Harnessing Intelligence from Malware Repositories

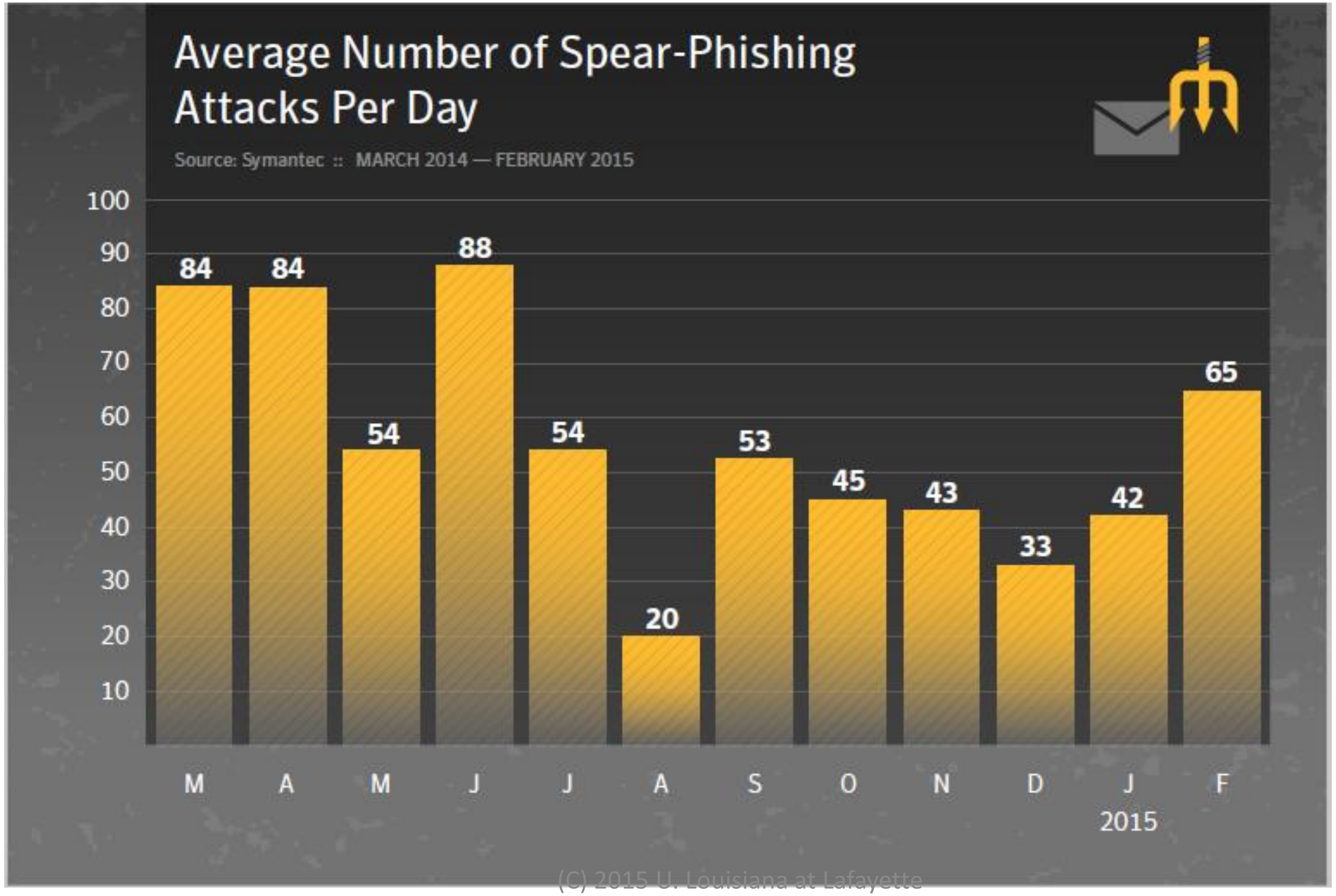
Arun Lakhotia and Vivek Notani
Software Research Lab
University of Louisiana at Lafayette
arun@louisiana.edu , vxn4849@louisiana.edu

Self Introduction

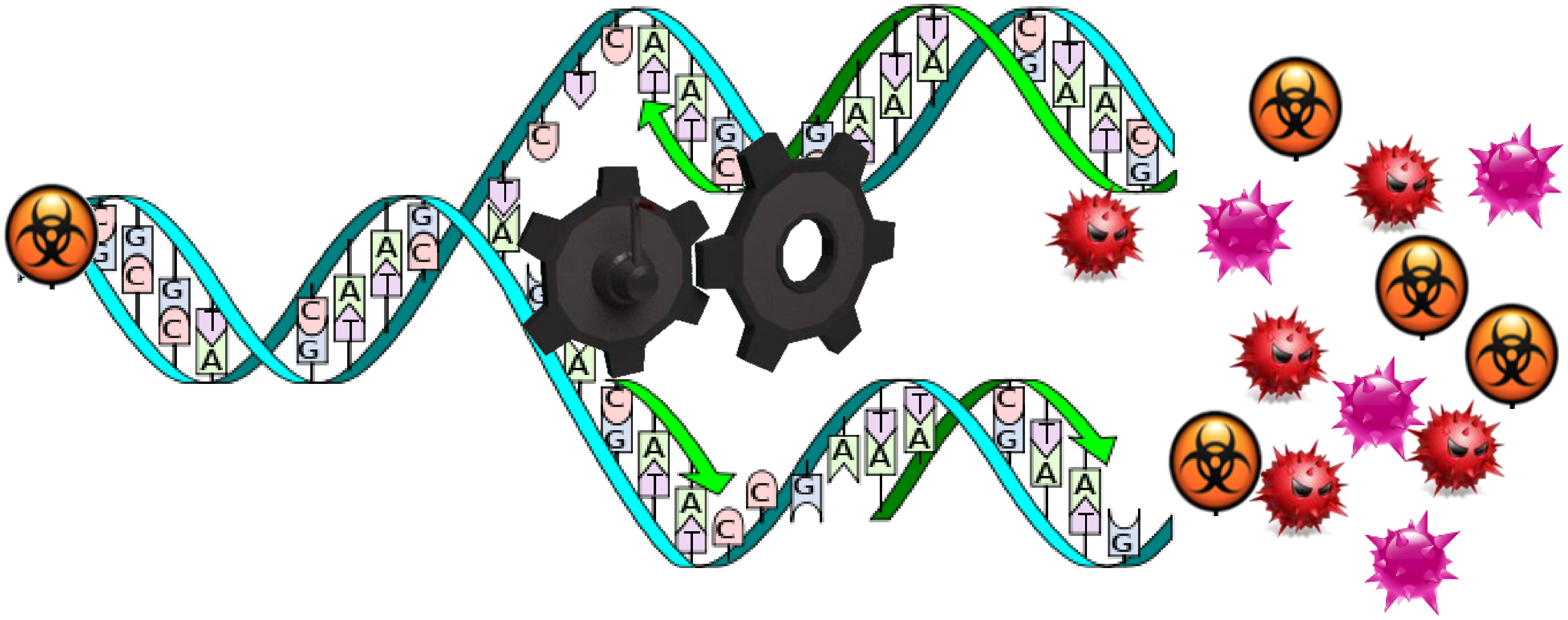
- Software Research Lab
  - 10 years research on Malware
  - Graduate course on malware analysis
  - Active interaction with industry
  - Funded by AFOSR, ARO, DARPA, ONR, and State of Louisiana
- Research Focus
  - How does malware evade detection?
  - How to detect stealthy malware?
  - Malware analysis in the large
- Results
  - Papers: 50+ peer-reviewed
  - Patents: one granted
  - Degrees: 6 Ph.D., 8 M.Sc.
  - Research Funding: $5MM+

UL Lafayette™

Targeted Attacks

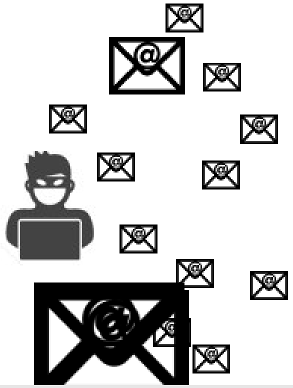


Machine Generation of Malware



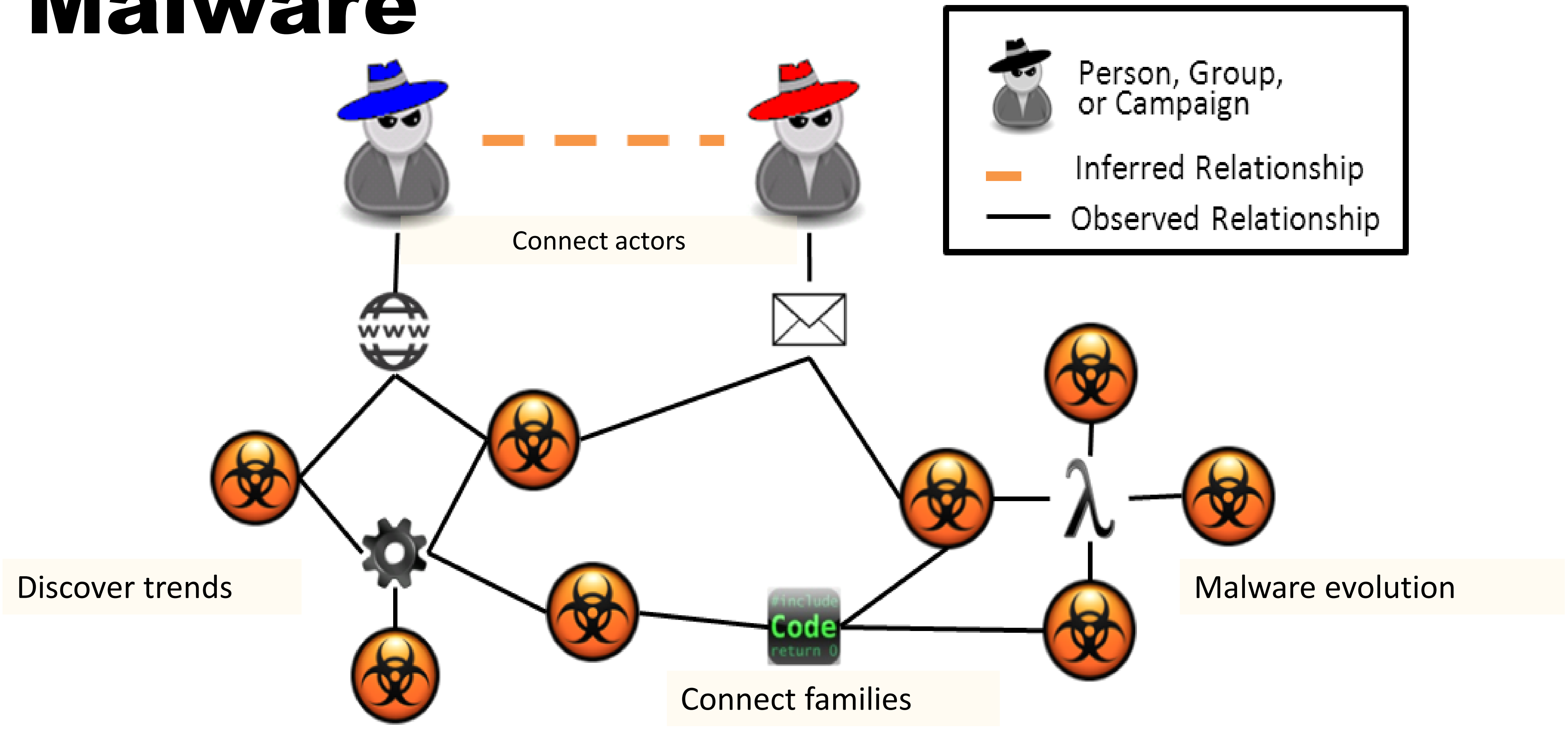
CyberSecurity Paradox

Physical world: FAILED attempts are INVESTIGATED



Cyber World: FAILED attempts are CELEBRATED

Extract Intelligence from Malware



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

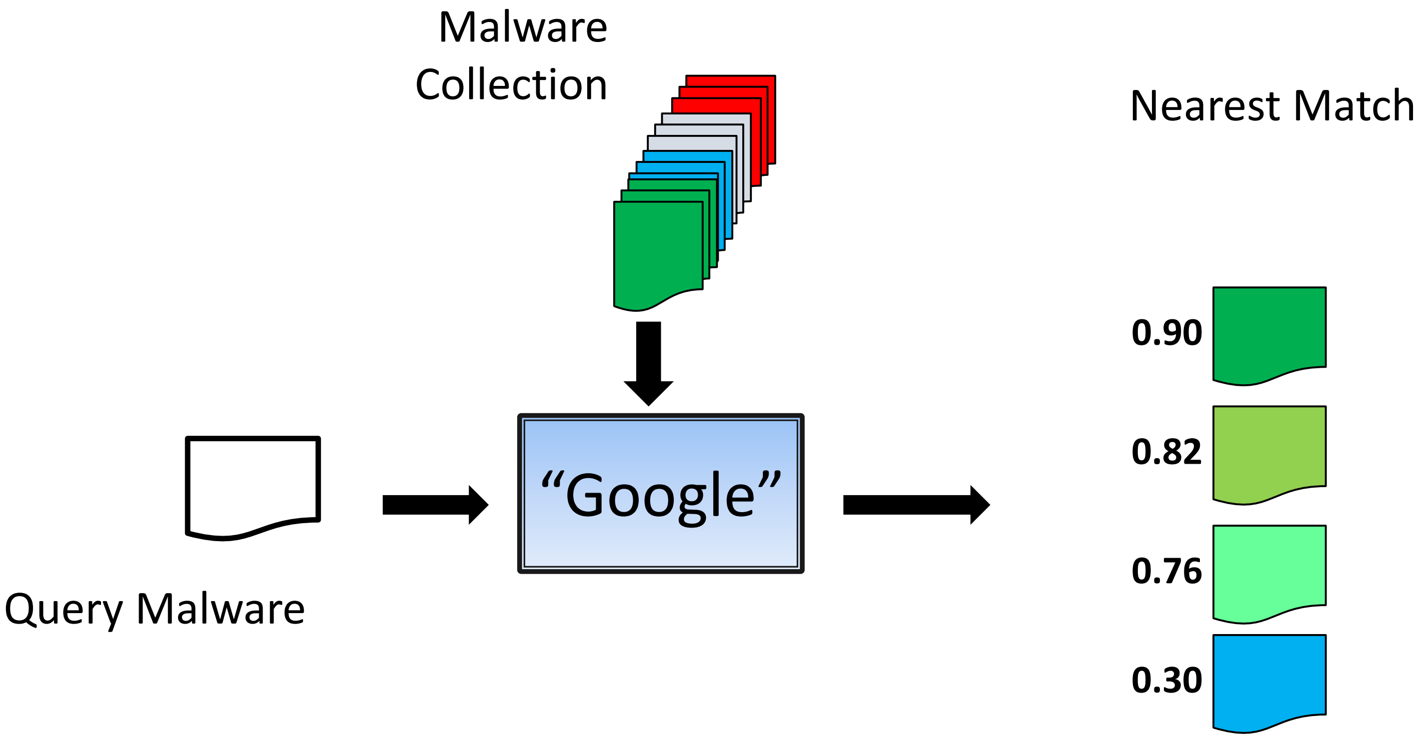
Parts of Duqu are nearly identical to Stuxnet,
but with a completely different purpose.

written by the same authors
(or .. using Stuxnet source code)

On October 14, 2… strong international connections alerted us to a … be very sim… …amed the threat "Duqu" [dyü-kyü] because it … prefi… …ab provided us with samples recovered f… …urope, as … …ort with their initial findings, includi… …tuxnet, which we … …onfirm. Parts of Duqu are near… …pletely different purpose.

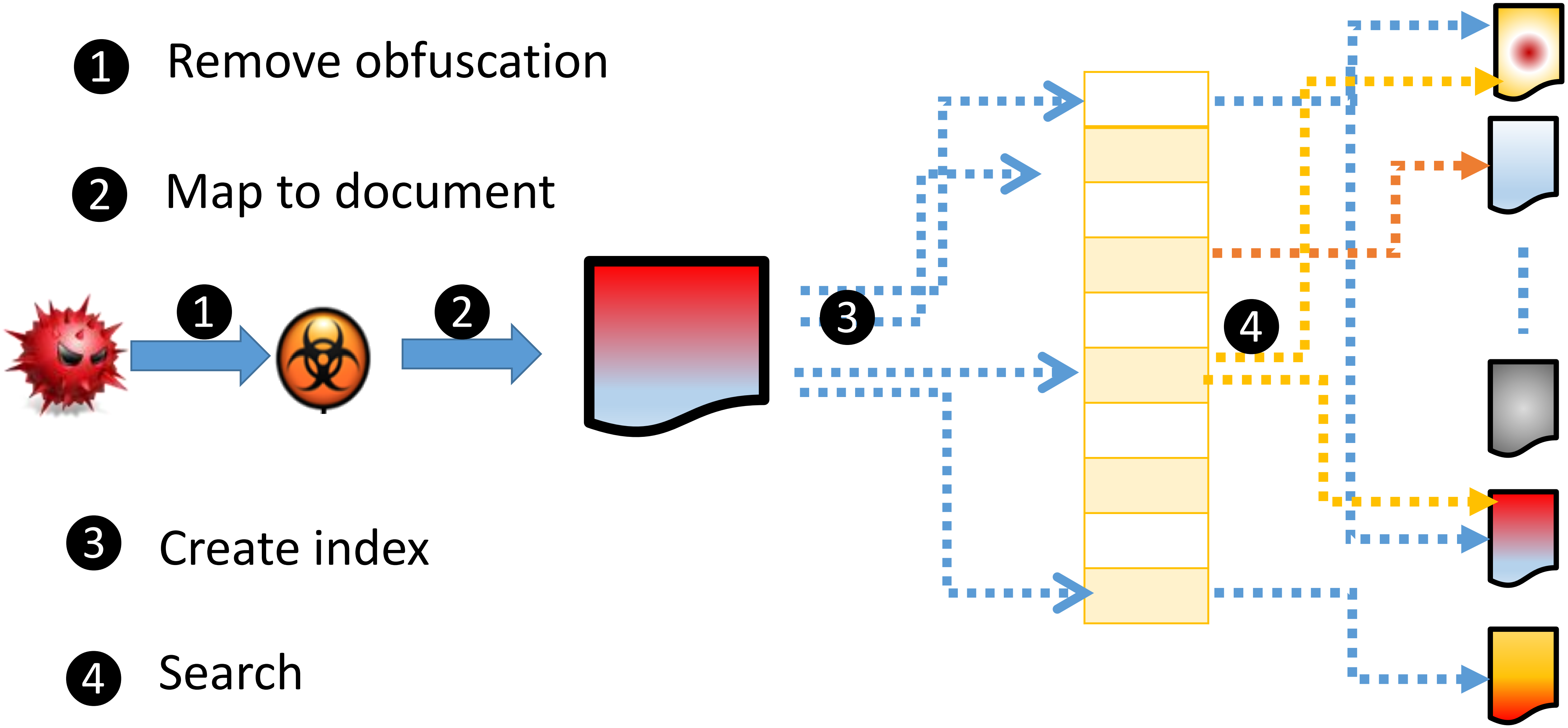
…u is essentially the precur… …The threat was written by the same authors (or those that have access to the S… …and appears to have been created since the last Stuxnet file was recovered. Duqu's purpose … …intelligence data and assets from entities, such as industrial control system manufacturers, in 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.

Requirement: “Google” for Malware



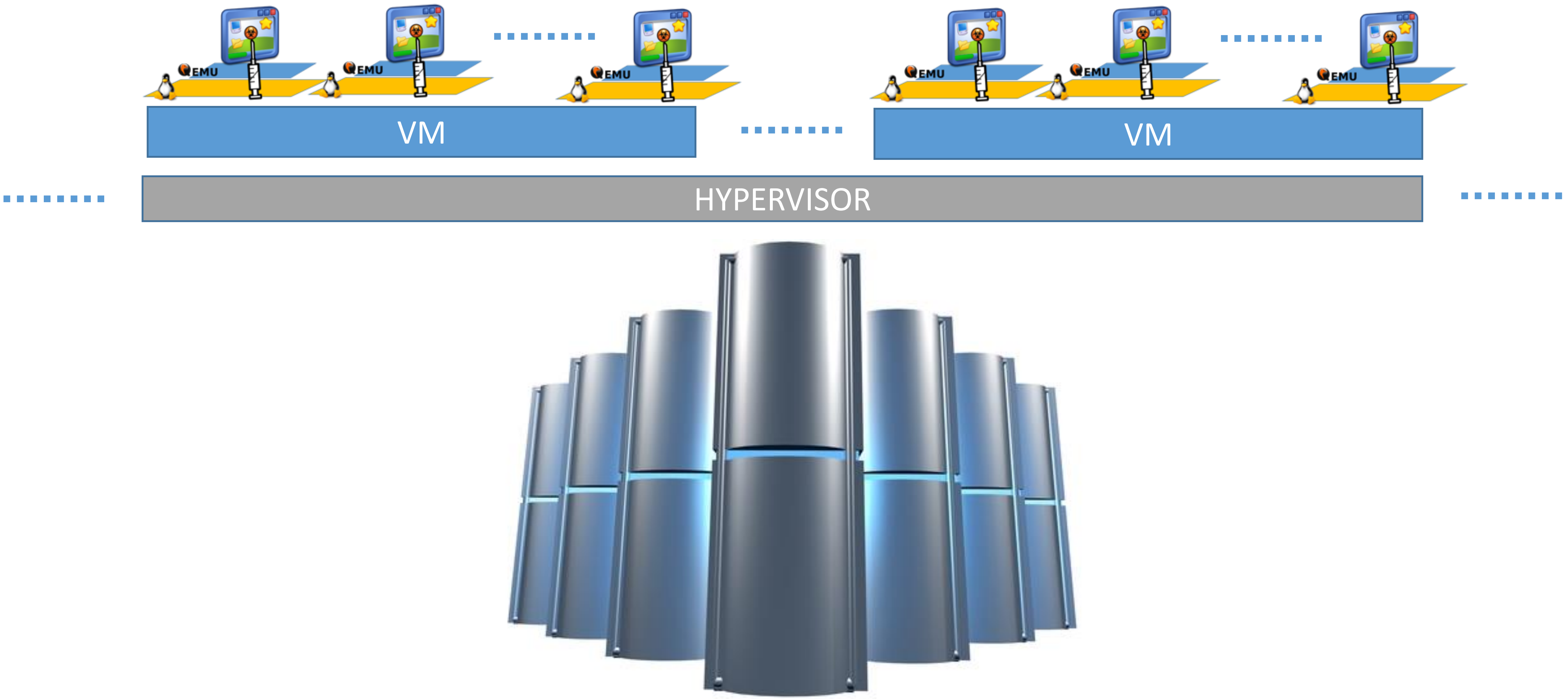
Challenges

1. Remove obfuscation
2. Map to document



3. Create index
4. Search

Key Innovation: VM Introspection in the Cloud



Key Innovation: Semantic Fingerprints

STATE OF PRACTICE

```
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
```

Bit shreds

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

Fingerprint

Susceptible to obfuscation

VIRUSBATTLE

```
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
```

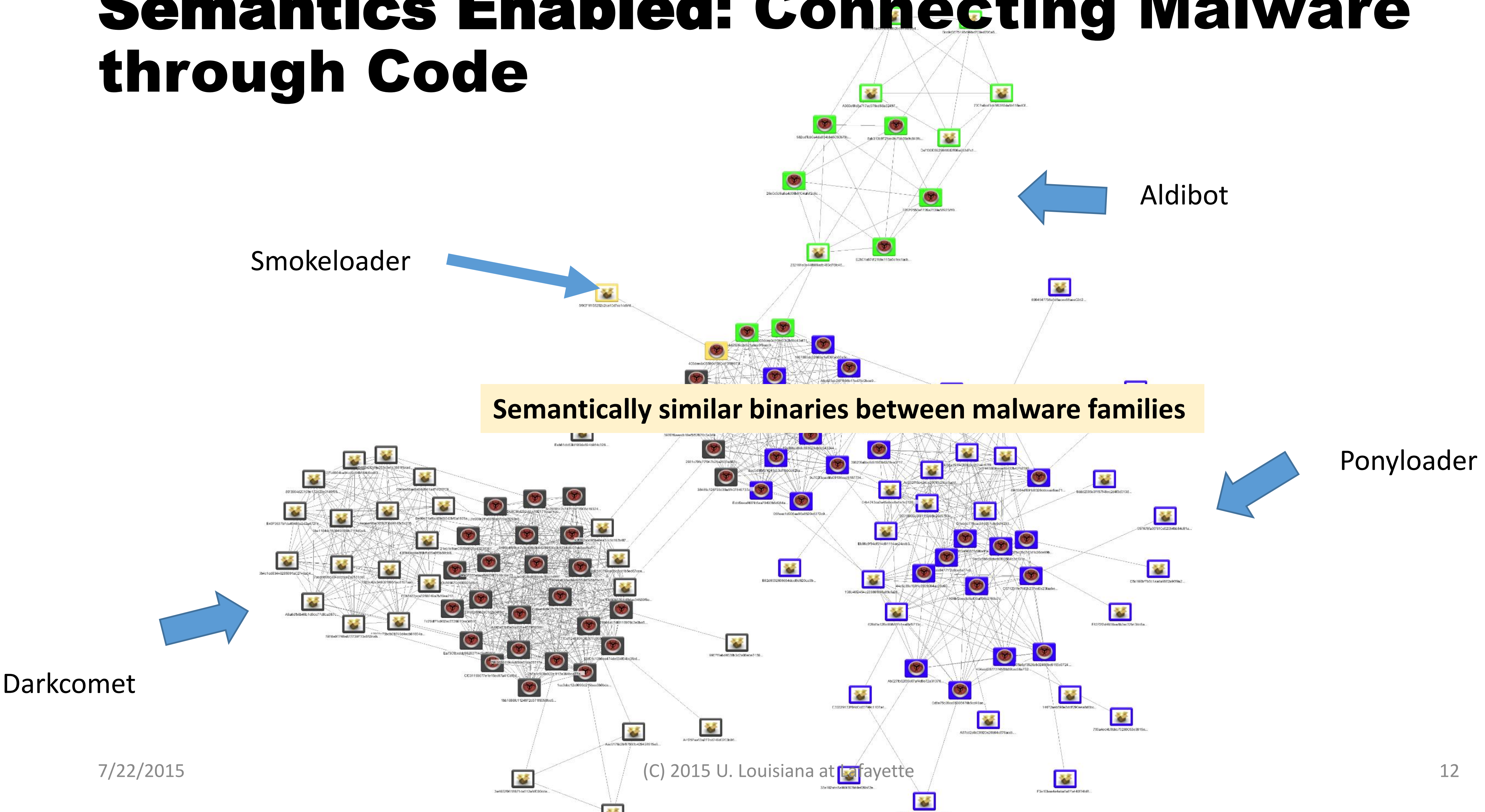
Semantics

```
              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)))
```

Fingerprint

Obfuscation resistent

Semantics Enabled: Connecting Malware through Code



Unpacking Malware

Challenge 1: Packing

Classes of Packers (Protectors)

- Classification parameter
  - Based on execution behavior
    - When and how much of the original code is decrypted
- Traditional Packer
  - Entire original code is decrypted at one time
    - Entire original code is in clear text before it is executed
- Paged Packer
  - Just-In-Time decryption of a page when it is executed
    - Only a ‘page’ of the original code is in clear text at any time
- Virtual Machine Protectors
  - Decrypt a single instruction at a time
    - None of the original code is ever in clear text

Unpacking: State-of-Practice

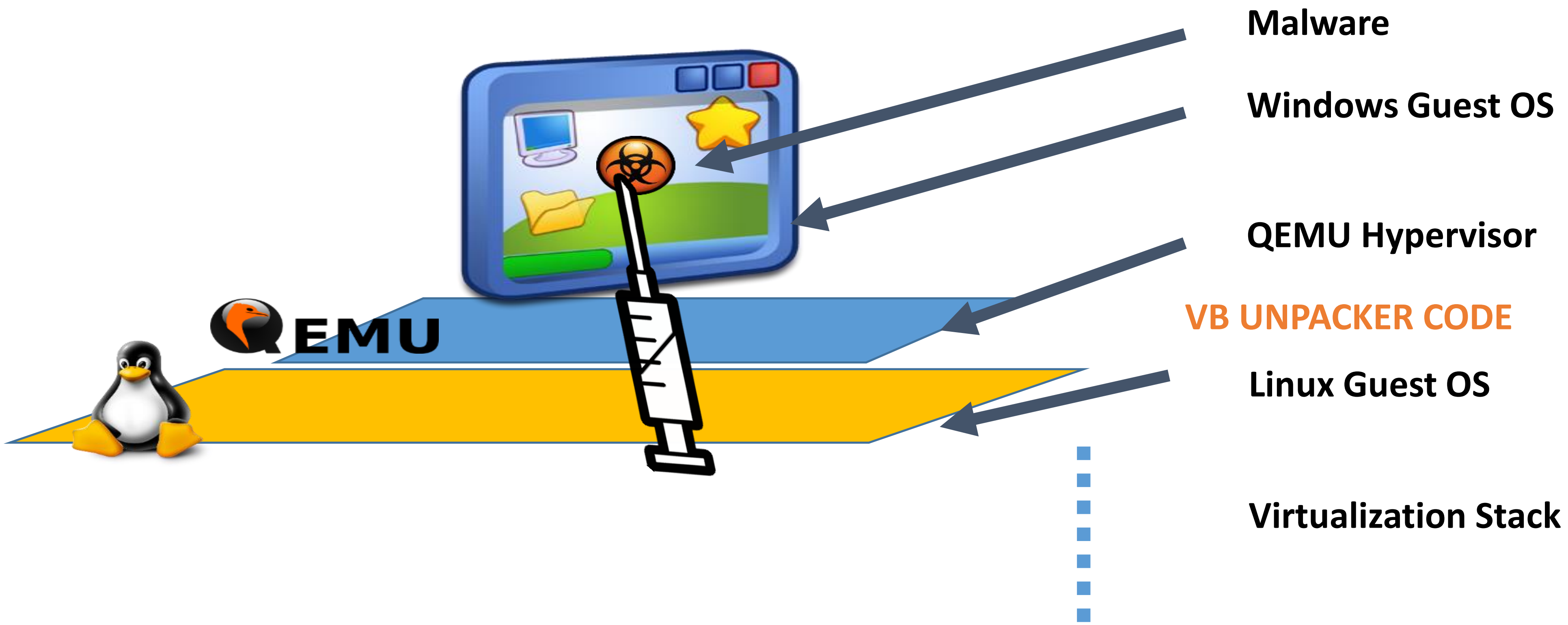
Malware

UNPACKER CODE

Windows Guest OS

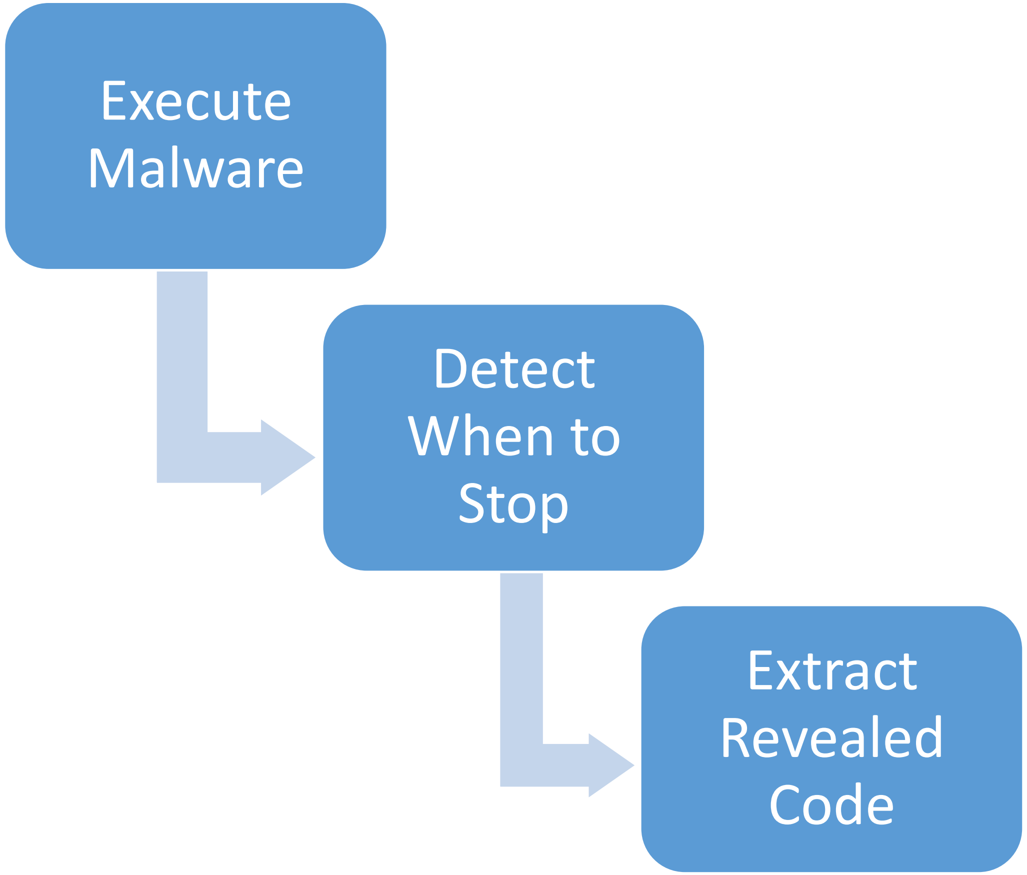
Virtualization Stack

Innovation: Unpacking using VM Introspection



Observe malware below ring 0

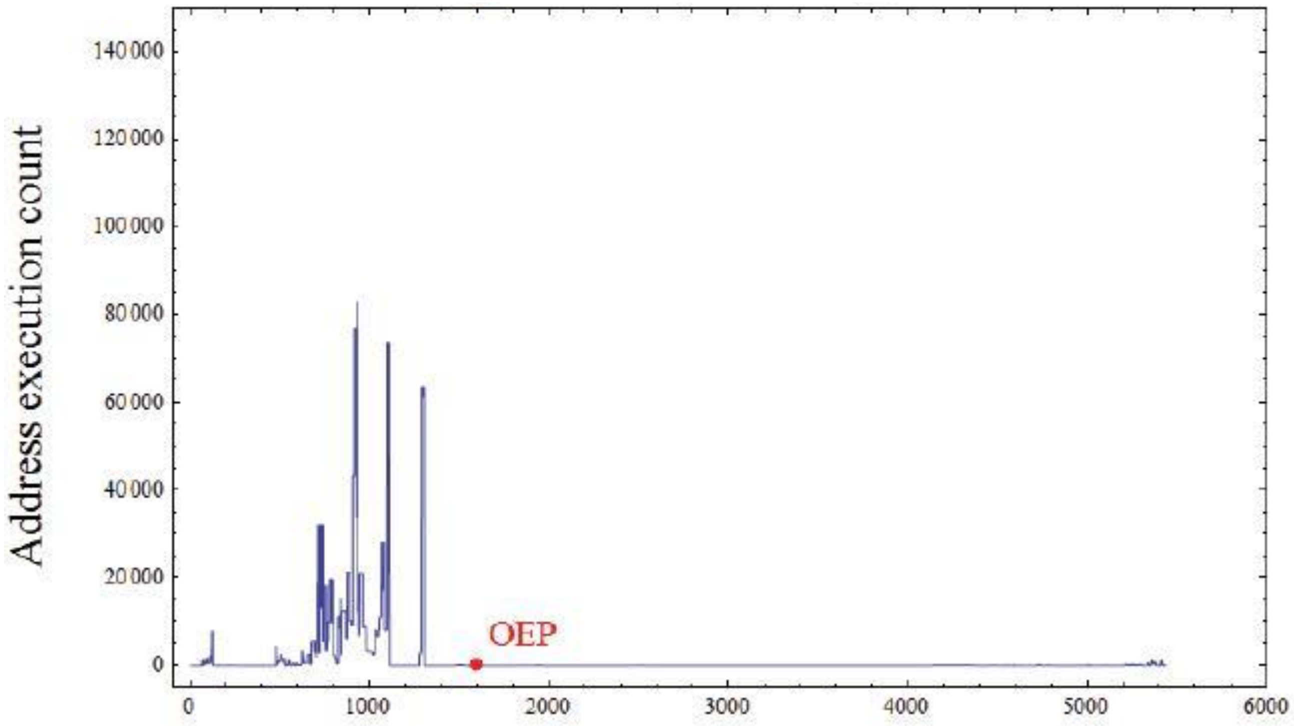
Unpacking Traditionally Packed Malware



When to Stop: Hump and Dump

- Traditional Packer
    - Decryption in a loop
        - High instruction execution frequency
        - Spike in frequency graph
- Hump & Dump Algorithm
    - Detect spike – hump
    - Detect end – flat

PE Compact2.5 (calc.exe), Linear Scale



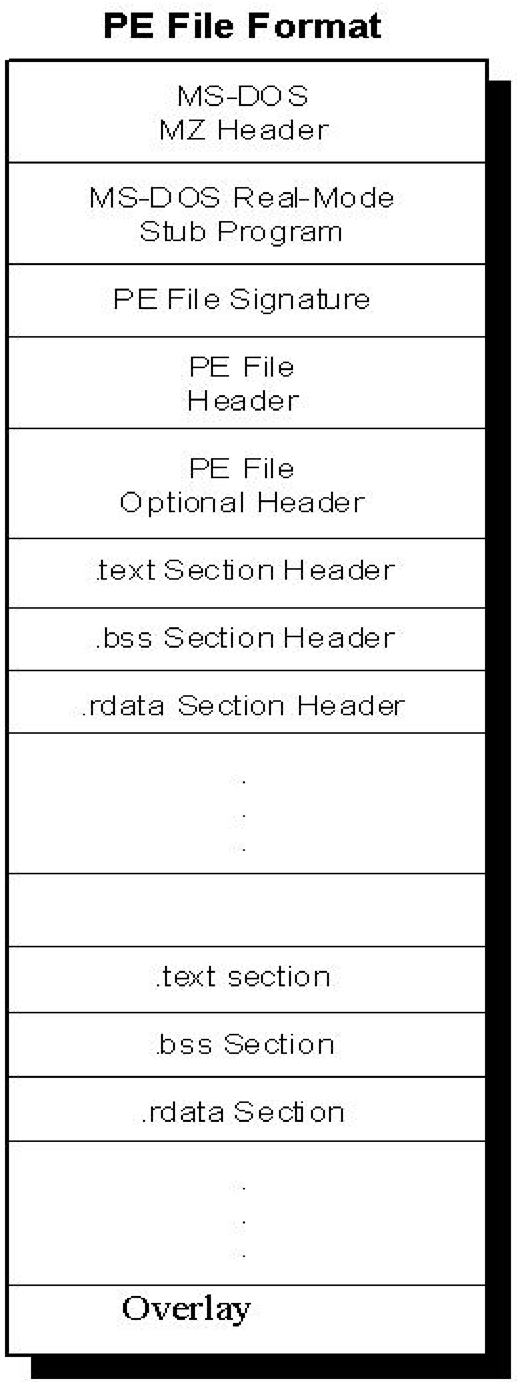
Addresses Ordered by last execution time

When to Stop: TimeOut

- What if Hump is never detected?
  - TimeOut
  - Limits execution time

Constructing PE

- Modify OEP using last PC value
- Fix Section Headers
- Copy Memory Contents to new PE



Extracting Memory Contents: Challenge

- Extracting memory through hypervisor
- Memory contents may be paged out by GuestOS
- Solution:
  - Determine memory is paged out
  - Analyze execution profile
  - Re-run unpacker with new parameters
    - Catch before memory is paged out

Case Study

Dataset Description:

- File Type: PE-32
- Source: FBI
- Availability: Upon request
- Collection period: 1 year

Bot Family	# Executables
Aldibot	19
Armageddon	1
Blackenergy	65
Darkcomet	339
Darkshell	379
Ddoser	5
Illusion	17
Nitol	11
Optima	160
Ponyloader	1,312
Smokeloader	31
Umbraloader	25
Yzf	4
Zeus	41
Total	**2,409**

Case Study: Results

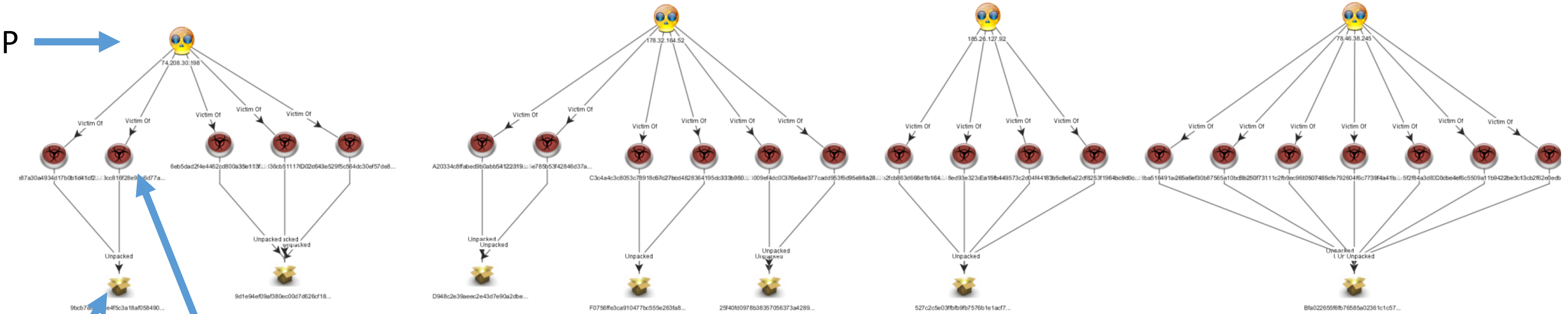
- Input : 2,409
- Unpacked : 2,354
- Output : 2,185

HEURISTIC	Original	Unique Unpacked	Poor Unpacking	Poor Unpacking (%)
Hump and Dump	1,671	1,523	205	12.27
TimeOut	515	500	46	8.93
Self-tuning	168	163	23	13.69
TOTAL	2,354	2,186	274	11.64

Unpacked Binary “very similar” to Original => Poor Unpacking

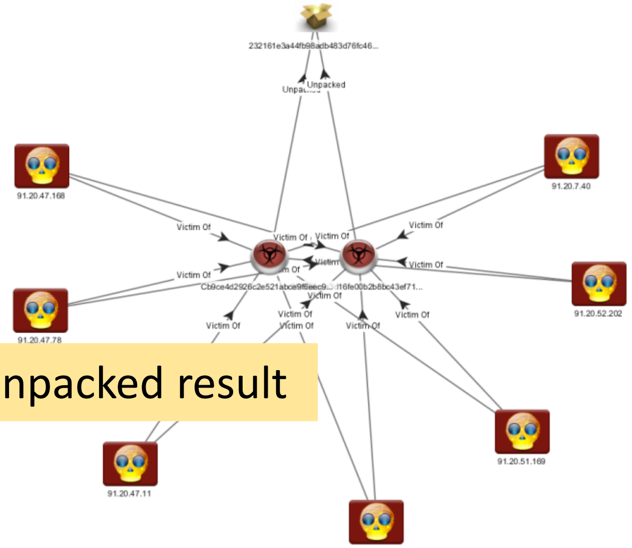
Unpacker's Impact: Analysis Cost Reduction

CNC IP



Unpacked

Malware



Multiple malware produce the same unpacked result

Malware Binaries	#	%
Original binaries	2,409	
Unpacked successfully	2,354	97.7%
Unique unpack result	2,185	92.8%
Reduction in analyst work	**169**	**7.2%**

Matching Code

Challenge 2: Code Obfuscation

```
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
```

→

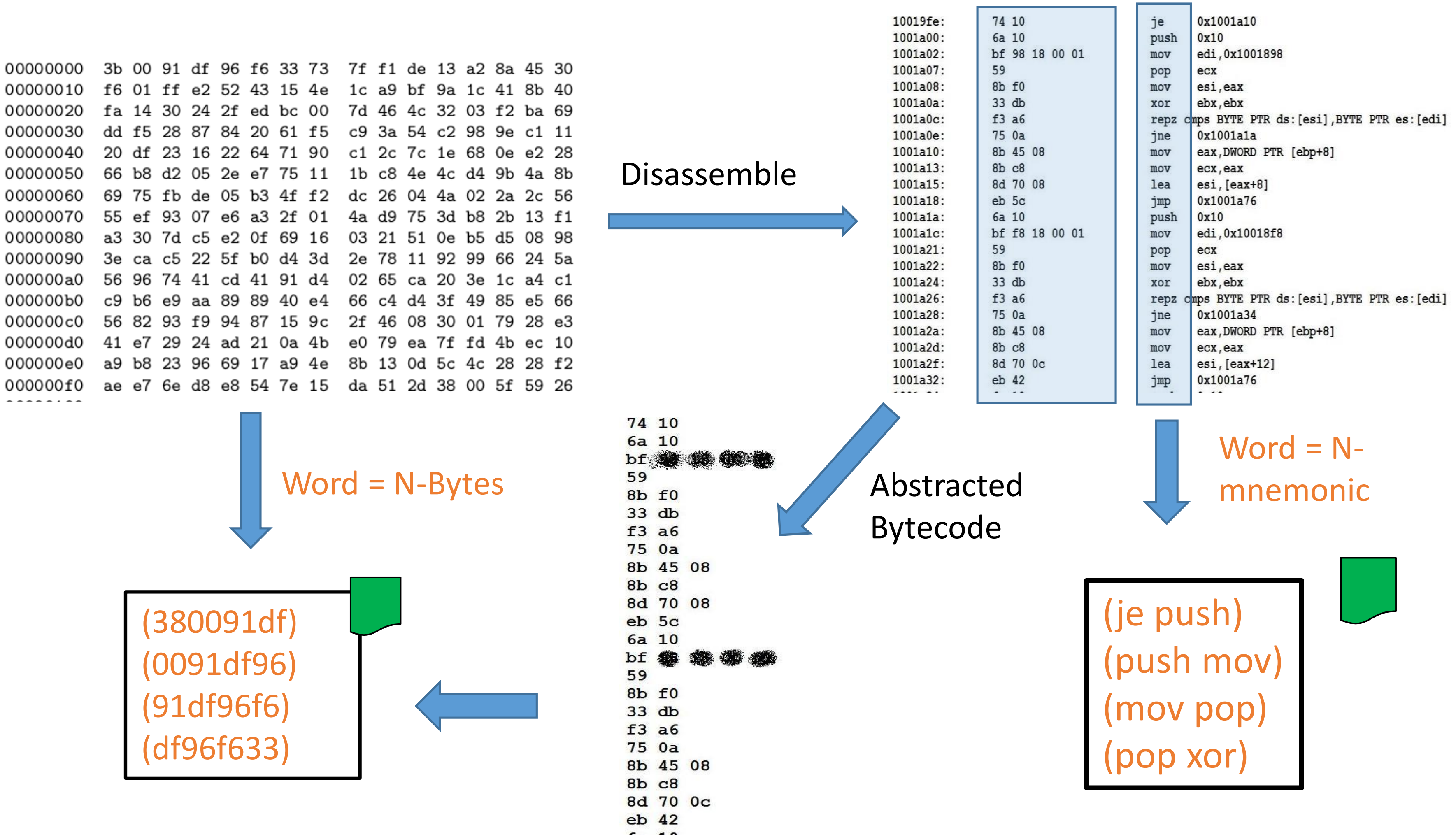
```
push ecx
mov ecx, [ebp + 10]
mov ecx, ebp
push eax
add eax, 2342
mov eax, 33
add ecx, eax
pop eax
mov eax, esi
push eax
mov esi, ecx
push edx
xor edx, 778f
mov edx, 34
sub esi, edx
pop edx
mov [esi-2], eax
pop esi
pop ecx
```

Requirements

- Scale requirement
  - Search in collection of thousands to millions of malware
- Performance requirement
  - Provide results in seconds, or less
- Quality requirement
  - Error rates should be comparable to pairwise matching

Representations for Matching Binaries

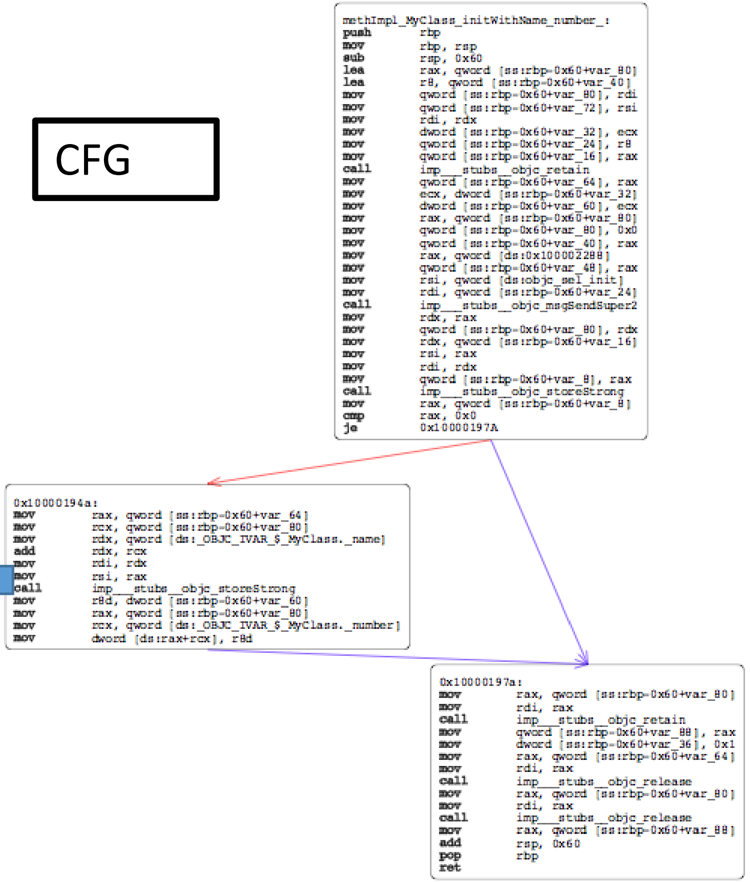
Map binary to ‘document’



VirusBattle Strategy

Map binary to CFG to Document

Binary → Disassembly → CFG



Abstracted Bytecode

Abstracted Disassembly

Semantics NEW

Juice NEW

Word = Block

Code to Semantics

Code

- Sequential
- Focus on operations

```
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

- Parallel
- Captures affect

```
                 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)))
```

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

State = LValue -> RValue

LValue = Register + Mem

RValue = Int

+ def(RValue)
+ RValue op Rvalue
+ op RValue

Value in previous state

Unsimplified

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

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 = Int
+ **def(RValue)**
+ **RValue op Rvalue**
+ **op RValue**

Computing Juice

```
401290: b8 05 00 00 00       mov   eax,0x5
401295: 81 c3 04 00 00 00    add   ebx,0x4
40129b: 6b c3                imul  eax,ebx
```

$$\begin{aligned} \text{eax} &= 5 \\ \text{ebx} &= (\text{def}(\text{ebx}) + 4) \times 5 \\ &= \text{def}(\text{ebx}) \times 5 + 20 \end{aligned}$$

$$\begin{aligned} A &= \text{N1} \\ B &= \text{def}(B) \times \text{N1} + \text{N2} \\ &\quad \text{where } \boxed{\text{N2} = \text{N1} \times \text{N3}} \\ &\quad \quad \text{and type}(A) = \text{type}(B) = \text{reg32} \end{aligned}$$

Problem: Establish constraints between induced variables?

Solution

1. Track simplification steps

2. Generalize simplification steps

Semantics and Juice

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

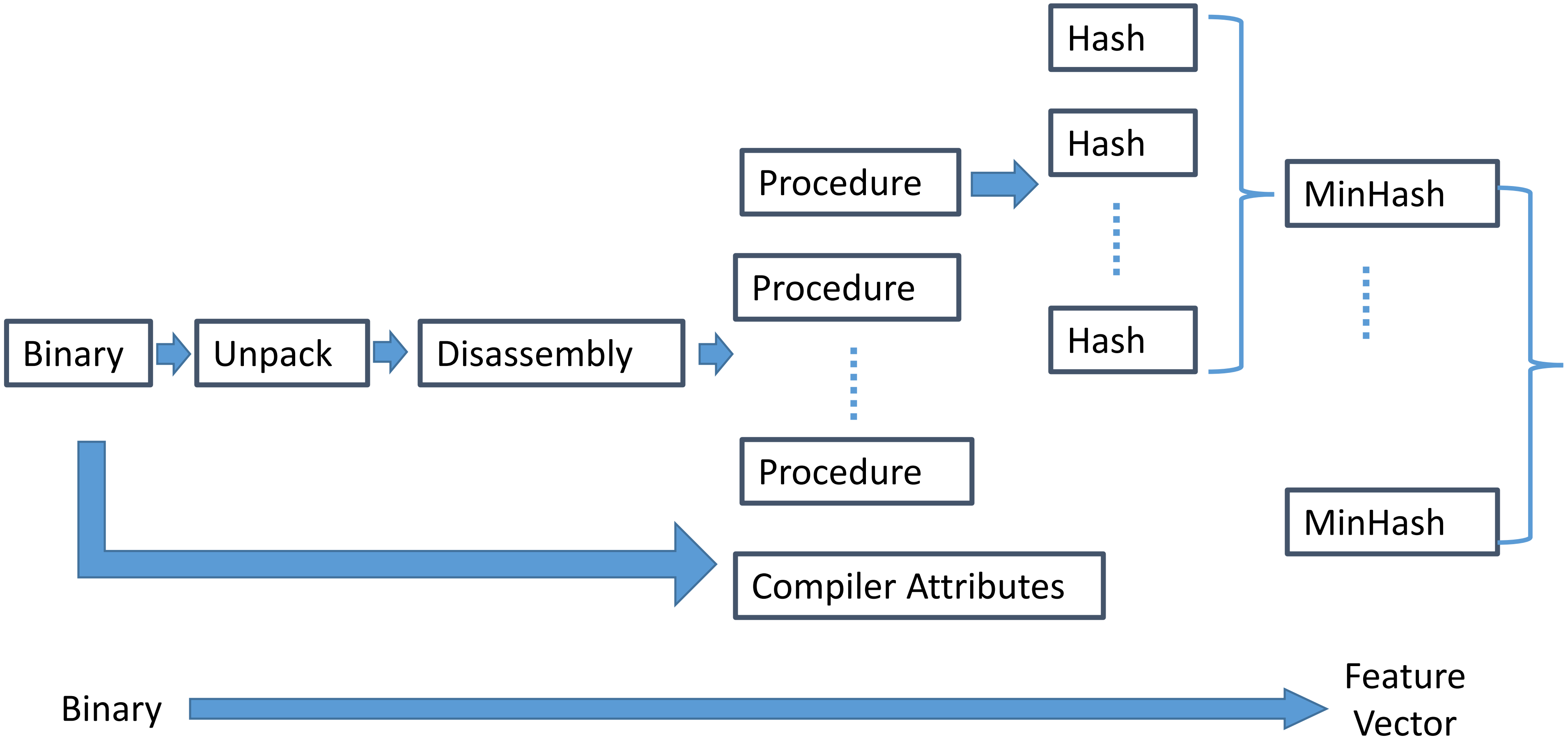
Juice

```
              A = def(B),
              B = N1+def(C),
              C = N2+def(C),
memdw(N2+def(C)) = def(B)
memdw(N1+def(C)) = def(B)
memdw(N3+def(C)) = def(memdw(def(C)))
      where  A, B, C are 'registers'
             N1, N2, N3 are 'Int'
```

RValue = Int
+ **def(RValue)**
+ **RValue op Rvalue**
+ **op RValue**
+ **Variable**

Challenge 3: Scalable Search

Featurization Process



MinHash: A form of LSH

A	Feature	B
Light Brown	Hair Color	Dark Brown
Long	Hair Length	Long
Brown	Eye Color	Brown

Feature = MinHash Function
Set of Features = MinHash Signature

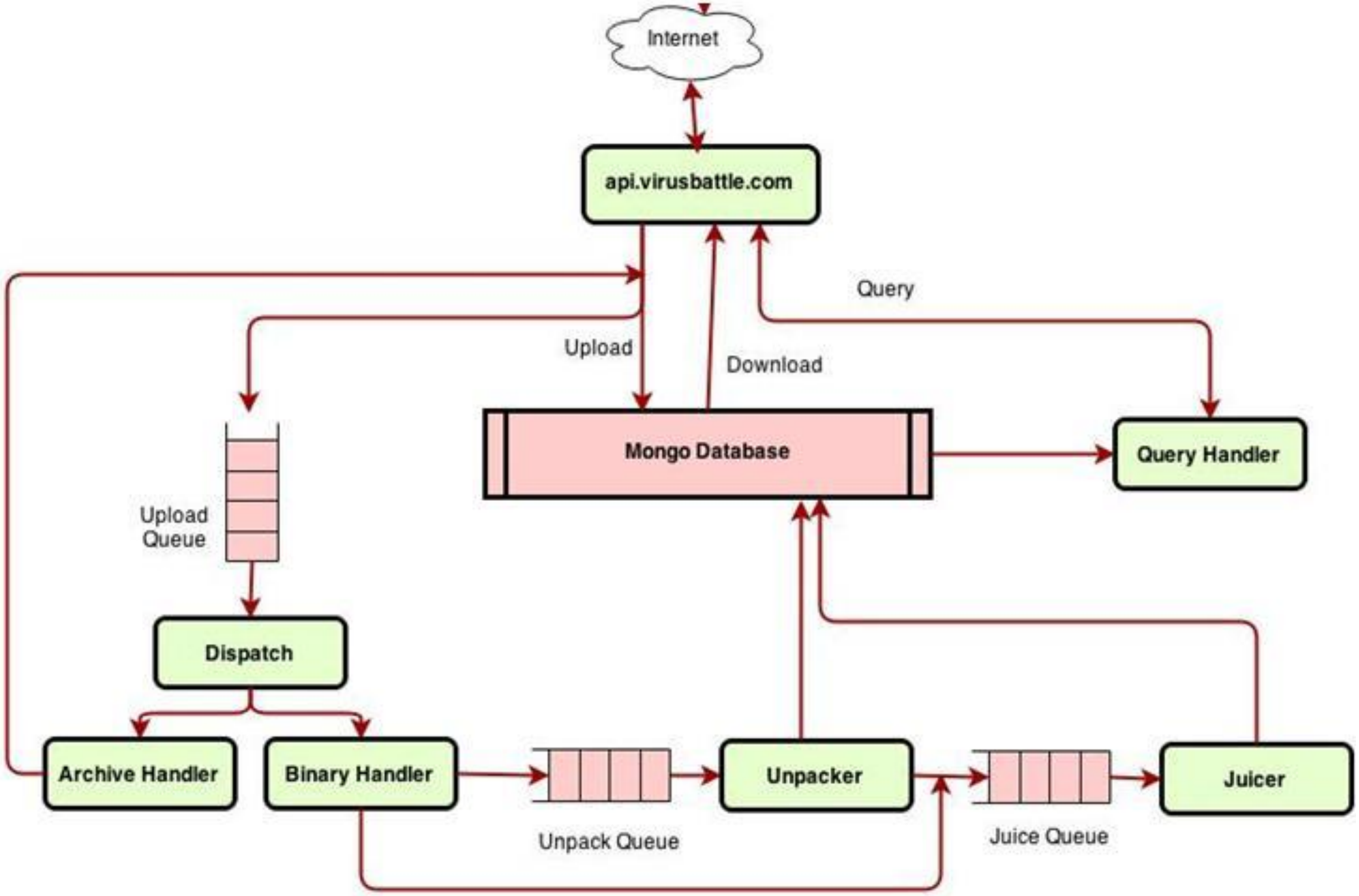
MinHash Signature-1

MinHash Signature-2

Compose for Deterministic manipulations

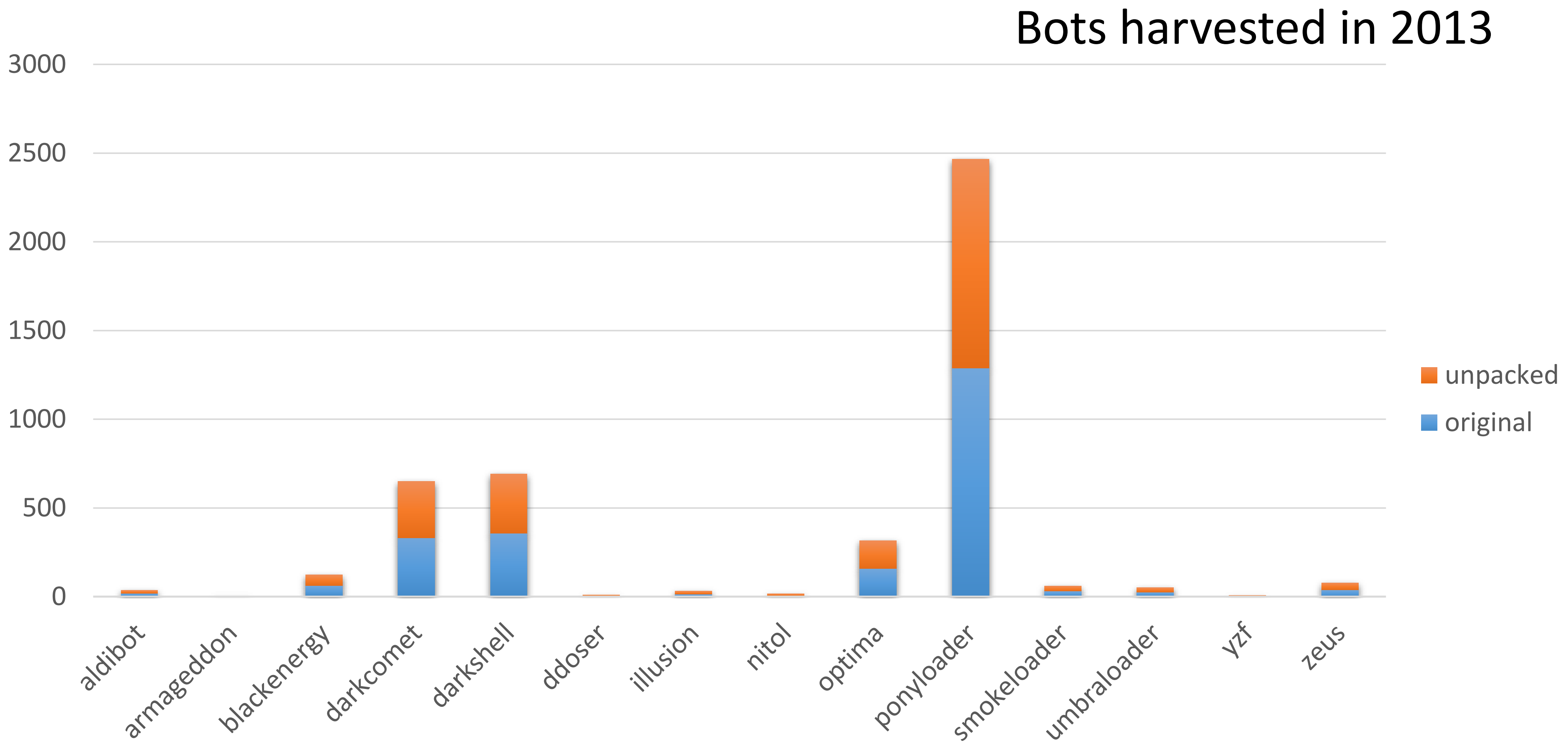
Architecture

VirusBattle Webservice Architecture

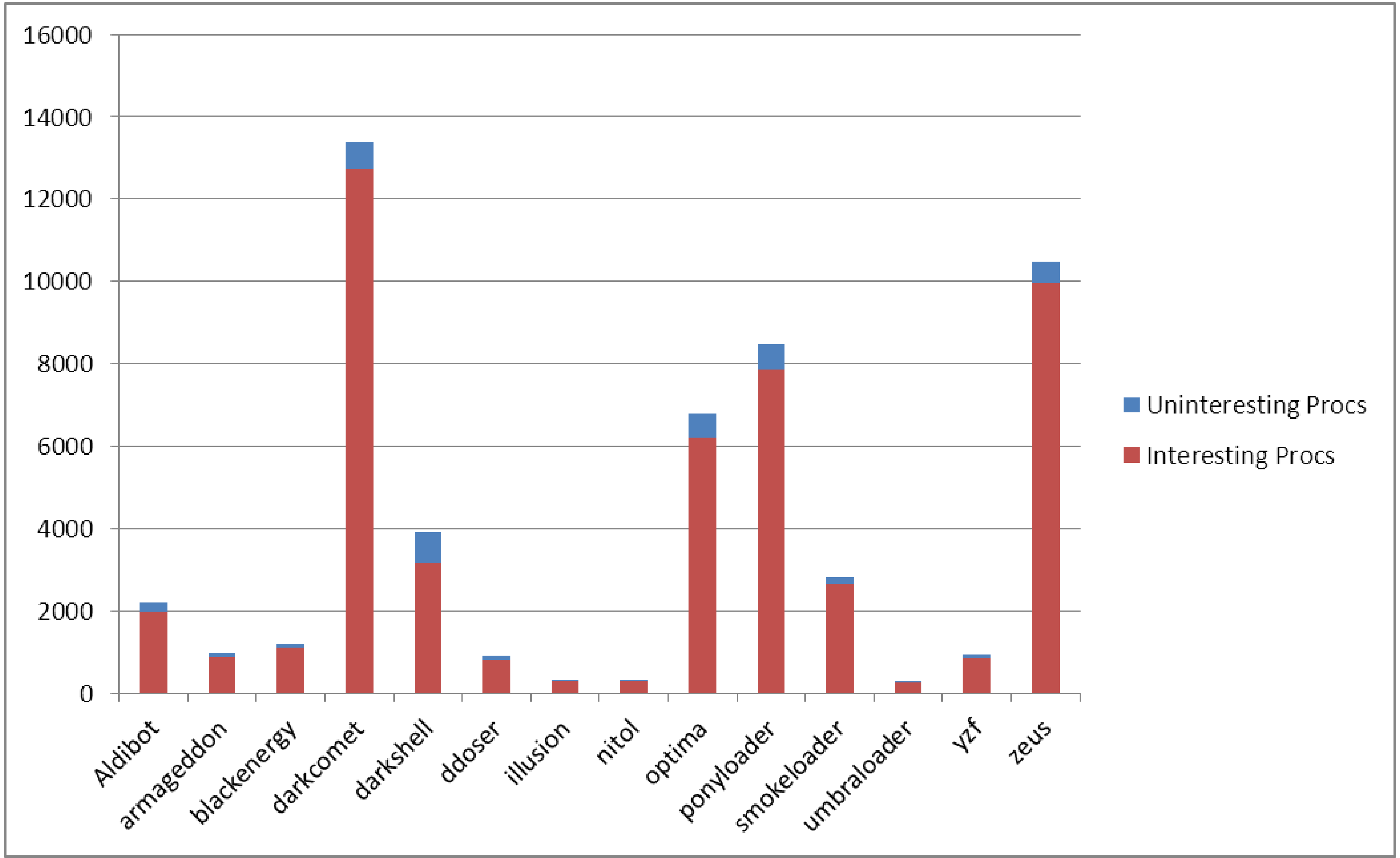


Empirical Results

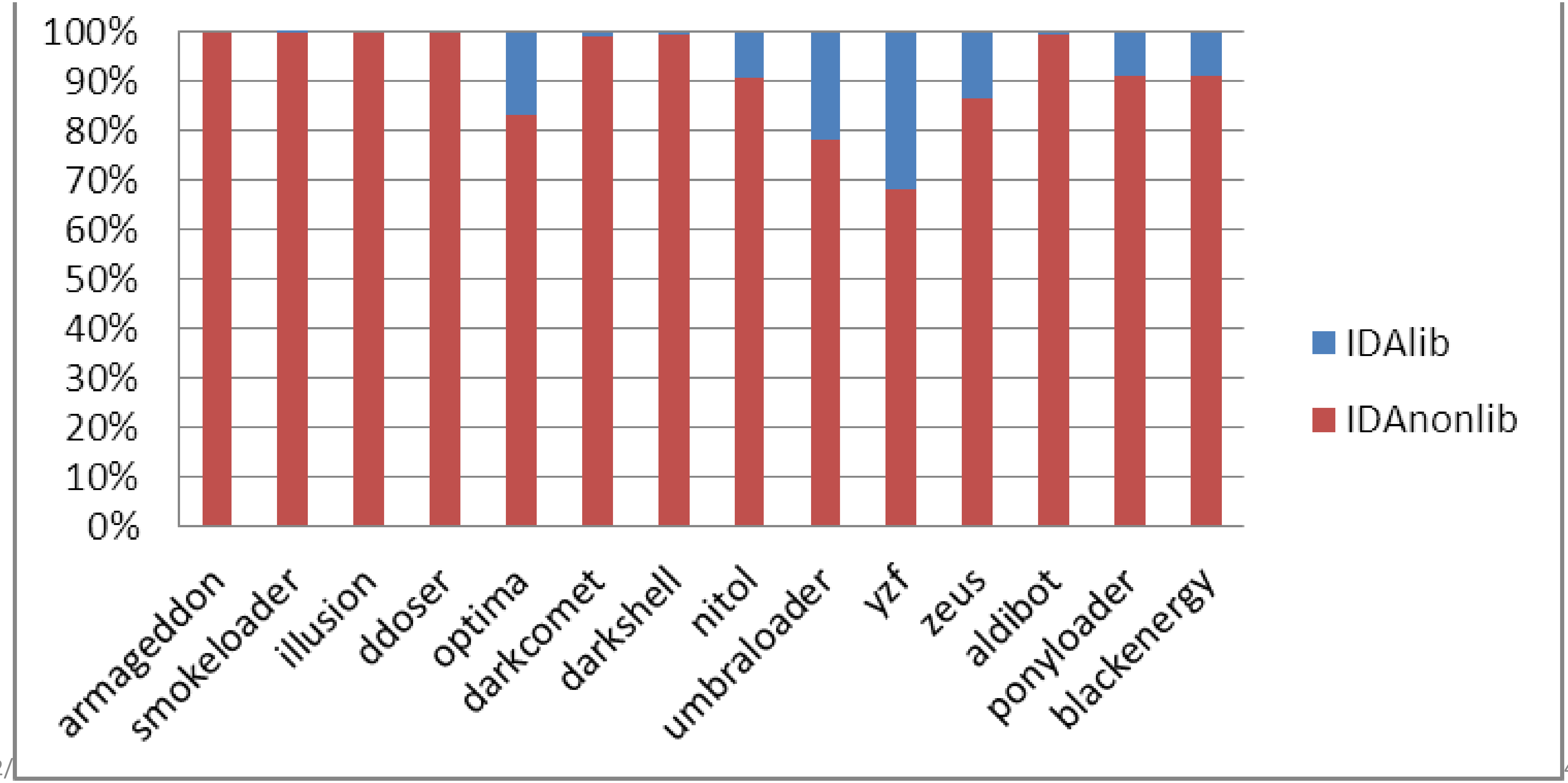
Dataset


Bots harvested in 2013
3000
2500
2000
1500
1000
500
0
aldibot
armageddon
blackenergy
darkcomet
darkshell
ddoser
illusion
nitol
optima
ponyloader
smokeloader
umbraloader
yzf
zeus
unpacked
original

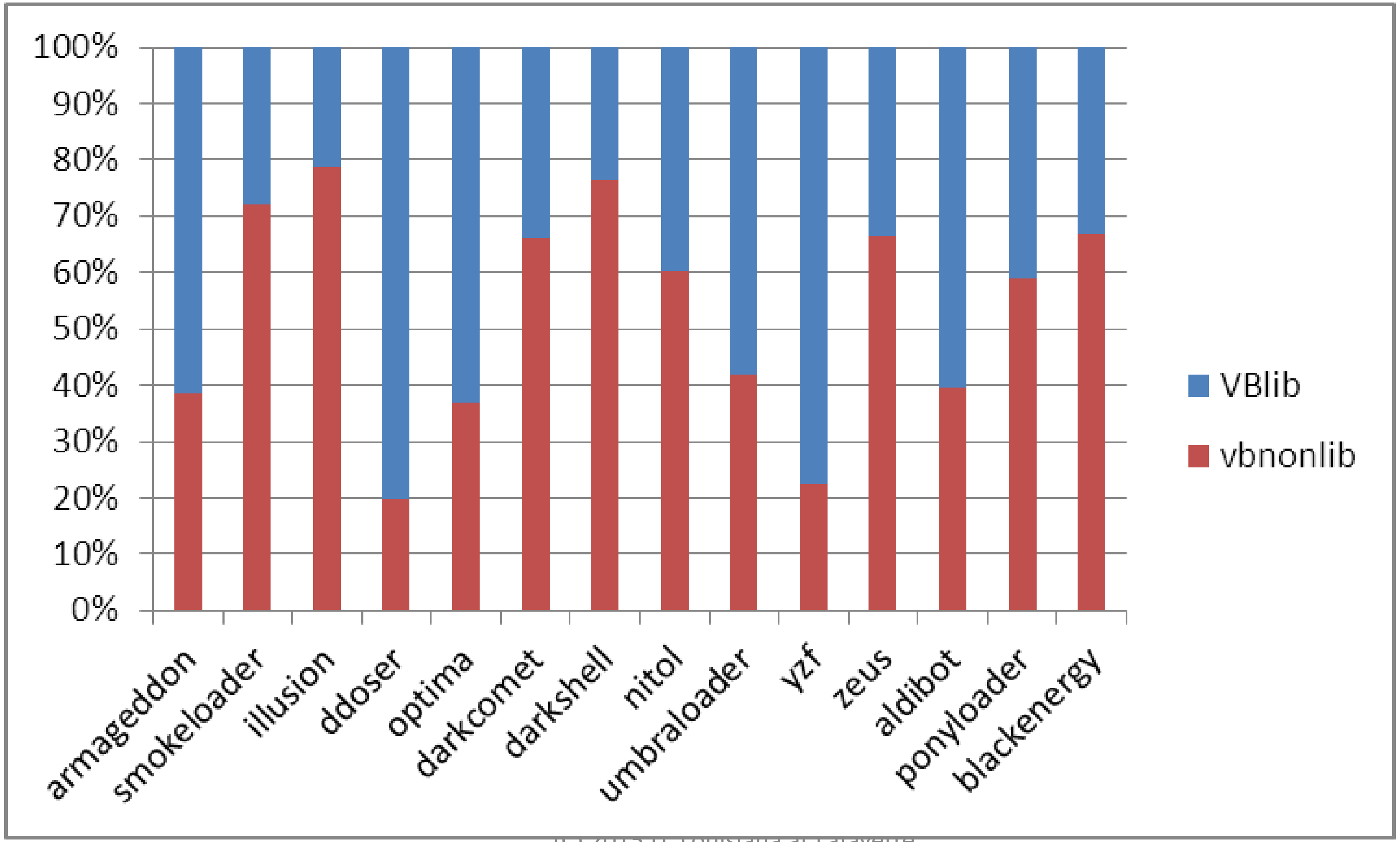
“Interesting” Procedures



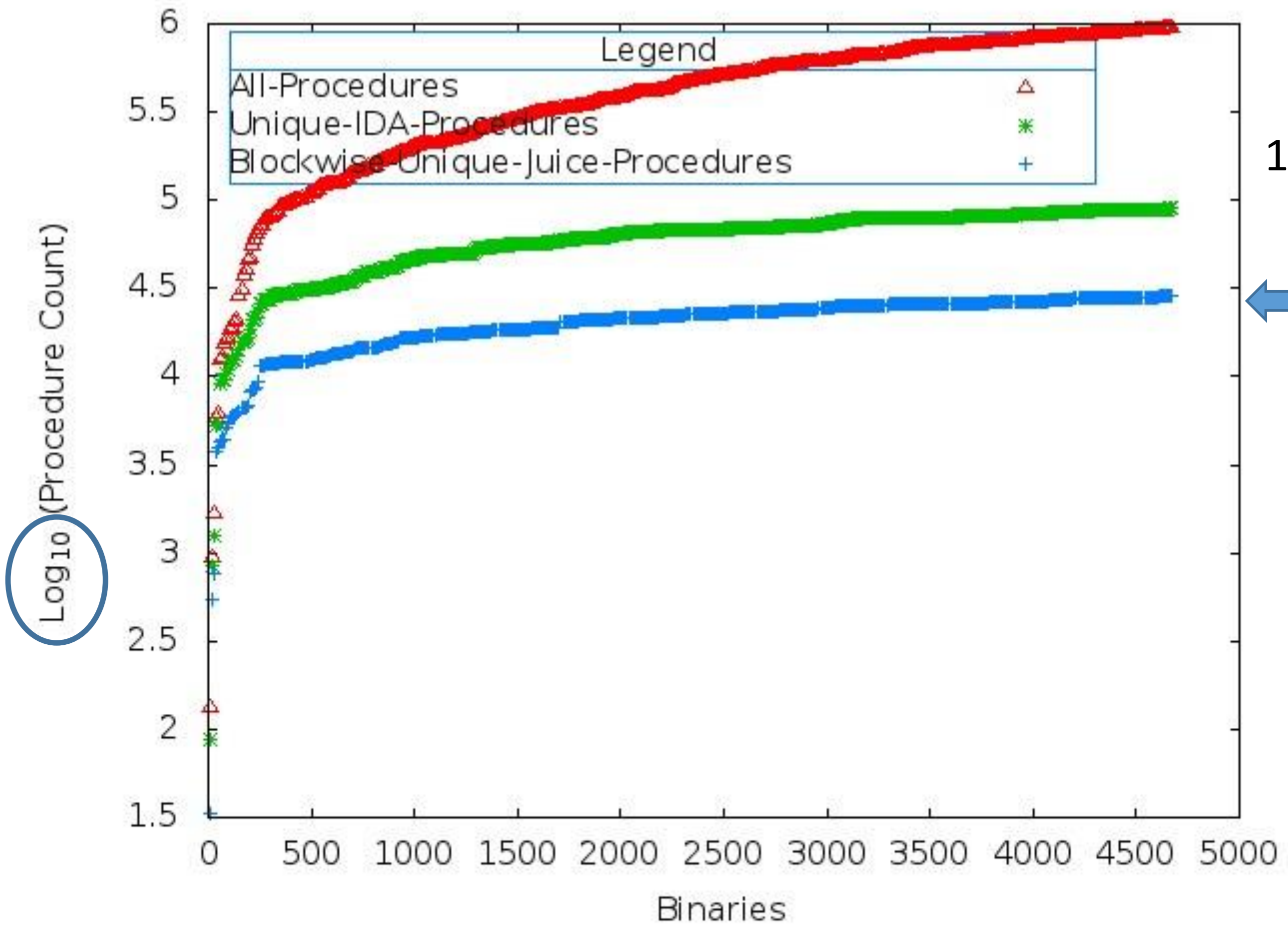
Libraries ID'ed by IDA



Transitive Library via Semantics



RE Cost Reduction



1.7 Million+ procedures

# of procedures in binaries

105K+ procedures

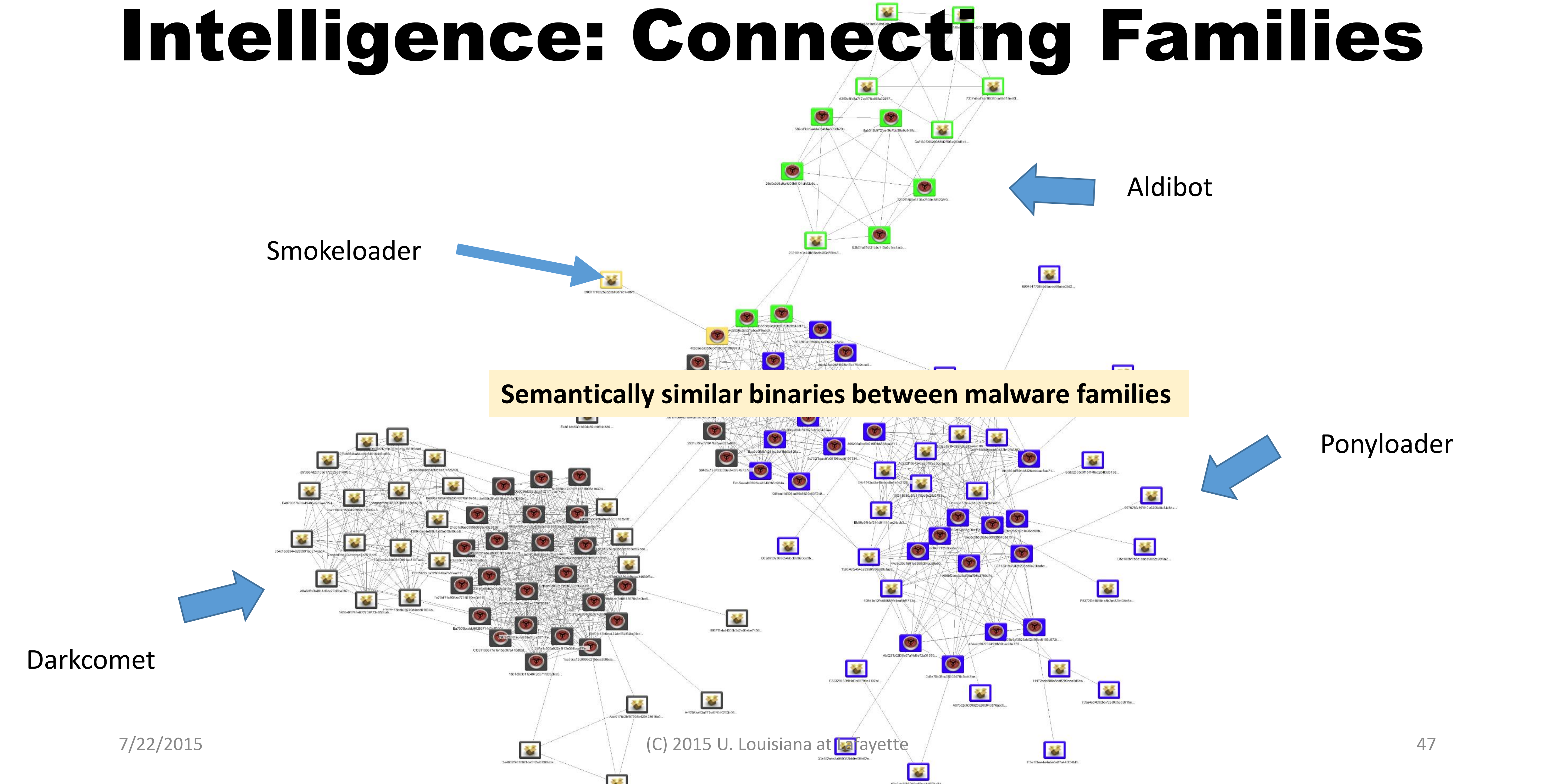
# of IDA unique procedures

# of semantically unique procedures

32K+ semantically unique procedures

Procedures	All	IDA Unique	Juice Unique
Lib Procs	65,113	11,482	4,382
Non Lib Procs	1,644,355 96.2%	93,916 89.1%	27,859 86.4%
Total	1,709,468	105,398	32,241

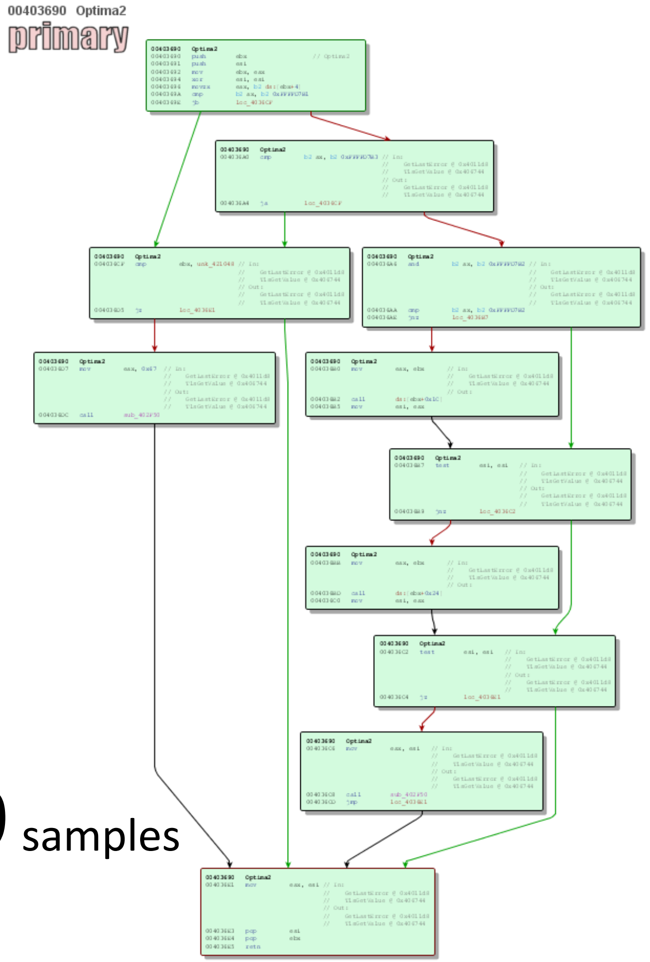
Intelligence: Connecting Families

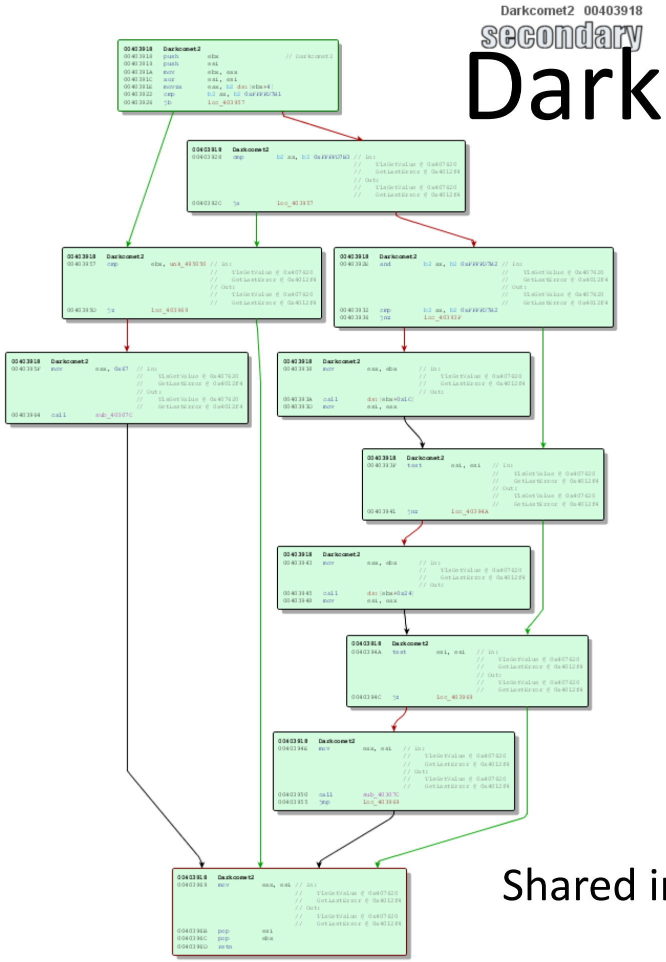


Intelligence: Code Sharing

Non-Lib Unpacked Procedure

Optima



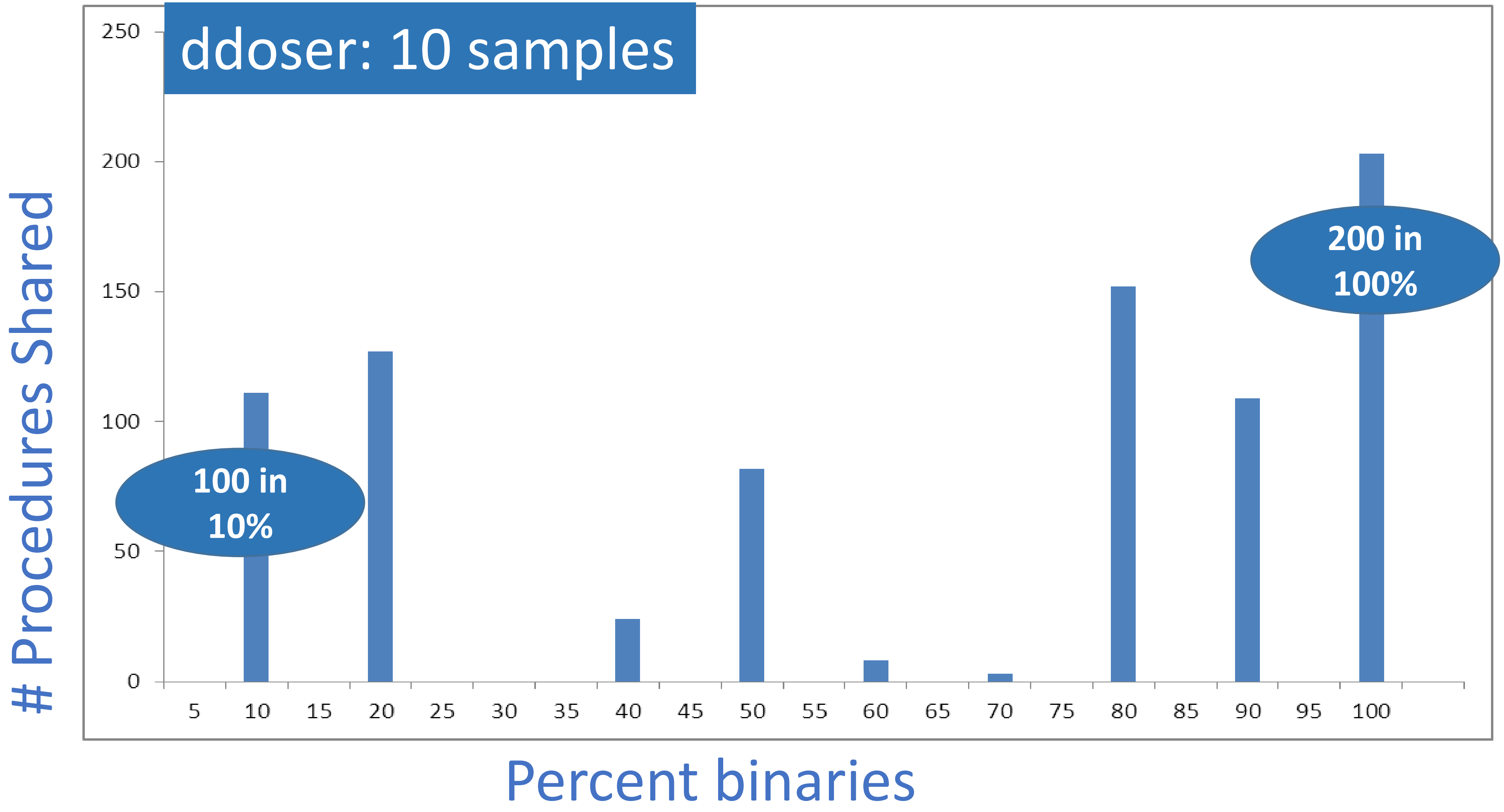


DarkComet

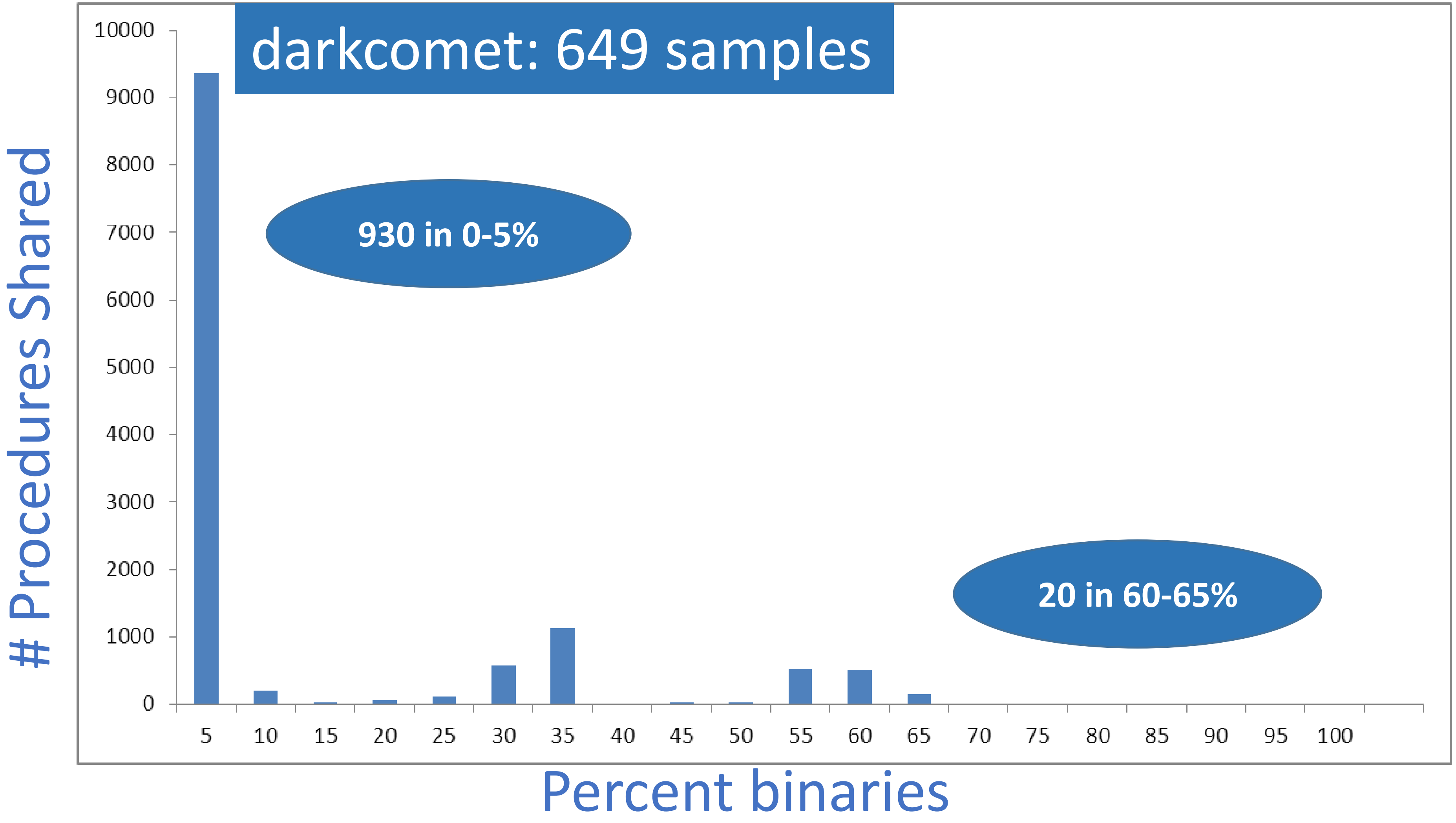
Shared in 13/159 samples

Shared in 65/319 samples

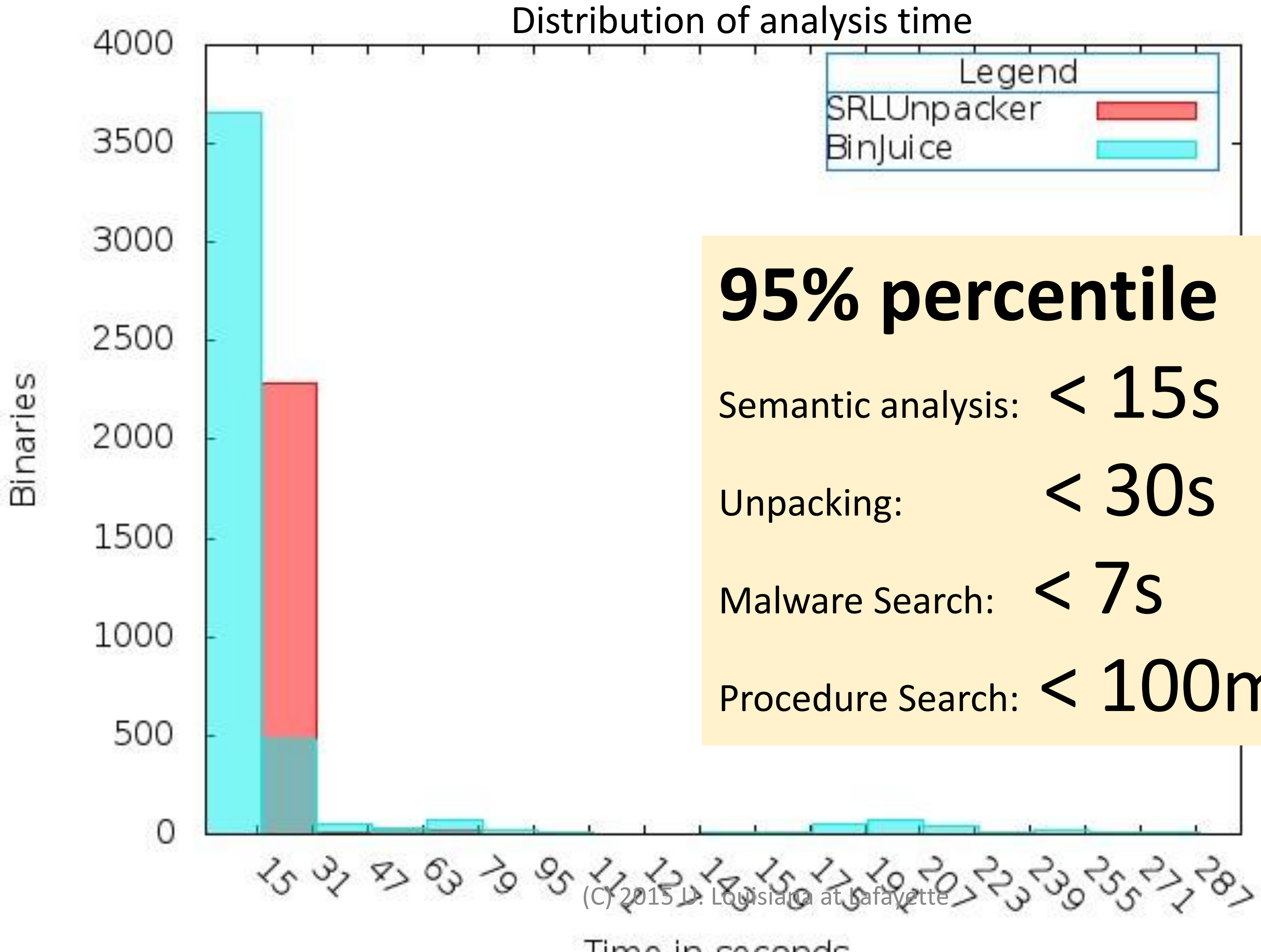
Intelligence: Code Evolution



Intelligence: Needle in Haystack



Performance



95% percentile

Semantic analysis: < 15s

Unpacking: < 30s

Malware Search: < 7s

Procedure Search: < 100ms

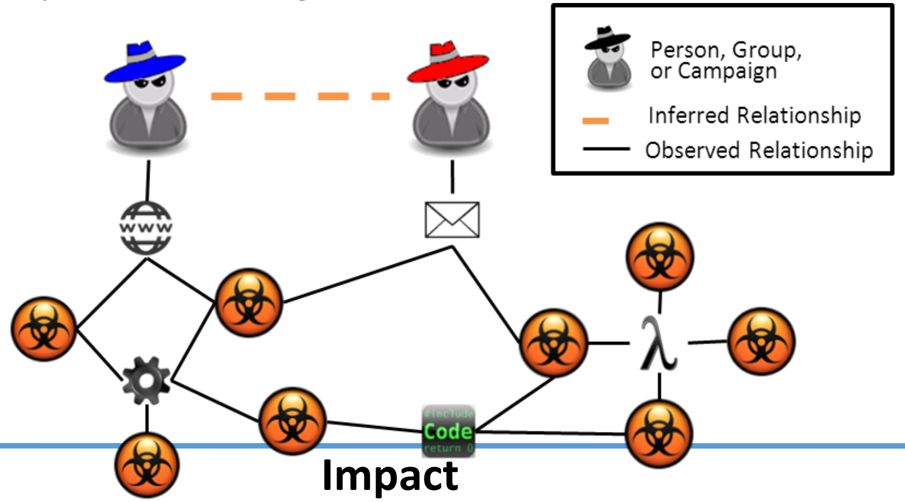
VirusBattle: In a nutshell

Key Innovations

- Automated unpacking using VM introspection
- Semantic fingerprints, as against bits-based fingerprints
- Innovative 2-tier search algorithm for fast searches
- Search at various granularity:
  Whole binary, procedures, blocks, strings
- Interfaces with Palantir's Forensic Investigation platform

Application

Rapidly extract intelligence from malware



Performance

Component	Time(*)	Accuracy
Unpacker	30 sec	97%
Semantic Juice	15 sec	
Binary search	7 sec	95%
Procedure search	100ms	

* Based on analysis of 2,500 botnets binaries; ** Max time to process 95% of files

Impact

- **Order of magnitude improvement in malware analysts capability**
  - Unpacking time:
    - Reduced from days/weeks to minutes
  - Analysis work:
    - Reduce efforts from weeks/months to minutes
  - New capability:
    - **Build knowledge base of analysis indexed on similar code**
    - **Share analysts' experience across malware families**

Blackhat Sound Bytes

- Malware repositories are great source of intelligence
- Semantic juice peers through code obfuscation
- Semantic hashing enables fast search over large repositories
- VM Introspection gives you X-Ray vision over malware
- VirusBattle.com: Malware Intelligence Mining in the Cloud

Contact

Prof. Arun Lakhotia

Vivek Notani

University of Louisiana at Lafayette, Lafayette, LA, USA

arun@louisiana.edu

vxn4849@louisiana.edu

Extras

MinHash: A form of LSH

- Consider Set A and Set B
- Let h(x)->int be a function that takes a member of A or B and gives an integer
- Let h_{min} (s) represent minimum member of set s w.r.t. h.
- Then,

$$Pr(h_{min}(A)=h_{min}(B)) = J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Problem: High Variance!

MinHash Signatures:

- Compose d minhash functions:
  - Signature Match then implies each of the d functions agree on match
  - Pr (sig(A)=sig(B)) = $J(A,B)^d$

Problem: Too many False Negatives!

- Check r minhash signatures:
  - A Match then implies atleast one of the r signatures agree on match
  - Pr (match(A,B)) = $1 - (1 - J(A,B)^d)^r$