



Security Holdings

# The Rule for Wild Mal-Gopher Families.

NTTセキュリティ・ジャパン

野村和也

平尾早智澄

1. Introduction
  2. Creating YARA Module
  3. Clustering Evaluation
  4. Applying to “Wild” Binaries
  5. Conclusions
- Appendix

# **1. Introduction**

**The Rule for Wild Mal-Gopher Families.**

## 野村和也

NTTセキュリティ・ジャパンのSOCアナリスト。主な業務はIPS/IDS/EDRでのアラート監視。  
NTTセキュリティ・ジャパンにおいてマルウェア解析とデータ可視化の記事を投稿。  
MWS2020論文賞受賞、SecHack2020優秀修了生。

## 平尾早智澄

NTTセキュリティ・ジャパンのSOCアナリスト。元金融系インフラエンジニア。  
SOCではNW/EDRのアラート監視の他マルウェア解析なども担当。

## 1. YARAモジュールの実装

- 大規模な検体群に対する高速かつ簡便な分類が可能に

## 2. 解析済みの検体を用いた精度評価

- ファミリ分類のために最適なパラメータを検討

## 3. VirusTotalに投稿された検体への適用

- 未解析の「野生の」検体への応用
- 最新のGolangマルウェアへの応用について考察

- **Golangマルウェアのファミリーは年々増加**

- 特徴的な構造による解析の複雑化
- マルチプラットフォーム向けにビルド可能



攻撃者のメリット

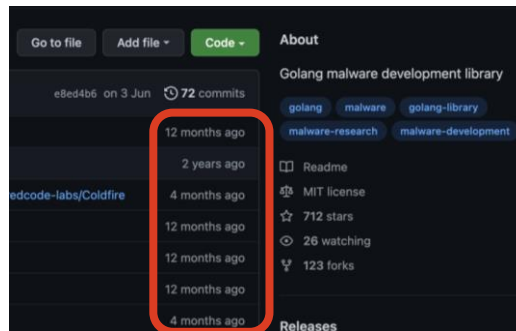


- **多種多様な検体が今後も増加し続ける**

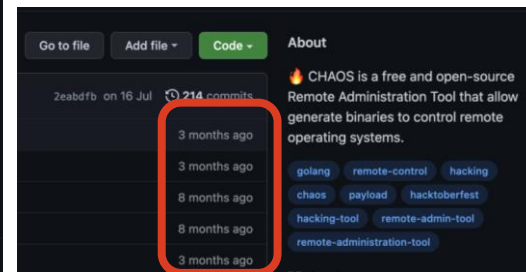
- 分類の効率化
- 解析済みの検体への帰結による解析の効率化

Golangマルウェアの分類・解析の効率化

- Golang向けの様々なマルウェア作成フレームワークが存在
  - Coldfire
  - CHAOS
  - EGESPLOIT
  - ARCANUS
- 開発が活発なフレームワークが多い



<https://github.com/redcode-labs/Coldfire>



<https://github.com/tiagorlampert/CHAOS>

## • Golangバイナリ版のimphash

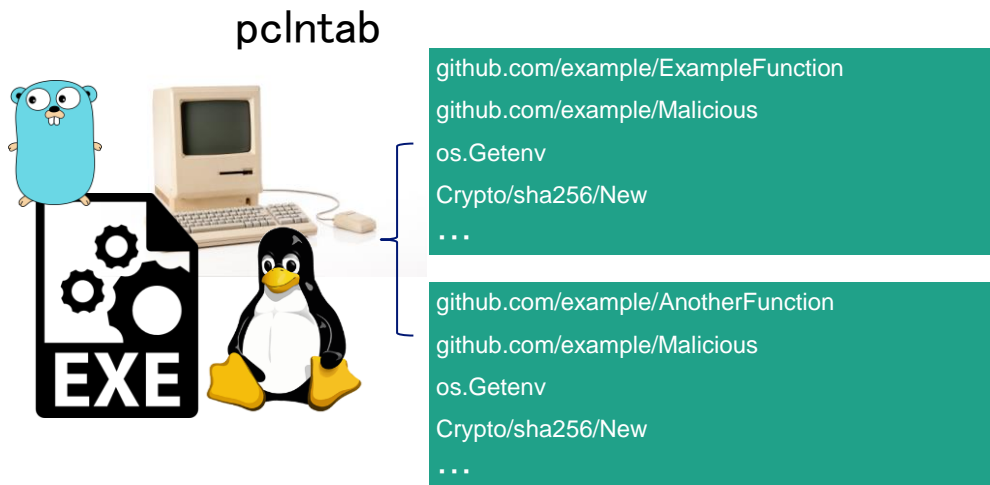
- Golangバイナリはプラットフォームに拠らずpcIntabと呼ばれる構造を持つ
- 依存するパッケージ名や関数名などを復元可能
- gimphashは復元したパッケージ名/関数名のうち、一部をSHA256でハッシュ化したもの
- マルウェアが依存する機能を一意に表現可能だが、**異なるハッシュの類似度比較は不可能**





- gimphashをファジーハッシュに

- SHA256は入力が1bitでも違えば出力は大きく変化
- Fuzzy Hashは似ている入力には似ている値を返す、“大雑把な”ハッシュを計算
- gimpfuzzyはssdeepを利用。検体間の類似度が測定可能に



gimpFuzzy

3:j4dwGIVWYvgxN4dw  
GIV3MNWzYKKKvTKrs  
:j4mGggximGCMcTjKrs

類似度の比較が可能

3:j4dwGIV++RSXzM9xN4d  
wGIV3MNWzYKKKvTKrs:j4  
mGduSXg9ximGCMcTjKrs

- **CODE BLUE 2022**

- 「Who is the Mal-Gopher? – Implementation and Evaluation of “gimpfuzzy” for Go Malware Classification」

- gimphashにFuzzy Hashを適用する手法を初めて提案
    - 解析済みの検体をgimphashにより分類し精度評価

- **JSAC2023 「The Rule for Wild Mal-Gopher Families. 」  
実際のオペレーションや解析での応用に着目し実装・評価**

- 実施に検体の分類が可能なYARAモジュールの作成
  - VTに投稿された「野生の」未解析の検体への適用と評価

## **2. Creating YARA Module**

**The Rule for Wild Mal-Gopher Families.**

- VTが開発しているマルウェア分類のためのツールキット<sup>[1]</sup>
  - 分類のルールを書くことで、ルールを満たす検体のみを探索可能
  - Cで実装されているため高速
- ファイル構造に応じた様々なモジュールが存在
- 以下のモジュールは存在しない
  - Golangバイナリを扱うモジュール
  - Fuzzy Hashingや文字列の類似度計算を行うモジュール
- gimpfuzzyによる検体の分類を手軽に行えるようにするため  
**新たにYARAモジュールを実装**



[1] <https://github.com/VirusTotal/yara>

以下の二つを実装

- **goモジュール : goLang製のPEバイナリを解析**
  - `go.gimpfuzzy()` : 抽出した関数名からgimpfuzzy計算
  - `go.function_names` : 抽出した関数名の文字列配列
- **fuzzyモジュール : Fuzzy Hashの類似度を計算**
  - `fuzzy.fuzzy()` : 引数の文字列からファジーハッシュ計算
  - `fuzzy.score()` : 引数の文字列二つを比較し編集距離を基にしたスコア算出

- YARAのルール例
  - gimpfuzzyを基にした検体の類似度に基づいた検索が可能に

```
import "go"
import "fuzzy"
import "pe"

rule GoFuzzyTest
{
  condition:
    pe.is_pe and
    fuzzy.score(go.gimpfuzzy(), "96:05iaa8UdGAq27F92...") > 80
}
```

類似度の  
スコア計算

検体のgimpfuzzy  
計算結果

比較する  
Fuzzy Hash

類似度が80超の  
検体を探索

- gimpfuzzyの類似度をベースに検体の検索が可能に

```
root@1f06b9d1f716:/malwares# yara /test.yara -r .
GoFuzzyTest ./Valhalla_hkctl_htran_golang/4550635143c9997d5499d1d4a4c860126ee9299311fed0f85df9bb304dca81ff
GoFuzzyTest ./Valhalla_hkctl_htran_golang/645622a85906da6304315ae9444046f2310609da933f53e87b54fbb206b53e3e
GoFuzzyTest ./Valhalla_hkctl_htran_golang/4e5468e36dc7bc5601384f22c032f990f2e8454d27f6b11e8e897fb0c6c5e0e5
GoFuzzyTest ./Valhalla_hkctl_htran_golang/65cfa86dec6f19cdbf5f9641ab835af023d34fa23b0e31a9f9b66c93a221d7a2
GoFuzzyTest ./Valhalla_hkctl_htran_golang/72549bdc9e857162603f3ce90f1bfc8eb761e7e9f399a24a2bba47468b6edfe3
GoFuzzyTest ./Valhalla_hkctl_htran_golang/91bce99e792db5c3da42da3f01f50a1021f9538b78f70544bedc9ca7508ce54e
GoFuzzyTest ./Valhalla_hkctl_htran_golang/d45a6f12d5956f0fb8ad17727c717b621e3be06fabf9ff27058cb86f8f108b7d
GoFuzzyTest ./Valhalla_hkctl_htran_golang/e70e0c8fb2727b35b65596a6e2838abd0b5f7351cdd4031b9971b91c22f5d15c
```

- 下記の関数を実装する

主な  
実装

関数	
module_initialize	YARAモジュールの初期化处理
module_finalize	YARAモジュールの終了処理
module_load	モジュールがファイルを読み込んだ際の処理 ファイルに対する実際の解析ロジックを実装
module_unload	モジュールがファイルを読み込んだ際の後処理 ハッシュテーブルの削除、構造体の開放など



- 各関数で例外が発生した場合きちんと対応したエラーを返すことが重要
  - 場合によってはYARA側が適切に後処理する

```
#ifndef ERROR_SUCCESS  
#define ERROR_SUCCESS 0  
#endif
```

} 成功処理

```
#define ERROR_INSUFFICIENT_MEMORY 1  
#define ERROR_COULD_NOT_ATTACH_TO_PROCESS 2  
#define ERROR_COULD_NOT_OPEN_FILE 3  
#define ERROR_COULD_NOT_MAP_FILE 4  
#define ERROR_INVALID_FILE 6  
#define ERROR_CORRUPT_FILE 7
```

} エラー処理

# デモ : YARAモジュール

## Demonstration

# **3. Clustering Evaluation**

## **The Rule for Wild Mal-Gopher Families.**

- 実際に観測された「野生の」検体を用いたクラスタリング評価

## 解析済みの検体を用いた評価

- マルウェアと判明した検体のみを分類
- クラスタリングの妥当性と精度を評価

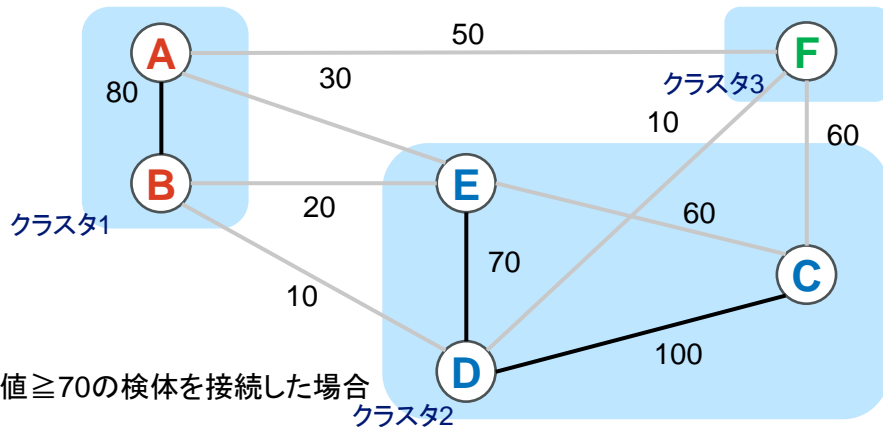


## 未解析の最新の検体を用いた評価

- マルウェアと判明していない検体も含む
- 実際のオペレーションでの利用を評価

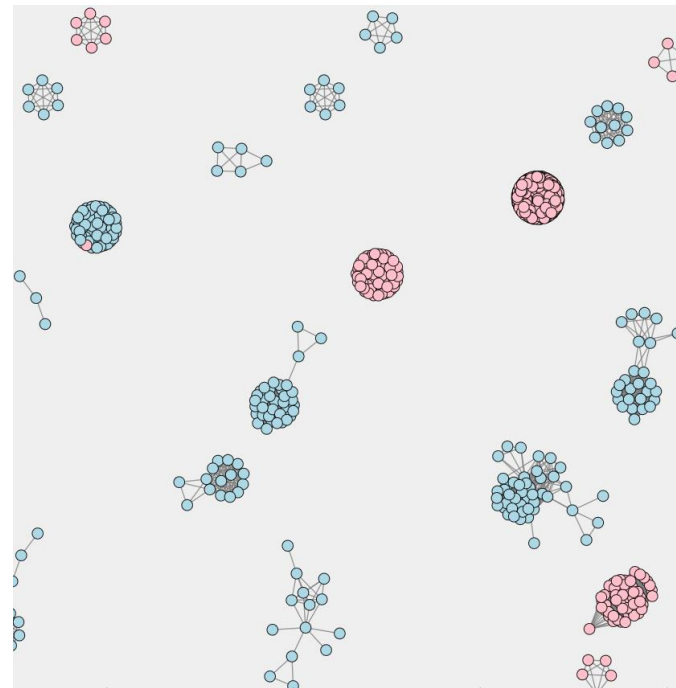
# Overview of Clustering Methods

- gimpfuzzyの類似度に基づくクラスタリング
  - クラスタリングを行う検体のgimpfuzzyを計算
  - クラスタリングを行う検体どうしでgimpfuzzyの類似度を計算
  - 閾値以上の類似度を持つ検体をエッジで接続
  - 接続された検体どうしをクラスタとみなす



閾値  $\geq 70$  の検体を接続した場合

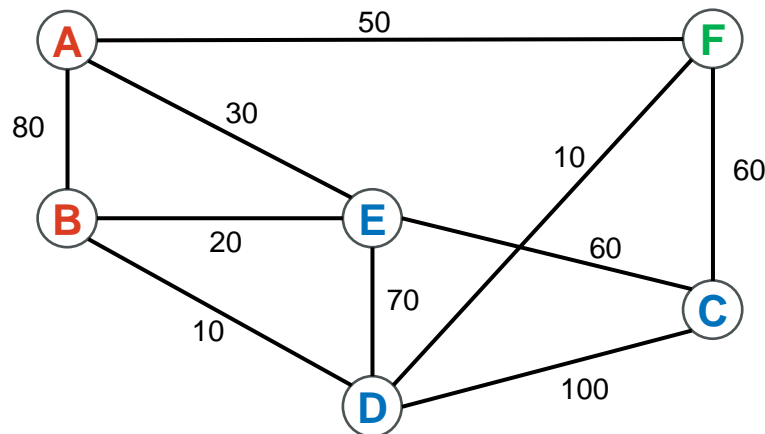
クラスタ2



- 検体のすべての組み合わせに対してスコア を計算
  - 編集距離をベースに、文字列の類似度を0~100までスコアリング
  - 「隣接行列」を作成し、無向グラフを考える

色はファミリーに対応

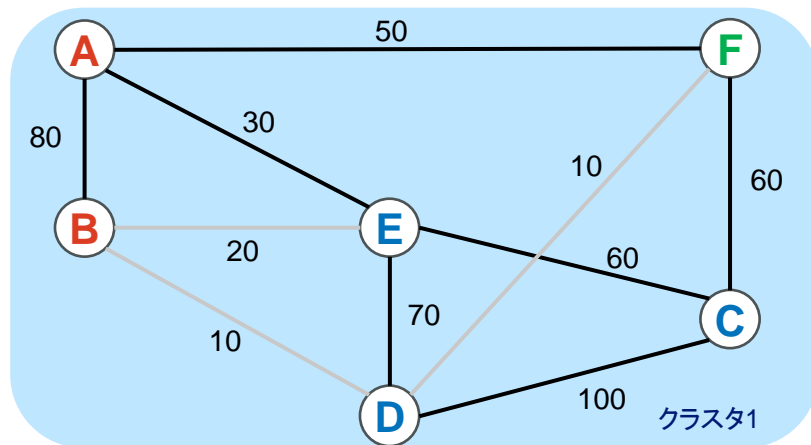
	A	B	C	D	E	F
A	0	80	0	0	30	50
B	80	0	0	10	20	0
C	0	0	0	100	60	60
D	0	10	100	0	70	10
E	30	20	60	70	0	0
F	50	00	60	10	0	0



# How to Evaluate Clustering

- 閾値の切り方: 何をもってよいクラスタリングとするか?

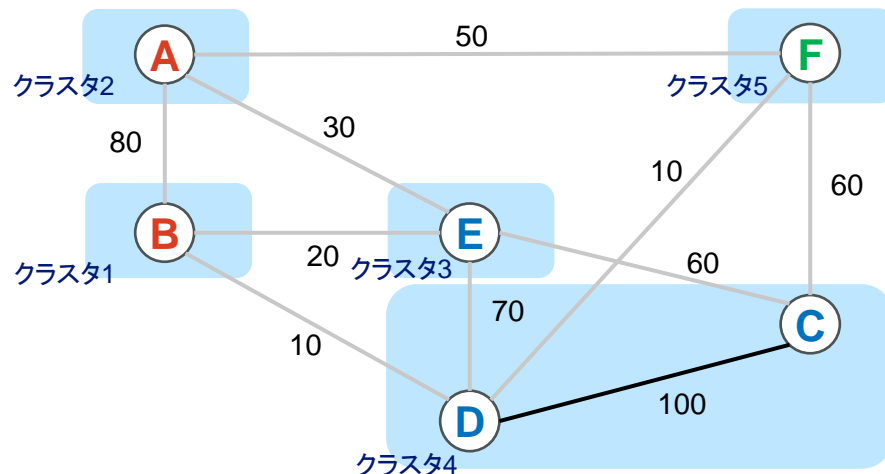
- 閾値が低い場合: 大きな少数のクラスタができる
- 閾値が高い場合: 小さな多数のクラスタができる



閾値  $\geq 30$

◎多くの検体を紐づけられる (完全性↑)

△クラスタ内の分類精度が低下 (同質性↓)



閾値  $\geq 90$

◎クラスタ内の分類精度が高い (同質性↑)

△クラスタが分かれすぎるので意味がなくなる (完全性↓)

- 両者のトレードオフで調和平均を考慮
- scikit-learnに実装されている**V-measure**<sup>[2]</sup>を評価に使用
  - 同質性スコア  $h$  : あるクラスタ内で単一の正解グループが占める割合が多いほど高い
  - 完全性スコア  $c$  : ある正解グループが単一のクラスタに分類されるほど高い

$$V_{\beta} = (1 + \beta) \cdot \frac{h \cdot c}{h + c}$$

$$h = \begin{cases} 1 & \text{if } H(C, K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases} \quad (1)$$

where

$$H(C|K) = - \sum_{k=1}^{|K|} \sum_{c=1}^{|C|} \frac{a_{ck}}{N} \log \frac{a_{ck}}{\sum_{c=1}^{|C|} a_{ck}}$$
$$H(C) = - \sum_{c=1}^{|C|} \frac{\sum_{k=1}^{|K|} a_{ck}}{n} \log \frac{\sum_{k=1}^{|K|} a_{ck}}{n}$$

$$c = \begin{cases} 1 & \text{if } H(K, C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{else} \end{cases} \quad (2)$$

where

$$H(K|C) = - \sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{a_{ck}}{N} \log \frac{a_{ck}}{\sum_{k=1}^{|K|} a_{ck}}$$
$$H(K) = - \sum_{k=1}^{|K|} \frac{\sum_{c=1}^{|C|} a_{ck}}{n} \log \frac{\sum_{c=1}^{|C|} a_{ck}}{n}$$

[2] [https://www.researchgate.net/publication/221012656\\_V-Measure\\_A\\_Conditional\\_Entropy-Based\\_External\\_Cluster\\_Evaluation\\_Measure](https://www.researchgate.net/publication/221012656_V-Measure_A_Conditional_Entropy-Based_External_Cluster_Evaluation_Measure)



- そもそも何を分類の正解とするか？
  - マルウェアファミリの細かい亜種やバージョン違い
  - マルウェアファミリ
  - 大雑把なマルウェアの機能
- paloaltoデータセット<sup>[3]</sup>による評価
  - YARAでファミリを分類した解析済みの検体
  - ファミリ名のつかなかった検体を除外
  - 分類した結果とファミリ名をV-measureで評価

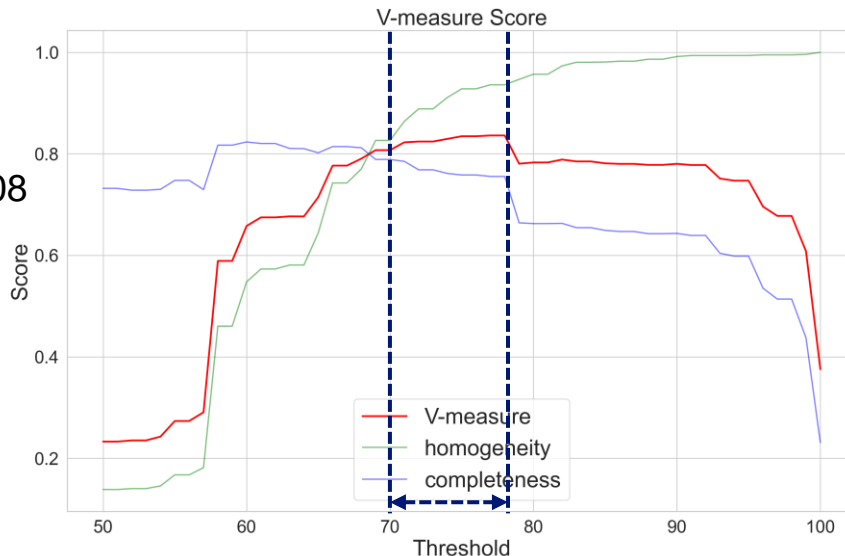
```
1  SHA256,YARA Hit
2  74fc63cfc60f3f9dd3c0d43f59052ba189fa0544ccf79a8fcd99a90ffdb6b0f0e,trojan_golang_hercules
3  99b89e9580af7fc70d8f6ac079358e6b716f7fd242a6547cf2ca932c4ad9c3df,trojan_golang_veil
4  c6d4fb8c4924863d61678df3aba57fe8efa19946f4c5ea678444ec3d7ada0152,trojan_golang_veil
5  f5798d675289fa5b96635635c94562b32c8ddb99ae12ad5af7b56cccd7c35062,N/A
6  738439ade9ae9e9e6d2f2aff3e63f4161722b3149bf7d02902715c127340c676,trojan_golang_veil
7  f25c859b8f2db7f9a7b40d9234885a1c0a8e2b36e091dbb88041f04f1c46c760,trojan_golang_chaos
8  e49125ac24e15a30619f07fe1ebc2dbce3c8137aabc86a88b5f1a57a89d03d5f,trojan_golang_infostealer
9  7a0598927921eb15980ee7d512fc2f20dd697642727eb4a38ba638bf4e7ce902,trojan_golang_goBot2
10 57ca3cb685eef7a1fa40f6bb42946adc3a018f8371d4d57204e98601f08d097d,N/A
11 53ded1467133e8c68c47aba33ea242a1751371031d727e8497a60bb9edb2abd3,trojan_golang_gobrut
12 2ad37fa2946780e99f049b8be7980c6a3483c91ccb3b90506e3fddc629a69039,N/A
13 437e5762c1a814c5934d5d36f1e4a077b14b63be7ffce86999b5503ba34f1aa0,trojan_golang_veil
14 56b110a95c2b16784ba053c693ffcd0bfffce1fd1f42214f71d61b9e0d59b9a42,N/A
15 55f4f5be742d8557956af3278f01825fb02cb90fa2f2f0c1f5160322c26a1af,trojan_golang_veil
16 e15c2dad9d8e9c788cb394aa04d5e070a50512c25ecbe4c5e1e99d69fb52d7ea,trojan_golang_veil
17 722f1c182bac229812107b6ab87853f886ff5c1f96fbdd343dc1847667fb7f79,trojan_golang_veil
18 d69f4caf27097e9a8d7241aa1334fa790d4f5a5708de12a1b8aabd5239724cd8,trojan_golang_veil
19 c680a89e218c74bde438119f9f3112c8725be59956a5fc3e53812165bfe556d2,N/A
```

検体の  
SHA256

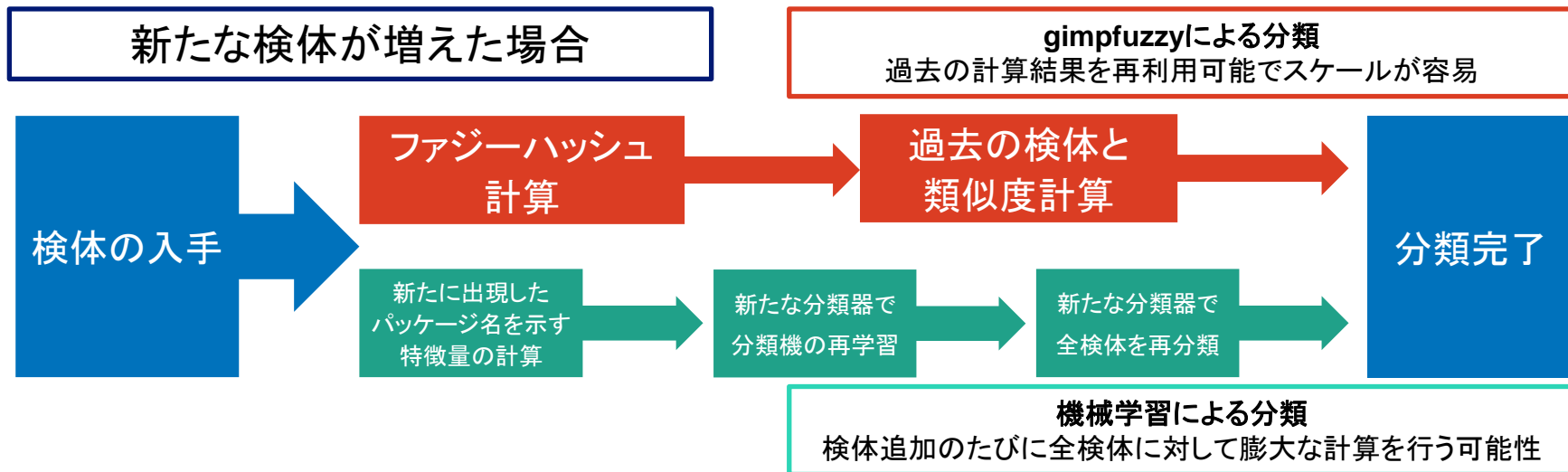
マッチした  
YARAルール

[3][https://github.com/pan-unit42/iocs/blob/master/golang\\_malware\\_results.csv](https://github.com/pan-unit42/iocs/blob/master/golang_malware_results.csv)

- paloaltoデータセット
  - 検体数: 10,700
  - うちVTからダウンロード可能な検体数: 7,088
  - うちファミリー名の表記あり、解析可能な検体: 5,808
- V-measureによる評価
  - ファミリによる分類を正解とした場合の評価
  - 閾値70~80程度で分類精度が最も良い



- 機械学習と比較したgimpfuzzyの類似度に基づくクラスタリングのメリット
  - 計算量が少ない
  - 時間変化の影響を受けにくい



## **4. Applying to “Wild” Binaries**

**The Rule for Wild Mal-Gopher Families.**

- 実際に観測された「野生の」検体を用いたクラスタリング評価

## 解析済みの検体を用いた評価

- マルウェアと判明した検体のみを分類
- クラスタリングの妥当性と精度を評価



## 未解析の最新の検体を用いた評価

- マルウェアと判明していない検体も含む
- 実際のオペレーションでの利用を評価

- VTのRetrohuntで以下のYARAにマッチングしたものをダウンロード
  - golang製のwindowsバイナリにマッチする検体

```
rule go_language_pe
{
  strings:
    $go1 = "go.buildid" ascii wide
    $go2 = "go.buildid¥¥" ascii wide
    $go3 = "Go build ID:" ascii wide
    $go4 = "Go buildinf:"
    $go5 = "runtime.cgo"
    $go6 = "runtime.go"
    $go7 = "GOMAXPRO"
    $str1 = "kernel32.dll" nocase
  condition:
    uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and 2 of ($go*) and all of ($str*)
}
```

- 検体の収集結果



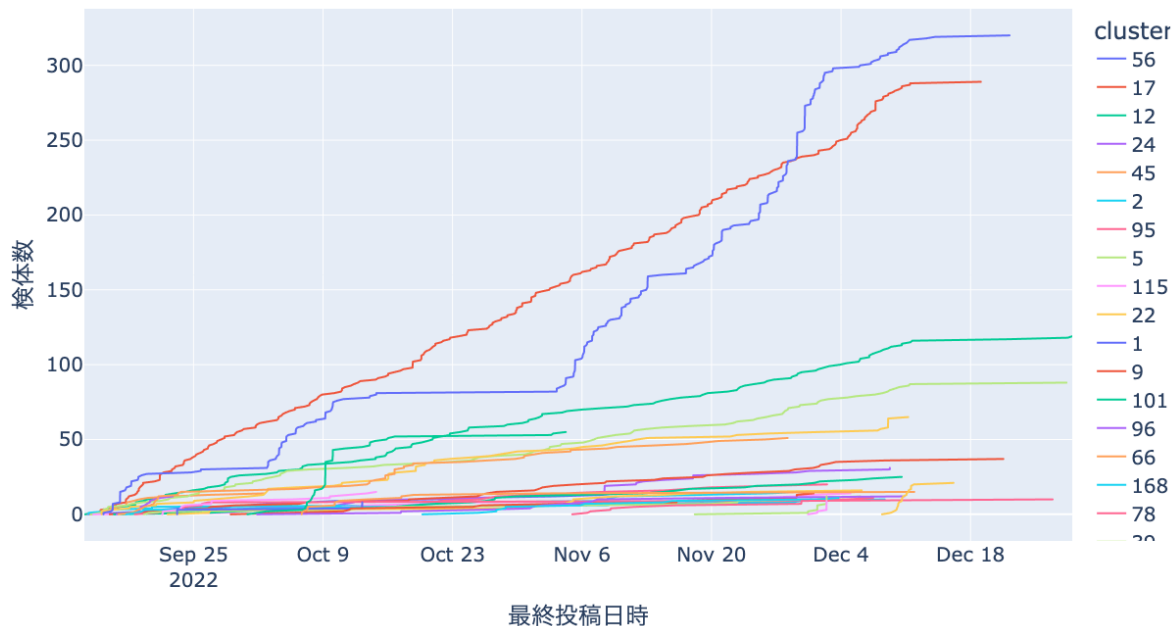
- Retrohuntによって収集された検体  
(2022/12からおよそ過去3か月分)

- Subfileなどの重複を除いてダウンロードした検体

ssdeep入力のバイト数不足の場合  
TLSH等を用いることで改善可能

- pclntabの解析+gimpfuzzy計算が可能だった検体
- UPXはアンパックし解析

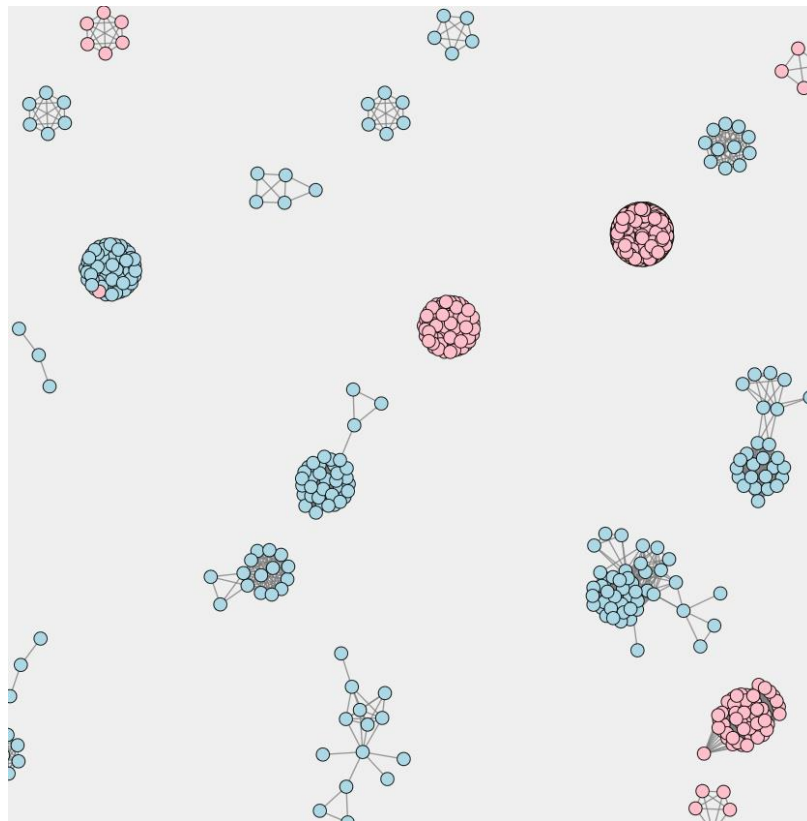
- 2867件の検体に対し1093件のクラスタを作成





# Clustering Result

- クラスタの可視化
  - Pythonのbokeh.ioで実装
  - インタラクティブに「動かせる」グラフを作成
  - グラフの赤色のノードはVTでの悪性判定数10以上の検体を示す



# デモ：クラスタの可視化

## Demonstration

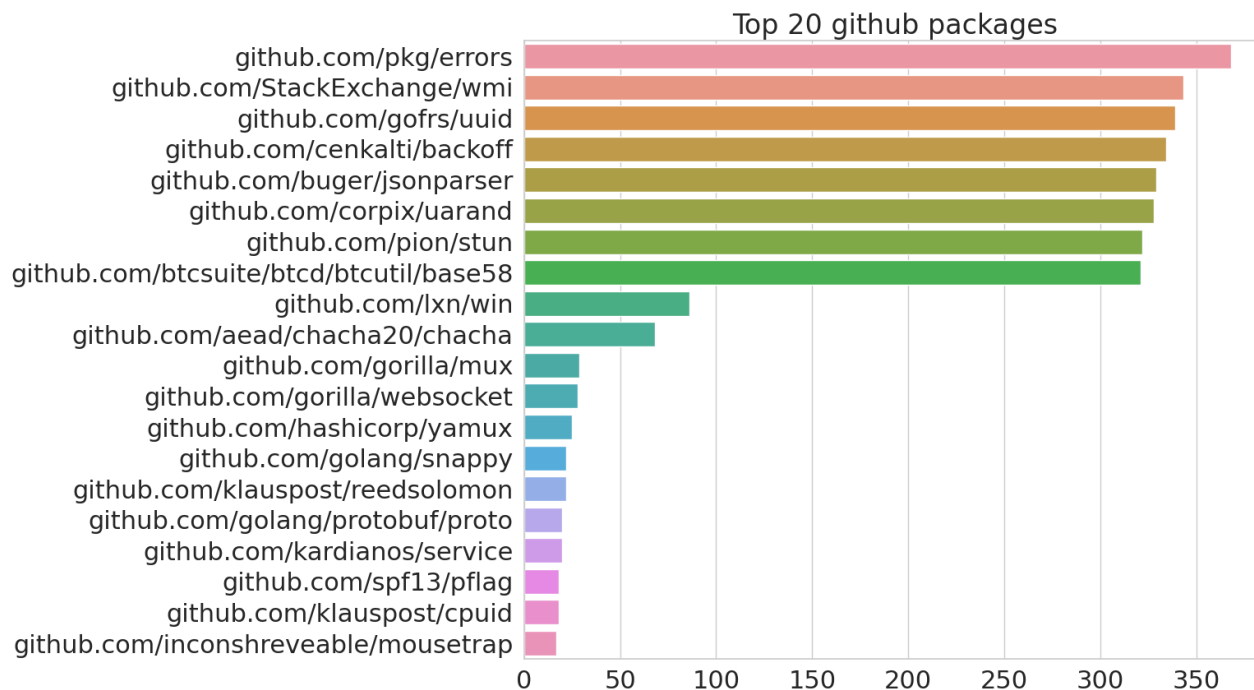
- Golangはパッケージにgithubのリポジトリを指定可能
- 悪性判定10件以上の検体に注目、705件のリポジトリ名を復元
  - プライベートと思われるリポジトリも含む
- 復元された興味深いリポジトリ名
  - ランダムなUAを生成するパッケージ (corpix/uarand等)
  - Process Invokingを行うパッケージ (inconshreveable/mousetrap等)
  - マルウェア作成フレームワーク (tiagorlampert/CHAOS等)
  - マルチホッププロキシ (Dliv3/Venom等)
  - Post-Exploitationフレームワーク (Ne0nd0g/merlin等)

# Case Study① : GitHub Packages



Security Holdings

## 悪性検体に出現したgithubのリポジトリTop20

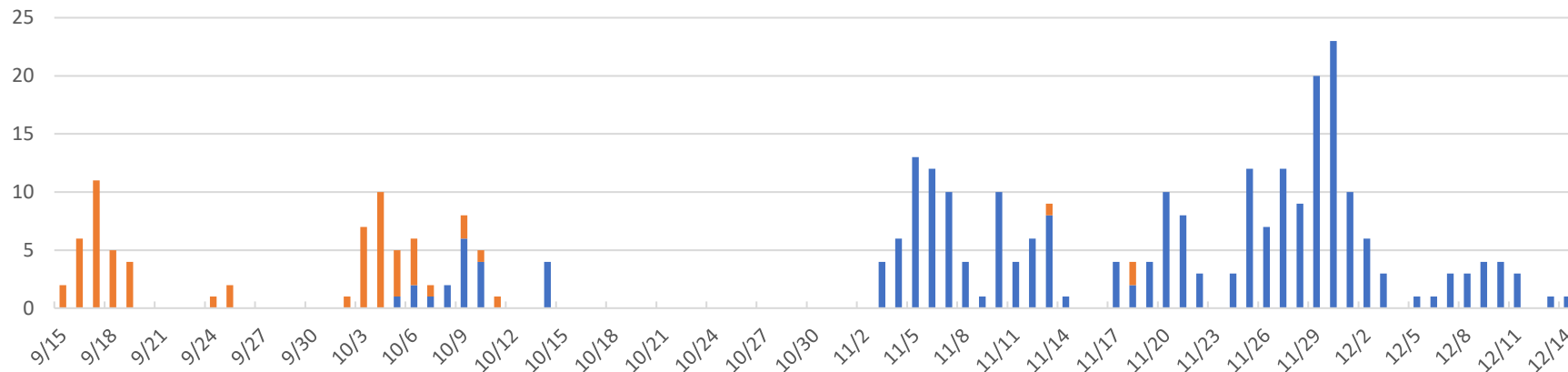


# Case Study②：マルウェアの機能追加を検出



Security Holdings

- 悪性検体クラスでGimpFuzzy値のわずかな変化を検出し時間的变化を観測
  - 768:KZZ99PdnRrLXT3UbhHPBj/RqJgvm+HHyScP0OhZIXCPINvxtWrX7G/VAmWeEX:iZXJRrLXT3iNGKINvxtWrX7G/VAmWeEX
  - 768:KZZ99PdnRrLXT3UbhHPBj/RqJgvm+HHyScP0OhZIXCPINvxtWrX7G/V3mWeEX:iZXJRrLXT3iNGKINvxtWrX7G/V3mWeEX



- 変更された関数はロジックも変更されていた

- 変更されていた関数名

- 768:~VAmWeEX






→ application/pesignaturetest/wincert.GetPostalCode

- 768:~V3mWeEX

→ application/pesignaturetest/wincert.Extract



- 関数名が変更されていない関数の中にもロジックが変更されているものがある

	Similarity %	Confidence	Address	Primary Name	Type	Address	Secondary Name	Type	Basic Blocks			Jumps		
	0.26	0.98	006A4080	main_reportInstallFailure	No...	006A3DC0	main_reportInstallFailure	No...	0	10	58	4	9	84
	0.45	0.97	0069DA10	main_getCampaignID	No...	0069D790	main_getCampaignID	No...	11	3	2	15	3	2
	0.81	0.96	0067B610	application_pesignature...	Normal	0067B610	application_pesignature...	Normal	7	39	0	22	32	13
	0.86	0.97	006B01B0	main_extractDistributor...	Normal	006B05E0	main_extractDistributor...	Normal	0	10	2	1	11	3
	0.91	0.99	006943D0	main_initializeConfig	Normal	006941D0	main_initializeConfig	Normal	0	21	3	2	22	6

- main\_reportInstallFailureは通信機能の追加が行われている

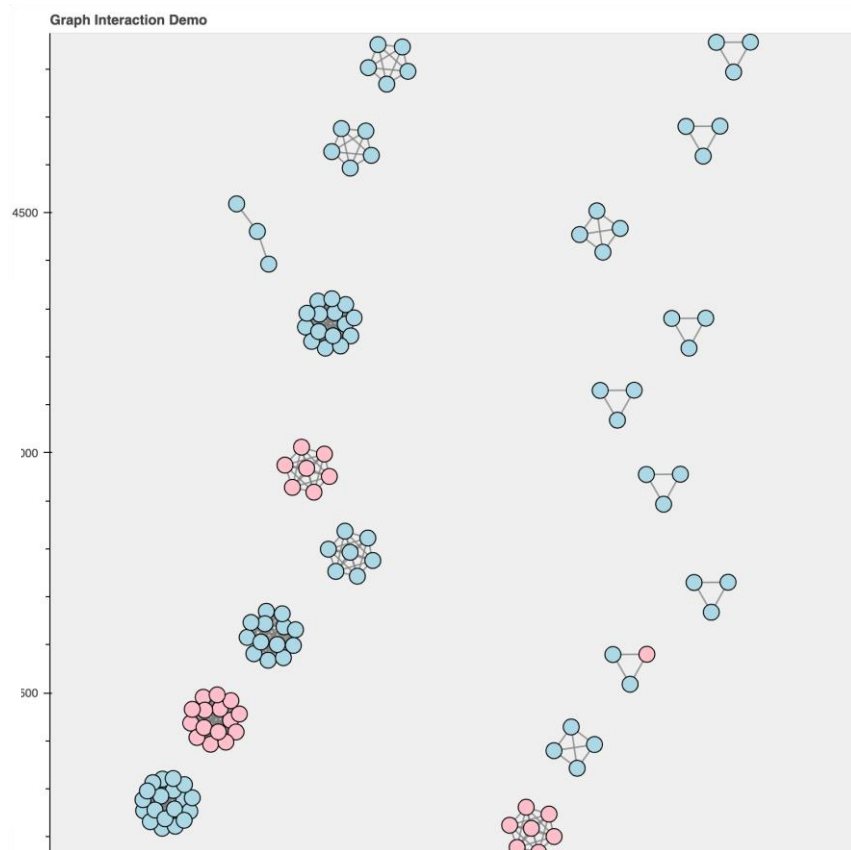
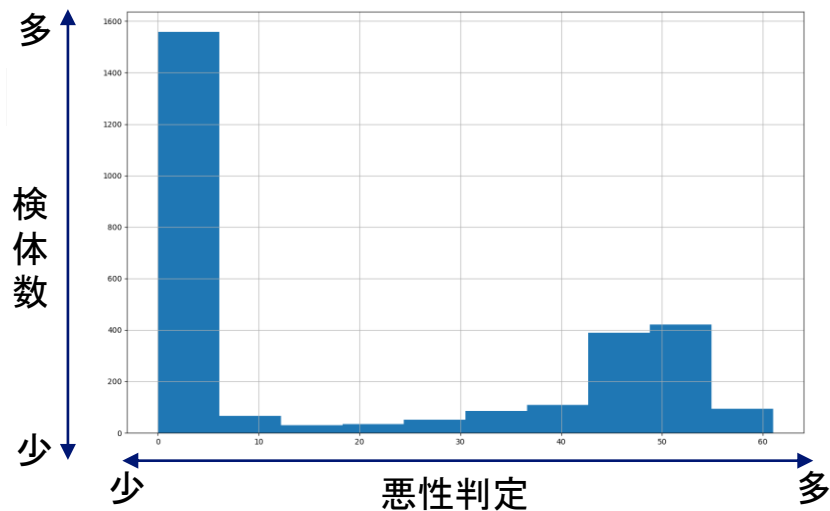
```

234  v42 = (http_Request *)net_http_NewRequestWithContext(
235      (int)&go_itab_ptr_context_emptyCtx_comma_ptr_context_Context,
236      dword_C76630,
237      (int)"POSTQEMU",
238      4,
239      (int)"https://fulusus.com/api/install-failure",
240      39,
241      v39,
242      v40);
243  if ( !v43 )
244  {
245      Header = (runtime_hmap *)v42->Header;
246      v56 = net_textproto_CanonicalMIMEHeaderKey((int)"Content-Type", 12);
247      v55 = (_DWORD *)runtime_newobject((int)&RTYPE_1_string);
248      v55[1] = 33;
249      *v55 = "application/x-www-form-urlencoded";

```

# Case Study③ : Legitimate Files

- 現実では正規ファイルが圧倒的に多い
  - VTに投稿された検体においても同様
  - 正規ファイルのクラスタリングも可能なのか？

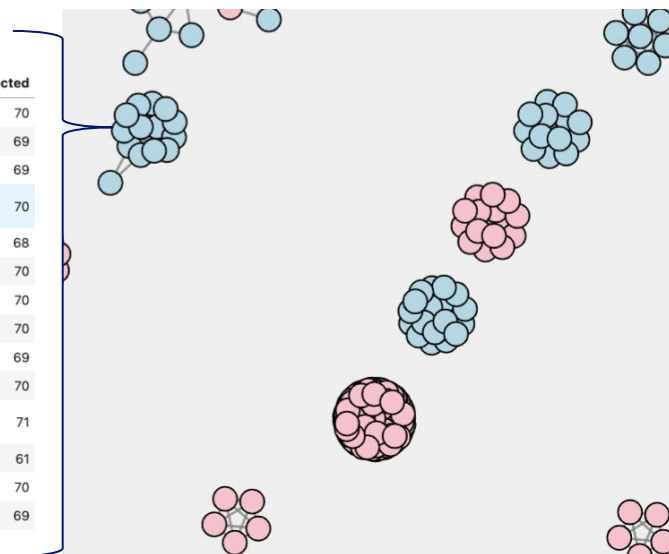




# Case Study③ : Legitimate Files

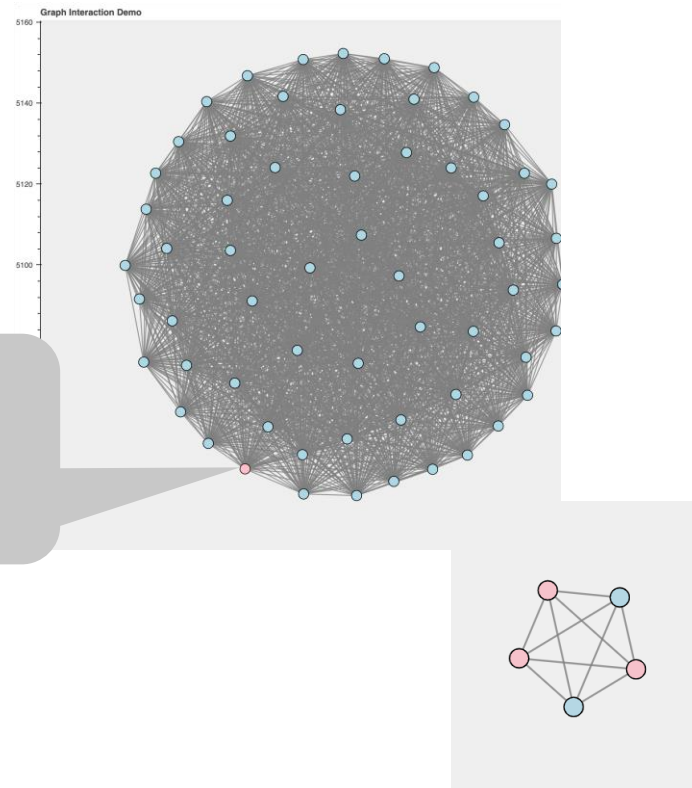
- 正規ファイルと思しき検体でもクラスタができる
  - VTに投稿されている検体は悪性ファイルのみであるとは限らない

	gimpfuzzy	cluster	last_submission_date	size	type_description	malicious	undetected
3072:ZVZoThQpAM+mBL+5CR61yLR2Mr8tA4ICAluEbXDbK...		106	2022-08-23 17:29:28	27937584	Win32 EXE	0	70
3072:ZVZoThQpAM+mBL+5CR61yLR2Mr8uA4ICAluybXDbK...		106	2022-08-29 10:48:57	27937072	Win32 EXE	0	69
3072:ZVZoThQpAM+mBL+5CR61yLR2Mr8uA4ICAluhbXDbK...		106	2022-09-02 02:07:33	27938096	Win32 EXE	0	69
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-05 12:12:53	27936048	Win32 EXE	0	70
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-06 05:52:38	27913216	Win32 EXE	2	68
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-08 11:21:42	27940144	Win32 EXE	0	70
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-14 07:23:13	27940144	Win32 EXE	0	70
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-16 09:19:15	27940136	Win32 EXE	0	70
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-16 13:15:25	27940136	Win32 EXE	0	69
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-18 12:38:06	27936048	Win32 EXE	0	70
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-19 07:44:26	27940144	Win32 EXE	0	71
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-19 17:28:46	27940128	Win32 EXE	0	61
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-21 05:35:09	27940128	Win32 EXE	0	70
3072:ZVZoThQpAM+zBL+5CR61yLR2Mr8uA4ICAluwbXRb1...		106	2022-09-28 16:48:43	27917312	Win32 EXE	1	69



# Case Study④ : Floxif

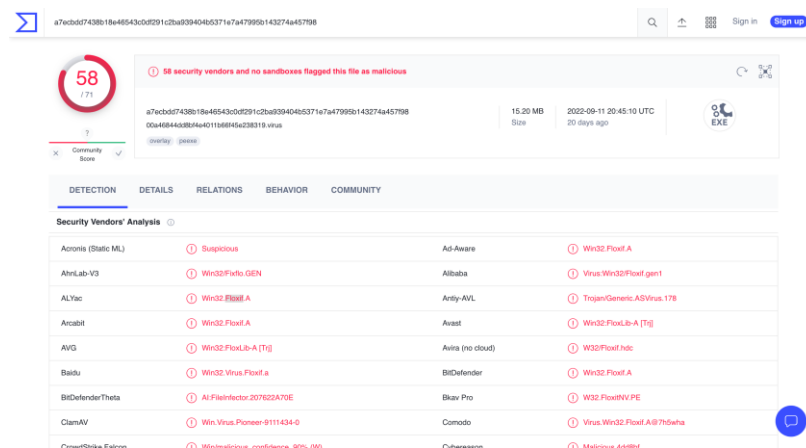
- 悪性 / 良性判定が混合するクラスター
  - gimpfuzzyが類似しているものの  
悪性判定がクラスター内で大きく異なる



# Case Study④ : Floxif

- 正規ファイルクラスタの中に潜む悪性度の高い検体
  - 以下のプログラムに擬態した悪性度の高いFloxifの検体を発見
    - psiphone-tunnel-core
    - Acronis Cyber Protect
  - 教師なしでのクラスタリングのみでは悪性/良性判定は難しい検体が存在
  - 検体のプロファイリングとしては正しい結果

4	2022-09-09 20:21:30	15857968	Win32 EXE	0	63
4	2022-09-10 08:11:54	15857968	Win32 EXE	0	70
4	2022-09-10 10:48:23	15857968	Win32 EXE	0	70
4	2022-09-10 13:50:39	15936247	Win32 EXE	58	13
4	2022-09-11 18:54:17	15857968	Win32 EXE	0	70
4	2022-09-12 07:56:33	15857968	Win32 EXE	0	70
4	2022-09-12 14:09:19	15857968	Win32 EXE	0	65
4	2022-09-12 21:44:51	16389424	Win32 EXE	0	70
4	2022-09-13 04:48:15	15857968	Win32 EXE	0	70
4	2022-09-13 19:51:55	15857968	Win32 EXE	0	70
4	2022-09-13 22:13:20	15857968	Win32 EXE	0	70



- gimpfuzzyを計算できない検体の存在
  - ssdeepの入力サイズの下限が存在 (>4KB)
    - › TLSHなどで置き換えが可能
  - パッキング、難読化などによる解析妨害
- 「教師なし」での分類の限界
  - 悪性 / 良性判定は難しい
  - クラスタによりある程度は分かれるが、混ざってしまうものも存在

# **5. Conclusions**

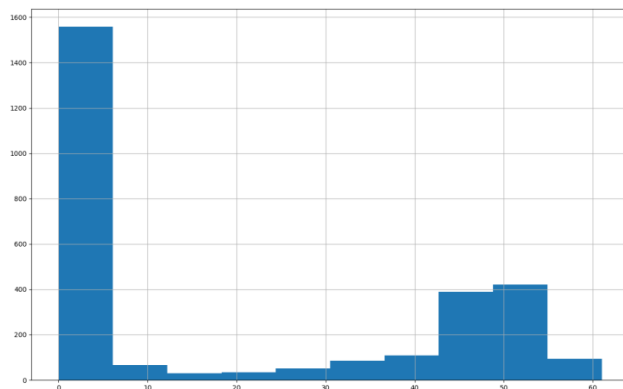
**The Rule for Wild Mal-Gopher Families.**

- gimpfuzzyを実際のオペレーションに応用するために以下の内容について発表
  - YARAモジュールの実装
  - 解析済みの検体を用いた精度評価
  - VirusTotalに投稿された検体への適用
- YARAモジュール及び可視化スクリプトを公開予定

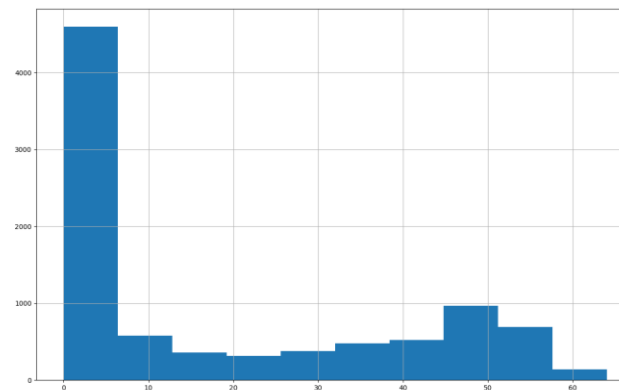
# **Appendix**

## **The Rule for Wild Mal-Gopher Families.**

- VTで収集した検体の悪性判定数
  - 圧倒的に悪性判定の少ないファイルが多い



解析済み検体：  
2835検体

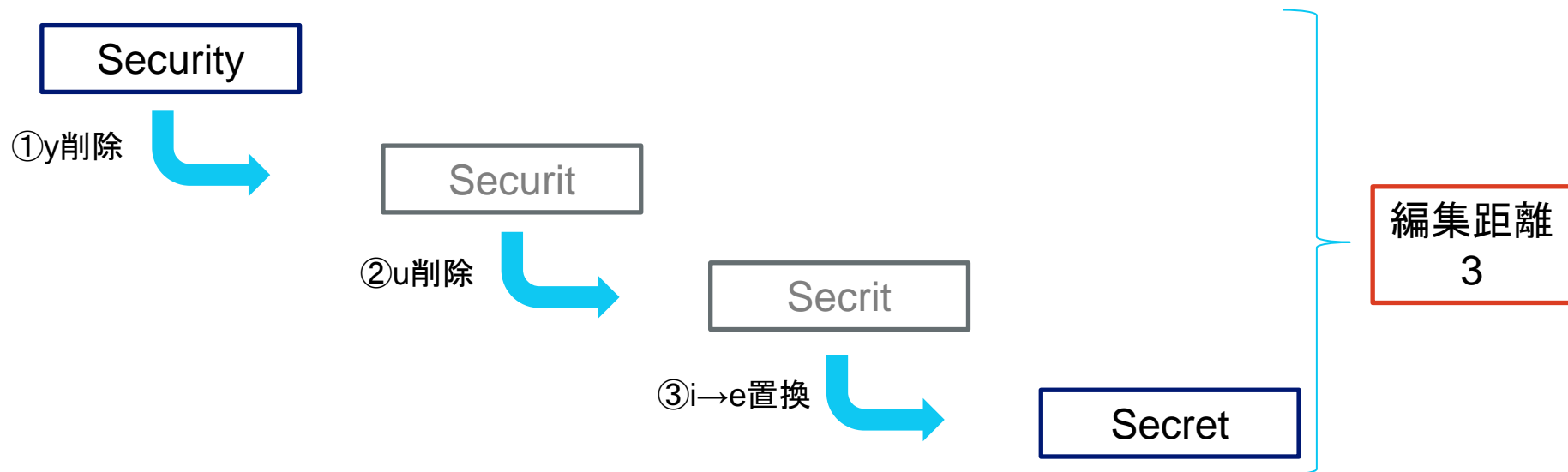


未解析の検体：  
9999検体



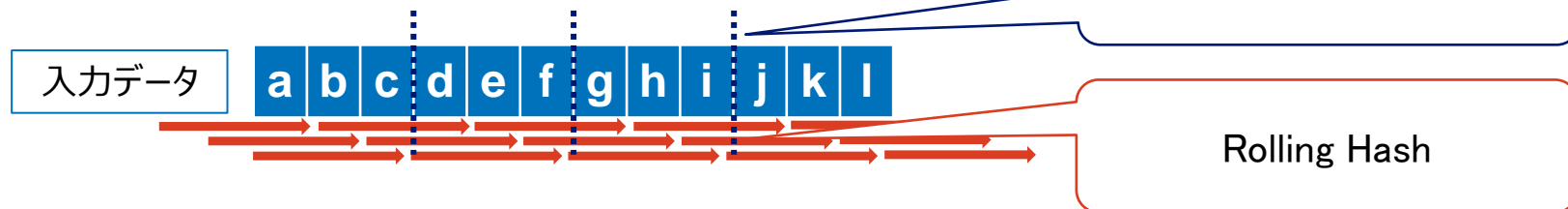
- 編集距離(レーベンシュタイン距離)

- 文字列の類似度を示す原始的な手法
- 1文字の挿入・削除・置換によって一方の文字列をもう片方の文字列に変換できる最小の回数



- **ssdeep (Context Triggered Piecewise Hashing)**

- Piecewise Hashing : 分割した部分データのハッシュ
- Rolling Hash : 固定長の部分データに対するハッシュ



- Rolling Hashが一定の値になった時に、そこで分割してPiecewise Hashingを行う
  - トリガーとなる値は入力データ長に応じて計算

