

RSA® Conference 2018

San Francisco | April 16–20 | Moscone Center

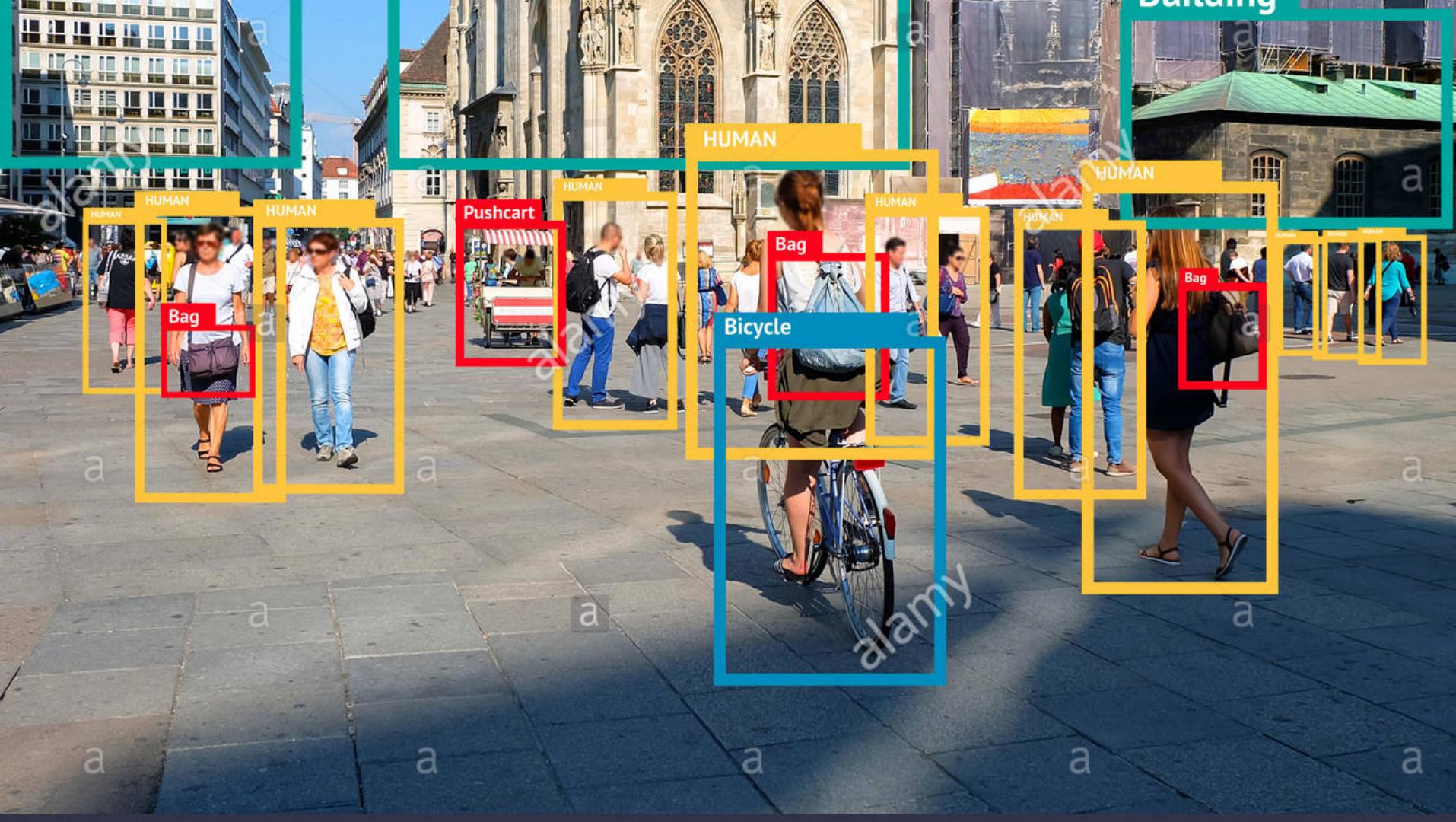
SESSION ID: HTA-F03

WINNING THE OS X MALWARE WAR: NEURALLY FINDING OUTBREAKS

Sean Park

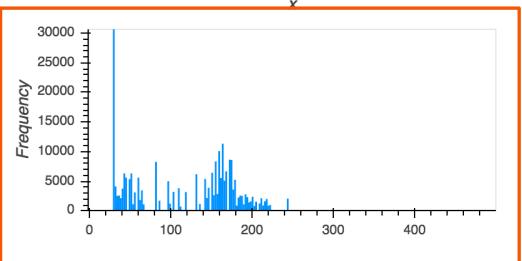
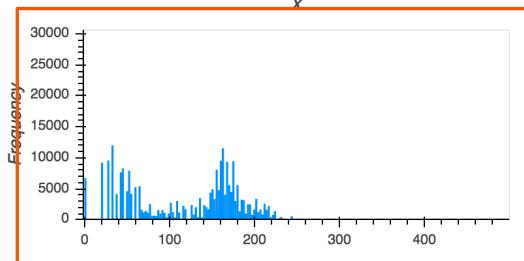
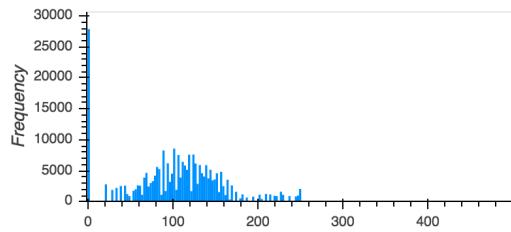
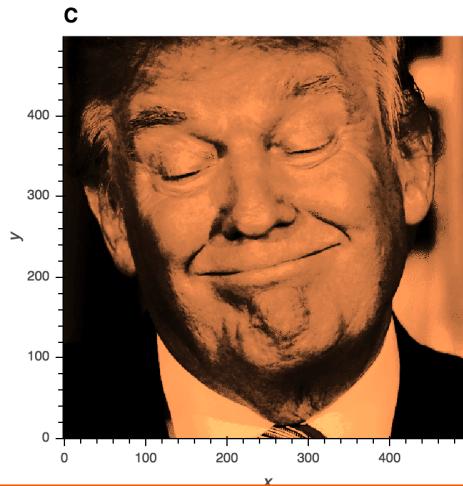
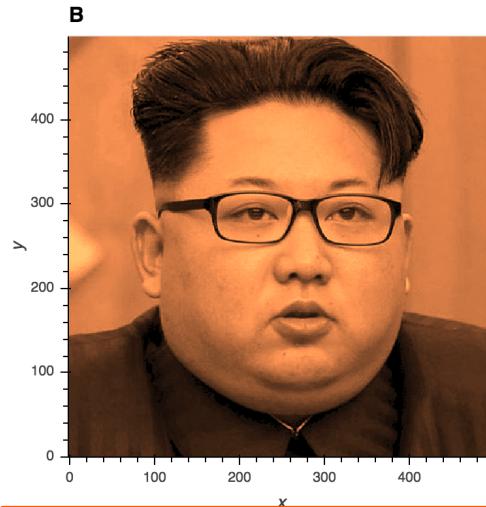
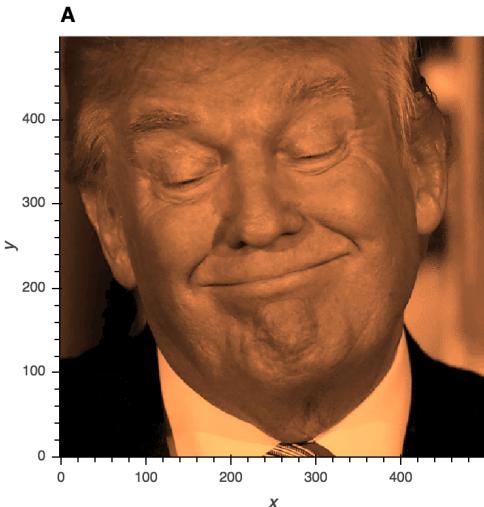
Senior Malware Scientist
Trend Micro





What we have tried...
and failed...

Features: Statistics





Features: Executable header

```
00000000000000001000    struct __macho_header64 {
                           0xfeedfacf,                                // mach magic number identifier
                           0x1000007,                                 // cpu specifier
                           0x80000003,                                // machine specifier
                           MH_EXECUTE,                                // type of file
                           5,                                         // number of load commands
                           632,                                       // the size of all the load commands
                           MH_NOUNDEFS|MH_DYLDLINK,                  // flags
                           0x0                                         // reserved
}
;
; Load Command 0
;

00000000000000001020    struct __macho_segment_command_64 {
                           LC_SEGMENT_64,                            // LC_SEGMENT_64
                           0x48,                                     // includes sizeof section_64 structs
                           "__PAGEZERO",   0, 0, 0, 0, 0, 0, 0, 0, // segment name
                           0x0,                                      // memory address of this segment
                           0x1000,                                    // memory size of this segment
                           0x0,                                      // file offset of this segment
                           0x0,                                      // amount to map from the file
                           0x0,                                      // maximum VM protection
                           0x0,                                      // initial VM protection
                           0x0,                                      // number of sections in segment
                           0                                           // flags
}
```

Features: Executable header



Evading Machine Learning Malware Detection

Hyrum
Enc
hyrum@

| | | | |
|-------------------|---------------------------|-----|--------------------|
| • add a used [11] | Random mutations | 13% | is never stored in |
| • manip | Black box attack | 16% | |
| • creat | Score-based attack | 14% | |

ABSTRACT

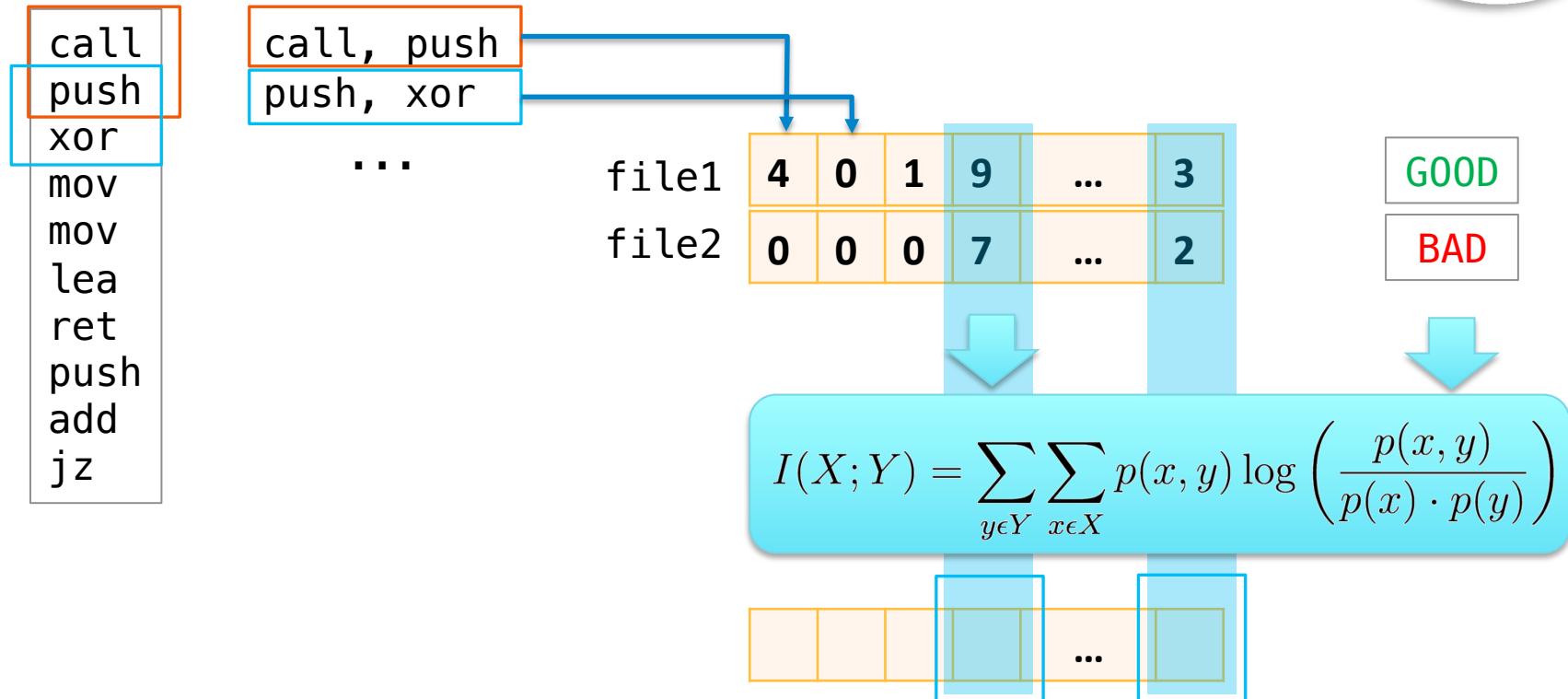
Machine learning is a primary tool for malware detection because it has been seen to be effective. This has resulted in its practical application by engines or supplementarily by vendors. Recent work has shown that models are vulnerable to other attacks. In this paper, we describe attacks that have been developed in information security research to have some degree of success. Importantly, even

- append bytes to extra space at the end of sections
- create a new entry point which immediately jumps to the original entry point
- manipulate (break) signature
- manipulate debug info
- pack or unpack the file
- modify (break) header checksum
- append bytes to the overlay (end of PE file)

by Filar
ame, Inc.
dgame.com

an attractive tool for anti-malware detection engines or classifiers. Properly regularized machine learning models generalize to new samples whose feature distribution is different from the training set. Supervised learning models automatically learn the relationships among features that are most discriminative for classifying malware. This allows defenders to quickly identify known malware variants that are manifest in the wild. However, sophisticated adversaries can bypass these defenses by attacking anti-malware engines, be

Features: N-gram



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



The screenshot shows a debugger interface with several windows:

- Program Segmentation**: Shows memory regions.
- Hex View-1**: The active window, showing assembly code and raw bytes.
- Structures**: Shows memory structures.
- Enums**: Shows memory enums.

The assembly code in the `_sha256_process endp` block is:

```
add    [rcx], al
add    [rax-78h], ecx
test   byte ptr [rcx-78h], 0BCh
jmp    near ptr 140FF47F8h
```

The **Hex View-1** window displays raw bytes. A blue box highlights the first few bytes of the raw dump:

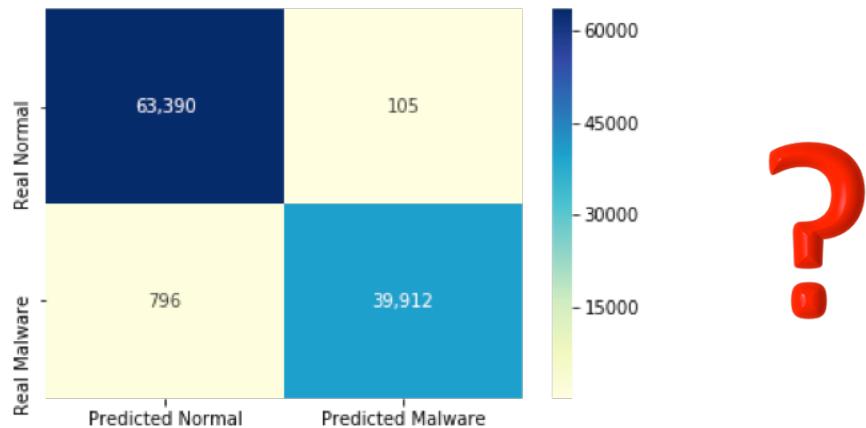
```
db 0Eh, 0B7h, 6, 49h, 0C0h, 0E1h, 19h
dq 0E0C049004EB70E40h, 46B70E40C0084911h, 0C9084909E1C04903h
dq 0C00849024EB70E40h, 49FEFFFD98C8849h, 490546B70E40CA88h
dq 44EB70E4019E1C0h, 49C0084911E0C049h, 0E40FEFFFF998C88h
dq 4909E1C0490746B7h, 49064EB70E40C908h, 0FEFE118C8849C008h
dq 46B70E40CF8848F Eh, 0870E4019E1C04909h, 84911E0C049084Eh
dq 0FEFEFFA18C8849C0h, 0E1C0490846B70E40h, 4EB70E40C9084909h
dq 894C8849C008490Ah, 0E1C0490D46B70E40h, 0C0490C4EB70E4019h
dq 8C8849C0084911E0h, 46B70E40FEFEFFA9h, 0C9084909E1C0490Fh
dq 0C008490E4EB70E40h, 46B70E40A14C8849h, 0B70E4019E1C04911h
dq 84911E0C049104Eh, 0FEFEFFB18C8849C0h, 0E1C0491346B70E40h
dq 4EB70E40C9084909h, 0A94C8849C0084912h, 0E1C0491546B70E40h
dq 0C049144E870E4019h, 8C8849C0084911E0h, 46B70E40FEFEFFB9h
dq 0C9084909E1C04917h, 0C00849164EB70E40h, 46B70E40C94C8849h
dq 0B70E4019E1C04919h, 84911E0C049184Eh, 0FEFEFE098C8849C0h
dq 0E1C0491B46B70E40h, 4EB70E40C9084909h, 0D14C8849C008491Ah
dq 49C88845314E8A4Dh, 884D07E8C049C888h, 0D708491BE7C049CFh
dq 4D0AEBC049CB8849h, 84914E1C049C988h, 18E8C049F1304901h
dq 4906E3C049CB884Dh, 0F98849C33049CB08h, 49DF004949718A49h
dq 8A4939598A40D700h, 0DE304DD688494151h, 8C49D63049CE204Dh
dq 8849438B2E99368Dh, 4829698A4DB14Ch, 49F68849EF8845CCh
dq 0C049EA884D07E0C0h, 0F68849FA08491BE2h, 49E8884D0AECC049h
dq 3049F8084914E0C0h, 0EB884C18EFC049D8h, 48F3084806E3C048h
dq 0C8304DD8884DCB30h, 4D8304DE8204Dh, 81548849CB0049F3h
dq 130D8C4B21718A4Dh, 703645900FB8C48h, 8849FA88C17C8849h
dq 0E5E8A887E861888h, 16558818187518h, 884981885518888h
```

Outbreaks

What do we care about in malware detection?

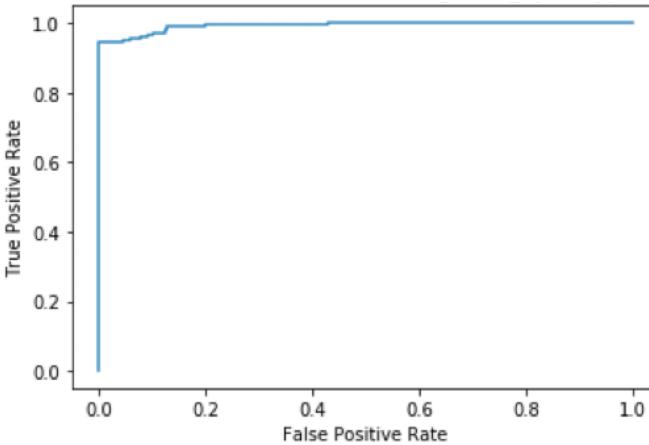


Confusion Matrix



?

ROC Curve



So, how do you detect outbreaks?



Outbreak
Detection

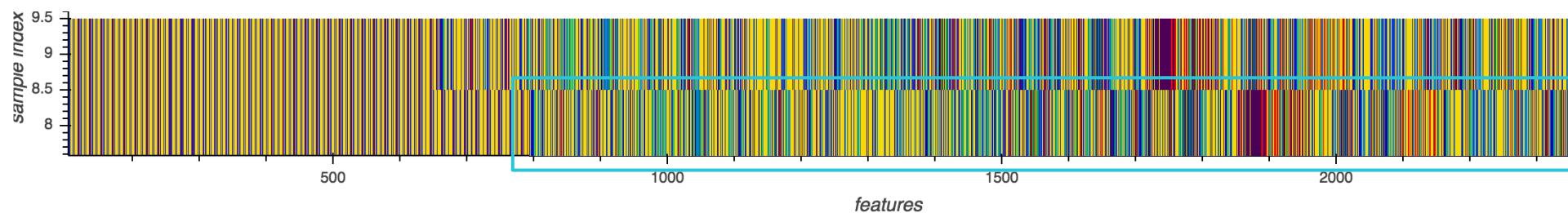
==

Clustering

What does an outbreak look like? : Functional Upgrade



| | name | id | file | filesize | offset | va | nfuncs | totalfuncs | size |
|-----|---|----|---|----------|--------|------|--------|------------|------|
| 312 | MAC.OSX.Trojan.FlashBack.AG db65c02586f7a6555ec86750ca6835a696394df8a73554... | | samples/2017-03_base /malware /db65c02586f7a6555... | 220784 | 0 | 7152 | 366 | 41757 | |
| 284 | MAC.OSX.Trojan.FlashBack.F b99b375c0fbe92c50760240a0eee43d175e2f774474706... | | samples/2017-03_base /malware /b99b375c0fbe92c50... | 132860 | 0 | 6448 | 280 | 16402 | |



What does an outbreak look like? : Functional Upgrade



#RSAC

The screenshot shows a debugger interface with two assembly code snippets side-by-side. The left pane contains the assembly for `sub_3CE2`, and the right pane contains the assembly for `sub_3CDA`. Both snippets are nearly identical, showing standard x86 assembly with `push ebp`, `mov ebp, esp`, `sub esp, 18h`, and `call _siglongjmp`. The right pane also includes a section for local variables (`var_44` to `arg_0`) and a series of `lea eax, [ebp+var_14]` instructions.

```
; void sub_3CE2(int)
sub_3CE2 proc near ; DATA XREF: sub_6C1A:loc_6C51
    push    ebp
    mov     ebp, esp
    sub     esp, 18h
    mov     dword ptr [esp+4], 1 ; int
    mov     dword ptr [esp], offset dword_34360 ; sigjmp_buf
    call    _siglongjmp
sub_3CE2 endp

; void sub_3CDA(int)
sub_3CDA proc near ; DATA XREF: sub_7206:loc_723D
    push    ebp
    mov     ebp, esp
    sub     esp, 18h
    mov     dword ptr [esp+4], 1 ; int
    mov     dword ptr [esp], offset dword_1F4E0 ; sigjmp_buf
    call    _siglongjmp
sub_3CDA endp

; ====== S U B R O U T I N E ======
; Attributes: bp-based frame

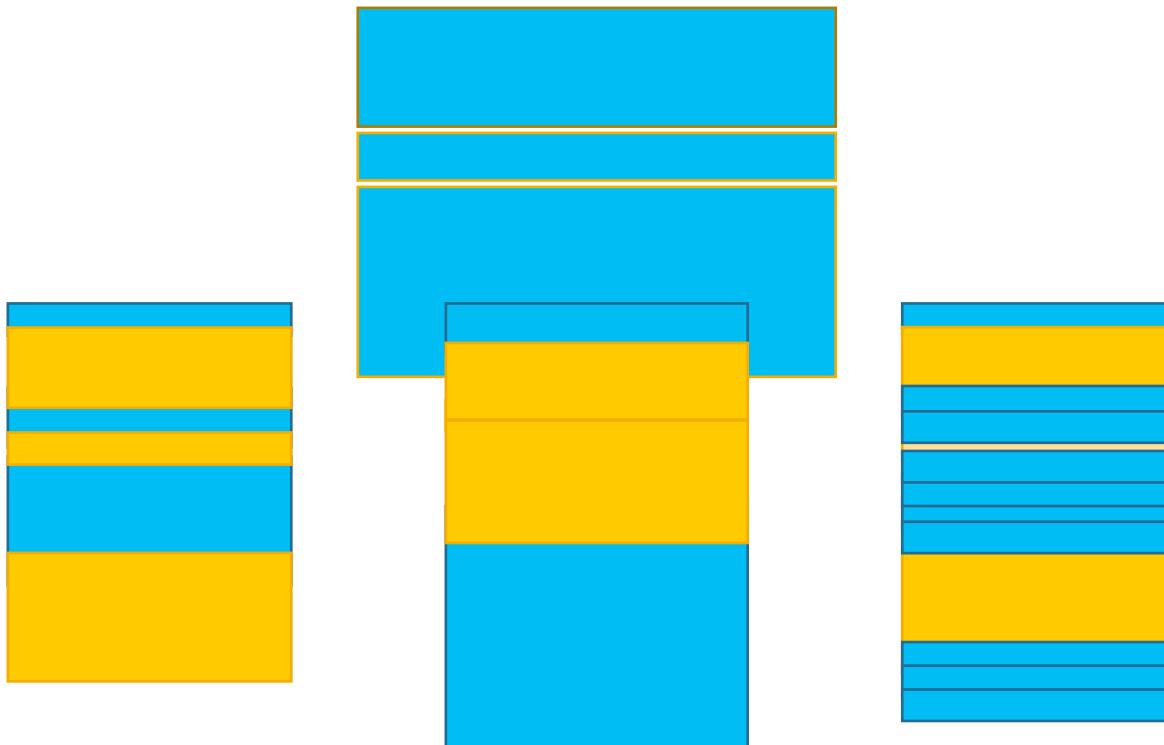
sub_3CF4 proc near ; CODE XREF: sub_47E6+842(19)p
    var_44 = dword ptr -44h
    var_14 = dword ptr -14h
    var_10 = dword ptr -10h
    var_C = dword ptr -0Ch
    arg_0 = dword ptr 8

    push    ebp
    mov     ebp, esp
    push    ebx
    sub     esp, 64h
    lea     eax, [ebp+var_14]
    lea     ebx, [ebp+var_44]
    mov     [ebp+var_C], 0FFFFFFFh
    mov     [ebp+var_14], 0Ch
    mov     [esp+8], eax ; size_t *
```

What does an outbreak look like? : Metamorphism



Original Code



What does an outbreak look like? : Metamorphism



```

mov    ecx, [ebp-38h]
add    ecx, 1E6h
mov    edx, [ebp-2Ch]
sub    edx, ecx
mov    [ebp-2Ch], edx
mov    eax, [ebp-18h]
sub    eax, 2CEh
test   eax, eax
jz     short loc_41DFAB
mov    ecx, [ebp-38h]
add    ecx, [ebp-38h]
mov    edx, [ebp-18h]
sub    edx, ecx
mov    [ebp-18h], edx
mov    eax, [ebp-18h]
add    eax, 16Ah
mov    ecx, [ebp-18h]
sub    ecx, eax
mov    [ebp-18h], ecx
mov    edx, [ebp-38h]
mov    eax, [ebp-18h]
lea    ecx, [eax+edx+288h]
mov    [ebp-18h], ecx
mov    edx, [ebp-2Ch]
sub    edx, [ebp-18h]
test   edx, edx
jz     short loc_41DFE3
mov    eax, [ebp-2Ch]
sub    eax, [ebp-2Ch]
mov    ecx, [ebp-18h]
sub    ecx, eax
mov    [ebp-18h], ecx
mov    edx, [ebp-18h]
sub    edx, 0BEh

```

```

mov    ecx, [ebp-38h]
add    ecx, 1E6h
mov    edx, [ebp-2Ch]
sub    edx, ecx
mov    [ebp-2Ch], edx
mov    eax, [ebp-18h]
sub    eax, 2CEh
test   eax, eax
jz     short loc_41DFAB
mov    ecx, [ebp-38h]
add    ecx, [ebp-38h]
mov    edx, [ebp-18h]
sub    edx, ecx
mov    [ebp-18h], edx
mov    eax, [ebp-18h]
add    eax, 16Ah
mov    ecx, [ebp-18h]
sub    ecx, eax
mov    [ebp-18h], ecx
mov    edx, [ebp-38h]
mov    eax, [ebp-18h]
lea    ecx, [eax+edx+288h]
mov    [ebp-18h], ecx
mov    edx, [ebp-2Ch]
sub    edx, [ebp-18h]
test   edx, edx
jz     short loc_41DFE3
mov    eax, [ebp-2Ch]
sub    eax, [ebp-2Ch]
mov    ecx, [ebp-18h]
sub    ecx, eax
mov    [ebp-18h], ecx
mov    edx, [ebp-18h]
sub    edx, 0BEh

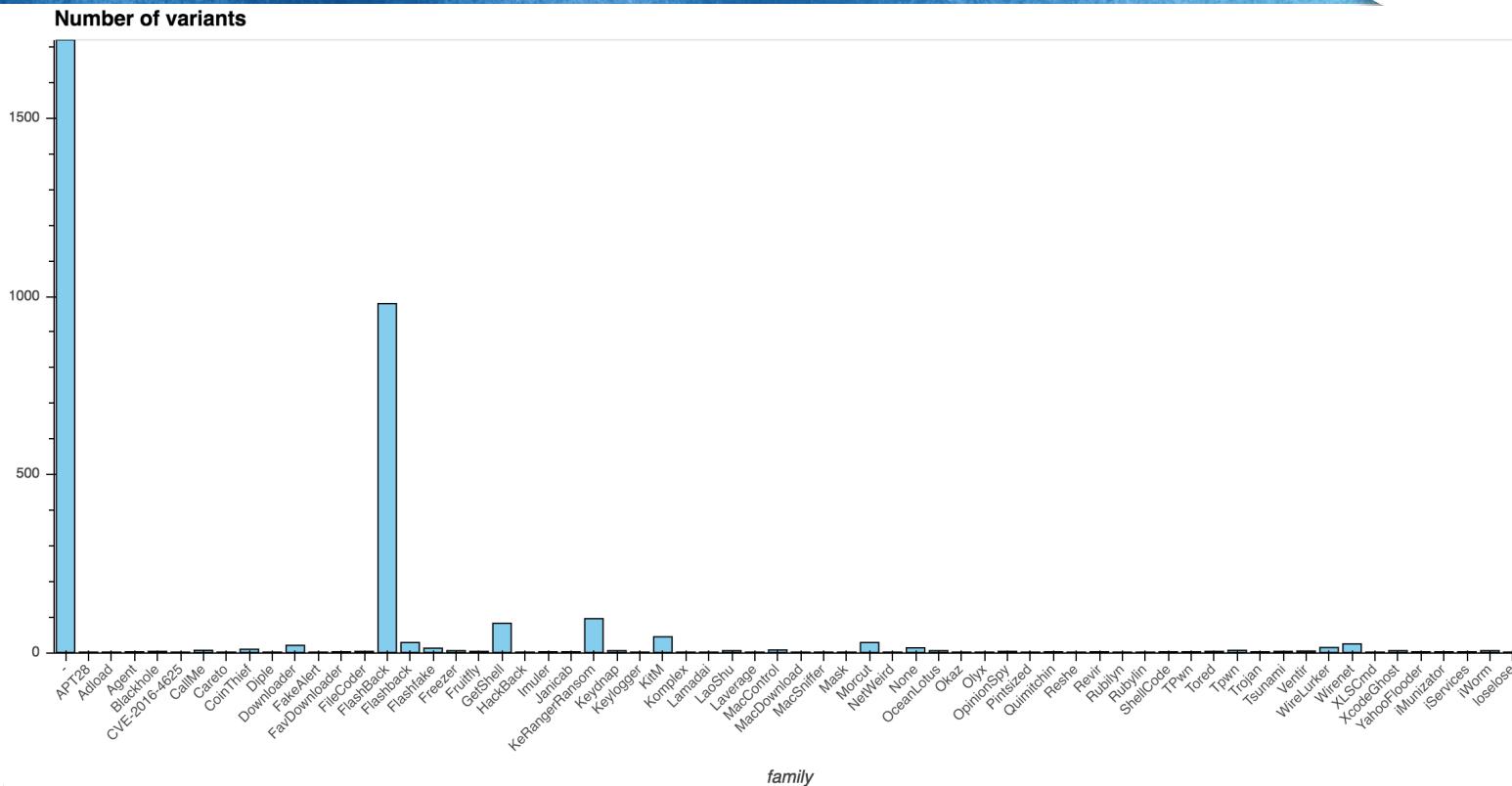
```

```

movzx  ecx, [ebp+var_18]
xor   ecx, 16h
mov    [ebp+var_2340], ecx
fld    ds:dbl_41CB70
fstp  [ebp+var_2318]
fld    ds:dbl_41CB68
fstp  [ebp+var_2310]
fld    ds:dbl_41CB60
fstp  [ebp+var_2308]
fld    ds:dbl_41CB58
fstp  [ebp+var_2300]
fld    ds:dbl_41CB50
fstp  [ebp+var_22F8]
fld    ds:dbl_41CB48
fstp  [ebp+var_22F0]
fld    ds:dbl_41CB40
fstp  [ebp+var_22E8]
fld    ds:dbl_41CB38
fstp  [ebp+var_22E0]
fld    ds:dbl_41CB30
fstp  [ebp+var_22D8]
fld    ds:dbl_41CB28
fstp  [ebp+var_22D0]
fld    ds:dbl_41CB20
fstp  [ebp+var_22C8]
fld    ds:dbl_41CB18
fstp  [ebp+var_22C0]
fld    ds:dbl_41CB10
fstp  [ebp+var_22B8]
push   ecx
call   _strcat
add    esp, 8
mov    [ebp+var_36C4], eax
fld    ds:dbl_41FB30
fstp  [ebp+var_36B0]
fld    ds:dbl_41FB28
fstp  [ebp+var_36A8]
fld    ds:dbl_41FB20
fstp  [ebp+var_36A0]
fld    ds:dbl_41FB18
fstp  [ebp+var_3698]
fld    ds:dbl_41FB28
fstp  [ebp+var_3690]
fld    ds:dbl_41FB10
fstp  [ebp+var_3688]
fld    ds:dbl_41FB18
fstp  [ebp+var_3680]
fld    ds:dbl_41FB08
fstp  [ebp+var_3678]
fld    ds:dbl_41FB00
fstp  [ebp+var_3670]
fld    ds:dbl_41FAF8
fstp  [ebp+var_3668]
fld    ds:dbl_41FAF0
fstp  [ebp+var_3660]
fld    ds:dbl_41FB18
fstp  [ebp+var_3658]
fld    ds:dbl_41FB28

```

OS X Malware Dataset



OS X Features



The screenshot shows the IDA Pro interface analyzing a binary file. The left pane displays the 'Functions window' with several functions listed:

| Function name | Segment |
|--------------------|---------------|
| sub_1000032AB | _text |
| sub_1000032C7 | _text |
| sub_1000032E5 | _text |
| sub_100003364 | _text |
| sub_1000033D4 | _text |
| sub_100003408 | _text |
| sub_10000342D | _text |
| sub_100003590 | _text |
| _CFRelease | _symbol_stub1 |
| _CStringGetCString | _symbol_stub1 |
| IOObjectRelease | _symbol stub1 |

The right pane shows the assembly code for one of the functions, specifically the start of the main routine:

```
0000F38 ; ====== S U B R O U T I N E =====
; Attributes: noreturn
start    public start
proc near
push    0
mov     rbp, rsp
and    rsp, OFFFFFFFFFFFFFFF0h
mov     rdi, [rbp+8]
lea     rsi, [rbp+10h]
mov     edx, edi
add    edx, 1
shl    edx, 3
add    rdx, rsi
mov     rcx, rdx
0000F54 0000000100000F54: start+1C
```

The bottom pane shows the 'Output window' displaying the loading message and the Python environment:

```
Database '0034e6d09e3adc7040bd77f2bcaaede6188b500a839f079a71f83a9c5c37152b.macn064' has been loaded.
Compiling file 'C:\Program Files (x86)\IDA 6.7\idc\ida.idc'...
Executing function 'main'...
-----
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)]
IDAPython 64-bit v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
-----
[Python]
```

Status bar at the bottom: AU: idle | Down | Disk: 18GB

OS X Features



```
def walk(self):
    text = ''

    prev_end = MinEA() + 1 # Account for prev_end-1 below.
    for i, fea in enumerate(Functions(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
```



```
{
    "va": 4294967295,
    "functions": [
        [
            4294971192,
            4294971251
        ],
        [ ... ] // 2 items
    ],
    "filesize": 27245,
    "file": "0034e6d09e3adc7040bd77f2bcaaede6188b500a839f079a71f83a9c5c37152b."
}
```

functions.json



```
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)

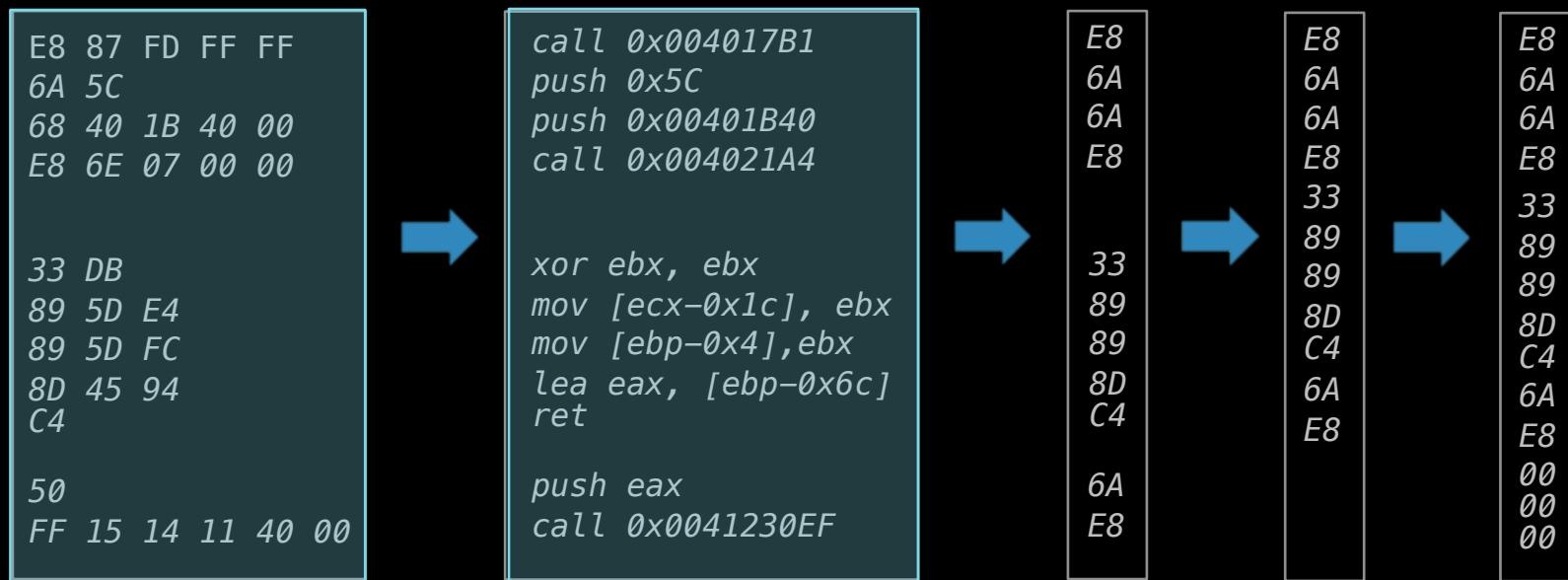
    for func in js['functions']:
        code = binascii.unhexlify(func['code']) # convert hex string to byte string.
        function = [ins.id for ins in md.disasm(code, len(code))]
        func_features.append(np.array(function, dtype=np.uint8))
```

program.npz



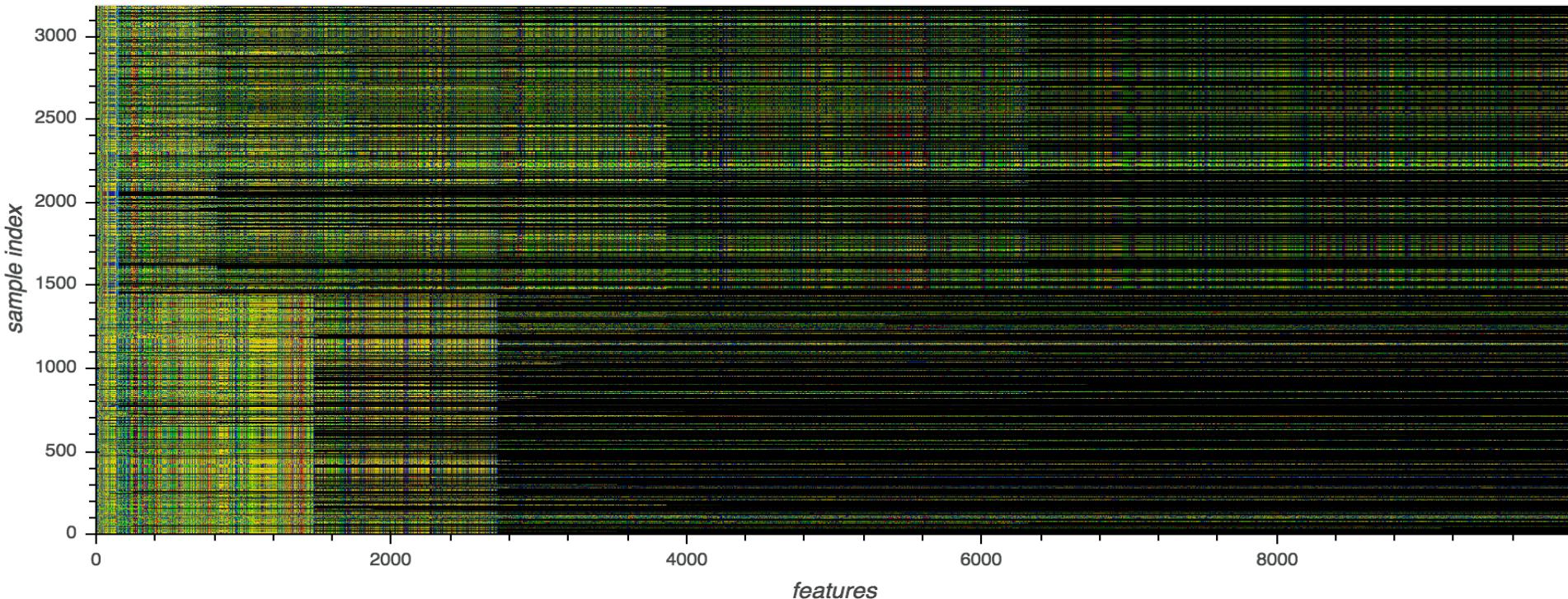


Feature Construction



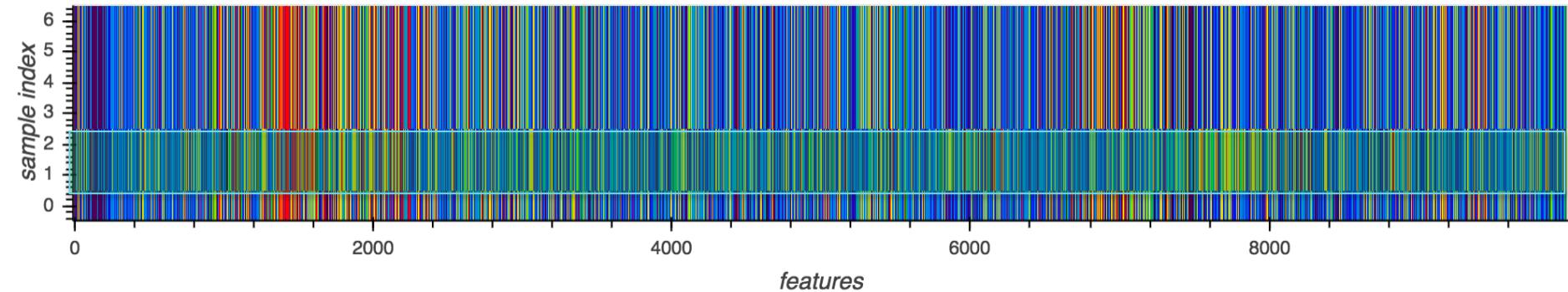


Features: Instructions





Locality Sensitive Hash (LSH)



7E227A6559E738025EDBF3E3964162076BA0DB124803F761798C5F6747A88B8F83C2DE

7E227A6559E738025EDBF3E3964162076BA0DB124803F761798C5F6747A88B8F83C2DE

Distance: 0

7E227A6559E738025EDBF3E3964162076BA0DB124803F761798C5F6747A88B8F83C2DE

D322567548E778866E97F3E35A82124BAB60EB120447F7417E8C5F524BAD874E02C2ED

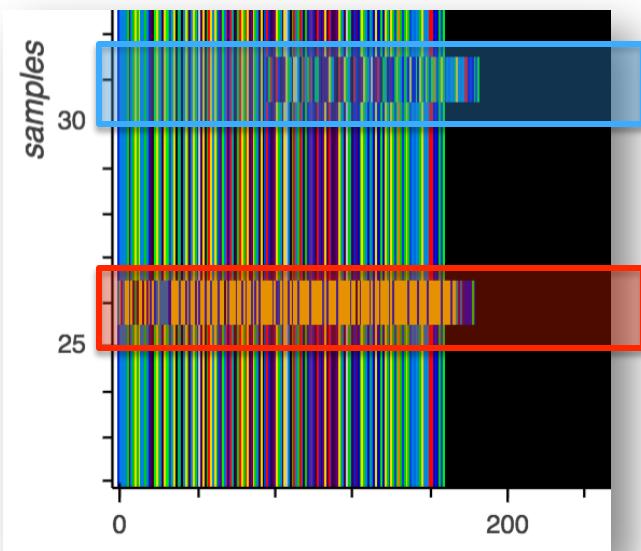
Distance: **92**

HDBSCAN



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

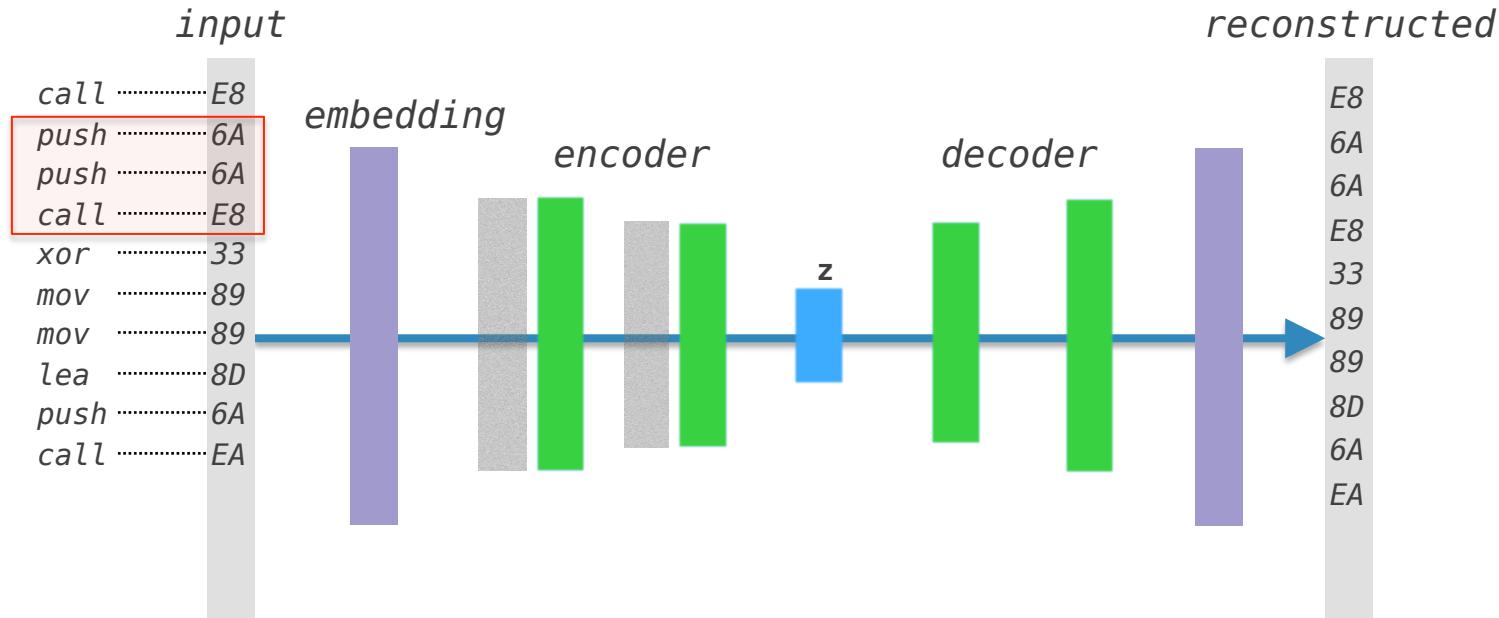
train Trojan.MAC.Tsunami.A(label=180)
id=a36209534623310b50d9460314f147e21a92c0e462f74faf046718ba
8fb18cbb
filesize=24576
va=0xffffffff
nfuncs=4
totalfuncsize=187
scan_date=2017-06-07-08:11:09

Sample 26 #471

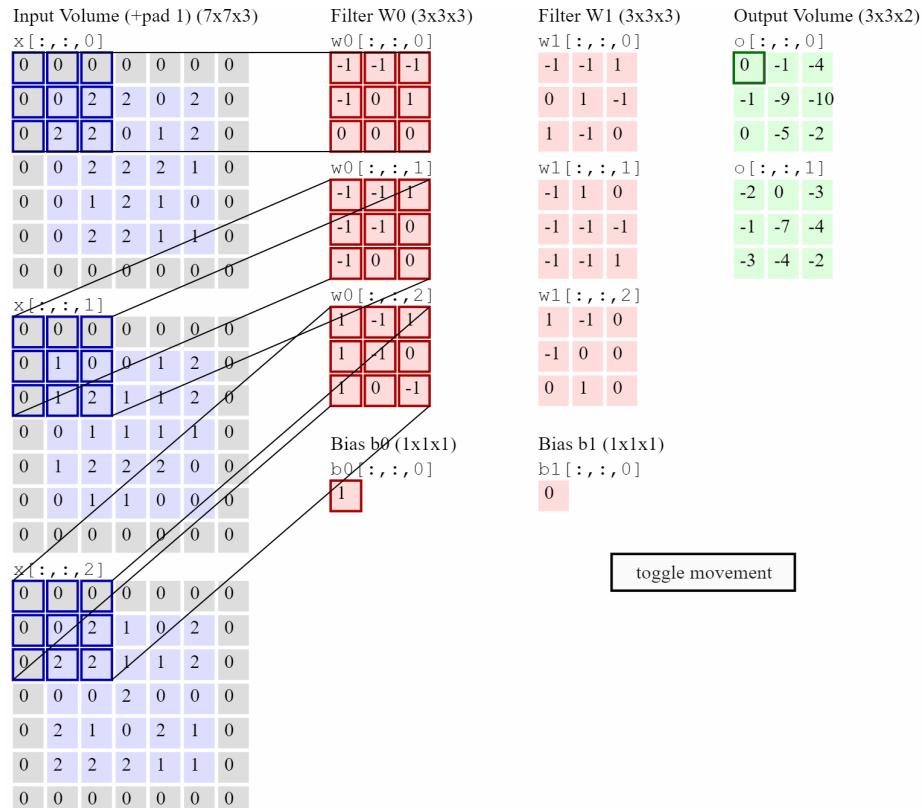
train Mac.OSX.iWorm.F(label=142)
id=6ed282e6e54ccbec80b16bb39fbdf465576215a94ff1b7875fc8b28d
29392fb6
filesize=177044
va=0x18f0
nfuncs=2
totalfuncsize=185
scan_date=2017-07-26-11:35:32

Neural Outbreak Detection

Stacked De-noising Autoencoder (SDA)

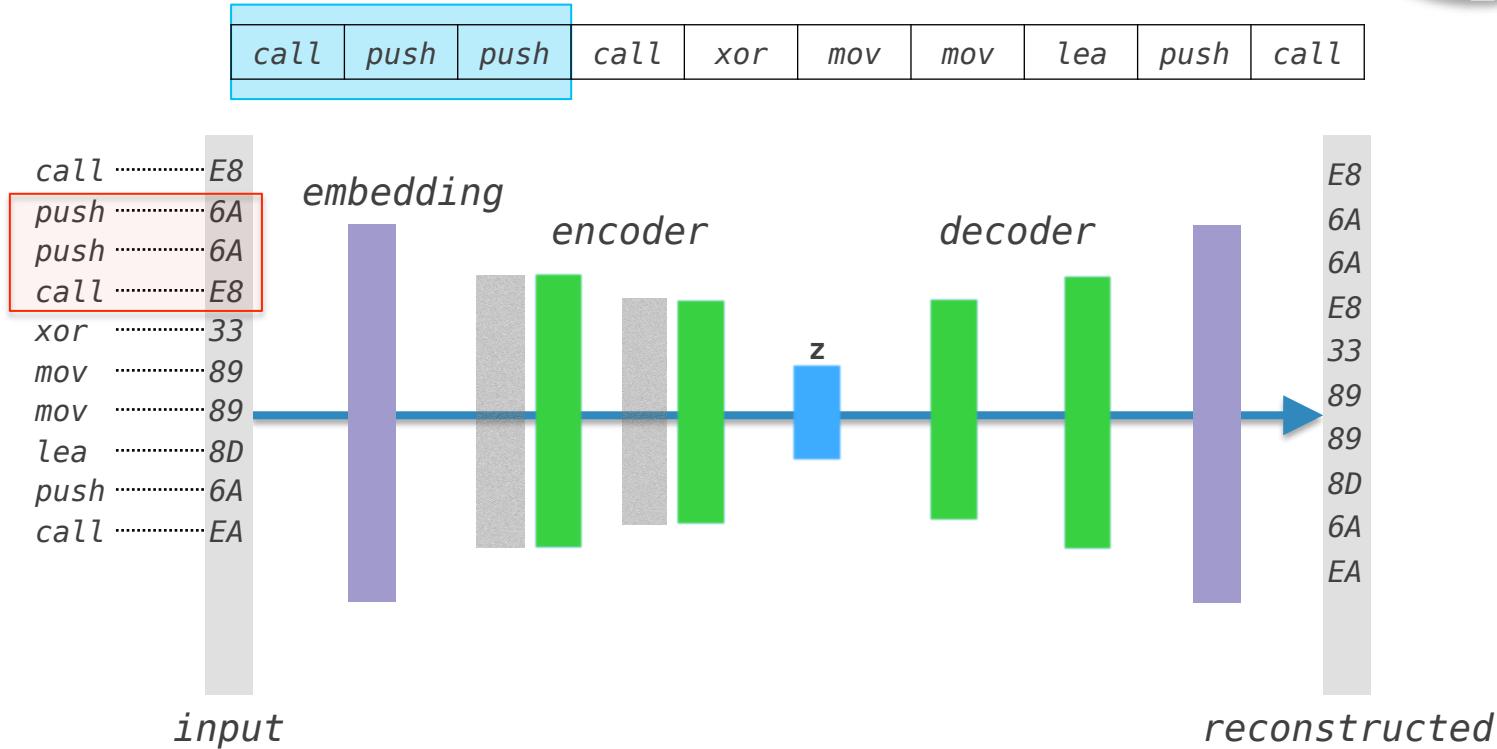


Convolutional Neural Network (CNN)

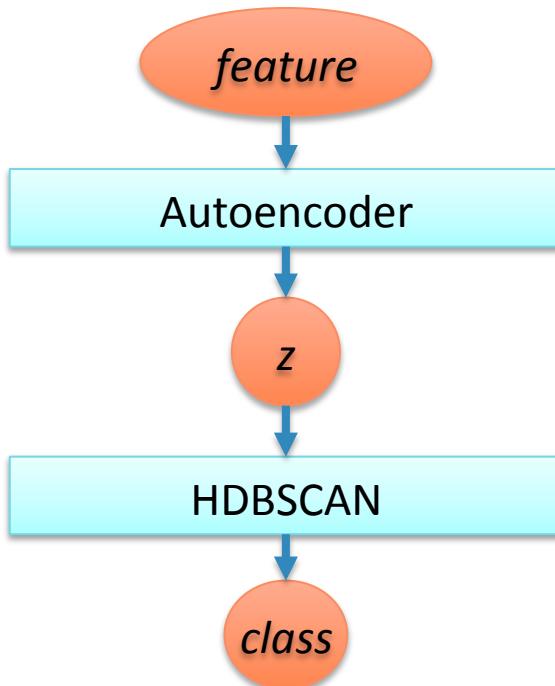


<http://cs231n.github.io/convolutional-networks/>

Sparse Convolutional Autoencoder



Putting them together



call, push, push, xor, mov, mov, lea, ret, ...

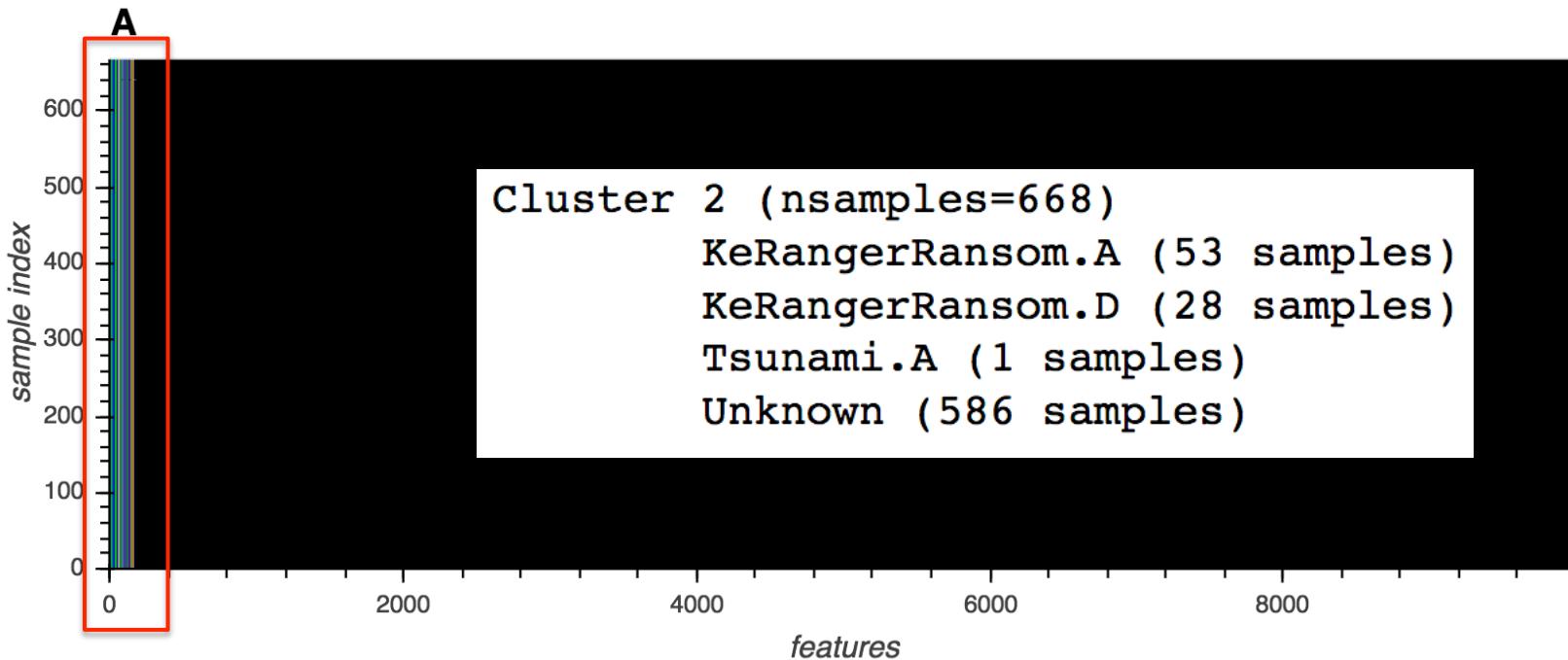
38, 99, 99, 186, 8, 8, 45, 127, ...

1.3, 0.5, -0.91, 2.12, -0.01

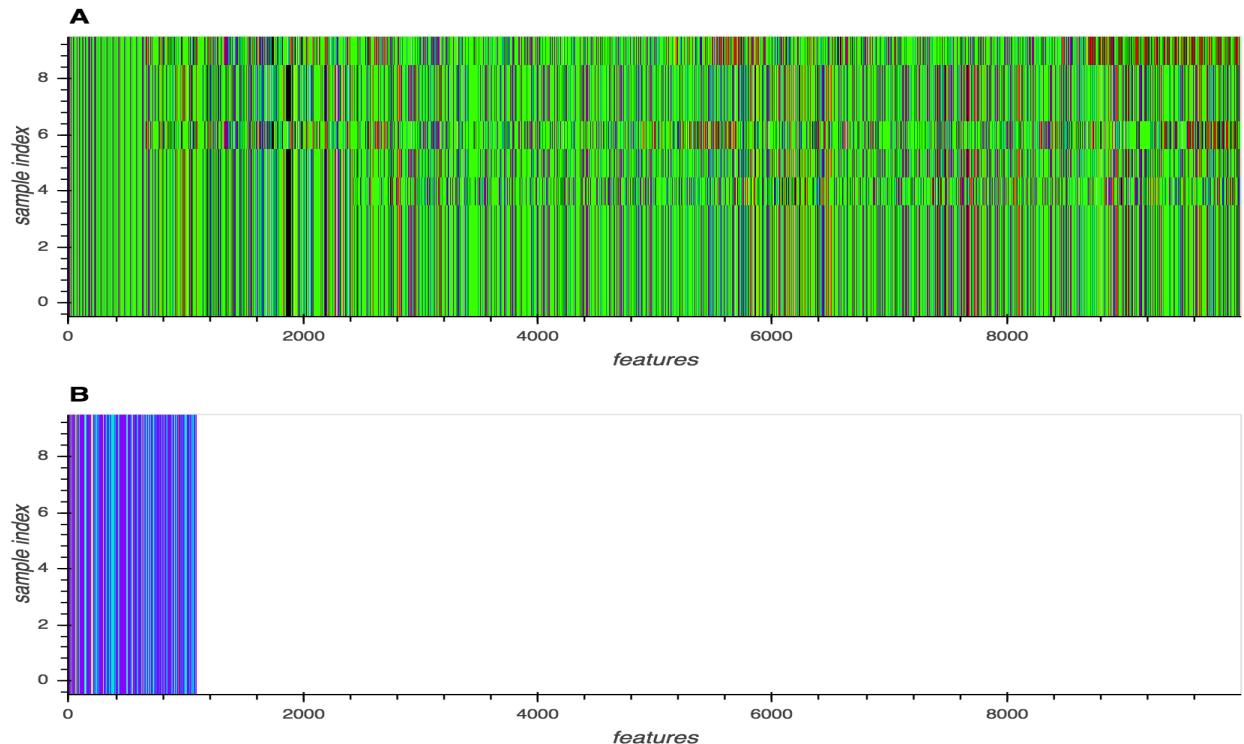
Class 7

Model Performance

Power of Instruction Features

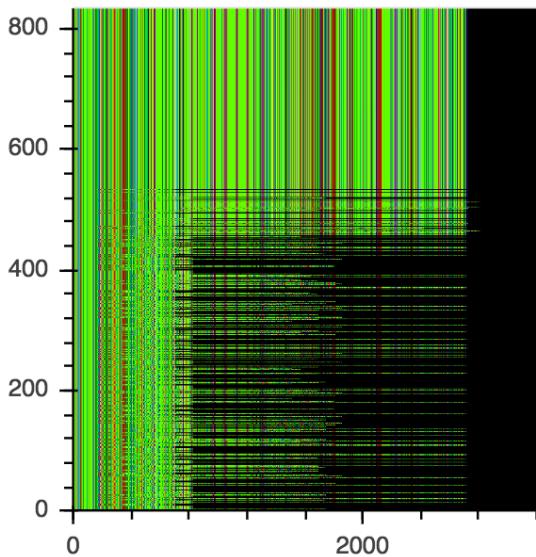
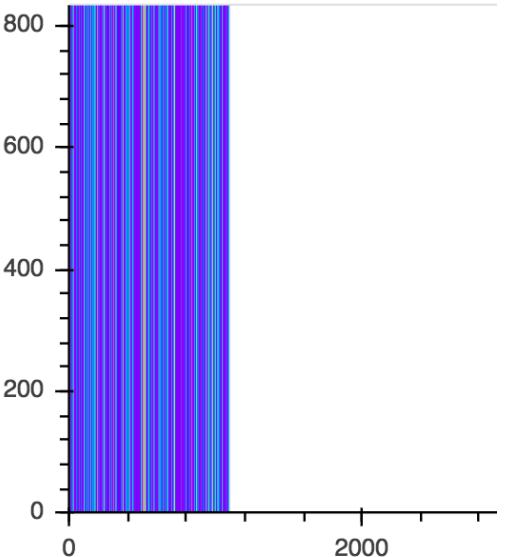


Power of Convolution



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)

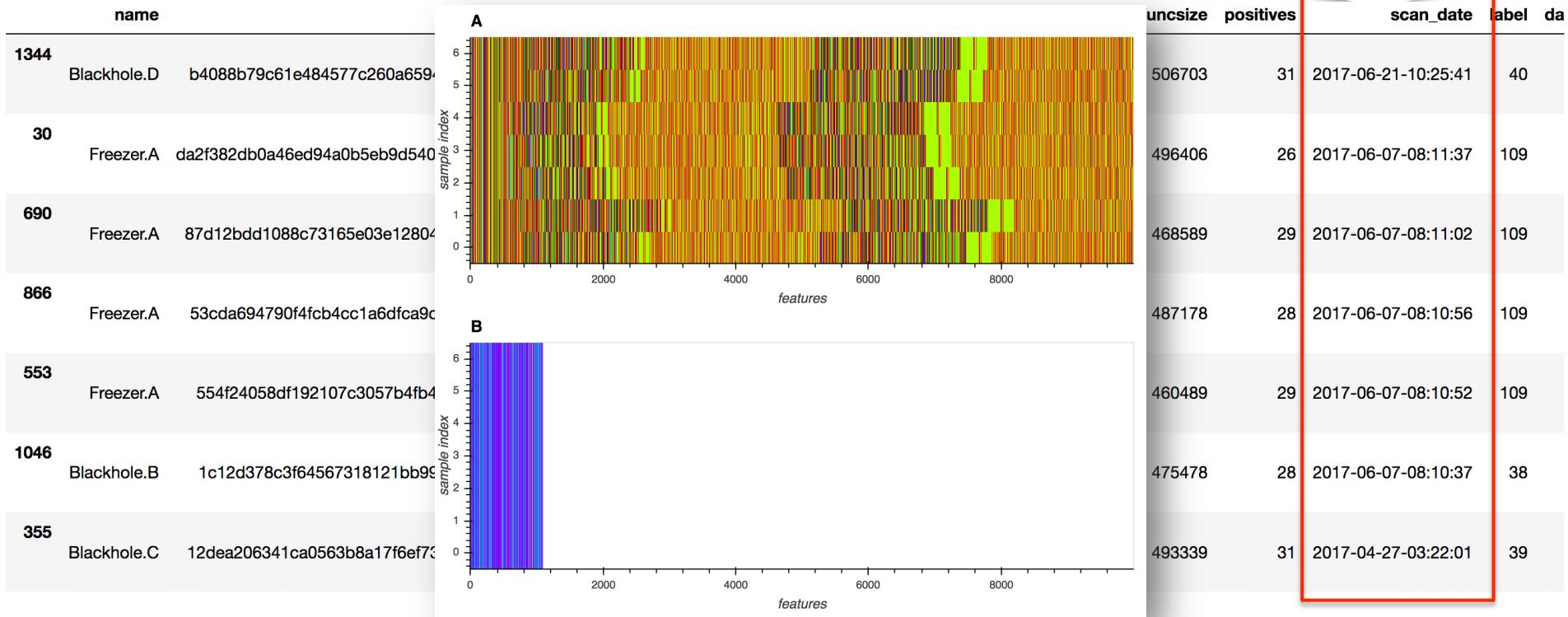
Unknown Sample Detection

**A****B**

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



Apply



- 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

- [1] Silver, David, et al. "Mastering the game of go without human knowledge." *Nature* 550.7676 (2017): 354.
- [2] Anderson, Hyrum S., et al. "Evading machine learning malware detection." *Black Hat* (2017).
- [3] Santos, Igor, et al. "Opcode sequences as representation of executables for data-mining-based unknown malware detection." *Information Sciences* 231 (2013): 64-82.
- [4] Lee, Honglak, et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009.
- [5] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.
- [6] Zhang, Yizhe, et al. "Deconvolutional paragraph representation learning." *Advances in Neural Information Processing Systems*. 2017.
- [7] L. McInnes, J. Healy, S. Astels, *hdbscan: Hierarchical density based clustering* In: *Journal of Open Source Software*, The Open Journal, volume 2, number 11. 2017
- [8] https://en.wikipedia.org/wiki/Locality-sensitive_hashing
- [9] <https://www.youtube.com/watch?v=llg3gGewQ5U>
- [10] <http://cs231n.github.io/convolutional-networks/>
- [11] <https://deeplearning4j.org/convolutionalnetwork>
- [12] <https://docs.gimp.org/en/plug-in-edge.html>

RSA® Conference 2018



#RSAC

QUESTIONS?

Sean Park, Senior Malware Scientist
spark@trendmicro.com