Malware Analysis – Connecting Variants and Versions

Arun Lakhotia
University of Louisiana at Lafayette

Demo

MAGIC Connect – Summary

FOLLOW THS LINK: http://www.virustotal.com/en/arunlakhotia

According to Cythereal's Malware Genomic Correlation (MAGIC) analysis, this file is similar to files with following sha1 hashes:
* 203a5a96357f4de279a9186f0f7845ba94e9ea7a
* b33d63d3b6046f86c8243d7994bf9867bfb49fac
* 156e0969d4fecd167e8010ca9b7315336284eb0e
* d52d8b4e8ec25281c28f8f48ab7e090cc0cce088

For further information, please contact magic@cythereal.com.

Posted 9 hours, 42 minutes ago by arunlakhotia file:ebf6a3bfd5d62b67a8718857881e90d97939ecf837711f36671a40b2d07d2a67

According to Cythereal's Malware Genomic Correlation (MAGIC) analysis, this file is similar to files with following sha1 hashes:
* 354b9fadd20c182adbb88cc1e33a1ee298f6bff0
* 2b1d23d0599c5d463fc56415eaa2bf12bfcb7968
* 4b5375a69bbec811983bd28f03aa1088ab7540b3
* 8e6c7a5fcf56413f30417496cd9e39b0940f431c
* ab99fa5034a17429bea1bfde2d3de97376dd09d5
* 7bad8ef82bc2d2090658885ecee4ab5accb0c8b8
* ae58a88e7747d097ed92496bd848c3019c04bcd2
* d5f7cd760496ca1a57e9e210bd8397a7a956d6d4
* 366483d6bf8fd184679da786800e7871ff916aa4
* 2245e306039555fa5b8a18d96fd4fc40c91113e7

and 16 other files.

For further information, please contact magic@cythereal.com.

MAGIC Connect: Full report

MAGIC Report: 1f1f560c29db6a61b05212eea0e

FOLLOW THIS LINK: http://beta.magic.cythereal.com/report/1f1f560c29db6a61b05212eea0e3c68de0b9d61e

File Info | Magic Matches | Malware Categories | Genomic Features

Show 10 entries

Matched Binary	Match Type	Similarity
2fc845f420939d77101f7b52e0df38bc3c0fe42e	similar_packer_similar_payload	0.9873
30d86ea21f9d259e1ed9c6de370aa9bbe5c553e0	similar_packer_similar_payload	0.9339
325fc074649a6c50b11f0e186a1f2f0f61369ed9	similar_packer_different_payload	0.9077
32bf511bbadf07651ddb9aa4a925e6e0719b67ed	similar_packer_similar_payload	0.9167
531245bf0ccbf9341dca56181388a8864a14eb03	different_packer_similar_payload	0.8997
7ab317c5afb6325463f4fd7f7b4815eab320ef0e	similar_packer_similar_payload	0.9357
7e2aae5b6f88cf297ed29358ac522b452e3deae7	similar_packer_similar_payload	0.9384

MAGIC Report via API

- https://api.magic.cythereal.com/magic/1cf646f9fa78a5c253647dd9220d0502/ff9790d7902fea4c910b182f6e0b00221a40d616/

Find Matching Procedures (via API)

- https://api.magic.cythereal.com/search/procs/1cf646f9fa78a5c253647dd9220d0502/ff9790d7902fea4c910b182f6e0b00221a40d616/0x1000

MAGIC Features, via API

- https://api.magic.cythereal.com/show/proc/1cf646f9fa78a5c253647dd9220d0502/ff9790d7902fea4c910b182f6e0b00221a40d616/0x1000

API Documentation

- https://api.magic.cythereal.com/docs
- http://docs.cythereal.com

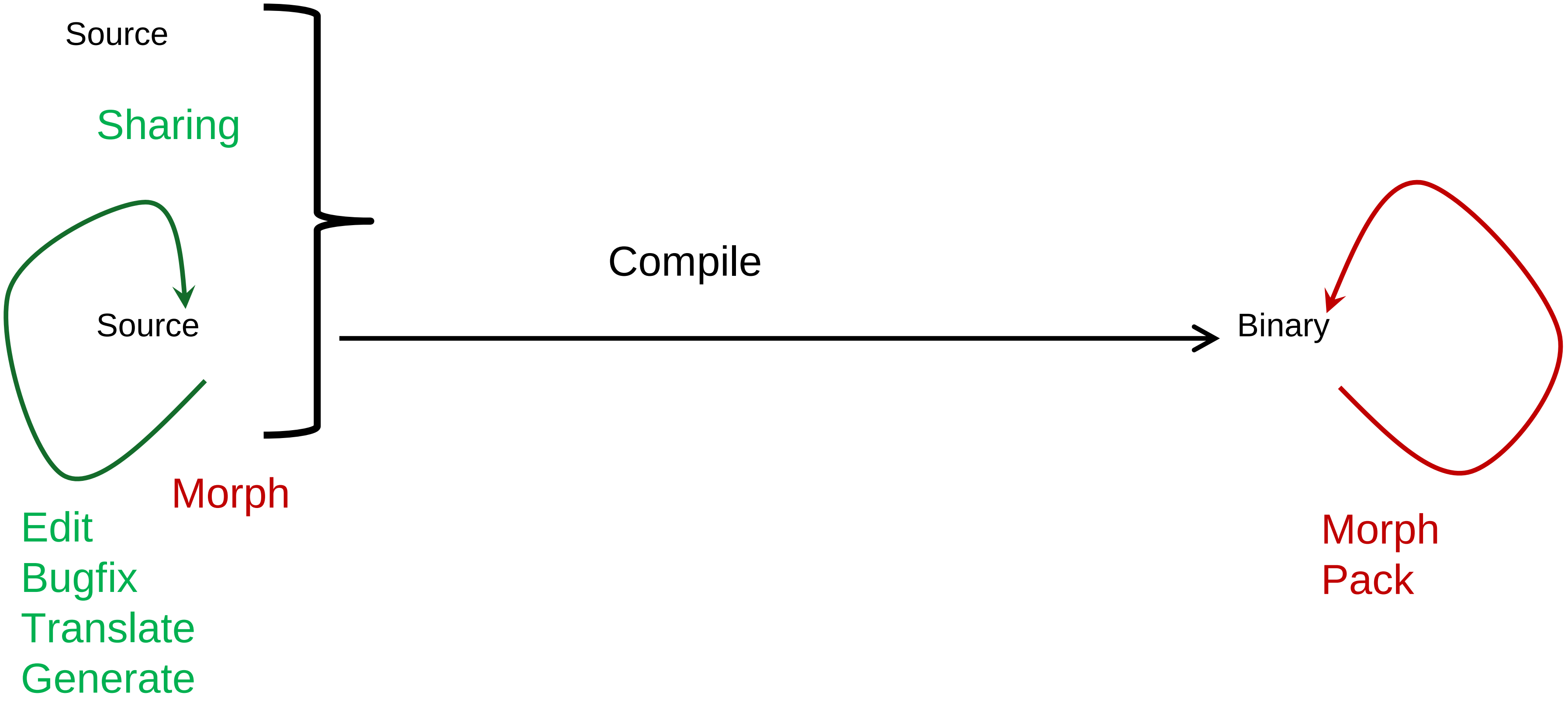
- Other links:
- http://www.virustotal.com/en/arunlakhotia
- http://beta.magic.cythereal.com/

Cythereal MAGIC API Key

- Temporary API Key for ISSISP
  - 1cf646f9fa78a5c253647dd9220d0502
- To get own key:
  - Visit https://api.magic.cythereal.com/docs/
  - Look for "Register"
  - Click on "Try It Out"
  - Fill form, and "Execute"

Problem Definition

Malware (software) Generative Process



Problem

- Given a collection of malware, consisting of VERSIONS and VARIANTS:
  - find malware similar to a given file
  - find functions (disassembled) similar to a given

Challenge: “Undo” Metamorphis

```
mov [ebp - 3], eax
```

→

```
push ecx
mov ecx,ebp
add ecx,33
mov [ecx-36],eax
pop ecx
```

→

```
push ecx
mov ecx,ebp
add ecx,33
push esi
mov esi,ecx
sub esi,34
mov [esi-2],eax
pop esi
pop ecx
```

→

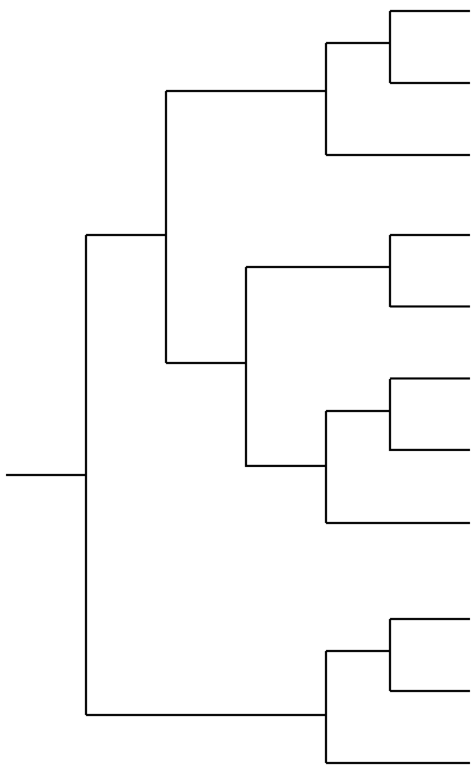
```
push ecx
mov ecx, ebp
push eax
mov eax, 33
add ecx, eax
pop eax

push esi
mov esi, ecx
push edx

mov edx, 34
sub esi, edx
pop edx
mov [esi - 2], eax
pop esi
pop ecx
```

→

```
push ecx
mov ecx, [ebp + 10]
mov ecx, ebp
push eax
add eax, 2342
mov eax, 33
add ecx, eax
pop eax
mov eax, esi
push eax
mov esi, ecx
push edx
xor edx, 778f
mov edx, 34
sub esi, edx
pop edx
mov [esi-2], eax
pop esi
pop ecx
```

BinJuice

Challenge: Similar Binaries

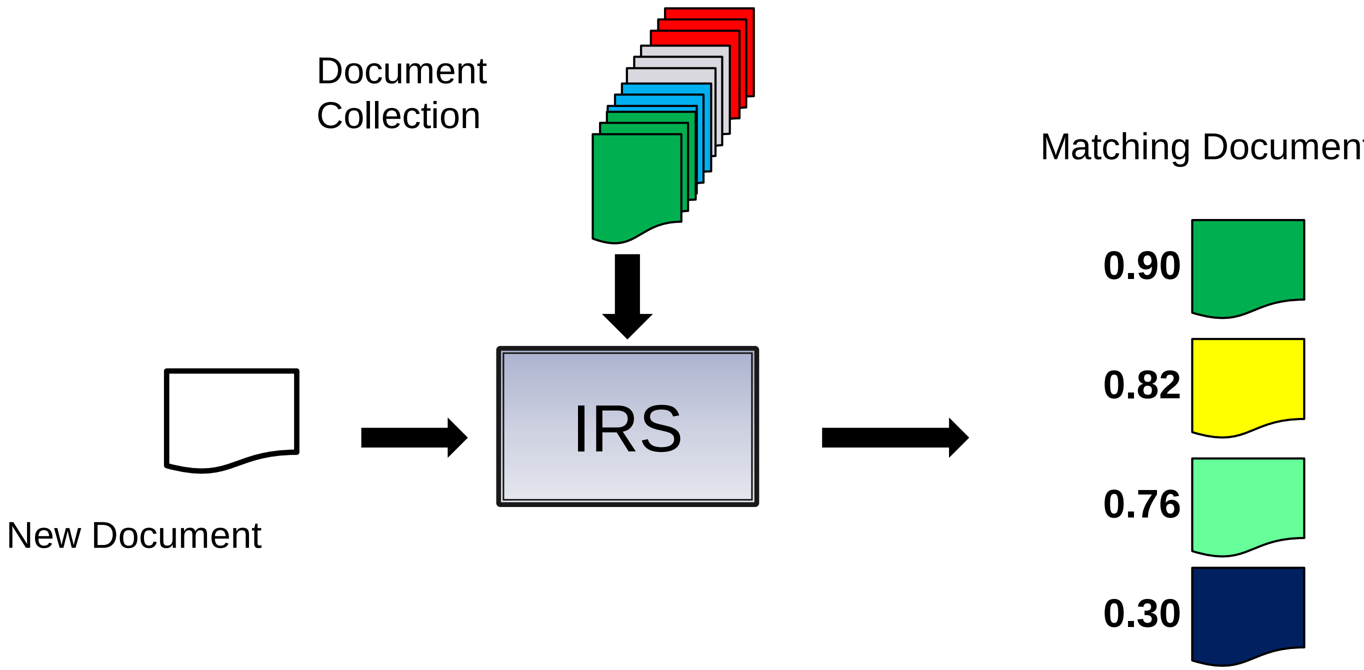


Symantec	McAfee
W32.NetSky.A	W32/NetSky.A
W32.NetSky.B	W32/NetSky.B
W32.NetSky.D	W32/Bugbear.17916intd
W32.Beagle.A@mm	W32/Bagle.a@mm
W32.Beagle.J@mm	W32/Bagle.j@mm
W32.Beagle.AO@mm	W32/Bagle.aq@mm
W32.Beagle.U@mm	W32/Bagle.u@mm
??	
W32.Klez.E@mm.enc	W32/Klez.e@MM
W32.Klez.F@mm	W32/Klez.f@MM
W32.Klez.I@mm	W32/Klez.i@MM

Information Retrieval

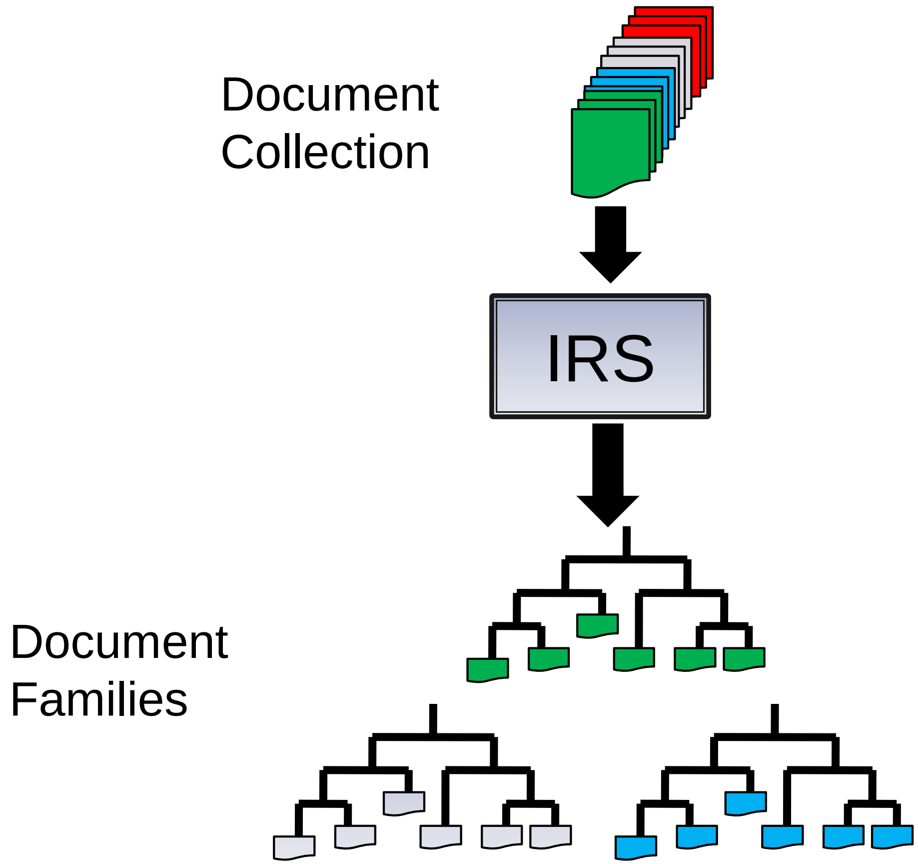
Info Retrieval: Use Case - I

- **Nearest Match (Unsupervised)**



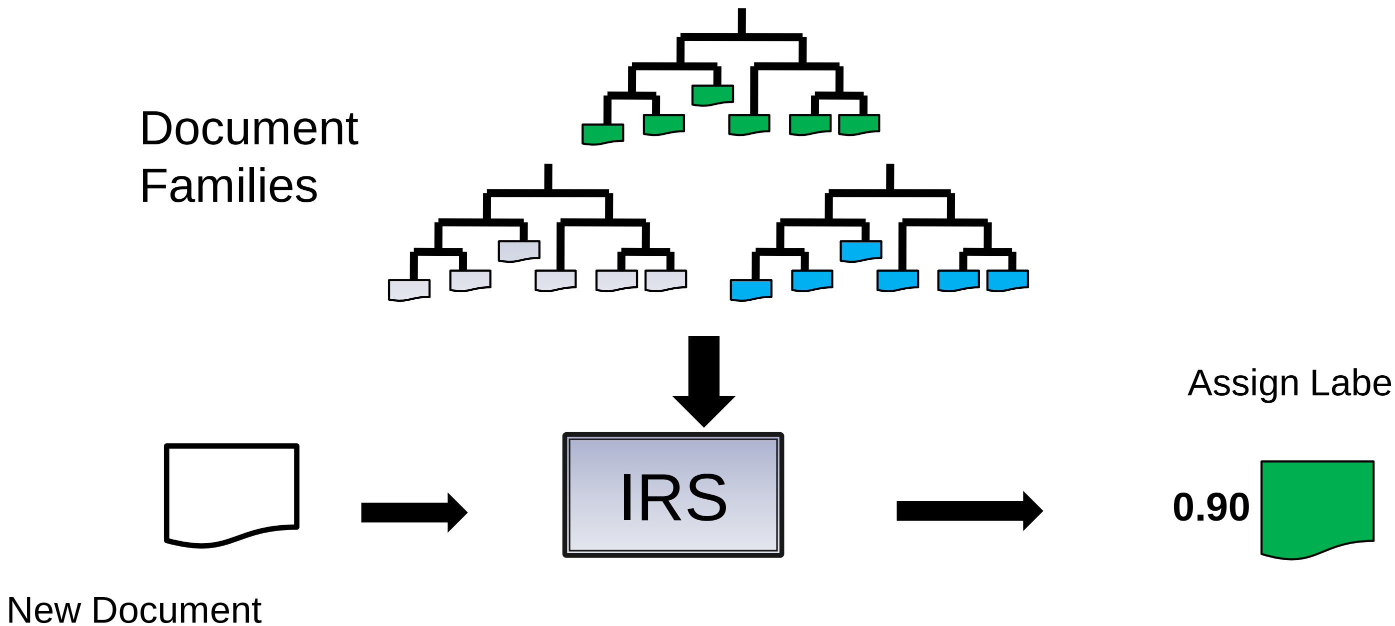
Info Retrieval: Use Case - 2

- **Partition Collection (Unsupervised)**



Info Retrieval: Use Case - 3

- **Match Label (Supervised)**



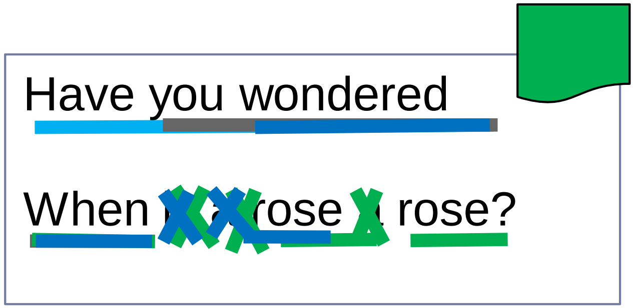
Step 1: Model ‘Documents’

Bag of features model

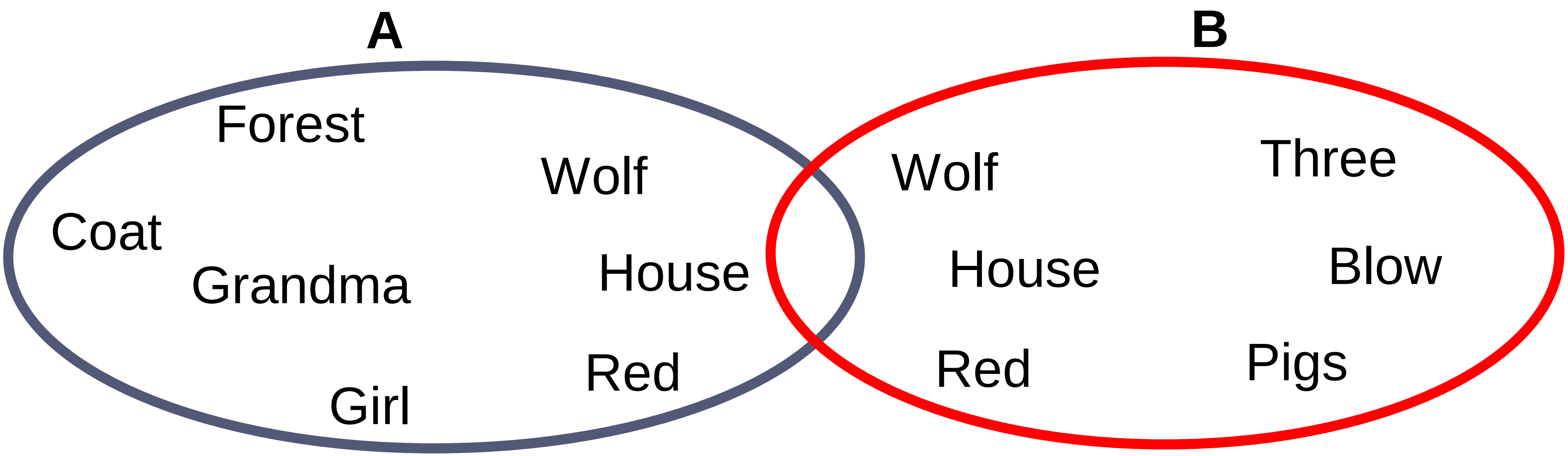
1. Define a method to identify “features”

   Example: k-consecutive words

2. Make a bag of features



Step 2: Define Similarity Function



$$\text{Similarity}(A,B) = |A \cap B| \,/\, |A \cup B|$$

$$= 3 / 10$$

$$= 0.3$$

Alternate: Vector Space Model

Vector Space: Ordered list of ALL of the words in ALL of the documents:

Blow x Coat x Forest x Girl x Grandma x House x Pigs x Red x Three x Wolf

Vector: A Boolean vector representing presence/absence of a word

A

[0, 1, 1, 1, 1, 1, 0, 1, 0, 1]

B

[1, 0, 0, 0, 0, 1, 1, 1, 1, 1]

Distance: Euclidian Distance between two points.

Benefits: Can use vector processors (Nvidia, Google Tensorflow)
Cons: Very, very large vectors

Step 3: Choose/create algorithm

- Supervised Learning
  - Neural Networks
  - Bayesian Statistics
  - Inductive Learning
  - Support Vector Machines
  - Regression
- Unsupervised Learning
  - K-Means Clustering
  - Hierarchical Clustering
  - K-Nearest Neighbor
- Semi-supervised
  - Use some labels to seed clusters

Modeling Malware as Documents

Modeling Malware as Documents

- Create a bag of features of binaries
  - such that `similar' programs have `similar' bags
- Similar programs:
  - Related through code evolution
    - New capability, bug fixes
    - Code reuse, shared libraries, shared strategies
    - Stealth – deliberate attempt to hide similarity

Malware Document: Byte N-gram

```
00000000  3b 00 91 df 96 f6 33 73  7f f1 de 13 a2 8a 45 30
00000010  f6 01 ff e2 52 43 15 4e  1c a9 bf 9a 1c 41 8b 40
00000020  fa 14 30 24 2f ed bc 00  7d 46 4c 32 03 f2 ba 69
00000030  dd f5 28 87 84 20 61 f5  c9 3a 54 c2 98 9e c1 11
00000040  20 df 23 16 22 64 71 90  c1 2c 7c 1e 68 0e e2 28
00000050  66 b8 d2 05 2e e7 75 11  1b c8 4e 4c d4 9b 4a 8b
00000060  69 75 fb de 05 b3 4f f2  dc 26 04 4a 02 2a 2c 56
00000070  55 ef 93 07 e6 a3 2f 01  4a d9 75 3d b8 2b 13 f1
00000080  a3 30 7d c5 e2 0f 69 16  03 21 51 0e b5 d5 08 98
00000090  3e ca c5 22 5f b0 d4 3d  2e 78 11 92 99 66 24 5a
000000a0  56 96 74 41 cd 41 91 d4  02 65 ca 20 3e 1c a4 c1
000000b0  c9 b6 e9 aa 89 89 40 e4  66 c4 d4 3f 49 85 e5 66
000000c0  56 82 93 f9 94 87 15 9c  2f 46 08 30 01 79 28 e3
000000d0  41 e7 29 24 ad 21 0a 4b  e0 79 ea 7f fd 4b ec 10
000000e0  a9 b8 23 96 69 17 a9 4e  8b 13 0d 5c 4c 28 28 f2
000000f0  ae e7 6e d8 e8 54 7e 15  da 51 2d 38 00 5f 59 26
```

Word = N-Bytes

(380091df)
(0091df96)
(91df96f6)
(df96f633)

Malware Document: Abstracted Bytes

```
00000000  3b 00 91 df 96 f6 33 73  7f f1 de 13 a2 8a 45 30
00000010  f6 01 ff e2 52 43 15 4e  1c a9 bf 9a 1c 41 8b 40
00000020  fa 14 30 24 2f ed bc 00  7d 46 4c 32 03 f2 ba 69
00000030  dd f5 28 87 84 20 61 f5  c9 3a 54 c2 98 9e c1 11
00000040  20 df 23 16 22 64 71 90  c1 2c 7c 1e 68 0e e2 28
00000050  66 b8 d2 05 2e e7 75 11  1b c8 4e 4c d4 9b 4a 8b
00000060  69 75 fb de 05 b3 4f f2  dc 26 04 4a 02 2a 2c 56
00000070  55 ef 93 07 e6 a3 2f 01  4a d9 75 3d b8 2b 13 f1
00000080  a3 30 7d c5 e2 0f 69 16  03 21 51 0e b5 d5 08 98
00000090  3e ca c5 22 5f b0 d4 3d  2e 78 11 92 99 66 24 5a
000000a0  56 96 74 41 cd 41 91 d4  02 65 ca 20 3e 1c a4 c1
000000b0  c9 b6 e9 aa 89 89 40 e4  66 c4 d4 3f 49 85 e5 66
000000c0  56 82 93 f9 94 87 15 9c  2f 46 08 30 01 79 28 e3
000000d0  41 e7 29 24 ad 21 0a 4b  e0 79 ea 7f fd 4b ec 10
000000e0  a9 b8 23 96 69 17 a9 4e  8b 13 0d 5c 4c 28 28 f2
000000f0  ae e7 6e d8 e8 54 7e 15  da 51 2d 38 00 5f 59 26
```

Disassemble

```
10019fe:   74 10            je     0x1001a10
1001a00:   6a 10            push   0x10
1001a02:   bf 98 18 00 01   mov    edi,0x1001898
1001a07:   59               pop    ecx
1001a08:   8b f0            mov    esi,eax
1001a0a:   33 db            xor    ebx,ebx
1001a0c:   f3 a6            repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]
1001a0e:   75 0a            jne    0x1001a1a
1001a10:   8b 45 08         mov    eax,DWORD PTR [ebp+8]
1001a13:   8b c8            mov    ecx,eax
1001a15:   8d 70 08         lea    esi,[eax+8]
1001a18:   eb 5c            jmp    0x1001a76
1001a1a:   6a 10            push   0x10
1001a1c:   bf f8 18 00 01   mov    edi,0x10018f8
1001a21:   59               pop    ecx
1001a22:   8b f0            mov    esi,eax
1001a24:   33 db            xor    ebx,ebx
1001a26:   f3 a6            repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]
1001a28:   75 0a            jne    0x1001a34
1001a2a:   8b 45 08         mov    eax,DWORD PTR [ebp+8]
1001a2d:   8b c8            mov    ecx,eax
1001a2f:   8d 70 0c         lea    esi,[eax+12]
1001a32:   eb 42            jmp    0x1001a76
```

Zap Address bytes

```
74 10
6a 10
bf [redacted]
59
8b f0
33 db
f3 a6
75 0a
8b 45 08
8b c8
8d 70 08
eb 5c
6a 10
bf [redacted]
59
8b f0
33 db
f3 a6
75 0a
8b 45 08
8b c8
8d 70 0c
eb 42
```

Word = N-Bytes of Abstracted Bytecode

Malware Document: Mnemonics

```
00000000  3b 00 91 df 96 f6 33 73  7f f1 de 13 a2 8a 45 30
00000010  f6 01 ff e2 52 43 15 4e  1c a9 bf 9a 1c 41 8b 40
00000020  fa 14 30 24 2f ed bc 00  7d 46 4c 32 03 f2 ba 69
00000030  dd f5 28 87 84 20 61 f5  c9 3a 54 c2 98 9e c1 11
00000040  20 df 23 16 22 64 71 90  c1 2c 7c 1e 68 0e e2 28
00000050  66 b8 d2 05 2e e7 75 11  1b c8 4e 4c d4 9b 4a 8b
00000060  69 75 fb de 05 b3 4f f2  dc 26 04 4a 02 2a 2c 56
00000070  55 ef 93 07 e6 a3 2f 01  4a d9 75 3d b8 2b 13 f1
00000080  a3 30 7d c5 e2 0f 69 16  03 21 51 0e b5 d5 08 98
00000090  3e ca c5 22 5f b0 d4 3d  2e 78 11 92 99 66 24 5a
000000a0  56 96 74 41 cd 41 91 d4  02 65 ca 20 3e 1c a4 c1
000000b0  c9 b6 e9 aa 89 89 40 e4  66 c4 d4 3f 49 85 e5 66
000000c0  56 82 93 f9 94 87 15 9c  2f 46 08 30 01 79 28 e3
000000d0  41 e7 29 24 ad 21 0a 4b  e0 79 ea 7f fd 4b ec 10
000000e0  a9 b8 23 96 69 17 a9 4e  8b 13 0d 5c 4c 28 28 f2
000000f0  ae e7 6e d8 e8 54 7e 15  da 51 2d 38 00 5f 59 26
```

Disassemble

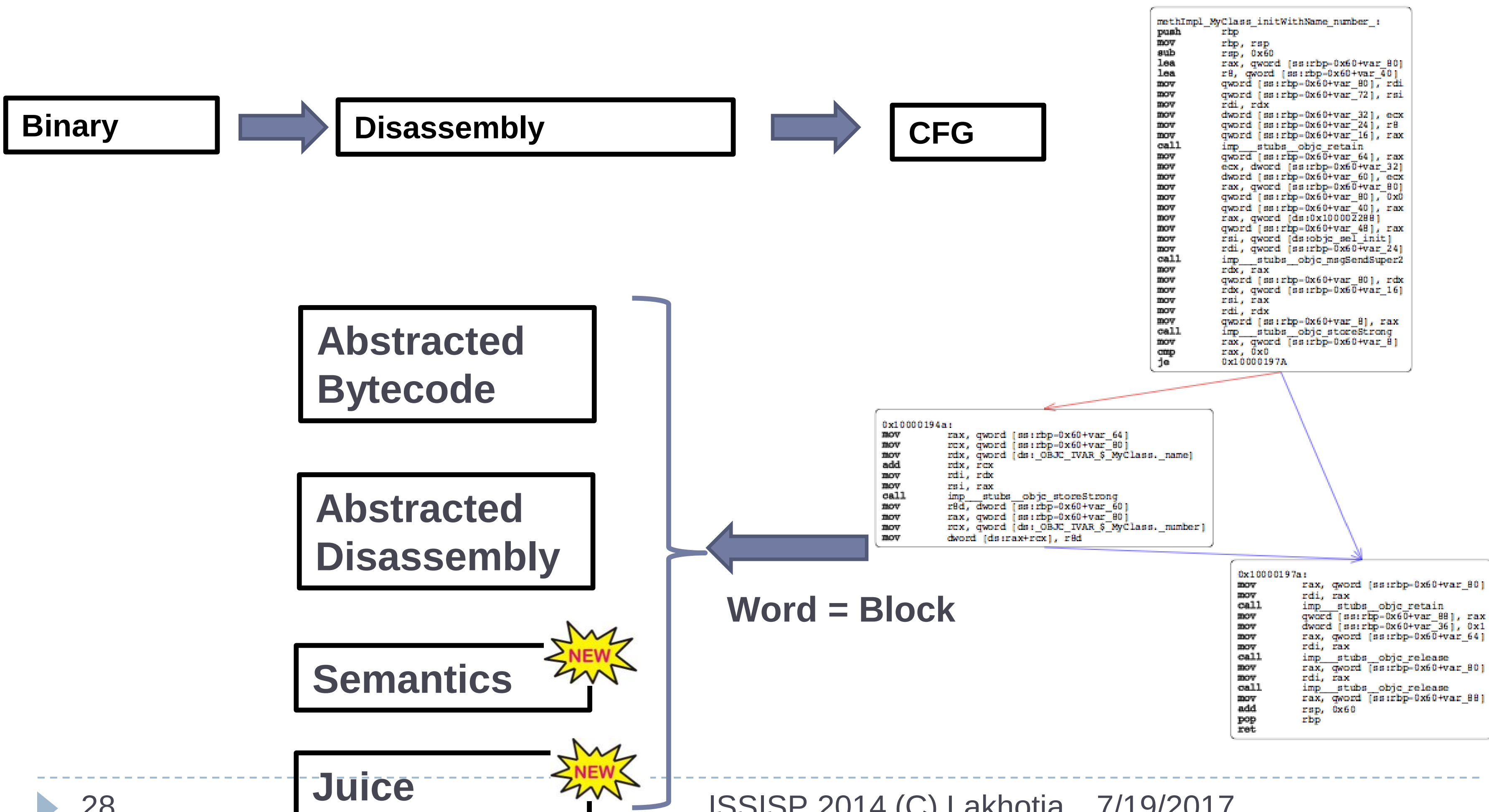
```
10019fe:   74 10            je    0x1001a10
1001a00:   6a 10            push  0x10
1001a02:   bf 98 18 00 01   mov   edi,0x1001898
1001a07:   59               pop   ecx
1001a08:   8b f0            mov   esi,eax
1001a0a:   33 db            xor   ebx,ebx
1001a0c:   f3 a6            repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]
1001a0e:   75 0a            jne   0x1001a1a
1001a10:   8b 45 08         mov   eax,DWORD PTR [ebp+8]
1001a13:   8b c8            mov   ecx,eax
1001a15:   8d 70 08         lea   esi,[eax+8]
1001a18:   eb 5c            jmp   0x1001a76
1001a1a:   6a 10            push  0x10
1001a1c:   bf f8 18 00 01   mov   edi,0x10018f8
1001a21:   59               pop   ecx
1001a22:   8b f0            mov   esi,eax
1001a24:   33 db            xor   ebx,ebx
1001a26:   f3 a6            repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]
1001a28:   75 0a            jne   0x1001a34
1001a2a:   8b 45 08         mov   eax,DWORD PTR [ebp+8]
1001a2d:   8b c8            mov   ecx,eax
1001a2f:   8d 70 0c         lea   esi,[eax+12]
1001a32:   eb 42            jmp   0x1001a76
```

Word = N-mnemonic

(je push)
(push mov)
(mov pop)
(pop xor)

Variation: N-perm

Malware Document: using semantics



Code to Semantics

Code
- Sequential
- Focus on operations

Semantics
- Parallel
- Captures affect

```
push ebp
mov  ebp,esp
sub  esp,4
mov  eax, DWORD ebp+4
mov  DWORD ebp+8,eax
mov  eax, DWORD ebp
mov  DWORD ebp-4,eax
```

```
                   eax = def(ebp)
                   ebp = -4+def(esp)
                   esp = -8+def(esp)
memdw(-8+def(esp))= def(ebp)
memdw(-4+def(esp))= def(ebp)
memdw(4+def(esp))  = def(memdw(def(esp)))
```

Concrete Semantics

Interpret

Instruction → State → State

Interpret

add ax, bx →

ax = 10
bx = 20
cx = 30
…
M[4000] = 50045
M[4004] = 20
M[4008] = 30
…

→

ax = **30**
bx = 20
cx = 30
…
M[4000] = 50045
M[4004] = 20
M[4008] = 30
…

Symbolic Semantics

Sym Interpret

Instruction → SymState → SymState

Sym Interpret

add ax, bx →

ax = **def(ax)**
bx = **20**
cx = **def(cx)**
…
M[4000] = **def(cx)**
M[4004] = 5005
M[4008] = **def(4008)**
…

→

ax = **def(ax)+20**
bx = 20
cx = def(cx)
…
M[4000] = def(cx)
M[4004] = 5005
M[4008] = def(4008)
…

Symbolic Semantics: Formal Sketch

Interpret: seq(Instruction) -> State -> State

where:

State = LValue -> RValue

LValue = Register + Mem

RValue = Number

+ def(RValue)

Previous state

+ RValue op Rvalue

+ op RValue

Unsimplified

Algebraic Simplification

- Num **op** Num => Num
- **op** Num => Num

Evaluate

- Expr **+** Num => Num **+** Expr
- Expr * Num => Num * Expr

Commute

- Exp1 * (Exp2 **+** Exp3) => Exp1 * Exp2 **+** Exp1 * Exp3

Distribute

- Exp1 **shift-right** Num => Exp1 * 2^Num

Equivalent

Semantic matches

```
mov(ecx,ebp)
sub(ecx,63)
mov(dptr(ecx+59),eax)
pop(ecx)
lea(eax,wptr(ebp-28))
push(edi)
mov(edi,1148415812)
```

=

```
push(esi)
mov(esi,-1545600507)
or(ecx,esi)
pop(esi)
push(edi)
mov(edi,ebp)
mov(ecx,edi)
pop(edi)
push(eax)
mov(eax,63)
sub(ecx,eax)
pop(eax)
mov(dptr(ecx+59),eax)
pop(ecx)
lea(eax,wptr(ebp-28))
push(edi)
mov(edi,880280128)
push(esi)
mov(esi,268135684)
add(edi,esi)
pop(esi)
```

Semantic matches

cmp(bptr(esi),al)	=	push(edx) mov(dl,al) cmp(bptr(esi),dl) pop(edx)
mov(ebx,1684957510)	=	mov(ebx,251658400) xor(ebx,1802398182)
mov(bptr(edi),al)	=	push(ecx) mov(cl,al) mov(bptr(edi),cl) pop(ecx)
mov(cl,0)	≠	mov(ecx,1342369920) mov(cl,69) sub(cl,69)]
cmp(al,0)	=	push(ebx) mov(bh,0) cmp(al,bh) pop(ebx)

Semantics to Word

esp = -8+def(esp)
eax = def(ebp)
memdw(-4+def(esp))= def(ebp)
memdw(4+def(esp)) = 20 + def(eax)
memdw(-8+def(esp))= def(ebp)
ebp = -4+def(esp)

memdw(-4+def(esp))= def(ebp)
ebp = -4+def(esp)
memdw(-8+def(esp))= def(ebp)
eax = def(ebp)
memdw(4+def(esp)) = def(eax) + 20
esp = -8+def(esp

SORT

eax = def(ebp)
ebp = -4+def(esp)
esp = -8+def(esp)
memdw(-8+def(esp))= def(ebp)
memdw(-4+def(esp))= def(ebp)
memdw(4+def(esp)) = def(eax) + 20

eax = def(ebp)
ebp = -4+def(esp)
esp = -8+def(esp)
memdw(-8+def(esp))= def(ebp)
memdw(-4+def(esp))= def(ebp)
memdw(4+def(esp)) = def(eax) + 20

HASH

0da5678afdgfh732

0da5678afdgfh732

Semantics to 'words'

- Challenge:
  - How to map equal semantics to the same `word'?

- Solution:
  - Define canonical ordering
    - RValue structures are ground
    - Use ordering over symbols
    - Account for commutativity
    - Sum-of-product form
    - Simplify
  - Word = Hash (md5, SHA1) of linearized semantics

RValue = Number
+ def(RValue)
+ RValue op Rvalue
+ op RValue

Limitations of (Block) Semantics

Should these be considered similar?

```
mov   bh, 0
cmp   al, bh
```

```
bh   = 0
flag = def(al) < 0
```

```
mov   ch, 0
cmp   al, ch
```

```
ch   = 0
flag = def(al) < 0
```

They produce different hash.
Determining similarity would be expensive.

Limitations of (Block) Semantics

- Does not capture:
  - Register renaming
  - Memory address reassignment
  - Code motion between blocks
  - Evolutionary changes
    - Hashes good for strict equality
- Solution:
  - Generalize semantics
    - Juice
  - Use n-Block semantics
  - Use fuzzy hashes

Generalized Semantics (aka Juice)

```
mov   bh, 0
cmp   al, bh
```

bh = 0
flag = def(al) < 0

A = N
B = def(C) < N

```
mov   ch, 0
cmp   al, ch
```

ch = 0
flag = def(al) < 0

A = 0
B = def(C) < N

Generalized Semantics

code

```
push ebp
mov  ebp,esp
sub  esp,4
mov  eax, DWORD ebp+4
mov  DWORD ebp+8,eax
mov  eax, DWORD ebp
mov  DWORD ebp-4,eax
```

semantics

```
                eax = def(ebp)
                ebp = -4+def(esp)
                esp = -8+def(esp)
memdw(-8+def(esp))= def(ebp)
memdw(-4+def(esp))= def(ebp)
memdw(4+def(esp))  = def(memdw(def(esp)))
```

- **Inductive Generalization**
  Replace registers and constants by variables

gen_semantics

```
                A = def(B),
                B = N2+def(C),
                C = N2+def(C),
memdw(E+def(C)) = def(B)
memdw(D+def(C)) = def(B)
memdw(F+def(C)) = def(memdw(def(C)))
        where  A, B, C are 'registers'
               N1 and N2 are 'Int'
```

Problem Hashing Juice

eax = 20
ebx = 40
mem(def(eax)) = def(ebx) + 30
mem(def(ebx)) = def(eax)

R1 = N1
R2 = N2
mem(def(R1)) = def(R2) **+ N3**
mem(def(R2)) = def(R1)

ebx = 40
ecx = 20
mem(def(ebx)) = def(ecx)
mem(def(ecx)) = def(ebx) + 30

R1 = N1
R2 = N2
mem(def(R1)) = def(R2)
mem(def(R2)) = def(R1) **+ N3**

Logically similar, but different hash

R1 = N1
R2 = N2
mem(def(R1)) = def(R2) **+ N3**
mem(def(R2)) = def(R1)

Hashing Juice

- **Challenge**:
  - Juice is non-ground
  - Variables are unordered
  - Similar juice may have different hash

May be reordered

```
R1 = N1
R2 = N2
mem(def(R1)) = def(R2)
mem(def(R2)) = def(R1)
```

Solution:
Unify variant terms

```
R=N
mem(def(R)) = def(N)
```

```
JRValue = Number
        + def(RValue)
        + RValue op Rvalue
        + op RValue
        + Variable
```

Juice after Unifying Variants

eax = 20
ebx = 40
mem(def(eax)) = def(ebx) + 30
mem(def(ebx)) = def(eax)

ebx = 20
ecx = 40
mem(def(ebx)) = def(ecx)
mem(def(ecx)) = def(ebx) + 30

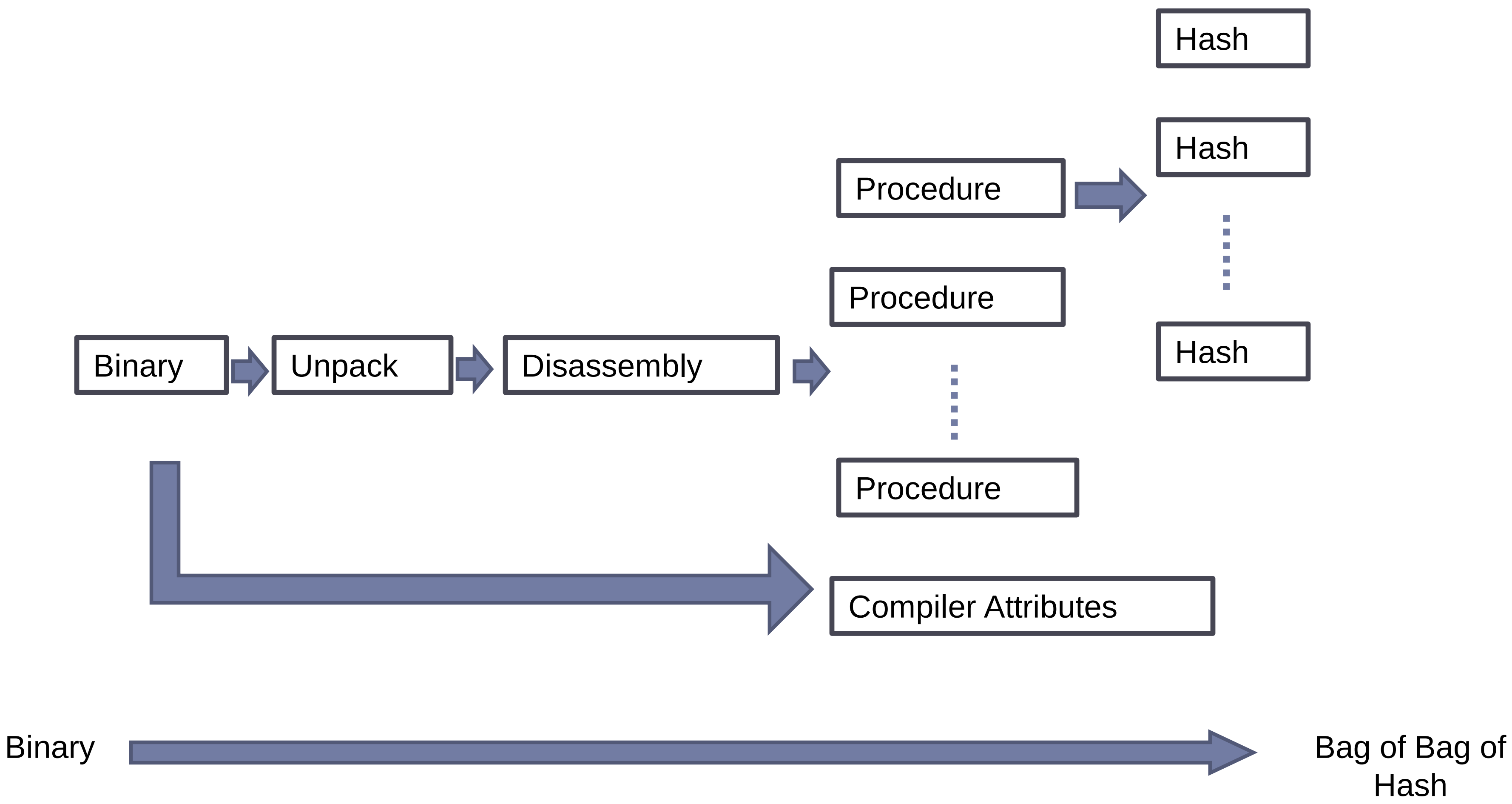
R1 = N1
R2 = N2
mem(def(R1)) = def(R2) **+ N3**
mem(def(R2)) = def(R1)

R1 = N1
R2 = N2
mem(def(R1)) = def(R2)
mem(def(R2)) = def(R1) **+ N3**

Loss of semantics, same hash

dup(R1 = N1, 2)
mem(def(R1)) = def(R1)
mem(def(R1)) = def(R1) **+ N3**

Malware as Document



APPLICATION

Cyber Threat Intelligence

- "Network defense techniques
- that leverage knowledge about the adversaries
- and decrease an adversary's likelihood of success"
- with each subsequent intrusion attempt."
- Cyber Squared Inc, 2013.

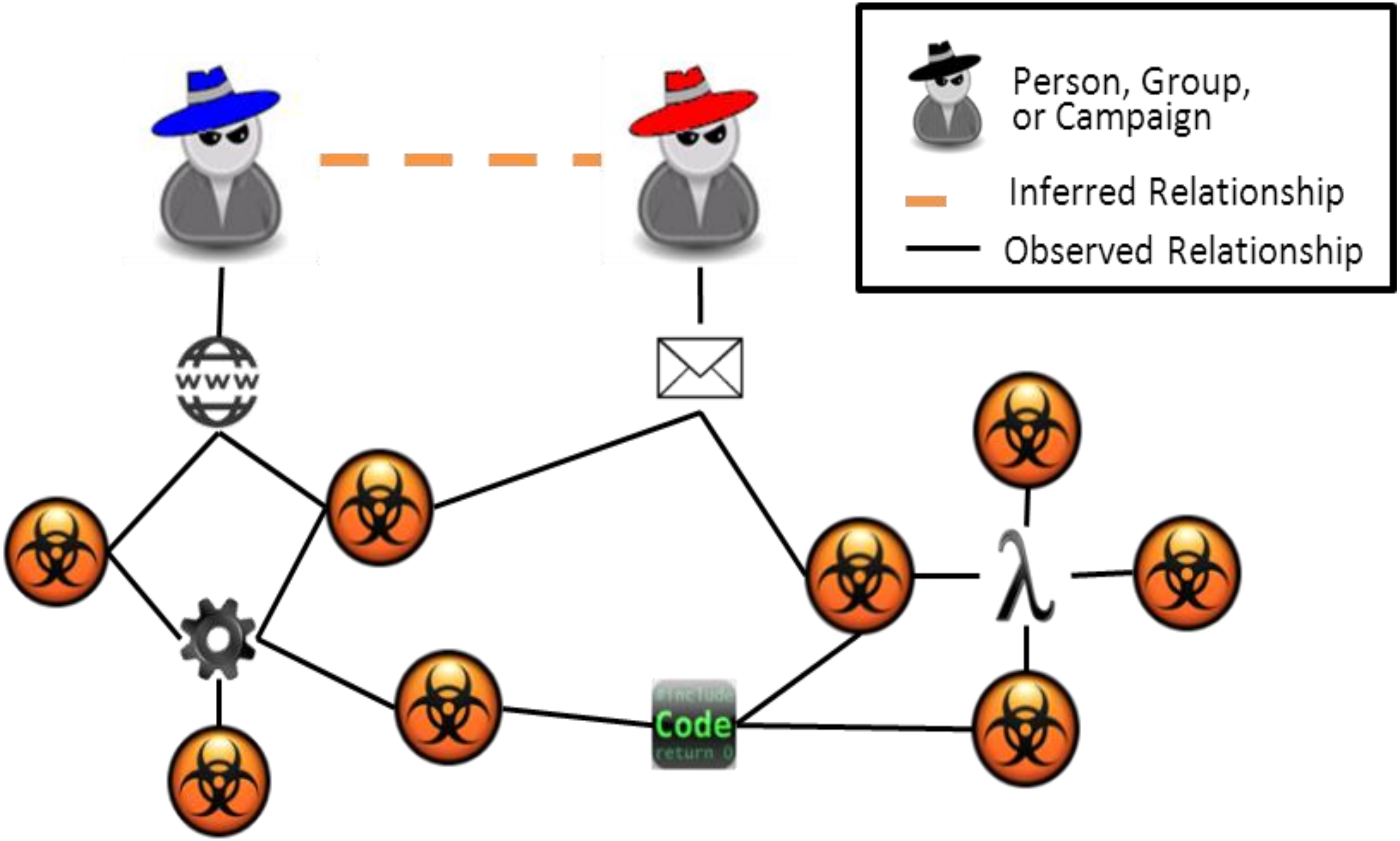
LEARN FROM AN ADVERSARY'S ATTEMPTS

Malware Intelligence

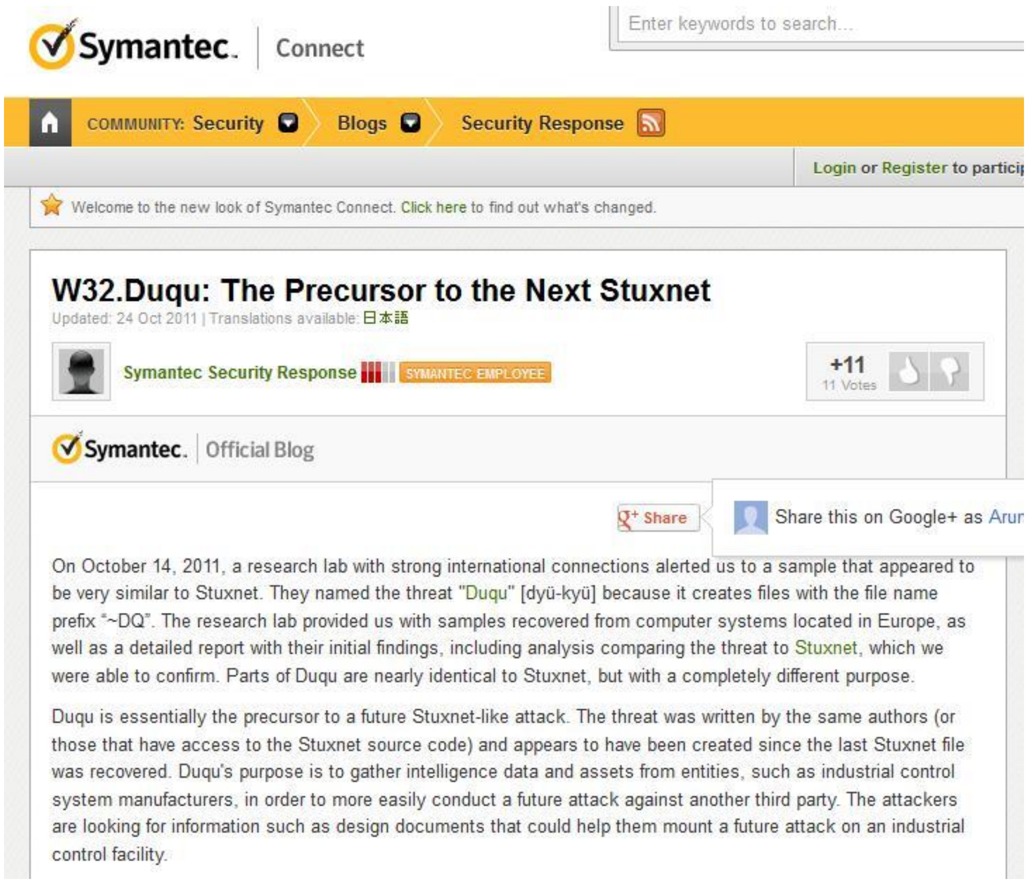
- MALWARE [ANALYSIS DRIVEN CYBER THREAT] INTELLIGENCE

LEARN FROM AN ADVERSARY’S MALWARE

Connecting Actors from Malware



Code connects Actors



Symantec. | Connect

Enter keywords to search...

COMMUNITY: Security Blogs Security Response

Login or Register to particip

Welcome to the new look of Symantec Connect. Click here to find out what's changed.

W32.Duqu: The Precursor to the Next Stuxnet

Updated: 24 Oct 2011 | Translations available: 日本語

Symantec Security Response SYMANTEC EMPLOYEE

+11 11 Votes

Symantec. Official Blog

Share Share this on Google+ as Arun

On October 14, 2011, a research lab with strong international connections alerted us to a sample that appeared to be very similar to Stuxnet. They named the threat "Duqu" [dyü-kyü] because it creates files with the file name prefix "~DQ". The research lab provided us with samples recovered from computer systems located in Europe, as well as a detailed report with their initial findings, including analysis comparing the threat to Stuxnet, which we were able to confirm. Parts of Duqu are nearly identical to Stuxnet, but with a completely different purpose.

Duqu is essentially the precursor to a future Stuxnet-like attack. The threat was written by the same authors (or those that have access to the Stuxnet source code) and appears to have been created since the last Stuxnet file was recovered. Duqu's purpose is to gather intelligence data and assets from entities, such as industrial control system manufacturers, in order to more easily conduct a future attack against another third party. The attackers are looking for information such as design documents that could help them mount a future attack on an industrial control facility.

Stuxnet, Duqu, … come from the same factory or factories

Stuxnet and Duqu were written on the same platform…by the same group of programmers.

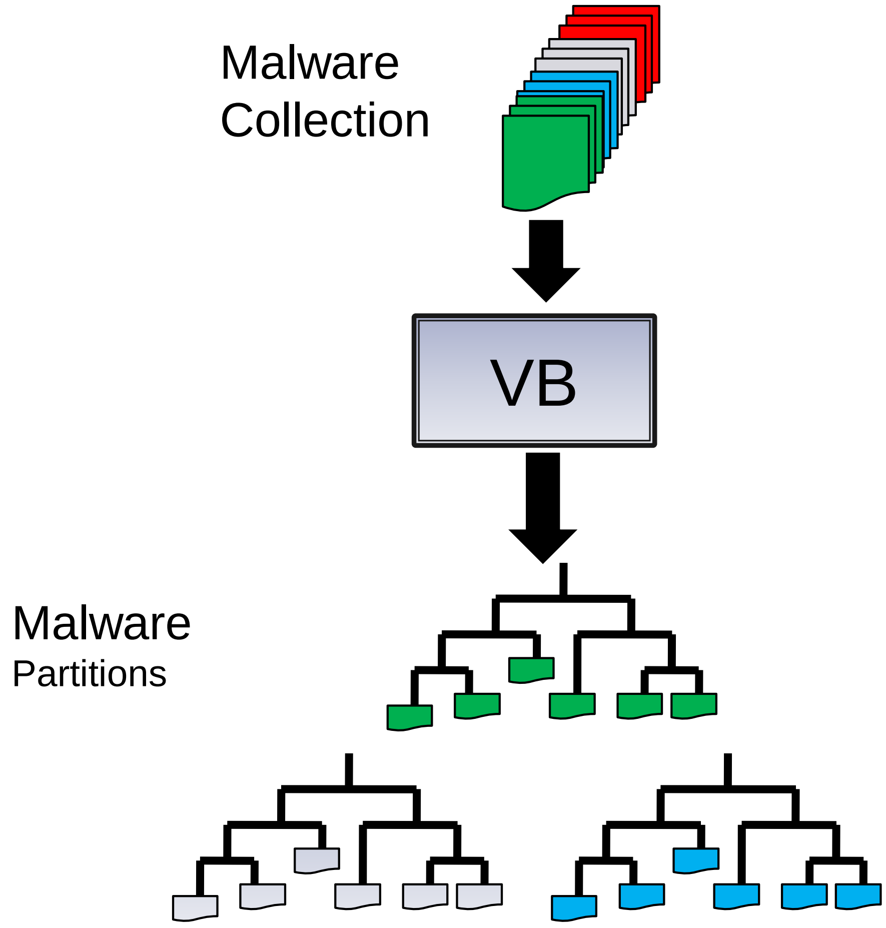
… linked specific portions of code

Case Study

Customer and Data

- Financial Services company profile
  - 120,000 servers, 60 countries
  - Have in-house, trained staff in malware analysis
  - Separate Security Op and Threat Investigation Op
- Data
  - Selection of 463 Binaries
  - VirusTotal first seen: Jun 2006 to April 2014
    - Unseen: 18 binaries
  - Size: 95 percentile – 700Kb

Partition Collection

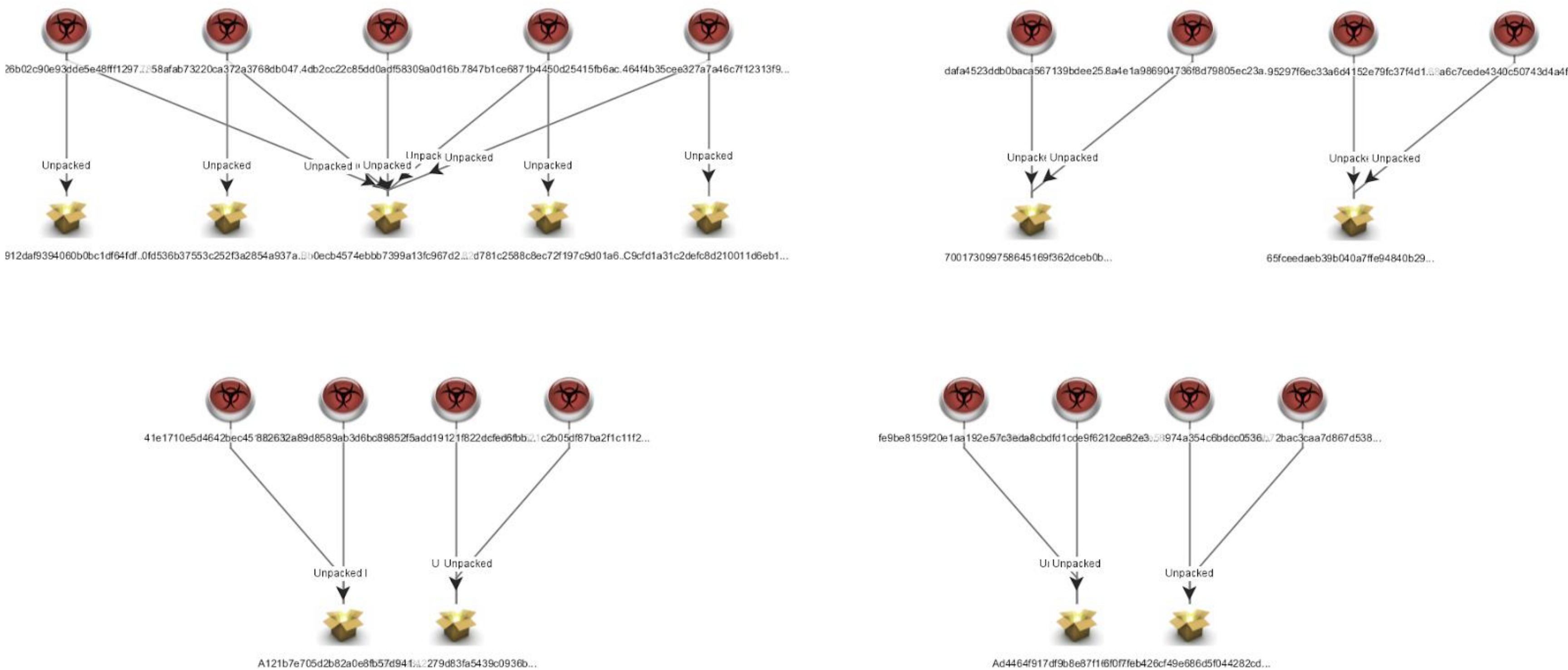


Unpacking

- Our approach
  - Run program in a virtual machine
  - Watch it's execution below the VM (in emulator)
    - Program doesn't know it's being watched
  - Determine when it's completed unpacking
  - Create a PE executable from memory image

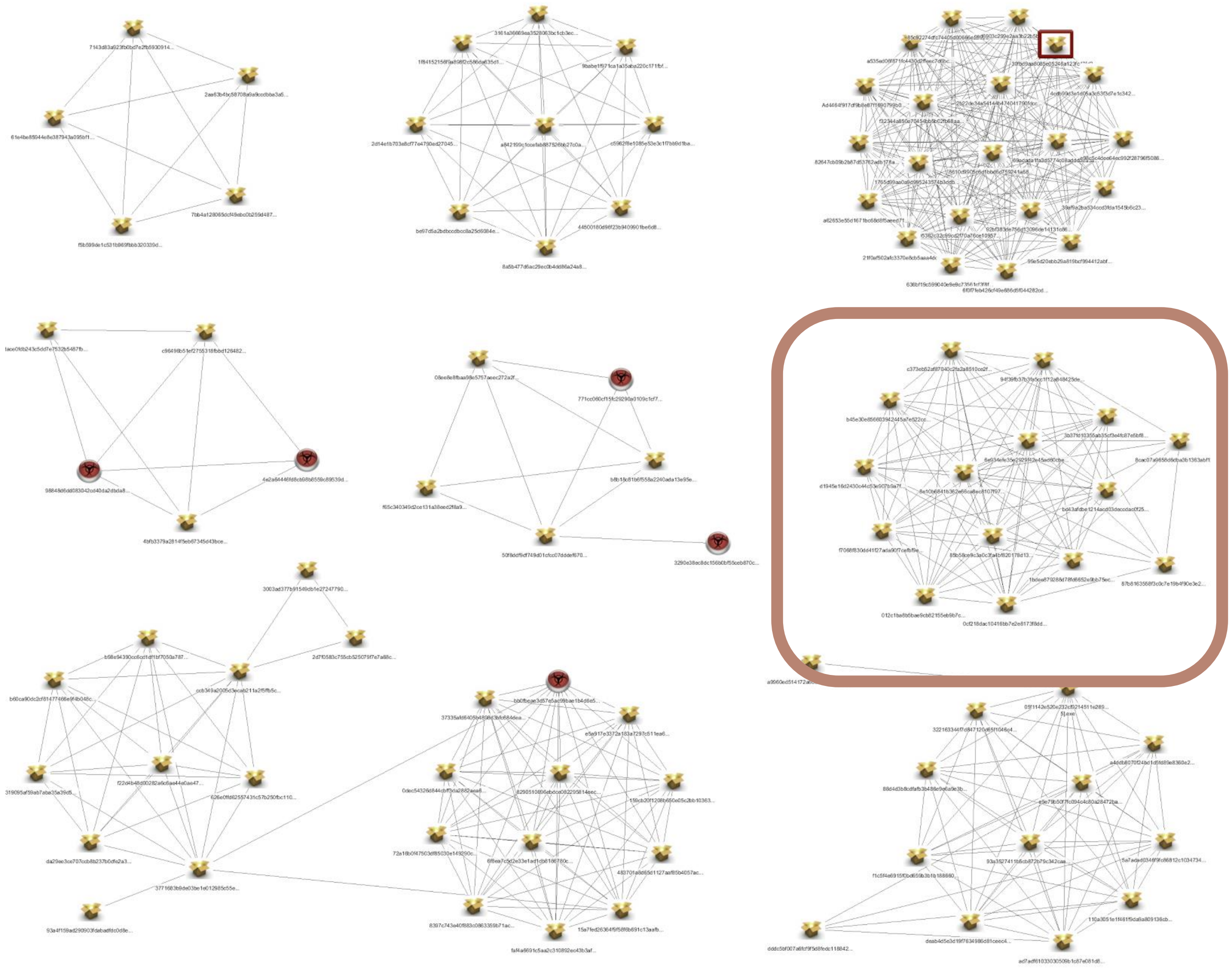
Similar binaries after unpacking

Unpacked 371/463 binaries

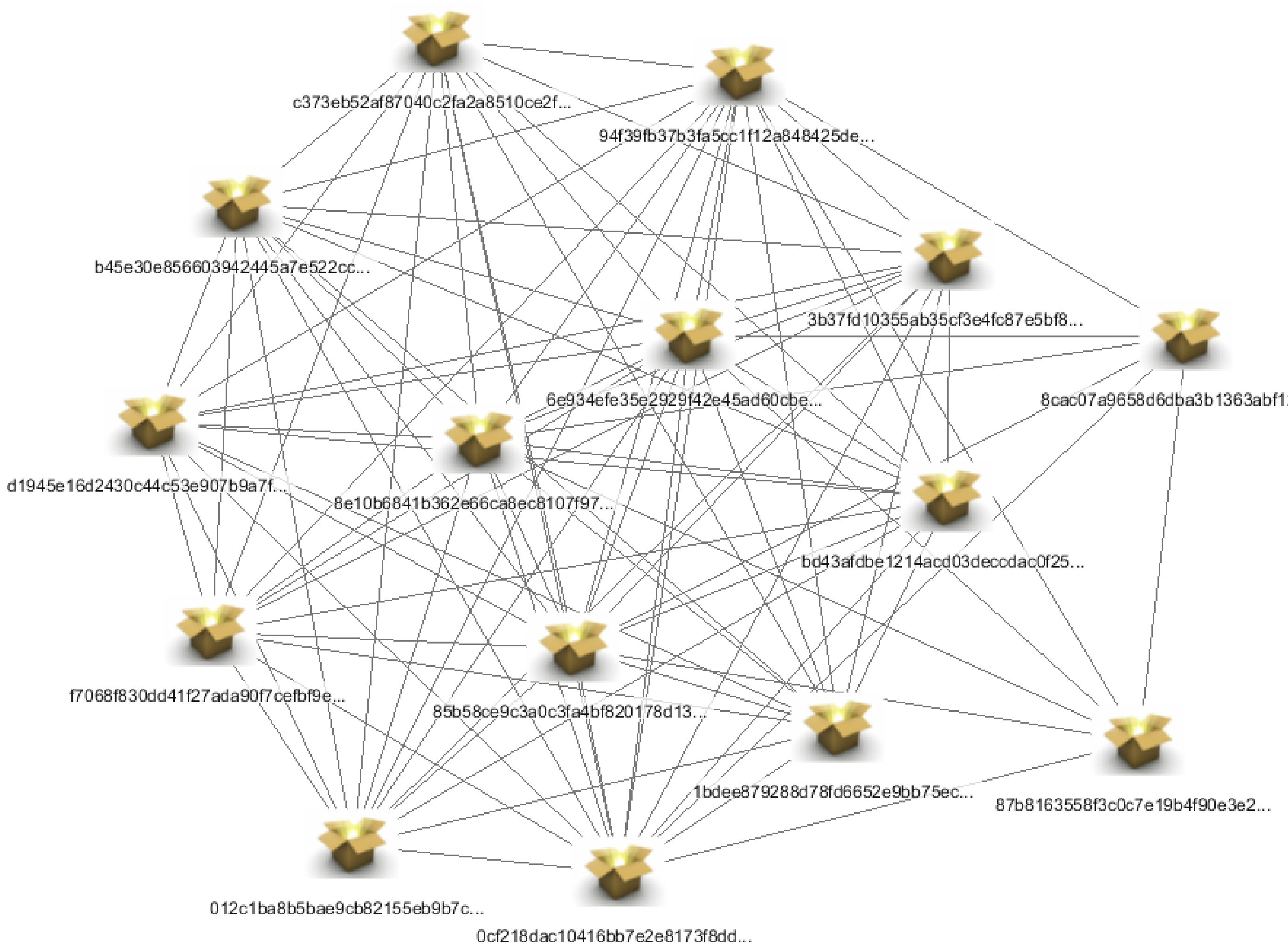


Different Binaries mapped to same MD5 after unpacking

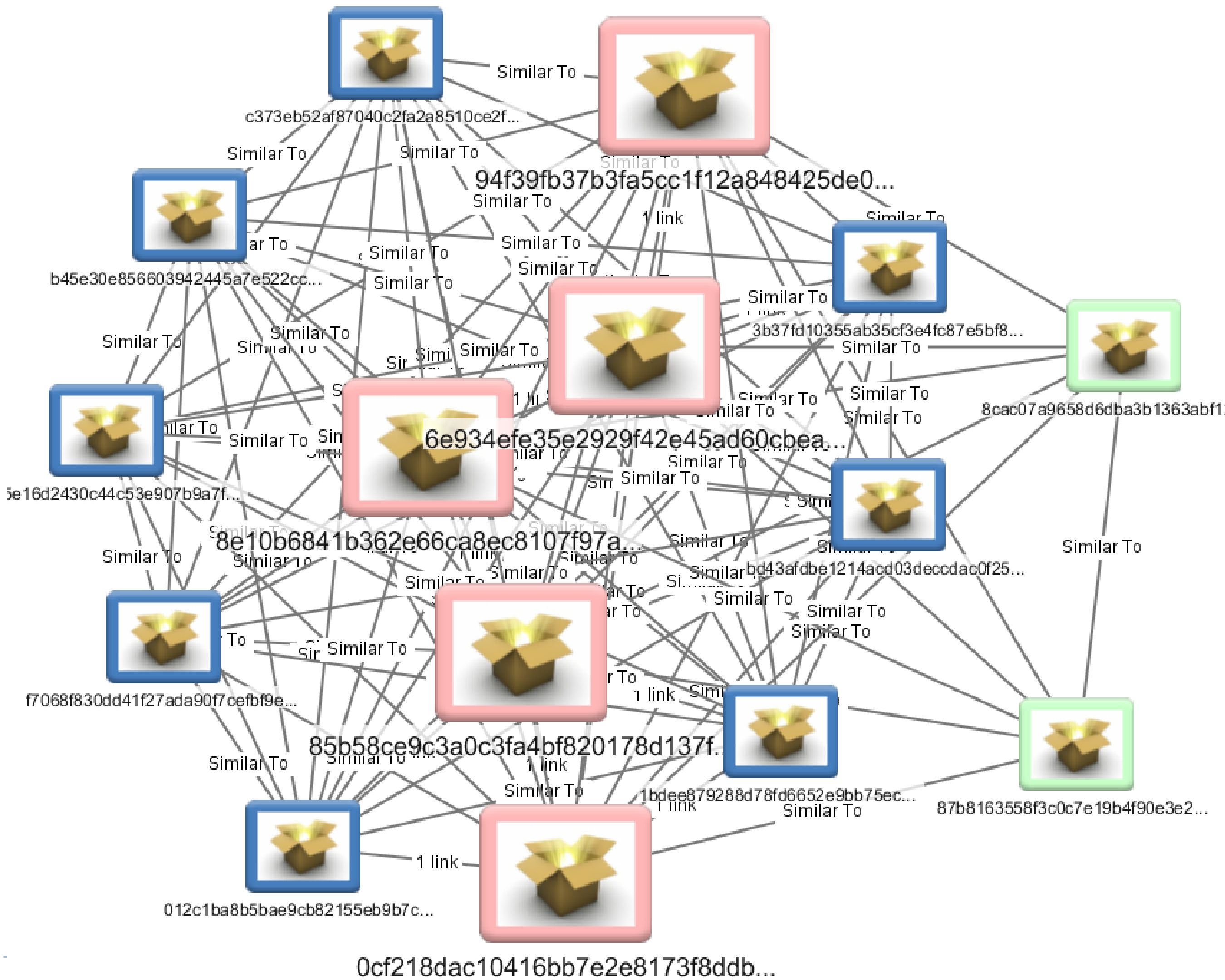
Case Study – Clusters found



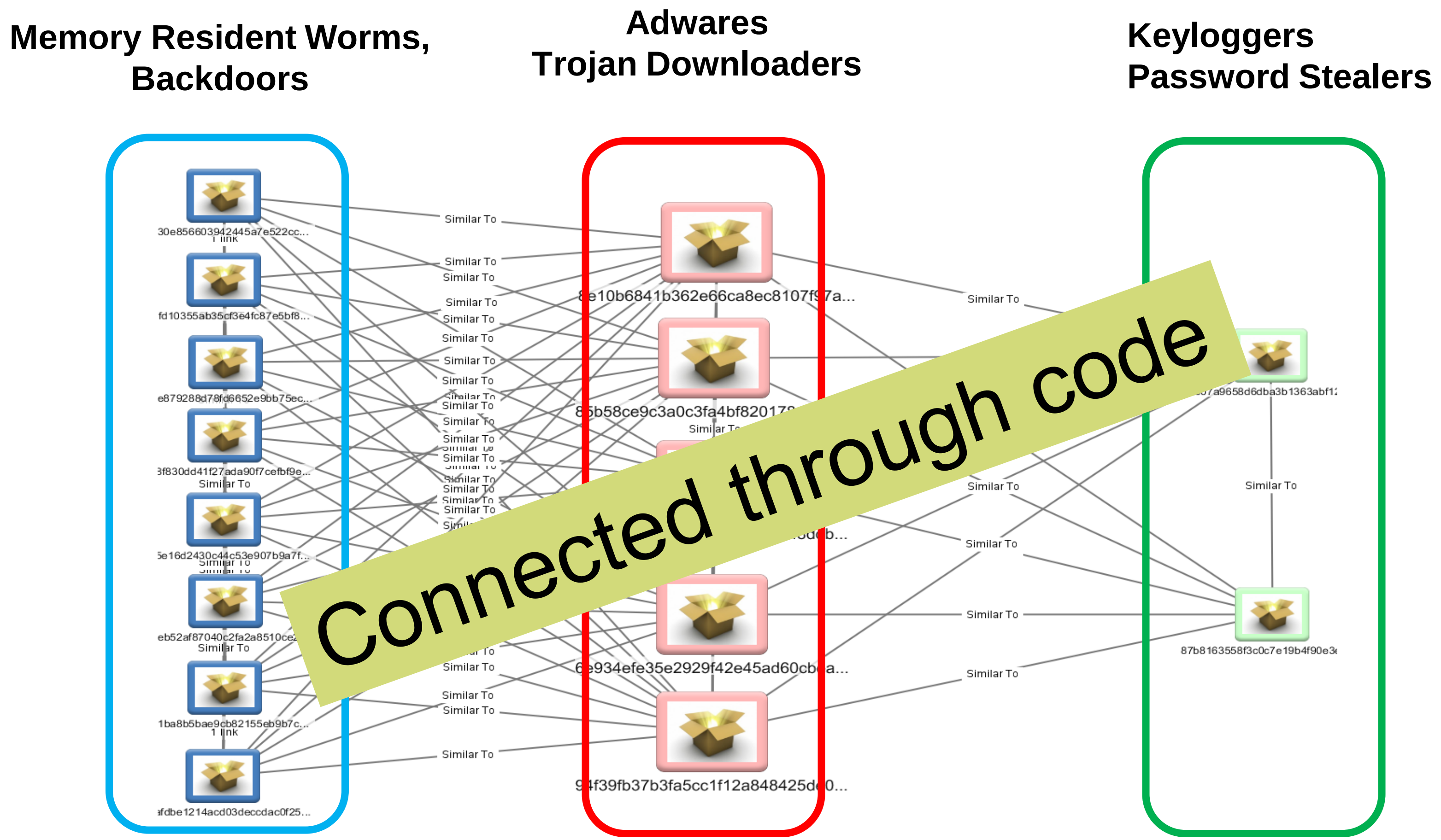
Selected cluster



Complete Subgraphs



Reorganize



Memory Resident Worms, Backdoors
Adwares
Trojan Downloaders
Keyloggers
Password Stealers
Connected through code
Similar To
30e856603942445a7e522cc...
fd10355ab35cf3e4fc87e5bf8...
e879288d78fd6652e9bb75ec...
6e10b6841b362e66ca8ec8107f97a...
87b8163558f3c0c7e19b4f90e3

Validation using Deep Inspection

Validating Clusters using Bindiff

- Select a pair of binaries matched by VirusBattle
- Perform side-by-side-comparison using Zynamics' BinDiff.
  - BinDiff is an interactive tool for comparing two binaries.
  - In contrast, VirusBattle helps in locating similar binaries in a large collection.

Investigating matches in two binaries

similarity		EA primary	name primary	EA secondary	name secondary
0.99	--------	00431CF4	sub_431CF4_22177	00431524	sub_431524_24147
0.99	--------	00431CC6	sub_431CC6_22176	004314F6	sub_4314F6_24146
0.99	--------	00431C98	sub_431C98_22175	004314C8	sub_4314C8_24145
0.99	--------	00431C58	sub_431C58_22174	00431488	sub_431488_24144
0.99	--------	00431C13	sub_431C13_22173	00431443	sub_431443_24143
0.99	--------	00431BD4	sub_431BD4_22172	00431404	sub_431404_24142
0.99	--------	00431B95	sub_431B95_22171	004313C5	sub_4313C5_24141
0.96	--------	00431AFB	sub_431AFB_22170	0043132B	sub_43132B_24140
0.99	--------	00431AAA	sub_431AAA_22168	004312DA	sub_4312DA_24138
0.90	--------	00431A47	sub_431A47_22165	00431277	sub_431277_24135
0.90	--------	00431A0D	sub_431A0D_22163	0043123D	sub_43123D_24133
0.99	--------	0043196F	sub_43196F_22162	0043119F	sub_43119F_24132
0.99	--------	00431920	sub_431920_22161	00431150	sub_431150_24131
0.99	--------	004318C5	sub_4318C5_22160	004310F5	sub_4310F5_24130
0.99	--------	0043186D	sub_43186D_22159	0043109D	sub_43109D_24129
0.99	--------	00431824	sub_431824_22158	00431054	sub_431054_24128
0.99	--------	004317D8	sub_4317D8_22157	00431008	sub_431008_24127
0.99	--------	00431792	sub_431792_22156	00430FC2	sub_430FC2_24126
0.99	--------	0043173E	sub_43173E_22155	00430F6E	sub_430F6E_24125
0.99	--------	00431624	sub_431624_22154	00430E54	sub_430E54_24124

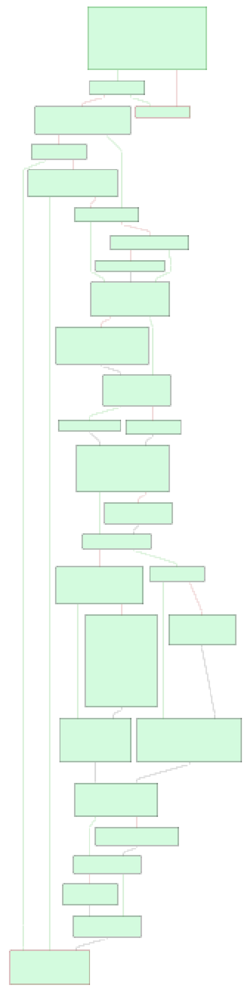
Procedures in one binary

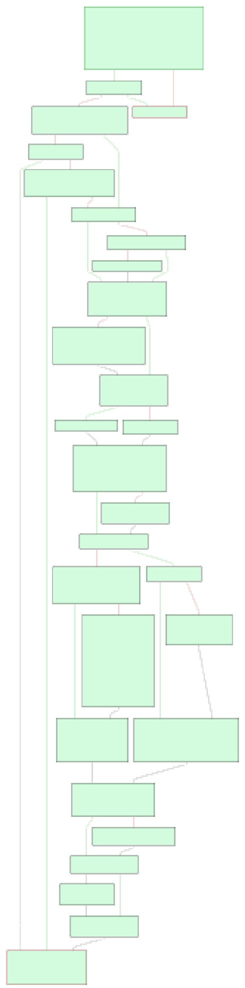
Matching procedures in second binary

Level of similarity

Drill down to matching two procedures

CFG of a procedure in one binary





CFG of a matching procedure in the second binary

Drilldown to matching code

Closing…

Summary

- Malware Variant Generation Process
  - Manual – usual lifecycle
  - Automated – for protection
- Managing very large collection of malware
  - Use information retrieval
  - Derive features from semantics
  - Normalize representation to enable string comparison
- Semantic analysis
  - Combine sound analysis (a la, compilers)
  - And unsound analysis (probabilistic)
- Application
  - Connect actors through shared code

Selected References

- LAKHOTIA, Arun, PREDA, Mila Dalla, et GIACOBAZZI, Roberto. Fast location of similar code fragments using semantic 'juice'. In : Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop. ACM, 2013. p. 5.
- DALLA PREDA, Mila, GIACOBAZZI, Roberto, LAKHOTIA, Arun, et al. Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables. In : ACM SIGPLAN Notices. ACM, 2015. p. 329-341.
- MILES, Craig, LAKHOTIA, Arun, LEDOUX, Charles, *et al.* VirusBattle: State-of-the-art malware analysis for better cyber threat intelligence. In : *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*. IEEE, 2014. p. 1-6.
- RUTTENBERG, Brian, MILES, Craig, KELLOGG, Lee, *et al.* Identifying shared software components to support malware forensics. In : *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Cham, 2014. p. 21-40.