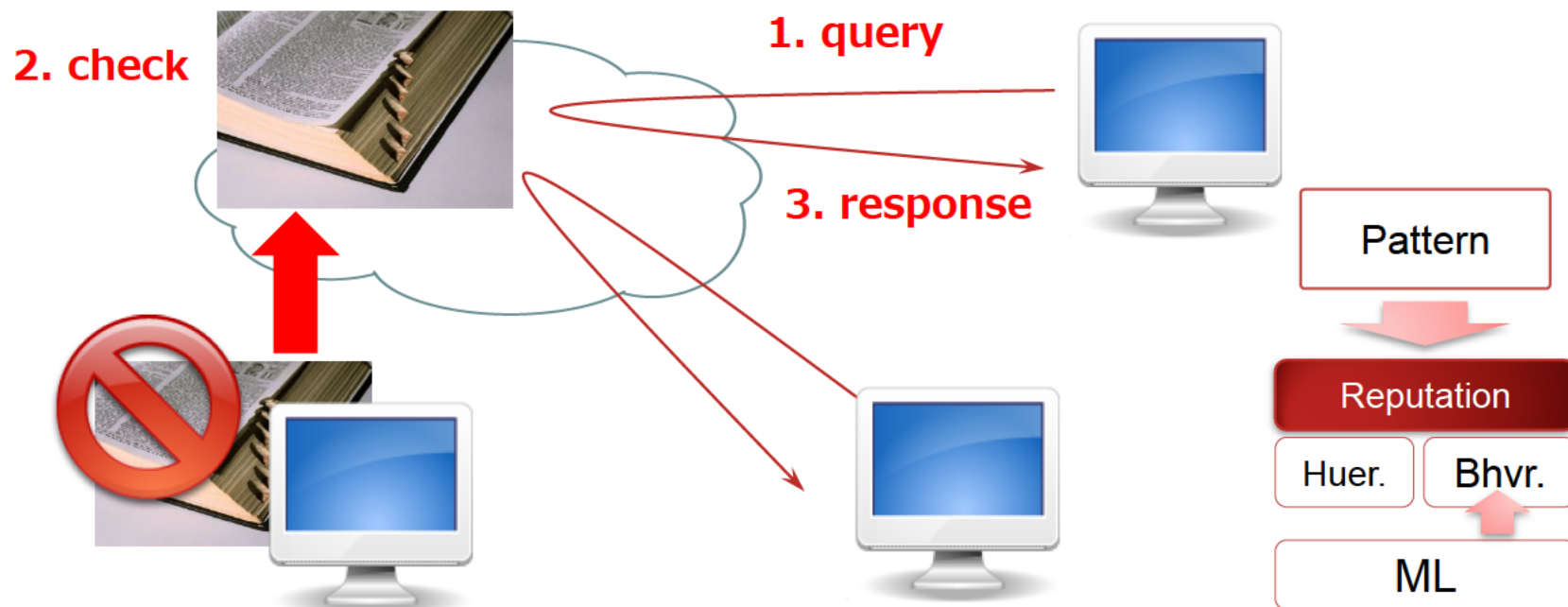
レピュテーション検知の強み

- 基本的なコンセプトはパターンマッチングと同じ (blacklist)
- エンドポイントはパターンを持つ必要がない
- 新しいパターンの反映が容易 (配信の必要がない)



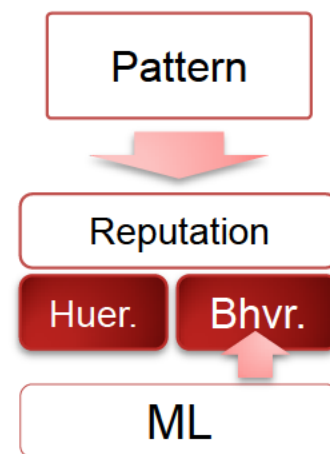
レピュテーションの弱み

- 「検知可能」 = 「既に誰かが攻撃されている」
- 自分達が初めての攻撃対象だった場合はどうか？
- 「真の標的型攻撃」に対してどの程度有効なのか？



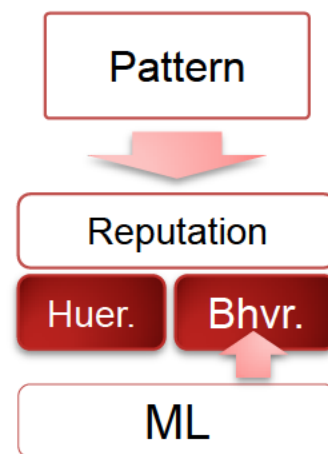
ヒューリスティック・ビヘイビア検知の強み

- 事前に定義した特徴、挙動に基づいて検知
 - OpenProcess → WriteProcessMemory → CreateRemoteThread
 - Runキー等の自動スタートポイントに自身を登録
 - Windows Firewallの無効化等
- 汎用的な検知機能を提供可能
- パターンが不要
(定常的なスキャン、アップデートが不要)



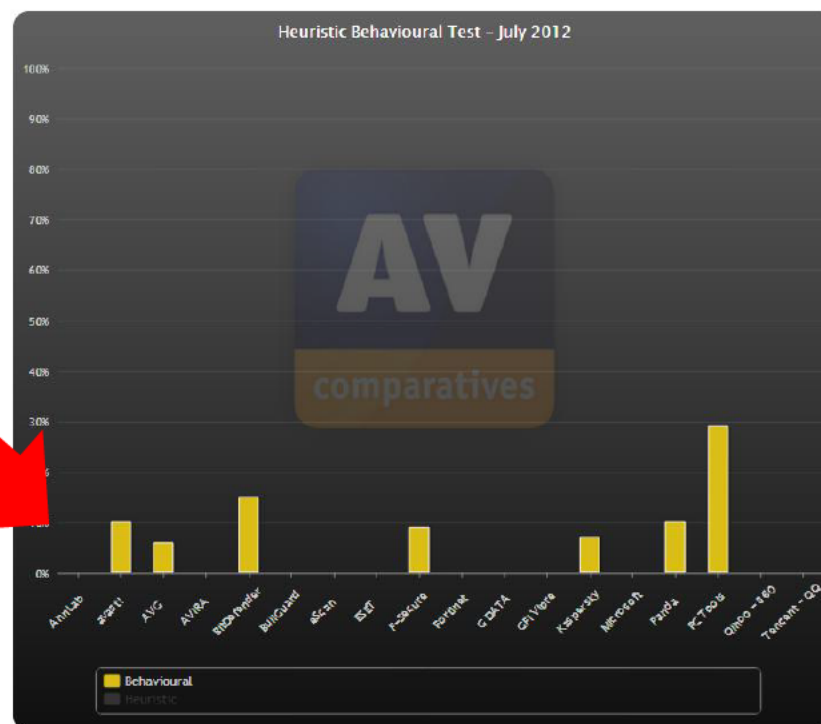
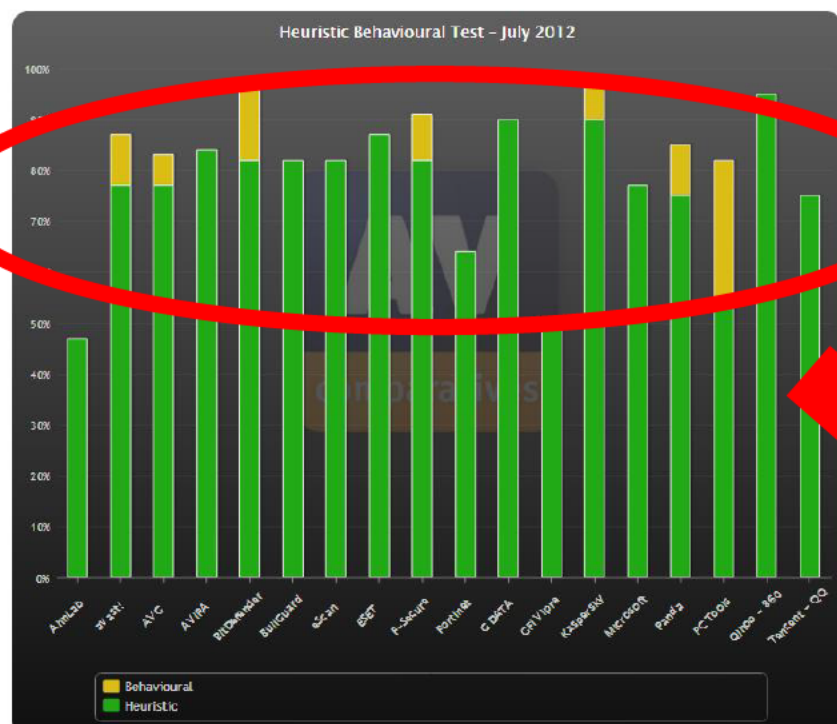
ヒューリスティック・ビヘイビア検知の弱み

- 完全な誤検知の回避が困難
- ユーザーに検出アクションを許可するか判断を仰ぐ場合も（ユーザーの判断依存）
- 継続的なマルウェア解析、ロジックの見直しが必要（人間よりもコンピュータに最適なタスク）



Heuristic Behavioral Test - July 2012

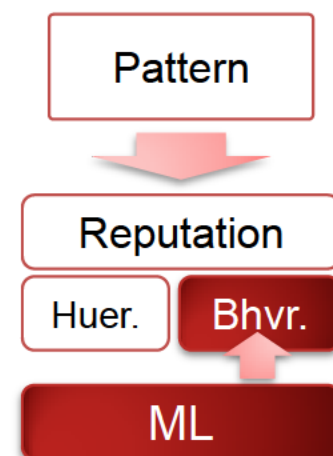
- AV-comparativesが2012年よりテストを実施、結果を公開
- ビヘイビア検知は検知にほとんど寄与していない(平均:4.8%)



<http://chart.av-comparatives.org/chart1.php>

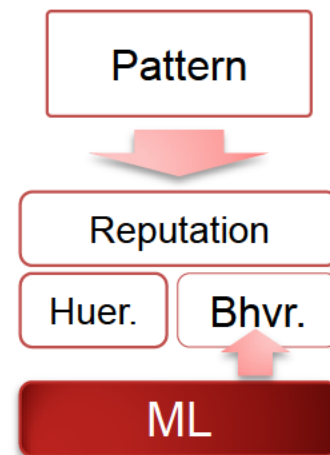
検出アプローチ

- 機械学習を利用したビヘイビア検出
- 既存アイデアだが産業界においてより実践的取り組み
- オープンソースを利用することで実現、自動化が容易
 - Cuckoo Sandbox
 - Jubatus

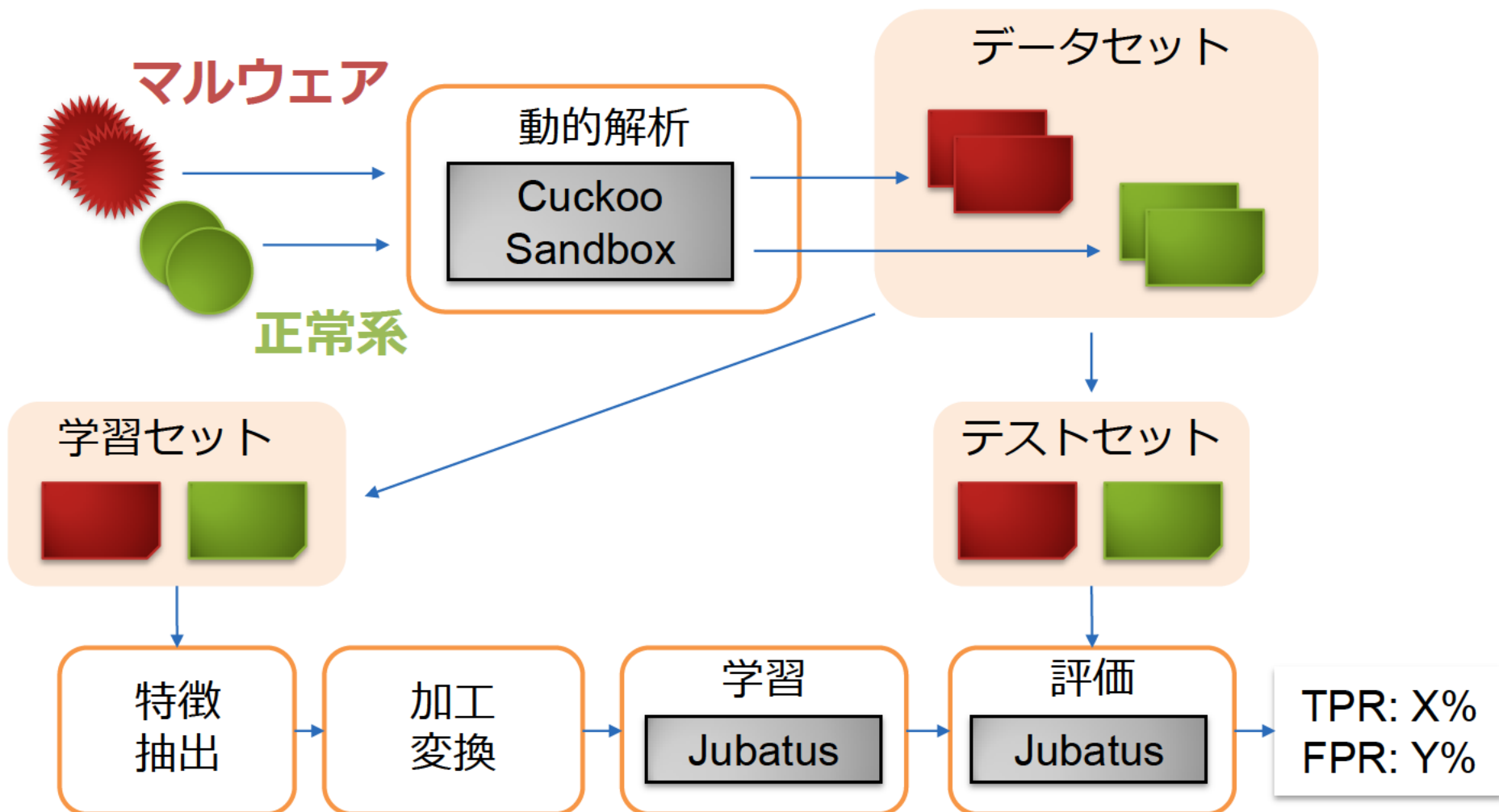


機械学習による検知

- 多くの研究が学术界で行われている
- 基本的には、分類に関する問題
- 下記の組み合わせに関する研究が主流



概要

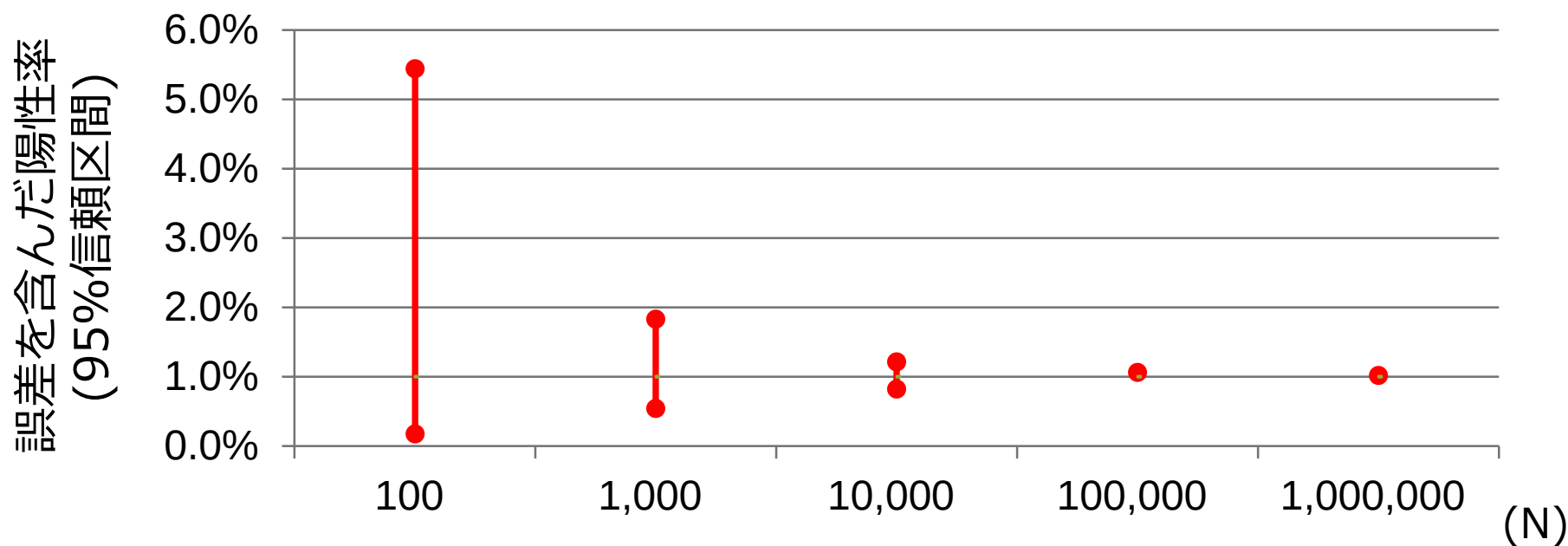


必要なサンプル数

- 信頼区間と呼ばれる概念
- どの程度誤差を許容するかに依存
- 下記はいずれも陽性率1%
 - 1/100 (N=100)
 - 10/1,000 (N=1,000)
 - 100/10,000 (N=10,000)
- 1%と断定する確信が異なる
 - それぞれ異なる誤差を持つ

必要なサンプル数

用意したサンプル数に応じて推定誤差を計算可能



マルウェア及び正常系データ

- 独自に収集している検体から無作為抽出
 - マルウェア: 15,000(5,000 = 学習用, 10,000 = テスト用)
 - 正常系: 15,000 (5,000 = 学習用, 10,000 = テスト用)
- ランダムであることが重要
 - 異なる期間（毎日の収集データから少量ずつ抽出）
 - 異なるソース
 - 抽出時にファイルタイプ、マルウェア種別は考慮しない

Cuckoo Sandbox - <http://www.cuckoosandbox.org/>

- オープンソースのマルウェア自動解析システム
 - 検体を仮想マシンにコピー
 - 検体を仮想マシン内で実行
 - 実行時の挙動をモニタリングし、データを記録
 - APIコール, ネットワーク通信, VirusTotalの結果等

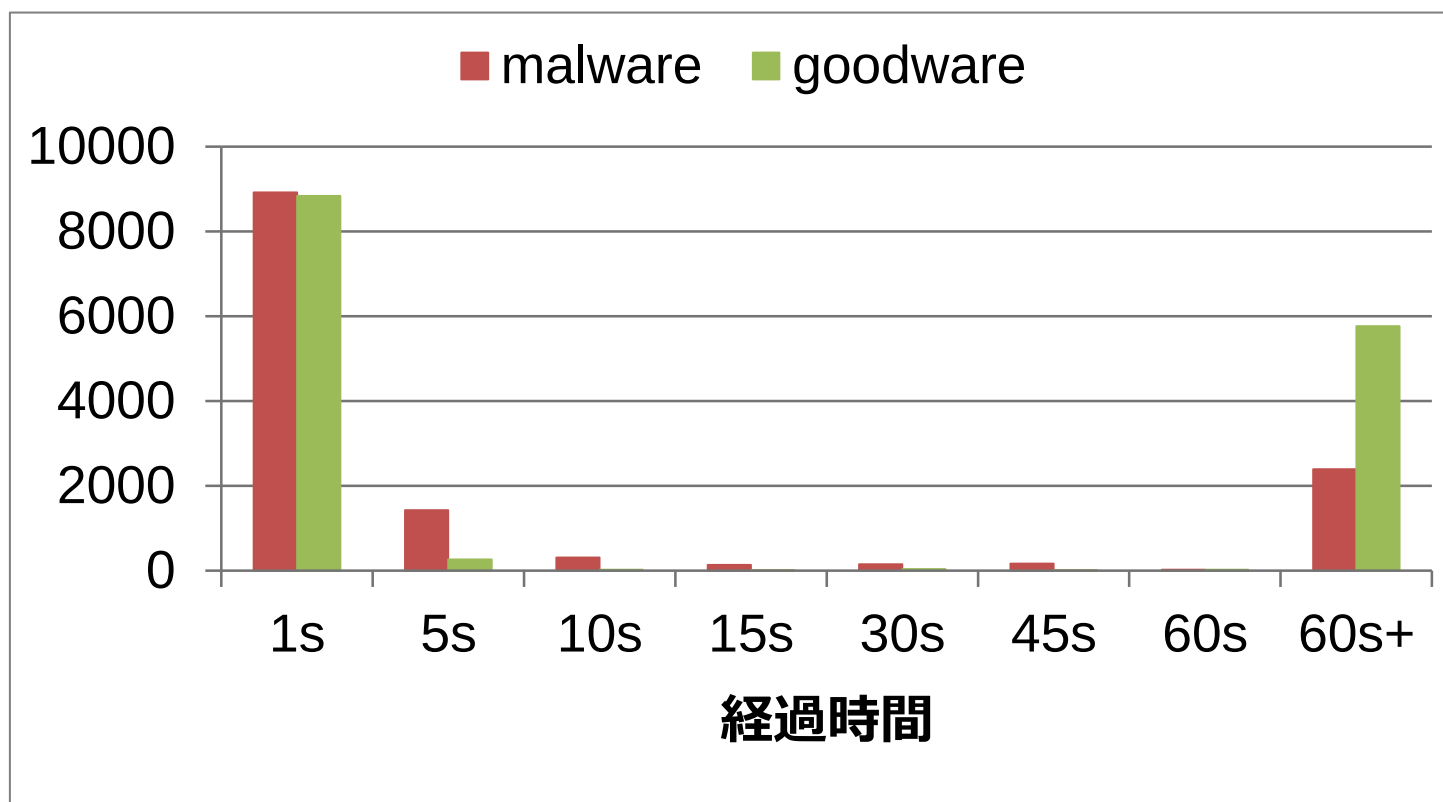
APIコール

```
"calls": [
  {
    "category": "system",
    "status": "FAILURE",
    "return": "0xc0000135",
    "timestamp": "2013-02-28 12:03:49,478",
    "thread_id": "420",
    "repeated": 0,
    "api": "LdrLoadDll",
    "arguments": [
      { "name": "Flags", "value": "1242916" },
      { "name": "FileName", "value": "C:\\WINDOWS\\system32\\VB6JP.DLL" },
      { "name": "BaseAddress", "value": "0x00000000" }
    ]
  },
  {
    "category": "registry",
    "status": "SUCCESS",
    "return": "0x00000000",
    "timestamp": "2013-02-28 12:03:49,528",
    "thread_id": "420",
    "repeated": 0,
    "api": "NtOpenKey",
    "arguments": [
      { "name": "KeyHandle", "value": "0x00000058" },
      { "name": "DesiredAccess", "value": "1" },
      { "name": "ObjectAttributes", "value": "Registry\\MACHINE\\System\\CurrentControlSet\\Services\\Tcpip\\Parameters\\Interfaces\\{...}" }
    ]
  }
],
```

APIコールの傾向

- 多くの検体が1秒以内にAPIコールを完了する
(または呼び出し続ける) -> 5秒以内に完了したものを対象に

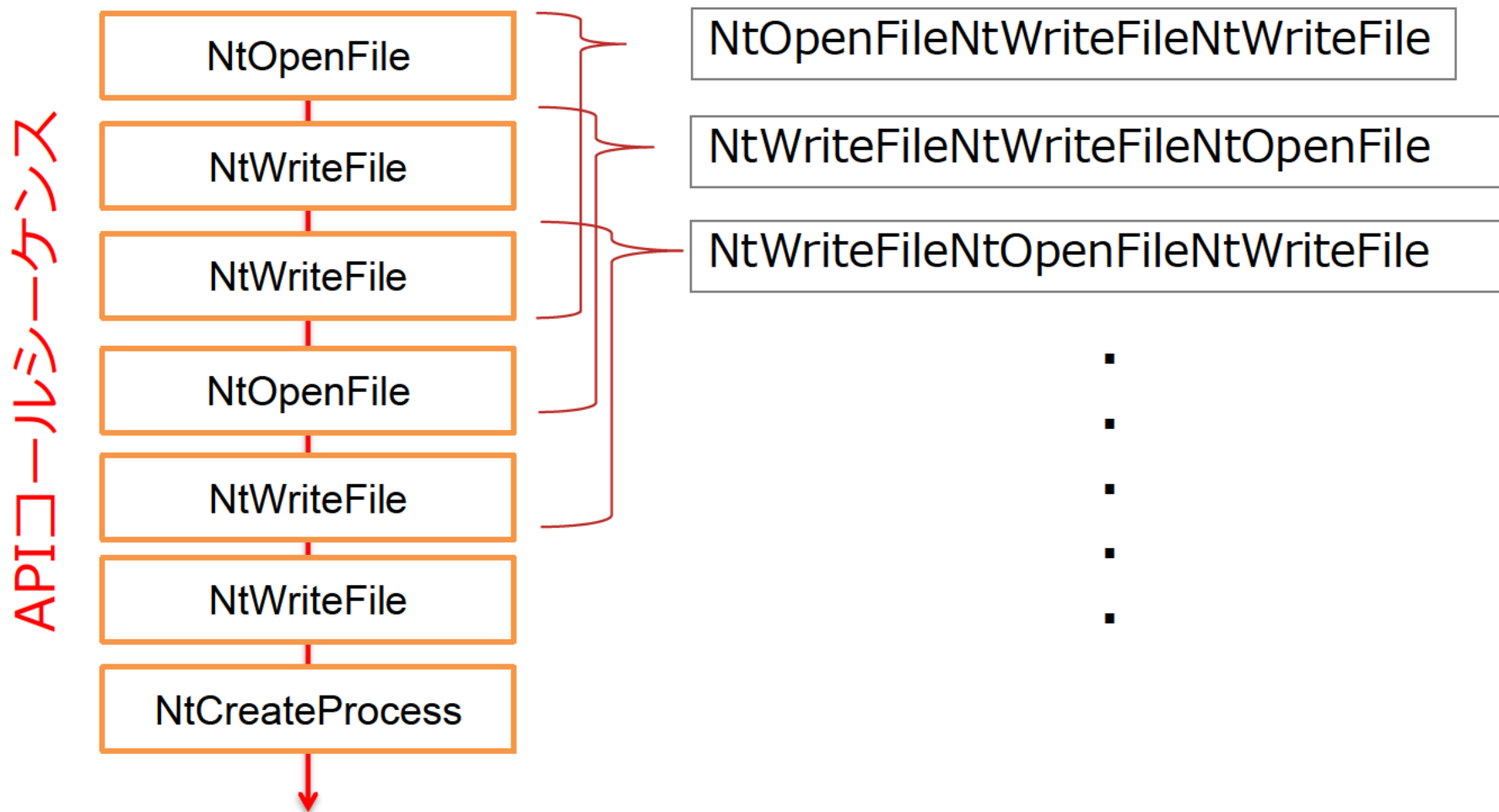
APIコールが完了した件数



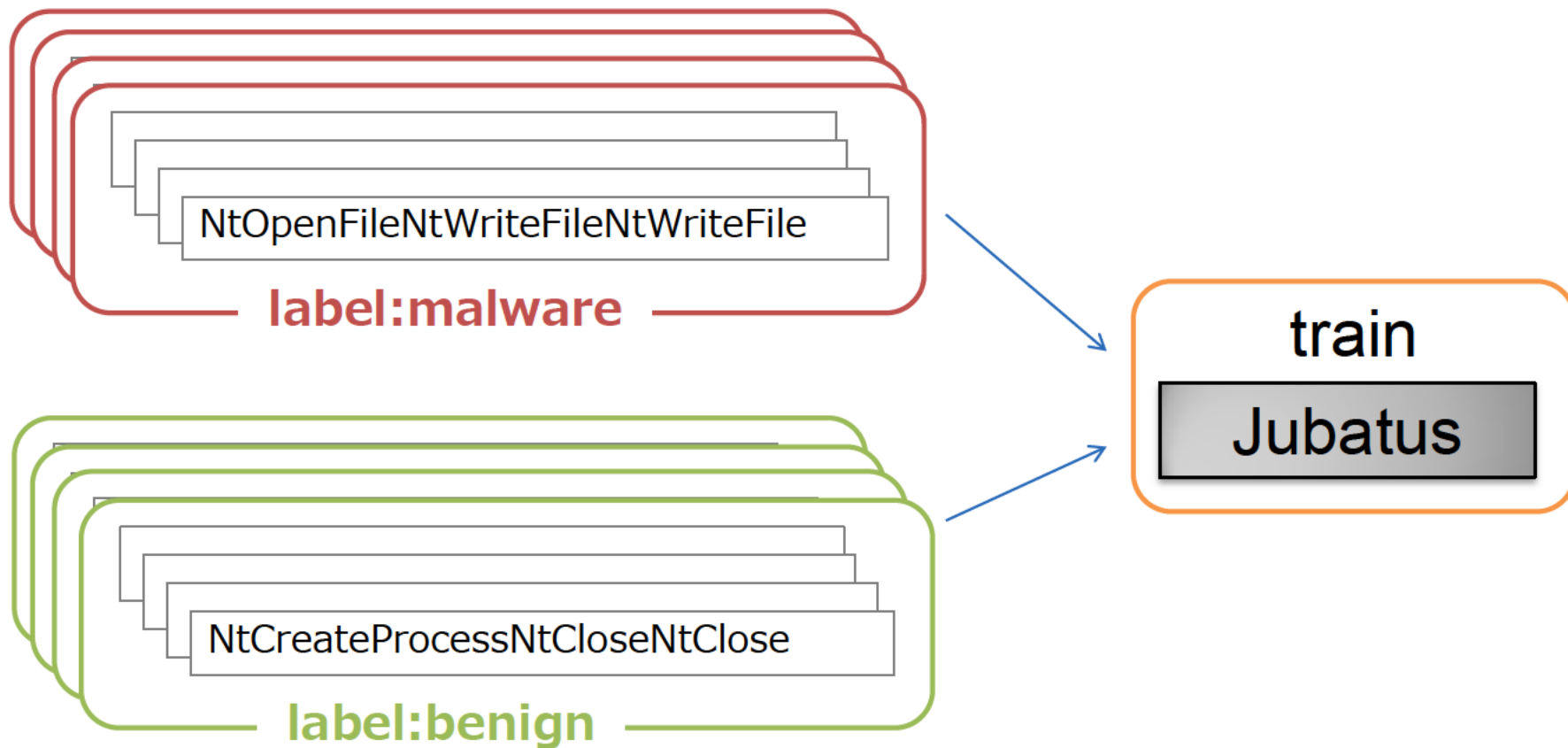
Jubatus - <http://jubat.us/en/>

- オンライン機械学習向け分散処理フレームワーク
 - 分散処理: スケーラブル
 - オンライン処理: 非バッチ処理、随時学習
- オープンソース, LGPL v2.1. 最新版0.4.5(22/07/2013)
- Preferred Infrastructure及びNTT Software Innovation Centerが開発
- 様々な機械学習をサポート
 - 分類、回帰分析、推薦、異常検知等
- 利用が容易（様々な言語バインディング、特徴ベクトル変換器等）

特徴抽出及び加工

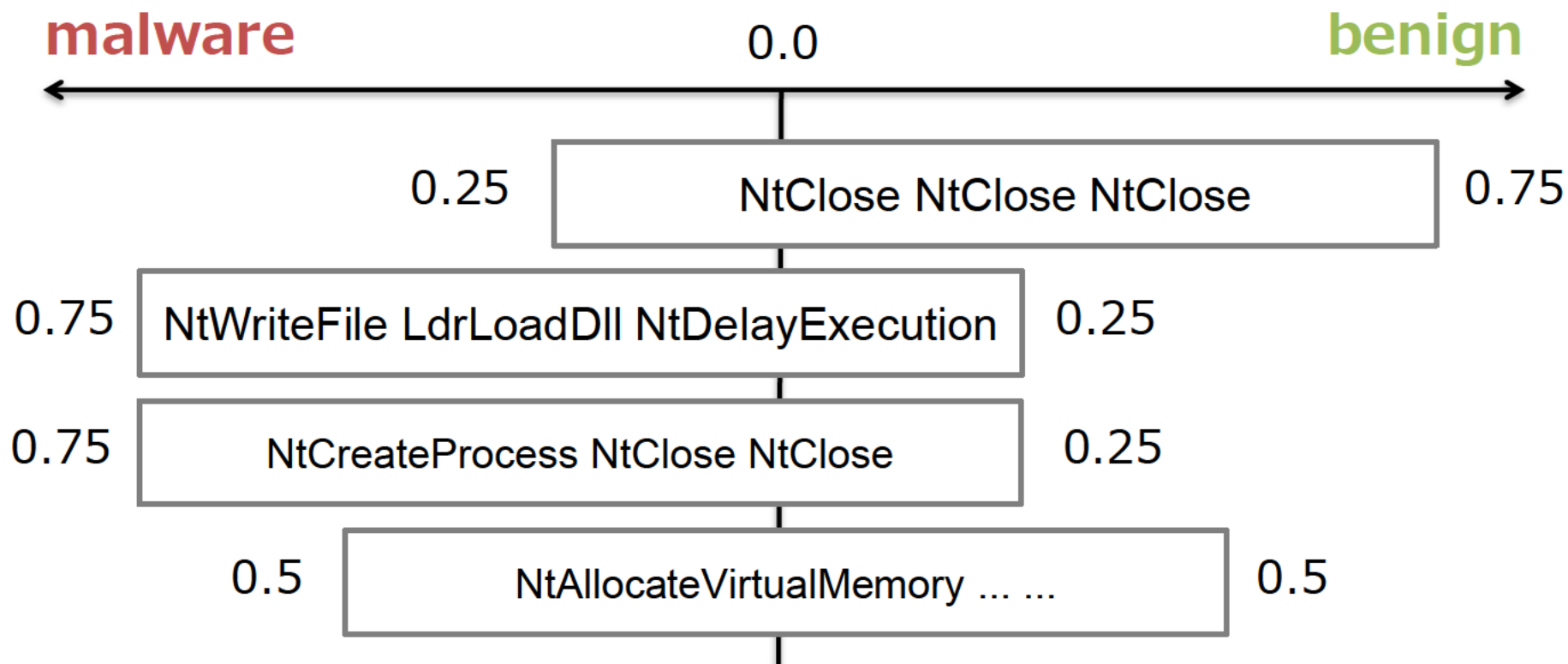


特徴抽出及び加工



(イメージ) 学習の内部状態

出現回数、ラベルに基づいて各特徴の重みを計算、更新



評価結果

- N of N-gram
 - 3~5-gram > 2-gram = 6-gram
- 最良値: 3-gram、log_tf-binによる重み付け
 - TRP: 72.33% [71.58 ~ 73.07 % (95%信頼区間)]
 - FPR: 0.77% [0.60 ~ 0.99% (95%信頼区間)]
- 上記はあくまで一つの結果
 - 様々な特徴の組み合わせが存在する
(今回はAPI名とそのシーケンスのみ特徴化)



デモ

学習モデルのダンプ

- <http://blog.jubat.us/2013/06/classifier.html>



jubalocal_storage_dump.cpp

<https://gist.github.com/t-abe/5746333>

API 3-gramにおけるマルウェアらしさ

```
[foo@nolife classifier]$ ./dump --input model --label "malware"
0.181128    api_call$VirtualProtectEx_VirtualProtectEx_VirtualProtectEx@space#log_tf/bin
0.142254    api_call$RegOpenKeyExA_NtOpenKey_NtOpenKey@space#log_tf/bin
0.137144    api_call$NtReadFile_NtReadFile_NtFreeVirtualMemory@space#log_tf/bin
0.134443    api_call$LdrLoadDll_LdrGetProcedureAddress_VirtualProtectEx@space#log_tf/bin
0.130287    api_call$LdrLoadDll_RegOpenKeyExA_NtOpenKey@space#log_tf/bin
0.130287    api_call$DeviceIoControl_LdrLoadDll_RegOpenKeyExA@space#log_tf/bin
0.122363    api_call$VirtualProtectEx_LdrLoadDll_LdrGetProcedureAddress@space#log_tf/bin
0.102545    api_call$NtFreeVirtualMemory_LdrGetDllHandle_NtCreateFile@space#log_tf/bin
0.102485    api_call$RegCloseKey_RegCloseKey_RegCloseKey@space#log_tf/bin
0.0983165   api_call$NtReadFile_NtFreeVirtualMemory_LdrLoadDll@space#log_tf/bin
0.0966545   api_call$NtSetInformationFile_NtReadFile_NtFreeVirtualMemory@space#log_tf/bin
0.094639    api_call$NtMapViewOfSection_NtFreeVirtualMemory_NtOpenKey@space#log_tf/bin
0.0933827   api_call$NtFreeVirtualMemory_LdrLoadDll_LdrGetProcedureAddress@space#log_tf/bin
0.0905402   api_call$DeviceIoControl_DeviceIoControl_NtWriteFile@space#log_tf/bin
0.0903766   api_call$DeviceIoControl_NtWriteFile_NtWriteFile@space#log_tf/bin
0.0884724   api_call$RegOpenKeyExW_RegOpenKeyExW_LdrGetDllHandle@space#log_tf/bin
0.0853282   api_call$LdrLoadDll_LdrLoadDll_LdrLoadDll@space#log_tf/bin
...
```

API 3-gramにおける正常系らしさ

```
[foo@nolife classifier]$ ./dump --input model --label "goodware"
0.268353      api_call$LdrGetDllHandle_LdrGetDllHandle_ExitProcess@space#log_tf/bin
0.268353      api_call$LdrGetDllHandle_ExitProcess_NtTerminateProcess@space#log_tf/bin
0.259838      api_call$NtWriteFile_LdrGetDllHandle_LdrGetDllHandle@space#log_tf/bin
0.25887 api_call$NtWriteFile_NtWriteFile_LdrGetDllHandle@space#log_tf/bin
0.135514      api_call$NtOpenFile_NtOpenFile_NtCreateFile@space#log_tf/bin
0.122445      api_call$DeviceIoControl_LdrLoadDll_LdrGetProcedureAddress@space#log_tf/bin
0.12242 api_call$DeviceIoControl_DeviceIoControl_LdrGetDllHandle@space#log_tf/bin
0.119231      api_call$GetSystemMetrics_LdrLoadDll_NtCreateMutant@space#log_tf/bin
0.115319      api_call$DeviceIoControl_LdrGetDllHandle_LdrGetProcedureAddress@space#log_tf/bin
0.109306      api_call$LdrGetProcedureAddress_NtOpenKey_LdrLoadDll@space#log_tf/bin
0.105579      api_call$NtReadFile_NtReadFile_NtReadFile@space#log_tf/bin
0.104565      api_call$NtCreateFile_NtCreateFile_NtWriteFile@space#log_tf/bin
0.103304      api_call$RegOpenKeyExA_LdrGetDllHandle_LdrGetProcedureAddress@space#log_tf/bin
0.10306       api_call$VirtualProtectEx_RegOpenKeyExA_LdrGetDllHandle@space#log_tf/bin
0.100701      api_call$NtFreeVirtualMemory_NtFreeVirtualMemory_GetSystemMetrics@space#log_tf/bin
...
```

コンピュータ vs. 人間

- “VirtualProtectEx_VirtualProtectEx_VirtualProtectEx”
マルウェアらしさに関連しているように見える
- “RegOpenKeyExA_NtOpenKey_NtOpenKey”はどうか？
- コンピュータは人間が認識できないものを認識している
(非常に強力な左脳)
- コンピュータと協力してみてはどうか？

コンピュータを利用する

- 機械学習により学習モデルを生成する
- 人間がモデルをチェックし、新ロジックを開発
(人間の右脳を利用)
- 機械学習による検知は御し難い場合も
 - 詳細な検出条件を指定できない
「コンピュータが悪と言ったから悪」 (理由は不明)
- 機械学習とロジックベースの検出の掛け合わせ

リアルタイム検知への応用

- 静的情報を特徴に利用した場合
 - プログラム実行前に検査可能
 - パフォーマンスは選定した特徴次第
- 動的情報を特徴に利用した場合
 - マルウェアは既に実行されている
 - 検知が手遅れになる場合も
 - この観点でも機械学習とロジックの掛け合わせは有効

まとめ

- 従来のパターンマッチングは限界を迎えている
- 現行のビヘイビア検知は検出にほとんど寄与していない
- ビヘイビア検知に機械学習を適用
 - TPRが改善
 - 人間には分からない特徴を発見（コンピュータが）
 - これらを人間が利用することが重要

Thank you!

Contact: research-feedback@ffri.jp

Twitter: @FFRI_Research