WINNING THE OS X MALWARE WAR: NEURALLY FINDING OUTBREAKS

Sean Park
Senior Malware Scientist
Trend Micro
What we have tried... and failed...
Features: Statistics
Features: Executable header

```c
struct __macho_header64 {
  0xfeedfacf,
  0x1000007,
  0x80000003,
  MH_EXECUTE,
  5,
  64,
  MH_NO_RENAME||MH_DYLDLINK,
  0x0
};

; Load Command 0

struct __macho_segment_command_64 {
  LC_SEGMENT_64,
  0x48,
  "PAGEZERO", 0, 0, 0, 0, 0,
  0x0,
  0x1000,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
          0
};
```

// mach magic number identifier
// cpu specifier
// machine specifier
// type of file
// number of load commands
// the size of all the load commands
// flags
// reserved

// LC_SEGMENT_64
// includes sizeof section_64 structs
// segment name
// memory address of this segment
// memory size of this segment
// file offset of this segment
// amount to map from the file
// maximum VM protection
// initial VM protection
// number of sections in segment
// flags
Evading Machine Learning Malware Detection

- Random mutations 13%
- Black box attack 16%
- Score-based attack 14%

- add a new entry point which is never used in the original binary
- manipulate (break) signature
- manipulate debug info
- pack or unpack the file
- modify (break) header checksum
- append bytes to the overlay (end of PE file)

ABSTRACT

Machine learning is a popular detection approach that is effective against malware that is never seen. However, in practice, the majority detection engines or supplemental to anti-malware vendors. Recent work has shown that models are trained on malware that has been classified before and the attack on new families of malware. In this paper, we present an attractive tool for anti-malware detection against new malware. We also show that modern detection engines are not always able to generalize to new samples whose characteristics differ from those in the training data. We then evaluate the performance of the model on the new malware-family samples. Interestingly, we find that the detection rate of the trained model on the new malware-family samples is lower than that of the original samples. This suggests that modern detection engines are not always able to generalize to new samples whose characteristics differ from those in the training data.
Features: N-gram

call, push, xor

\[ I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left( \frac{p(x, y)}{p(x) \cdot p(y)} \right) \]
Features: N-gram

call
push
xor
mov
ret
push
call
push

= = = =

call
push
mov
mov
call
push
mov
sub
pop
jz
add
mov
ret
Features: Raw bytes
Outbreaks
What do we care about in malware detection?

**Confusion Matrix**

- Real Normal: 63,390
- Real Malware: 796
- Predicted Normal: 105
- Predicted Malware: 39,912

**ROC Curve**

- True Positive Rate vs. False Positive Rate
So, how do you detect outbreaks?

Outbreak Detection == Clustering
What does an outbreak look like?

### Functional Upgrade

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>file</th>
<th>filesize</th>
<th>offset</th>
<th>va</th>
<th>nfuncs</th>
<th>totalfuncsize</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC.OSX.Trojan.FlashBack.AG</td>
<td>312</td>
<td>samples/2017-03_base</td>
<td>220784</td>
<td>0</td>
<td>7152</td>
<td>366</td>
<td>41757</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/malware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>db65c02586f7a6555ec68750ca6835a696394df8a73554...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC.OSX.Trojan.FlashBack.F</td>
<td>284</td>
<td>samples/2017-03_base</td>
<td>132860</td>
<td>0</td>
<td>6448</td>
<td>280</td>
<td>16402</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/malware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b99b375c0cbe92c50760240a0eee43d175e2f774474706...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

TRENDS MICRO

RSA Conference 2018
What does an outbreak look like? : Functional Upgrade
What does an outbreak look like? : Metamorphism
What does an outbreak look like?

: Metamorphism
OS X Malware Dataset
OS X Features

def walk(self):
    text = ''

    prev_end = MinEA() + 1 # Account for prev_end-1 below.
    for i, fea in enumerate(Functioins(MinEA(), MaxEA())):
        start = GetFunctionAttr(fea, FUNCATTR_START)
        end = GetFunctionAttr(fea, FUNCATTR_END)
        name = GetFunctionName(fea)
        lib = self.islib(fea)

        if not lib:
            self.add_function(name, start, end)

        prev_end = end

functions.json

```
{
    "va": 4294967295,
    "functions": [
        [4294971192,
        4294971251
        ],
        // 2 items
    ],
    "filesize": 27245,
    "file": "0034e6d09e3adc7040bd77f2bcaede6188b500a839f079a71f83a9c5c37152b",
}
```

from capstone import *
from capstone.x86 import *

# Process IDA generated JSON
func_features = []
filepath = os.path.join(dirpath, filename)
with open(filepath) as f:
    js = json.load(f)

    if js['arch'] == 'x86':
        md = Cs(CS_ARCH_X86, CS_MODE_32)
    elif js['arch'] == 'x64':
        md = Cs(CS_ARCH_X86, CS_MODE_64)

    for func in js['functions']:
        code = binascii.unhexlify(func['code']) # convert hex string to byte string.
        function = [ins.id for ins in md.disasm(code, len(code))]
        func_features.append(np.array(function, dtype=np.uint8))

program.npz
Feature Construction

E8 87 FD FF FF
6A 5C
68 40 1B 40 00
E8 6E 07 00 00
33 DB
89 5D E4
89 5D FC
8D 45 94
C4
50
FF 15 14 11 40 00

call 0x004017B1
push 0x5C
push 0x00401B40
call 0x004021A4
xor ebx, ebx
mov [ecx-0x1c], ebx
mov [ebp-0x4], ebx
lea eax, [ebp-0x6c]
ret
push eax
call 0x0041230EF
Features: Instructions
Locality Sensitive Hash (LSH)

Distance: 0

Distance: 92
Class 38 (nsamples=669)
Mac.OSX.iWorm.F (1 samples) [471]
Trojan.MAC.KeRangerRansom.A (53 samples)
Trojan.MAC.KeRangerRansom.D (28 samples)
Trojan.MAC.Tsunami.A (1 samples)
Unknown (586 samples)

Sample 31 #598
train Trojan.MAC.Tsunami.A(label=180)
id=a36209534623310b50d9460314f147e21a92c0e462f74f046718ba8fb18cbb
filesize=24576
va=0xffffffff
nfuncs=4
totalfuncsize=187
scan_date=2017-06-07-08:11:09

Sample 26 #471
train Mac.OSX.iWorm.F(label=142)
id=6ed282e6e54ccbec80b16bb39fbdf465576215a94ff1b7875fc8b28d29392fb6
filesize=177044
va=0x18f0
nfuncs=2
totalfuncsize=185
scan_date=2017-07-26-11:35:32
Neural Outbreak Detection
Stacked De-noising Autoencoder (SDA)
Convolutional Neural Network (CNN)

Input Volume (+pad 1) (7x7x3)

Filter W0 (3x3x3)

Filter W1 (3x3x3)

Output Volume (3x3x2)

Bias b0 (1x1x1)

Bias b1 (1x1x1)

http://cs231n.github.io/convolutional-networks/
Sparse Convolutional Autoencoder

```
call   push   push   call   xor   mov   mov   lea   push   call
```

```
call       E8
push       6A
push       6A
call       E8
xor        33
mov        89
mov        89
lea        8D
push       6A
call       EA
```

```
embedding
encoder

\( z \)

decoder

reconstructed
```

```
call     E8
6A
6A
E8
33
89
89
89
8D
6A
EA
```
Putting them together

```
feature
  └── Autoencoder
    └── z
        └── HDBSCAN
            └── class
                Class 7
```

- `call, push, push, xor, mov, mov, lea, ret, ...`
- `38, 99, 99, 186, 8, 8, 45, 127, ...`
- `1.3, 0.5, -0.91, 2.12, -0.01`

Class 7
Model Performance
Cluster 2 (nsamples=668)
  KeRangerRansom.A (53 samples)
  KeRangerRansom.D (28 samples)
  Tsunami.A (1 samples)
  Unknown (586 samples)
Cluster 11 (nsamples=10)
FlashBack.AE (1 samples)
FlashBack.AG (1 samples)
FlashBack.AP (1 samples)
FlashBack.E (1 samples)
FlashBack.F (3 samples)
FlashBack.P (2 samples)
FlashBack.Q (1 samples)
Cluster 49 (nsamples=836)
- FlashBack.AF (2 samples)
- FlashBack.L (344 samples)
- FlashBack.M (5 samples)
- FlashBack.Q (2 samples)
- Flashback.E (1 samples)
- Flashback.J (1 samples)
- Flashback.K (1 samples)
- Flashback.L (1 samples)
- Flashback.M (10 samples)
- Flashback.N (7 samples)
- Flashback.O (1 samples)
- Flashback.P (1 samples)
- Flashback.Q (1 samples)
- Trojan-Downloader.Flashfake.a

Unknown (447 samples)
Early Detection
Use convolutional autoencoder and HDBSCAN to identify dynamically morphing malware variants across outbreaks.

Grab malware samples in your repository, extract functions dataset using IDA Pro python script, train it with tensorflow or your favorite deep learning API, cluster it with HDBSCAN, visualize the clusters using holomap, and name all those ‘unknown’ samples.
Credits

Jon Oliver
Lili Diao
Will Zhuo
Steven Du
Gavin Gan
MacX team
MacTRT team
References

[9] https://www.youtube.com/watch?v=Ilg3gGewQ5U
QUESTIONS?

Sean Park, Senior Malware Scientist
spark@trendmicro.com