RISING 瑞星

MALWARE DETECTION
based on
MACHINE LEARNING

Application and practice of machine learning in anti-malware

Ye Chao
Beijing Rising

Experience

2012

x86 Instruction Flow based Predictor

To d

“ ACK”

OBSOLETED!

PDF Exploit Predictor

To de

what

contains javascript

OBSOLETED!

2012

Malware Predictor based on Decision-Tree

For Windows PE

Millions of tr s

Featu file structure

the Decision-Tree

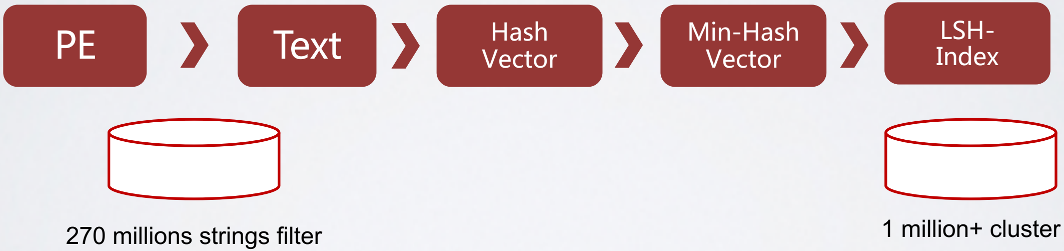
OBSOLETED!

2013

Min-Hash & LSH based Clustering

find similar historical samples quickly and fall into one cluster

always select the latest sample to represent the cluster



RISING 瑞星

2016

RDM+
malware predictor based on Random-Forest

For Windows PE
Tens of millions of training samples
Features are extracted from file structure/content/analysis
Use the Random Forest

RDM+

- A cautious predictor for malware detection
- It relies on file structure and part of the content
- It doesn't look so smart, but it improves through high frequency learning.

Feature Engineering

It is often said that

“In the application of machine learning, the feature engineering determines the upper limit of the model and algorithm performance.”

4778-D
Features Array

For RDM+

describes a file from multiple aspects
from file content and file analysis results

Program Structure and Properties

Section Table Analysis

Entropy

'Size' Fields

Compiler

Relative Position of Important Data

……

Import/Export Symbol Names

Embody the intent of the program

An algorithm called IMPHASH is widely used in malware classification

Hash Trick

there is no need to create an encoding for each name
count the names by name hash

1024slots

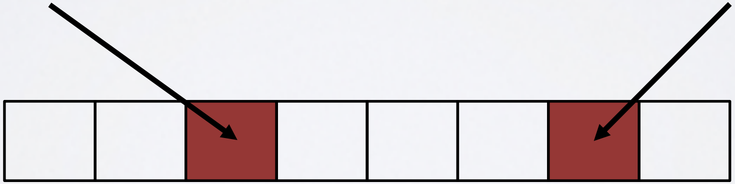
For IMPORT names

1024slots

For EXPORT names

hash(CreateFileA)%1024

hash(setGlobalCallBack)%1024



Instructions Started from Entry Point and Export Functions

1102

OPCODES

Frequently used instructions are grouped, others are completely reserved.

117

OPERAND-TYPES

In the obfuscated code, both the immediate number and the register are heavily used.

Strings in Section-Tables/Resources/Signature

Use "Alnum" table

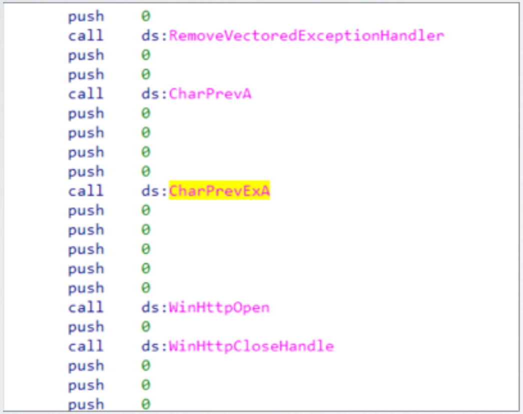
“Micorsoft Windows ”

M	i	c	r	o	s	f	t	W	n	d	w
1	2	1	3	3	2	1	1	1	1	1	1

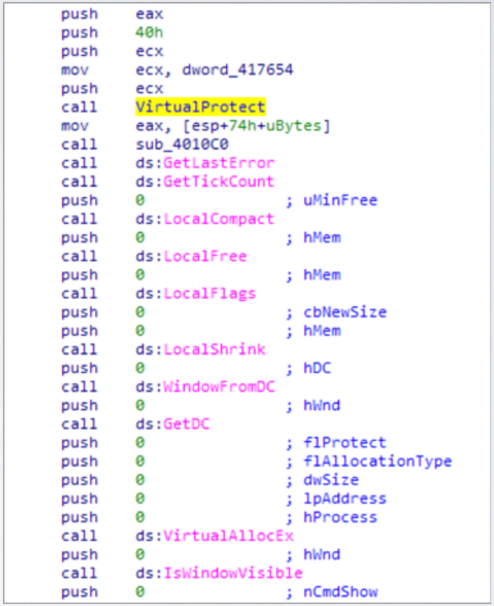
Features from Analysis

CASE 1

Insert many fake API calls in code to avoid the detection of some antivirus software, such as: Injector, Loader, Kryptik, XPACK, Crypter



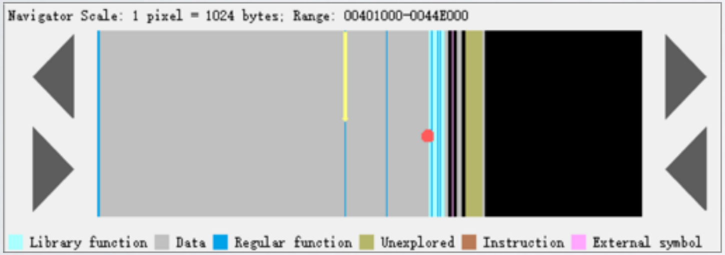
```
push    0
call    ds:RemoveVectoredExceptionHandler
push    0
push    0
call    ds:CharPrevA
push    0
push    0
push    0
push    0
call    ds:CharPrevExA
push    0
push    0
push    0
push    0
push    0
call    ds:WinHttpOpen
push    0
call    ds:WinHttpCloseHandle
push    0
push    0
push    0
```



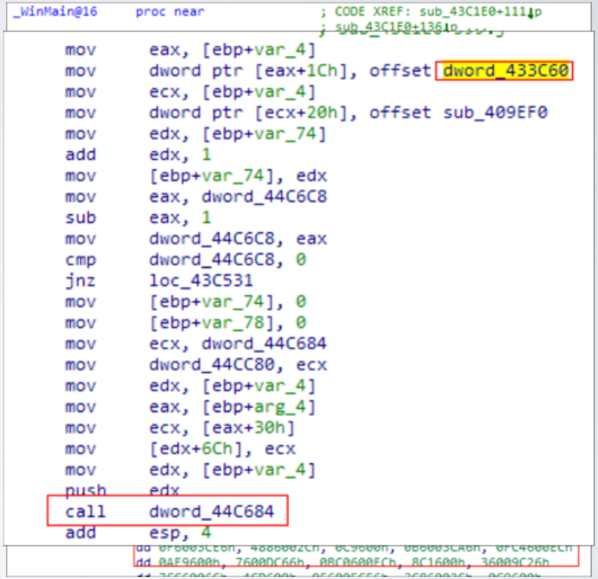
```
push    eax
push    40h
push    ecx
mov     ecx, dword_417654
push    ecx
call    VirtualProtect
mov     eax, [esp+74h+uBytes]
call    sub_4010C0
call    ds:GetLastError
call    ds:GetTickCount
push    0               ; uMinFree
call    ds:LocalCompact
push    0               ; hMem
call    ds:LocalFree
push    0               ; hMem
call    ds:LocalFlags
push    0               ; cbNewSize
push    0               ; hMem
call    ds:LocalShrink
push    0               ; hDC
call    ds:WindowFromDC
push    0               ; hWnd
call    ds:GetDC
push    0               ; flProtect
push    0               ; flAllocationType
push    0               ; dwSize
push    0               ; lpAddress
push    0               ; hProcess
call    ds:VirtualAllocEx
push    0               ; hWnd
call    ds:IsWindowVisible
push    0               ; nCmdShow
```

CASE 2

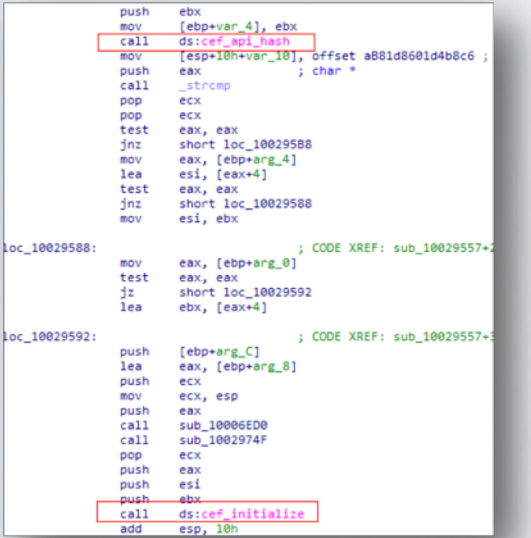
The program is compiled by the ordinary compiler,but there is a lot of high entropy data in the code. After execution, the data is decoded into code and executed, such as :Injector, Loader, Kryptik, Crypter


Navigator Scale: 1 pixel = 1024 bytes; Range: 00401000-0044E000

CDD85B79900FC8FB82768808576A8F38
Malware.XPACK-HIE/Heur!1.9C48

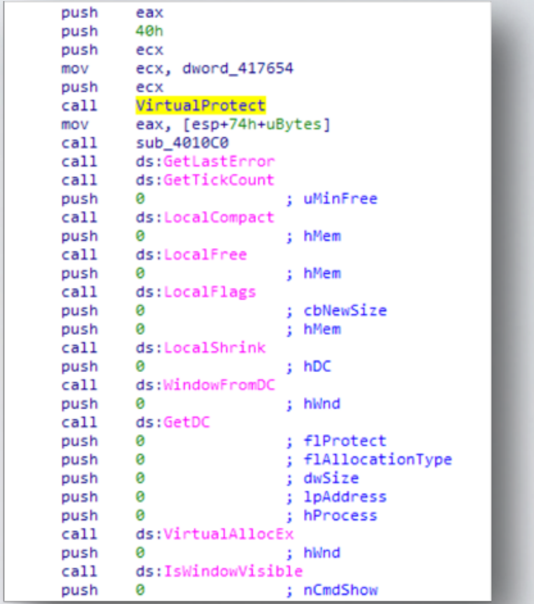

```
_WinMain@16     proc near                ; CODE XREF: sub_43C1E0+111↓p
                                         ; sub_43C1E0+136↓p ...
        mov     eax, [ebp+var_4]
        mov     dword ptr [eax+1Ch], offset dword_433C60
        mov     ecx, [ebp+var_4]
        mov     dword ptr [ecx+20h], offset sub_409EF0
        mov     edx, [ebp+var_74]
        add     edx, 1
        mov     [ebp+var_74], edx
        mov     eax, dword_44C6C8
        sub     eax, 1
        mov     dword_44C6C8, eax
        cmp     dword_44C6C8, 0
        jnz     loc_43C531
        mov     [ebp+var_74], 0
        mov     [ebp+var_78], 0
        mov     ecx, dword_44C684
        mov     dword_44CC80, ecx
        mov     edx, [ebp+var_4]
        mov     eax, [ebp+arg_4]
        mov     ecx, [eax+30h]
        mov     [edx+6Ch], ecx
        mov     edx, [ebp+var_4]
        push    edx
        call    dword_44C684
        add     esp, 4
                dd 0F6003CE6h, 4886002Ch, 0C9600h, 0B6003CA6h, 0FC4600ECh
                dd 0AF9600h, 7600DC66h, 0BC0600ECh, 8C1600h, 36009C26h
```

Symbols distribution is sparse in clean program



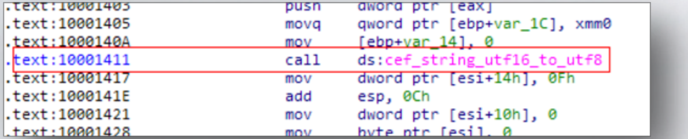
```
                     push    ebx
                     mov     [ebp+var_4], ebx
                     call    ds:cef_api_hash
                     mov     [esp+10h+var_10], offset aB81d8601d4b8c6 ;
                     push    eax             ; char *
                     call    _strcmp
                     pop     ecx
                     pop     ecx
                     test    eax, eax
                     jnz     short loc_100295B8
                     mov     eax, [ebp+arg_4]
                     lea     esi, [eax+4]
                     test    eax, eax
                     jnz     short loc_10029588
                     mov     esi, ebx

loc_10029588:                                ; CODE XREF: sub_10029557+2
                     mov     eax, [ebp+arg_0]
                     test    eax, eax
                     jz      short loc_10029592
                     lea     ebx, [eax+4]

loc_10029592:                                ; CODE XREF: sub_10029557+3
                     push    [ebp+arg_C]
                     lea     eax, [ebp+arg_8]
                     push    ecx
                     mov     ecx, esp
                     push    eax
                     call    sub_10006ED0
                     call    sub_1002974F
                     pop     ecx
                     push    eax
                     push    esi
                     push    ebx
                     call    ds:cef_initialize
                     add     esp, 10h
```

Symbols is densely distributed in some malware

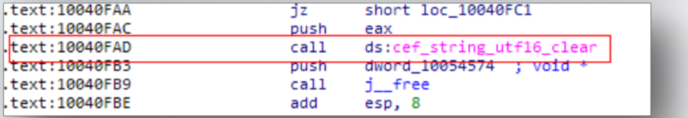


```
        push    eax
        push    40h
        push    ecx
        mov     ecx, dword_417654
        push    ecx
        call    VirtualProtect
        mov     eax, [esp+74h+uBytes]
        call    sub_4010C0
        call    ds:GetLastError
        call    ds:GetTickCount
        push    0               ; uMinFree
        call    ds:LocalCompact
        push    0               ; hMem
        call    ds:LocalFree
        push    0               ; hMem
        call    ds:LocalFlags
        push    0               ; cbNewSize
        push    0               ; hMem
        call    ds:LocalShrink
        push    0               ; hDC
        call    ds:WindowFromDC
        push    0               ; hWnd
        call    ds:GetDC
        push    0               ; flProtect
        push    0               ; flAllocationType
        push    0               ; dwSize
        push    0               ; lpAddress
        push    0               ; hProcess
        call    ds:VirtualAllocEx
        push    0               ; hWnd
        call    ds:IsWindowVisible
        push    0               ; nCmdShow
```

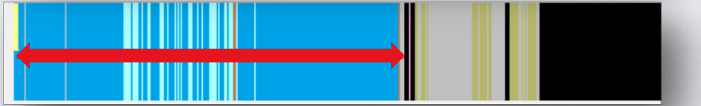
The code between the first symbol and the last symbol almost fills the entire code section



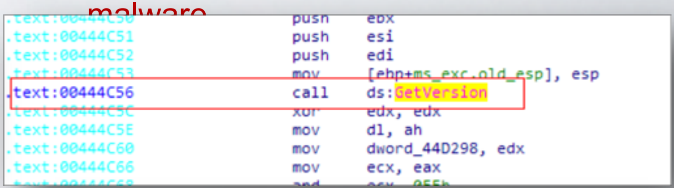
```
.text:10001403                 push    dword ptr [eax]
.text:10001405                 movq    qword ptr [ebp+var_1C], xmm0
.text:1000140A                 mov     [ebp+var_14], 0
.text:10001411                 call    ds:cef_string_utf16_to_utf8
.text:10001417                 mov     dword ptr [esi+14h], 0Fh
.text:1000141E                 add     esp, 0Ch
.text:10001421                 mov     dword ptr [esi+10h], 0
.text:10001428                 mov     byte ptr [esi], 0
```



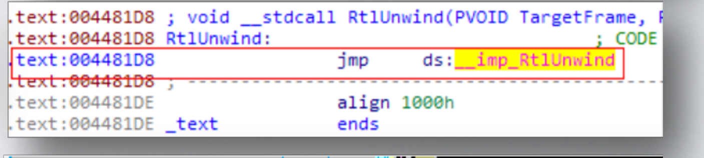
```
.text:10040FAA                 jz      short loc_10040FC1
.text:10040FAC                 push    eax
.text:10040FAD                 call    ds:cef_string_utf16_clear
.text:10040FB3                 push    dword_10054574  ; void *
.text:10040FB9                 call    j__free
.text:10040FBE                 add     esp, 8
```



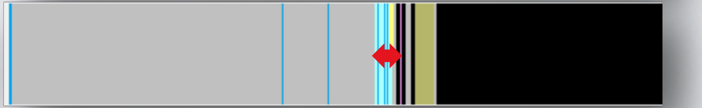
Very little code between the first and last symbols in some malware



```
.text:00444C50                 push    ebx
.text:00444C51                 push    esi
.text:00444C52                 push    edi
.text:00444C53                 mov     [ebp+ms_exc.old_esp], esp
.text:00444C56                 call    ds:GetVersion
.text:00444C5C                 xor     edx, edx
.text:00444C5E                 mov     dl, ah
.text:00444C60                 mov     dword_44D298, edx
.text:00444C66                 mov     ecx, eax
.text:00444C68                 and     ecx, 0FFh
```



```
.text:004481D8 ; void __stdcall RtlUnwind(PVOID TargetFrame, P
.text:004481D8 RtlUnwind:                              ; CODE
.text:004481D8                 jmp     ds:__imp_RtlUnwind
.text:004481D8 ; ---------------------------------------------
.text:004481DE                 align 1000h
.text:004481DE _text           ends
```



Features List

ISRR: imported symbols referenced ratio.

ISCR: imported symbols invoked ratio.

ILRR: imported libraries referenced ratio.

ISDD(Max/Min): the density of symbols distribution in file.

RPOS1: the offset of first symbol divided by the section size.

EDCR: the compression rate of the executable data in program.

IBR: the ratio of branch instructions to total instructions (200).

IDR: to measure whether an instruction can be statically tracked.

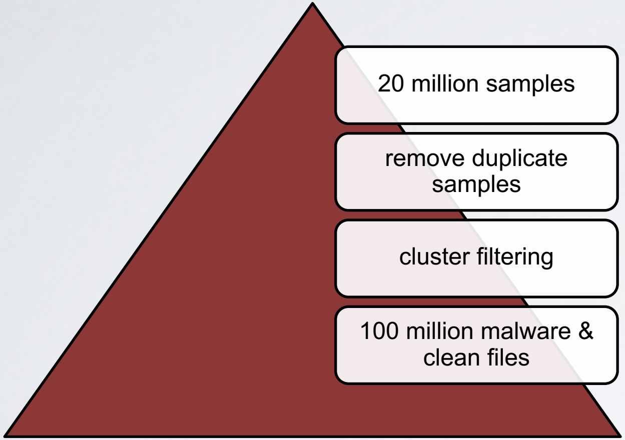
DER: how many export symbols are in the data segment.

BSR: the ratio of BSS section size to image size.

MSGR: the ratio of the maximum size between two symbols and the code section size.

Model Training and Combination

Training Samples Set



20 million samples

remove duplicate samples

cluster filtering

100 million malware & clean files

~700G

actual number of bytes

Algorithm Selection

SVM

- Not suitable for a large number of samples
- Unable to complete training

Random-Forest

- Good effect on training set
- Key features can be found
- The training process is long

Decision-Tree

- Under-fitting
- The output is too simple to concatenate

Model Combination

4778-D

100 Trees

in forest

Takes 120+ Hours

Unable to meet the hourly update

Model for **Prediction**

Model for dimensionality reduction

Model for Dimensionality Reduction	Model for Prediction
4778-D input	100-D input
100-D output	100-D output
Dimensionality reduction tool	Prediction tool
Updated every few months	Hourly update

After dimensionality reduction, the training difficulty is greatly reduced.

Prediction Model Training

Basic Samples & Latest Samples

BS: A set of historical samples after filtering and dimensionality reduction

+

LS: Recent major malware and clean files set, includes FPs

=

5 million samples

covering about 50 million files

Prediction Model Training Time

0.78 hour

✓ Hourly update

✓ Model fine-tuning

Mitigating false positives

Missing malware is better than false positives!

How do we do that?

Choosing the right algorithm

In order to mitigate the false positives, we think that over-fitting is the advantage.

Masking false positives using hash value of features

In a production environment, the key-value database is used to mask false positives before predictions

Carefully selected training samples

Select the right malware files and more clean files into the training set

The cloud service

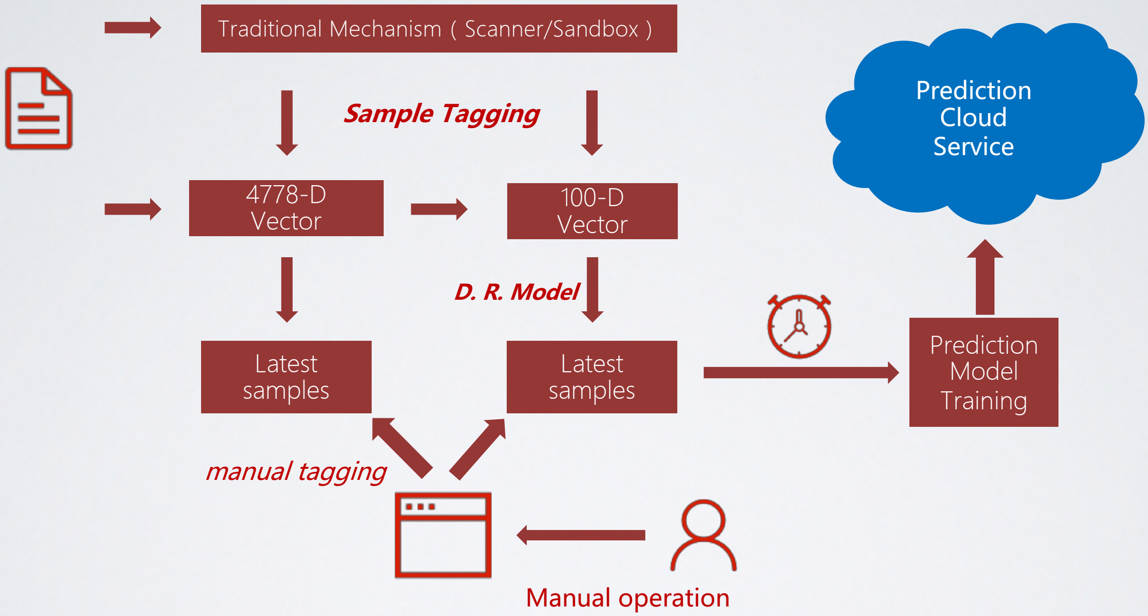
Compensating for model defects

Random-Forest cause the "model explosion" problem, making the model unsuitable for distribution to the host.

Requires high frequency updates

One is to maintain the most timely training and update, the second is to maintain timely false positives removal.

Operation Process



Traditional Mechanism (Scanner/Sandbox)

Sample Tagging

4778-D Vector

100-D Vector

D. R. Model

Latest samples

Latest samples

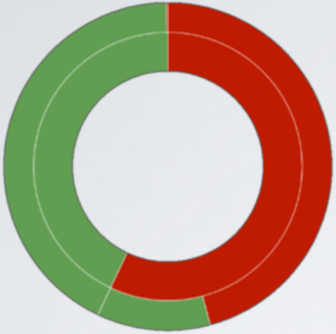
manual tagging

Manual operation

Prediction Model Training

Prediction Cloud Service

Performance



in the 1st month

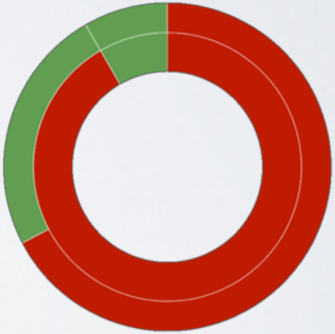
80~90% Positives

~0.2% FPs

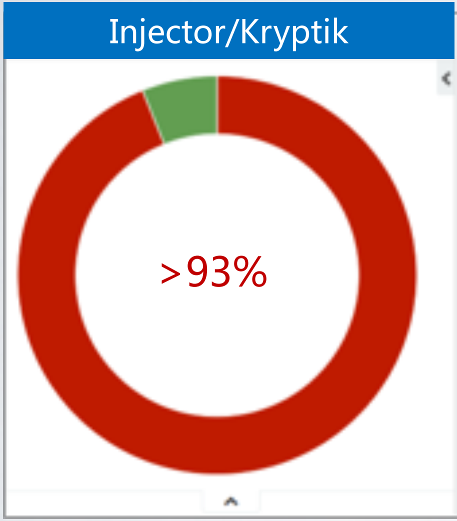
60~70% Positives

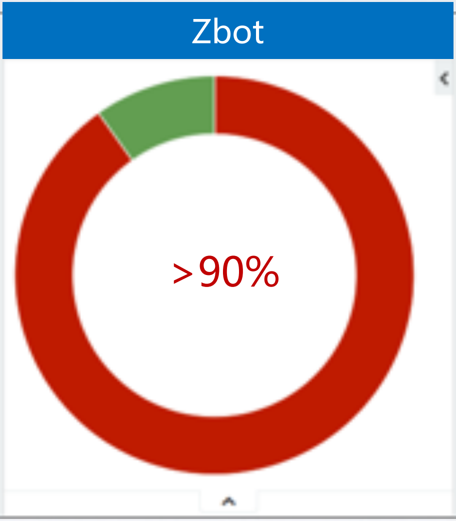
<0.1% FPs

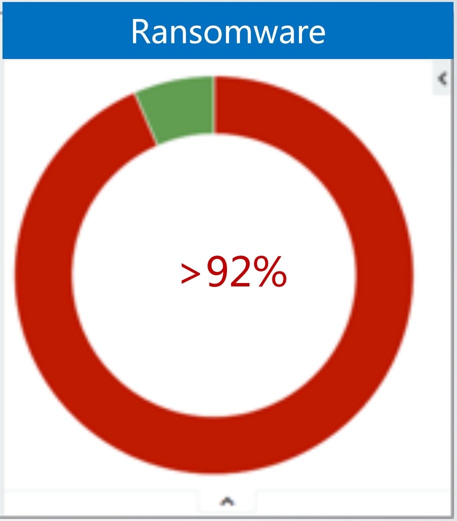
after 3 months

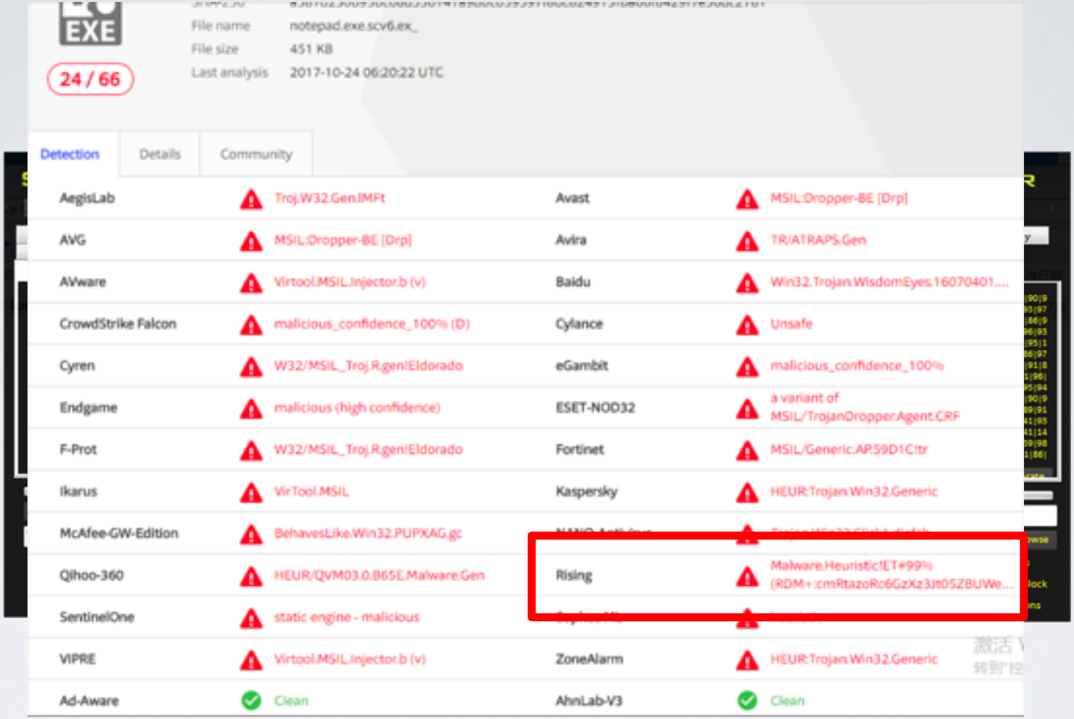


In the 1^{st} month







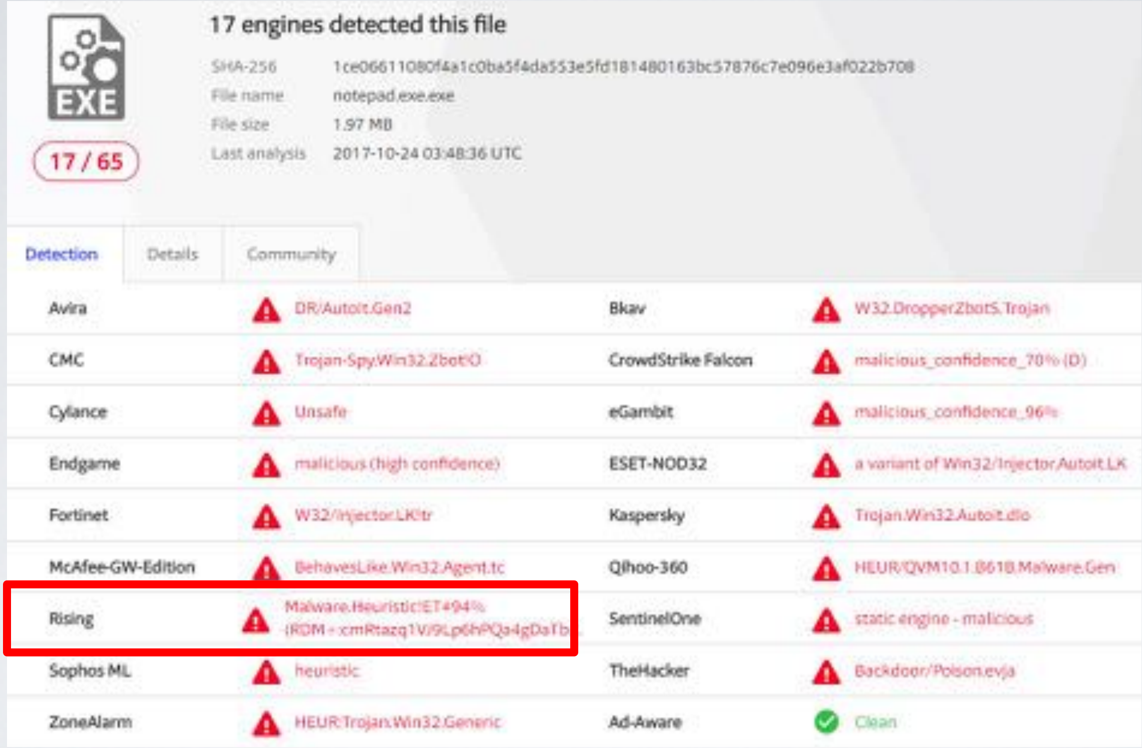


EXE

24 / 66

File name	notepad.exe.scv6.ex_
File size	451 KB
Last analysis	2017-10-24 06:20:22 UTC

AegisLab	⚠ Troj.W32.Gen.lMFt	Avast	⚠ MSIL:Dropper-BE [Drp]
AVG	⚠ MSIL:Dropper-BE [Drp]	Avira	⚠ TR/ATRAPS.Gen
AVware	⚠ Virtool.MSIL.Injector.b (v)	Baidu	⚠ Win32.Trojan.WisdomEyes.16070401....
CrowdStrike Falcon	⚠ malicious_confidence_100% (D)	Cylance	⚠ Unsafe
Cyren	⚠ W32/MSIL_Troj.R.gen!Eldorado	eGambit	⚠ malicious_confidence_100%
Endgame	⚠ malicious (high confidence)	ESET-NOD32	⚠ a variant of MSIL/TrojanDropper.Agent.CRF
F-Prot	⚠ W32/MSIL_Troj.R.gen!Eldorado	Fortinet	⚠ MSIL/Generic.AP.59D1C!tr
Ikarus	⚠ VirTool.MSIL	Kaspersky	⚠ HEUR:Trojan.Win32.Generic
McAfee-GW-Edition	⚠ BehavesLike.Win32.PUPXAG.gc	NANO-Antivirus	⚠ [illegible]
Qihoo-360	⚠ HEUR/QVM03.0.B65E.Malware.Gen	Rising	⚠ Malware.Heuristic!ET#99% (RDM+:cmRtazoRc6GzXz3Jt05ZBUWe...
SentinelOne	⚠ static engine - malicious	[illegible]	⚠ [illegible]
VIPRE	⚠ Virtool.MSIL.Injector.b (v)	ZoneAlarm	⚠ HEUR:Trojan.Win32.Generic
Ad-Aware	✓ Clean	AhnLab-V3	✓ Clean



EXE

17 / 65

17 engines detected this file

SHA-256 1ce06611080f4a1c0ba5f4da553e5fd181480163bc57876c7e096e3af022b708
File name notepad.exe.exe
File size 1.97 MB
Last analysis 2017-10-24 03:48:36 UTC

Avira	⚠ DR/AutoIt.Gen2	Bkav	⚠ W32.DropperZbotS.Trojan
CMC	⚠ Trojan-Spy.Win32.Zbot!O	CrowdStrike Falcon	⚠ malicious_confidence_70% (D)
Cylance	⚠ Unsafe	eGambit	⚠ malicious_confidence_96%
Endgame	⚠ malicious (high confidence)	ESET-NOD32	⚠ a variant of Win32/Injector.AutoIt.LK
Fortinet	⚠ W32/Injector.LK!tr	Kaspersky	⚠ Trojan.Win32.AutoIt.dlo
McAfee-GW-Edition	⚠ BehavesLike.Win32.Agent.tc	Qihoo-360	⚠ HEUR/QVM10.1.8618.Malware.Gen
Rising	⚠ Malware.Heuristic!ET#94% (RDM+:cmRtazq1V/9Lp6hPQa4gDaTb	SentinelOne	⚠ static engine - malicious
Sophos ML	⚠ heuristic	TheHacker	⚠ Backdoor/Poison.evja
ZoneAlarm	⚠ HEUR:Trojan.Win32.Generic	Ad-Aware	✓ Clean

Other File Formats

RISING 瑞星

Different Formats vs. Different Features Engineering

SWF EXPLOIT

Features are extracted from flash structure and 3-grams of strings in ABC. Recent 30-Day performance: 520/563 ~ **92%,** defeated almost all EXP-KITs.

Obfuscated Script

After special normalization, extract script skeleton features. It is still being improved because it often conflicts with '***.min.js**'.

PDF EXPLOIT

Features come from PDF keywords and embedded JS. About 88% of PDF exploits/phishing can be detected.

Conclusion

- AI/ML can improve the productivity of all aspects of anti-malware.
- The goal of using ML needs to be clear.
- In our application, the feature engineering directly affects the final effect.
- It's important to mitigate false positives.

Continue To Challenge

Try to create a low-dimensional RDM+

More Feature Engineering

Behavior sequence + LSTM

Understanding API Calls

and so on

THANK YOU