Statistical Similarity of Binaries

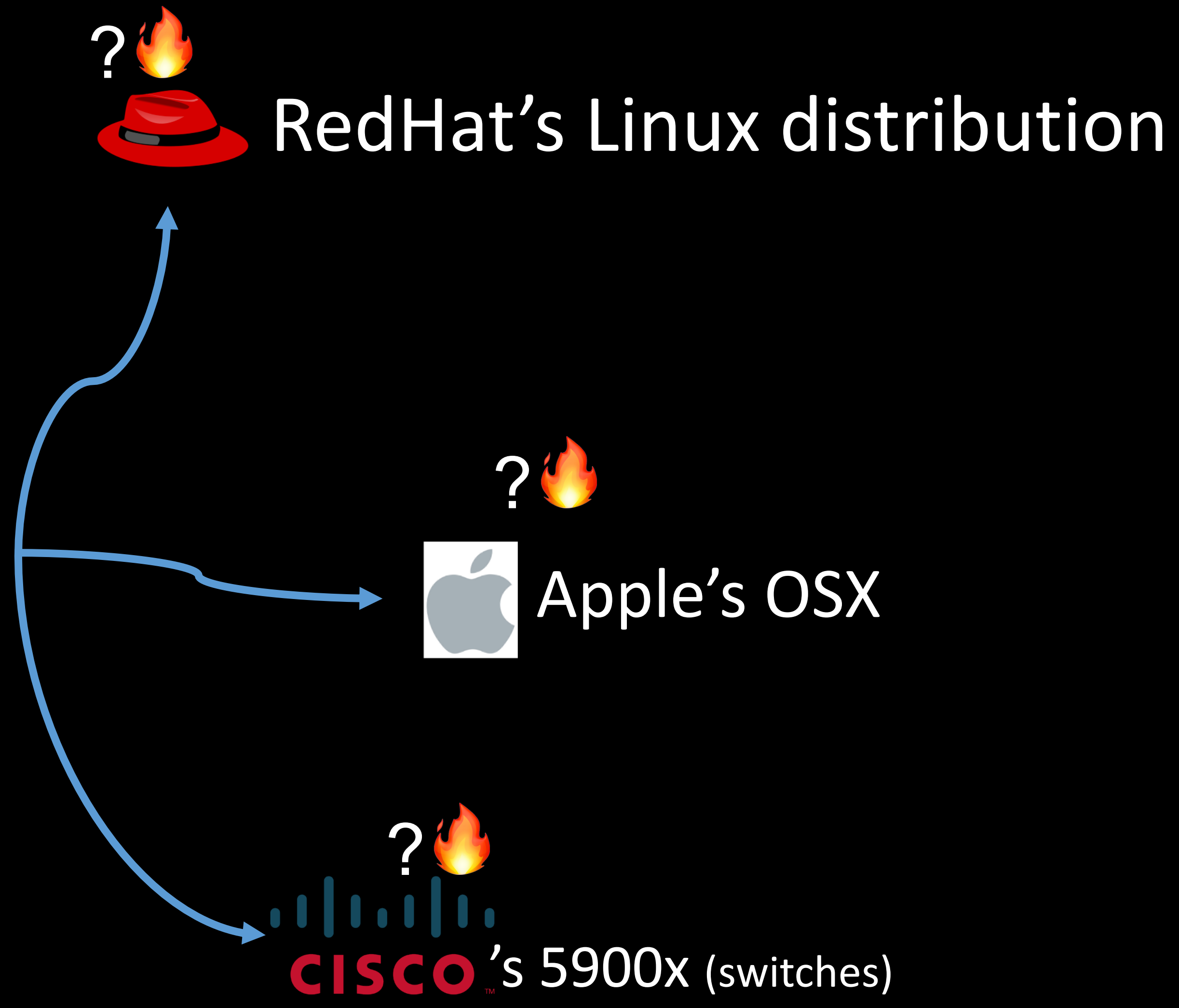
Yaniv David, Nimrod Partush, Eran Yahav @ TECHNION Israel Institute of Technology

*The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement n° 615688– ERC- COG-PRIME.

Motivation

Network time protocol (***ntpd***)

Version	Release Date
ntp-4.2.8	Dec 2014
ntp-4.2.6	Dec 2009
ntp-4.2.4	Dec 2006
ntp-4.2.2	Jun 2006
ntp-4.2.0	Oct 2003
ntp-4.1.2	Jul 2003
ntp-4.1.1	Feb 2002
ntp-4.1.0	Aug 2001
ntp-4.0.99	Jan 2000
ntp-4.0.90	Nov 1998
ntp-4.0.73	Jun 1998
ntp-4.0.72	Feb 1998
ntp-4.0	Sep 1997
xntp3-5.86.5	Oct 1996
xntp3.5f	Apr 1996
xntp3.3wy	Jun 1994
xntp3	Jun 1993
xntp2	Nov 1989



(table source: https://queue.acm.org/detail.cfm?id=2717320)

Semantic Similarity Wish List

- Given q (query) and set T (targets) rank targets based on similarity to q
- Precise - avoid false positives
- Flexible – find similiarities across
  - Different compiler versions
  - Different compiler vendors
  - Different versions of the same code
- Work on stripped binaries

Challenge: Finding Similar Procedures

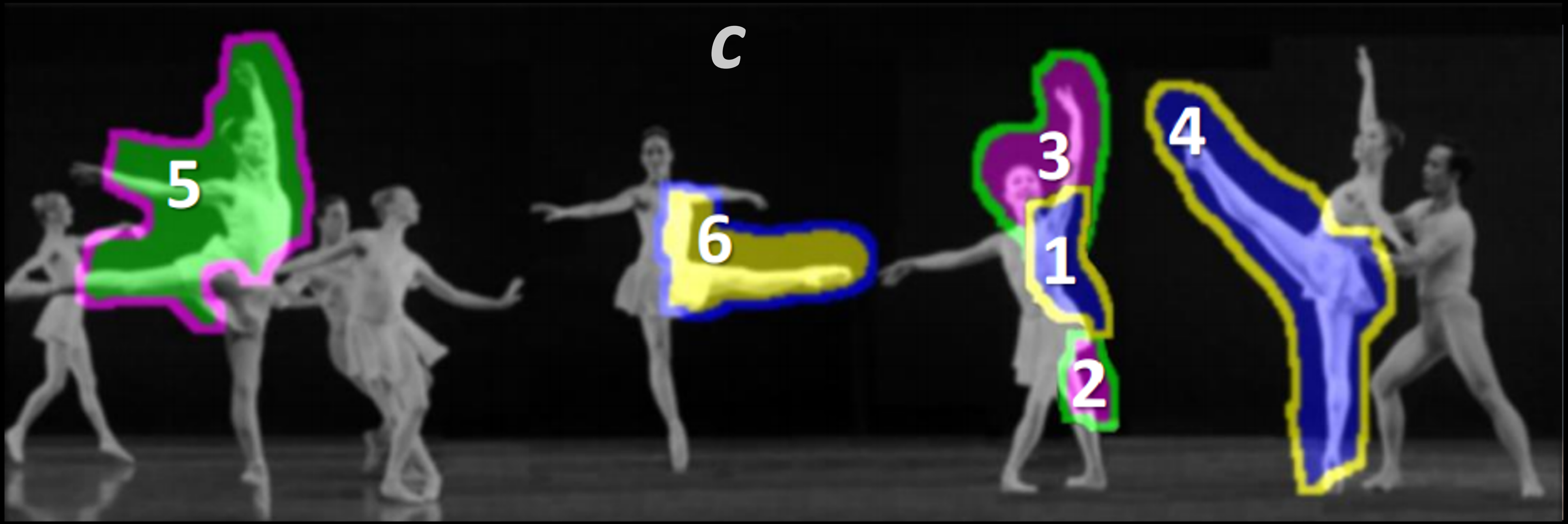
```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```

? ≈

```
mov   r9, 13h
mov   r12, rbx
add   rbp, 3
mov   rsi, rbp
lea   rdi, [r12+3]
mov   [r12+2], bl
lea   r13d, [rcx+r9]
shr   eax, 8
```

Heartbleed, gcc v.4.9

Heartbleed, clang v.3.5



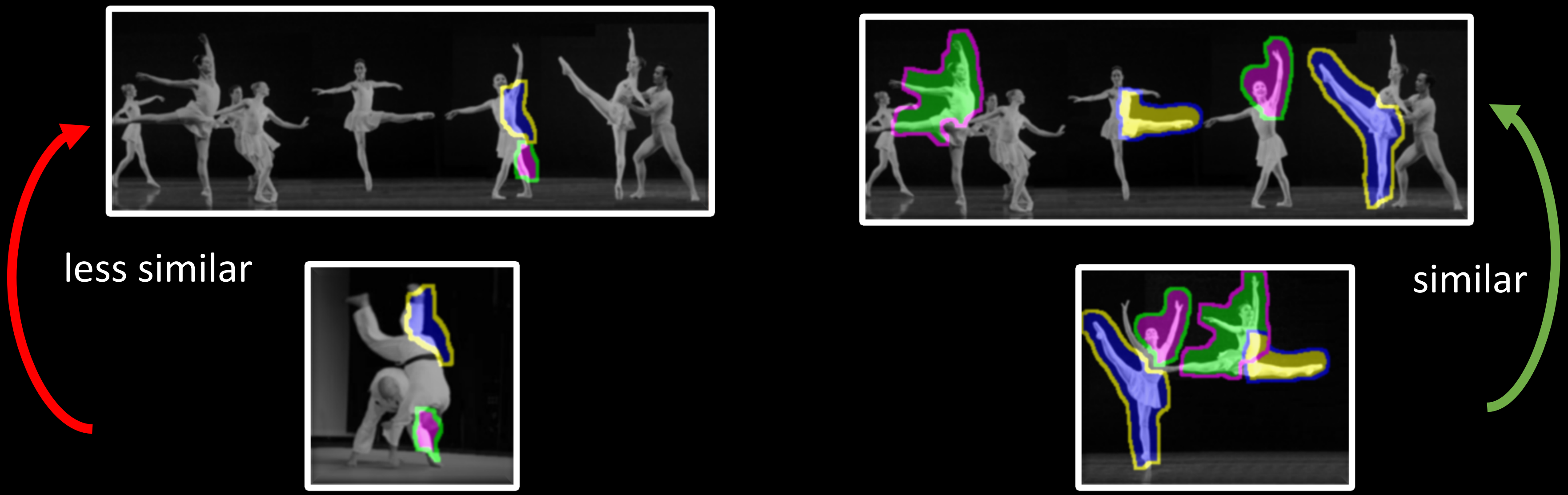
a

b

Images courtesy of Irani et al.

Similarity by Composition - Irani et al. [2006]

- *image1* is similar to a *image2* if you can compose *image1* from the segments of *image2*



- Segments can be transformed
  - rotated, scaled, moved
- Segments of (statistical) significance, give more evidence
  - black background should be much less accounted for

Similarity of Binaries: 3 Step Recipe

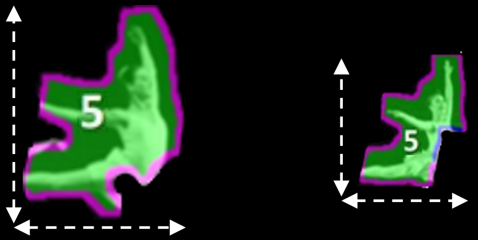
1. Decomposition


```
① shr   eax, 8
② lea   r14d, [r12+13h]
③ mov   r13, rbx
  lea   rcx, [r13+3]
④ mov   [r13+1], al
⑤ mov   [r13+2], r12b
```

Heartbleed, gcc v.4.9 -03

---

2. Pairwise Semantic Similarity



Heartbleed, gcc v.4.9 -03

```
③ mov   r13, rbx
  lea   rcx, [r13+3]
```

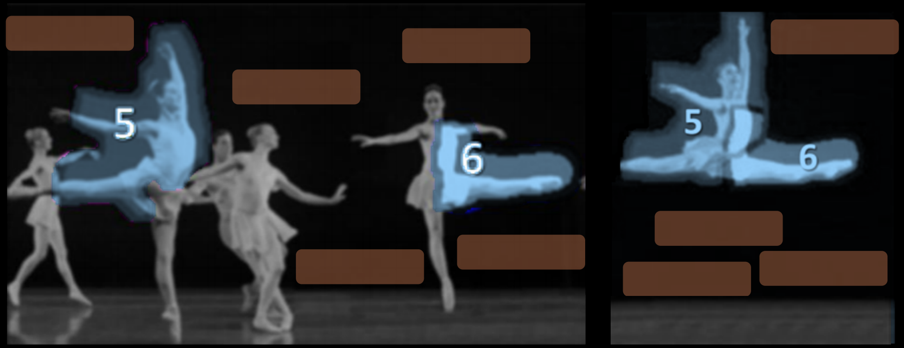
≈?

```
③ mov   r12, rbx
  lea   rdi, [r12+3]
```

Heartbleed, clang v.3.5 -03

---

3. Statistical Similarity Evidence

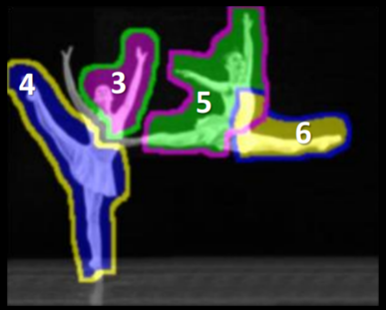



```
① shr   eax, 8
③ mov   r13, rbx
  lea   rcx, [r13+3]
                      , 8
③ mov   r12, rbx
  lea   rdi, [r12+3]
① shr   eax, 8
```

CORPUS

Similarity of Binaries: 3 Step Recipe

1. Decomposition

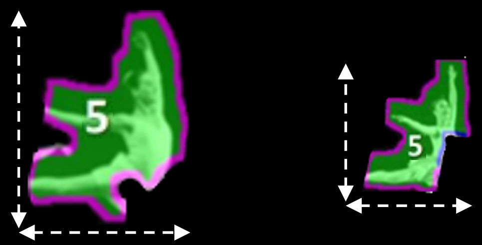



```
① shr   eax, 8
② lea   r14d, [r12+13h]
③ mov   r13, rbx
  lea   rcx, [r13+3]
④ mov   [r13+1], al
⑤ mov   [r13+2], r12b
```

Heartbleed, gcc v.4.9 -03

---

2. Pairwise Semantic Similarity



Heartbleed, gcc v.4.9 -03

```
③ mov   r13, rbx
  lea   rcx, [r13+3]
```

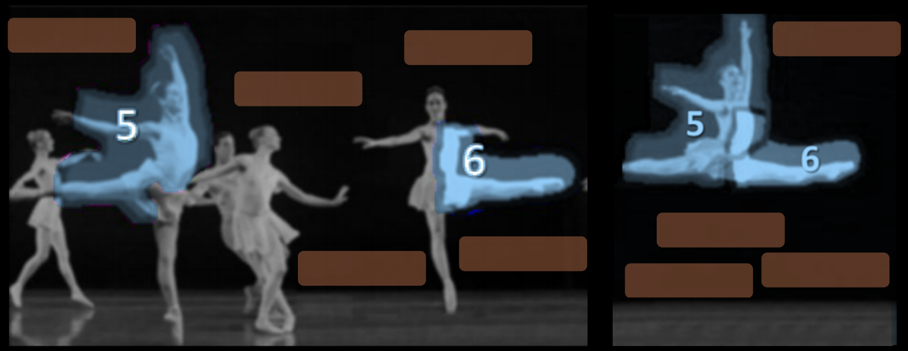
≈?

```
③ mov   r12, rbx
  lea   rdi, [r12+3]
```

Heartbleed, clang v.3.5 -03

---

3. Statistical Similarity Evidence




```
① shr   eax, 8
③ mov   r13, rbx
  lea   rcx, [r13+3]
              , 8
③ mov   r12, rbx
  lea   rdi, [r12+3]
① shr   eax, 8
```

CORPUS

Step 1 - Procedure Decomposition

We need to decompose procedures into comparable units

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr     eax, 8
lea     r14d, [r12+13h]
mov     r13, rbx
lea     rcx, [r13+3]
mov     [r13+1], al
mov     [r13+2], r12b
mov     rdi, rcx
```

Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```

Step 1 - Procedure Decomposition

Inputs: rbx

```
mov    r13, rbx
lea    rcx, [r13+3]
mov    rdi, rcx
```

Vars: rdi,rcx,r13

Step 1 - Procedure Decomposition

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```

Step 1 - Procedure Decomposition

- Aapplying program slicing on the basic-block level until all variables are covered

```
1: shr    eax, 8
2: lea    r14d, [r12+13h]
3: mov    r13, rbx
4: lea    rcx, [r13+3]
5: mov    [r13+1], al
6: mov    [r13+2], r12b
7: mov    rdi, rcx
```

- We call these basic-block slices **Strands**

Similarity of Binaries: 3 Step Recipe

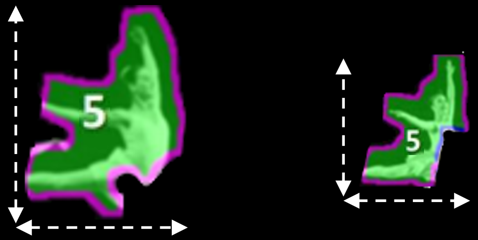
1. Decomposition


```
① shr   eax, 8
② lea   r14d, [r12+13h]
③ mov   r13, rbx
  lea   rcx, [r13+3]
④ mov   [r13+1], al
⑤ mov   [r13+2], r12b
```

Heartbleed, gcc v.4.9 -03

---

2. Pairwise Semantic Similarity



Heartbleed, gcc v.4.9 -03

```
③ mov   r13, rbx
  lea   rcx, [r13+3]
```

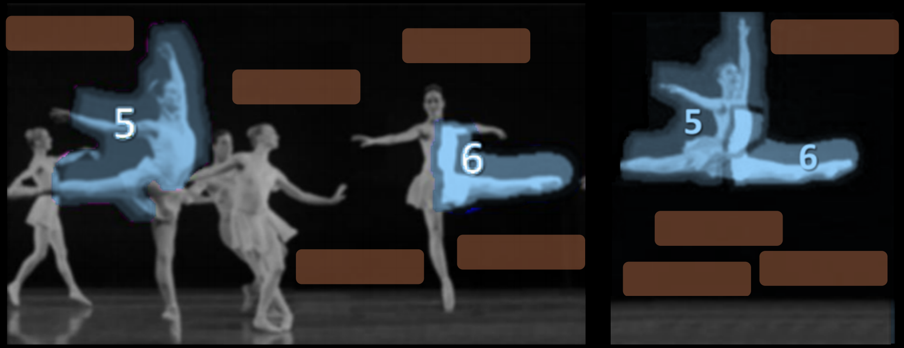
≈?

```
③ mov   r12, rbx
  lea   rdi, [r12+3]
```

Heartbleed, clang v.3.5 -03

---

3. Statistical Similarity Evidence




```
① shr   eax, 8
③ mov   r13, rbx
  lea   rcx, [r13+3]
                      , 8
③ mov   r12, rbx
  lea   rdi, [r12+3]
① shr   eax, 8
```

CORPUS

Step 2 – Pairwise Semantic Similairity

```
mov    r13, rbx
lea    rcx, [r13+3]
mov    rdi, rcx
```

BAP + Smack

```
v1    := rbx
r13   := v1
v2    := r13 + 3
v3    := int_to_ptr(v2)
rcx   := v3
v4    := rcx
rdi   := v4
```

Strand 3 @Heartbleed, gcc v.4.9 -03

Strand 3 in Boogie representation

Step 2 – Pairwise Semantic Similairity

```
v1    := r12
v2    := 13h + v1
v3    := int_to_ptr(v2)
r14   := v3
v4    := 18h
rsi   := v4
v5    := v4 + v3
rax   := v5
```

? ≈

```
v1     := 13h
r9     := v1
v2     := rbx
v3     := v2 + v3
v4     := int_to_ptr(v3)
r13    := v4
v5     := v1 + 5
rsi    := v5
v6     := v5 + v4
rax    = v6
```

Heartbleed, gcc v.4.9 -03
Strand 6

Heartbleed, clang v.3.5 -03
Strand 11

Step 2 – Pairwise Semantic Similairity

```
v1     := r12
v2     := 13h + v1
v3     := int_to_ptr(v2)
r14    := v3
v4     := 18h
rsi    := v4
v5     := v4 + v3
rax    := v5
```

Heartbleed, gcc v.4.9
Strand 6

? ≈

```
v1        := 13h
r9        := v1
v2        := rbx
v3        := v2 + v3
v4        := int_to_ptr(v3)
r13       := v4
v5        := v1 + 5
rsi       := v5
v6        := v5 + v4
rax       = v6
```

Heartbleed, clang v.3.5
Strand 11

Step 2 – Pairwise Semantic Similairity

Strand $s_q \in q$
Inputs: $r12_q$

```
v1_q   := r12_q
v2_q   := 13h + v1_q
v3_q   := int_to_ptr(v2_q)
r14_q  := v3_q
v4_q   := 18h
rsi_q  := v4_q
v5_q   := v4_q + v3_q
rax_q  := v5_q
```

Variables: $v1_q, v2_q, v3_q, r14_q, v4_q, rsi_q, v5_q, rax_q$

Strand $s_t \in t \in T$
Inputs: rbx_t

```
v1_t   := 13h
r9_t   := v1_t
v2_t   := rbx_t
v3_t   := v2_t + v3_t
v4_t   := int_to_ptr(v3_t)
r13_t  := v4_t
v5_t   := v1_t + 5
rsi_t  := v5_t
v6_t   := v5_t + v4_t
rax_t  := v6
```

Variables: $v1_t, r9_t, v2_t, v3_t, v4_t, r13_t, v5_t, rsi_t, v6_t, rax_t$

Step 2 – Pairwise Semantic Similarity

assume:
$r12_q == rbx_t$
$s_q; s_t;$

assert:
$v_{1_q} == v_{1_t}$
…
$rax_q == raxt$

assert:
…
…
…

assert:
$v_{1_q} == rax_t$
…
$rax_q == v_{1_t}$

Max number of equal variables

Step 2 – Pairwise Semantic Similairity

assume $r12_q == rbx_t$

$v1_q := \mathbf{r12_q}$
$v2_q := 13h + v1_q$
$v3_q := int\_to\_ptr(v2_q)$
$\mathbf{r14_q} := v3_q$
$v4_q := 18h$
$\mathbf{rsi_q} := v4_q$
$v5_q := v4_q + v3_q$
$\mathbf{rax_q} := v5_q$

$v1_t := 13h$
$\mathbf{r9_t} := v1_t$
$v2_t := \mathbf{rbx_t}$
$v3_t := v2_t + v3_t$
$v4_t := int\_to\_ptr(v3_t)$
$\mathbf{r13_t} := v4_t$
$v5_t := v1_t + 5$
$\mathbf{rsi_t} := v5_t$
$v6_t := v5_t + v4_t$
$\mathbf{rax_t} := v6$

assert
$v1_q==v2_t$, $v2_q==v3_t$, $v3_q==v4_t$, $r14_q==r13_t$
$v4_q==v5_t$, $rsi_q==rsi_t$,$v5_q==v6_t$, $rax_q==rax_t$

Step 2 - Quantify Semantic Similarity

- $\mathrm{VCP}(s_q, s_t)$ = MaxEqualVars(s_q, s_t) / $|s_q|$
  - Variable Containment Proportion
  - An *asymmetric* relation
  - Using dataflow information and optimizations make this calculation feasible

Step 2 – Pairwise Semantic Similairity

assume $r12_q$ == rbx_t

```
v1_q   = r12_q
v2_q   = 13h + v1_q
v3_q   = int_to_ptr(v2_q)
r14_q  = v3_q
v4_q   = 18h
rsi_q  = v4_q
v5_q   = v4_q + v3_q
rax_q  = v5_q
```

```
v1_t   = 13h
r9_t   = v1_t
v2_t   = rbx_t
v3_t   = v2_t + v3_t
v4_t   = int_to_ptr(v3_t)
r13_t  = v4_t
v5_t   = v1_t + 5
rsi_t  = v5_t
v6_t   = v5_t + v4_t
rax_t  = v6
```

$\text{VCP}(s_q; s_t) = 8/8$

assert

$v1_q{==}v2_t$, $v2_q{==}v3_t$, $v3_q{==}v4_t$, $r14_q{==}r13_t$

$v4_q{==}v5_t$, $rsi_q{==}rsi_t$, $v5_q{==}v6_t$, $rax_q{==}rax_t$

Step 2 – Pairwise Semantic Similairity

assume $r12_q == rbx_t$

$v1_q = \mathbf{r12}_q$
$v2_q = 13h + v1_q$
$v3_q = int\_to\_ptr(v2_q)$
$\mathbf{r14}_q = v3_q$
$v4_q = 18h$
$\mathbf{rsi}_q = v4_q$
$v5_q = v4_q + v3_q$
$\mathbf{rax}_q = v5_q$

$v1_t = 13h$
$\mathbf{r9}_t = v1_t$
$v2_t = \mathbf{rbx}_t$
$v3_t = v2_t + v3_t$
$v4_t = int\_to\_ptr(v3_t)$
$\mathbf{r13}_t = v4_t$
$v5_t = v1_t + 5$
$\mathbf{rsi}_t = v5_t$
$v6_t = v5_t + v4_t$
$\mathbf{rax}_t = v6$

$VCP(s_q; s_t) = 8/8$

$VCP(s_t; s_q) = 8/10$

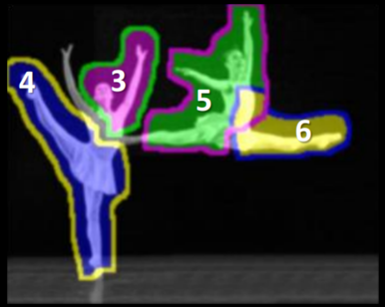
assert

$v1_q == v2_t$, $v2_q == v3_t$, $v3_q == v4_t$, $r14_q == r13_t$

$v4_q == v5_t$, $rsi_q == rsi_t$, $v5_q == v6_t$, $rax_q == rax_t$

Similarity of Binaries: 3 Step Recipe

1. Decomposition

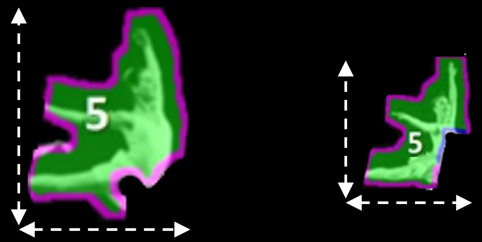



```
① shr   eax, 8
② lea   r14d, [r12+13h]
③ mov   r13, rbx
  lea   rcx, [r13+3]
④ mov   [r13+1], al
⑤ mov   [r13+2], r12b
```

Heartbleed, gcc v.4.9 -03

---

2. Pairwise Semantic Similarity



Heartbleed, gcc v.4.9 -03

```
③ mov   r13, rbx
  lea   rcx, [r13+3]
```

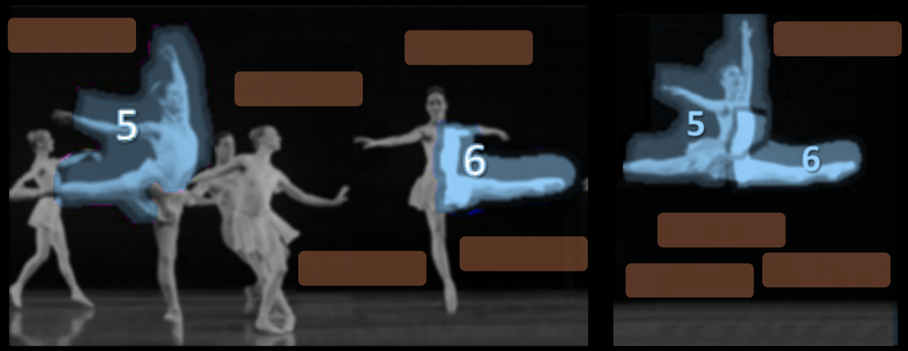
≈?

```
③ mov   r12, rbx
  lea   rdi, [r12+3]
```

Heartbleed, clang v.3.5 -03

---

3. Statistical Similarity Evidence



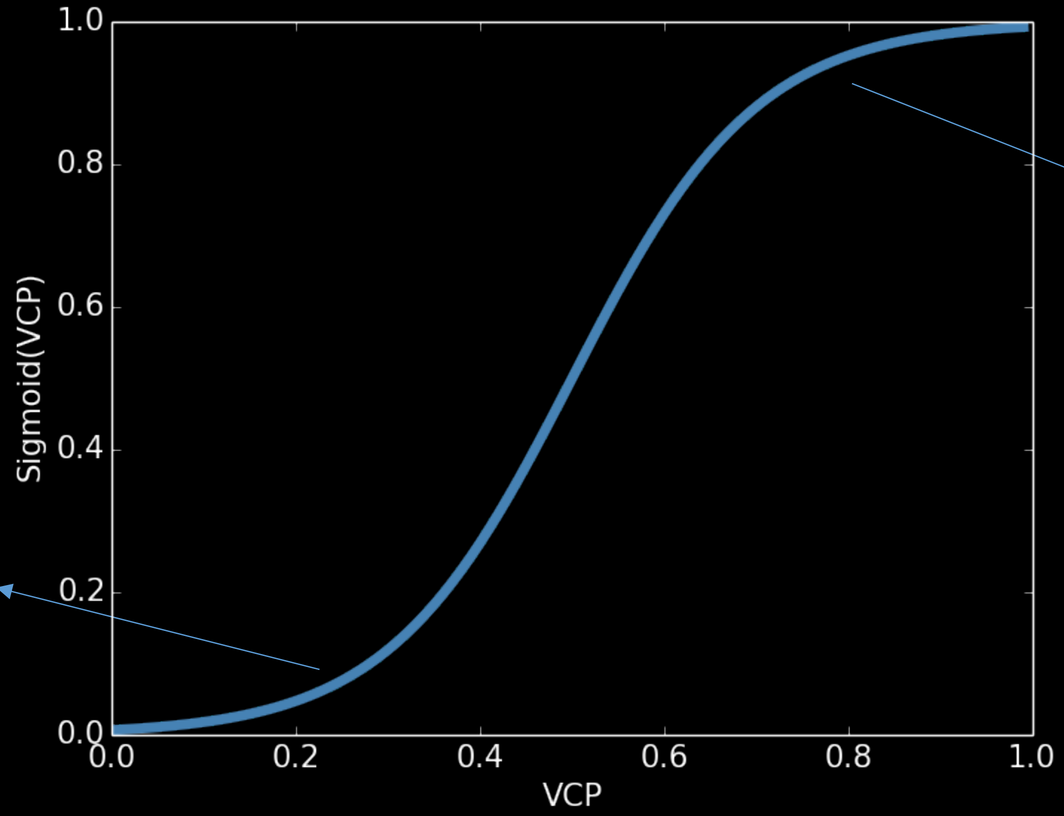

```
① shr   eax, 8
③ mov   r13, rbx
  lea   rcx, [r13+3]
                        , 8
③ mov   r12, rbx
  lea   rdi, [r12+3]
① shr   eax, 8
```

CORPUS

Step 3 – Statistical Evidence

- We need to turn VCP into a *probability* that s_q is input-output equivalent to s_t
- $\Pr(s_q|s_t) = sigmoid\left(VCP(s_q, s_t)\right) = \frac{1}{1+e^{-k(VCP((s_q,s_t)-0.5)}}$

"Throw" bad results close to 0



"Throw" good results close to 1

Step 3 – Statistical Evidence

- We need to know how *significant* is s_q
- To do that we use *all* the comparison data available

$q\ VS\ t_1$:

$\Pr\left(s_q | s_{1_{t_1}}\right)$

$\Pr\left(s_q | s_{2_{t_1}}\right)$

…

…

…

$q\ VS\ t_n$:

$\Pr\left(s_q | s_{1_{t_n}}\right)$

$\Pr\left(s_q | s_{2_{t_n}}\right)$

…

$$\Pr\left(s_q \middle| H_0\right) = \frac{\sum_{s_t \in T} \Pr(s_q | s_t)}{|T|}$$

Step 3 – Statistical Evidence

- Define a *Local Evidence Score* to quantify the statistical significance of matching each strand

$$LES(s_q|t) = \log \frac{\max_{s_t \in t} \Pr(s_q|s_t)}{\Pr(s_q|H_0)}$$

Step 3 – Statistical Evidence

① `shr eax, 8` ① `shr eax, 8`

$$\frac{\max_{s_t \in t} \Pr(s_q | s_t)}{\Pr(s_q | H_0)} = \frac{1}{0.08} = 12.5$$

---

③
```
mov     r13, rbx
lea     rcx, [r13+3]
```

③
```
mov     r12, rbx
lea     rdi, [r12+3]
```

$$\frac{\max_{s_t \in t} \Pr(s_q | s_t)}{\Pr(s_q | H_0)} = \frac{1}{0.001} = 1000$$

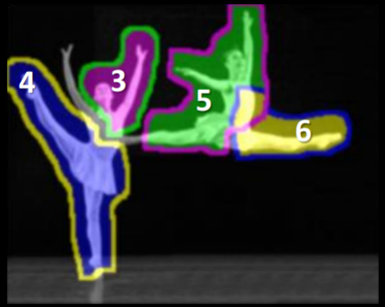
Step 3 - Global Similarity

- Procedures are similar if one can be composed using non-trivial, significantly similar parts of the other

$$GES(q|t) = \sum_{s_q \in q} LES(s_q|t)$$

Similarity of Binaries: Recap

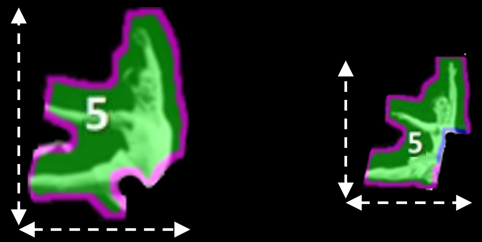
1. Decomposition




```
① shr   eax, 8
② lea   r14d, [r12+13h]
③ mov   r13, rbx
  lea   rcx, [r13+3]
④ mov   [r13+1], al
⑤ mov   [r13+2], r12b
```

Heartbleed, gcc v.4.9 -03

2. Pairwise Semantic Similarity



Heartbleed, gcc v.4.9 -03

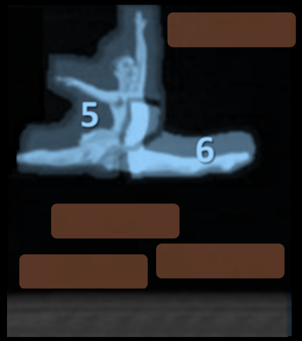
```
③ mov   r13, rbx
  lea   rcx, [r13+3]
```

≈?

```
③ mov   r12, rbx
  lea   rdi, [r12+3]
```

Heartbleed, clang v.3.5 -03

3. Statistical Similarity Evidence




```
① shr   eax, 8
③ mov   r13, rbx
  lea   rcx, [r13+3]
            , 8
③ mov   r12, rbx
  lea   rdi, [r12+3]
① shr   eax, 8
```

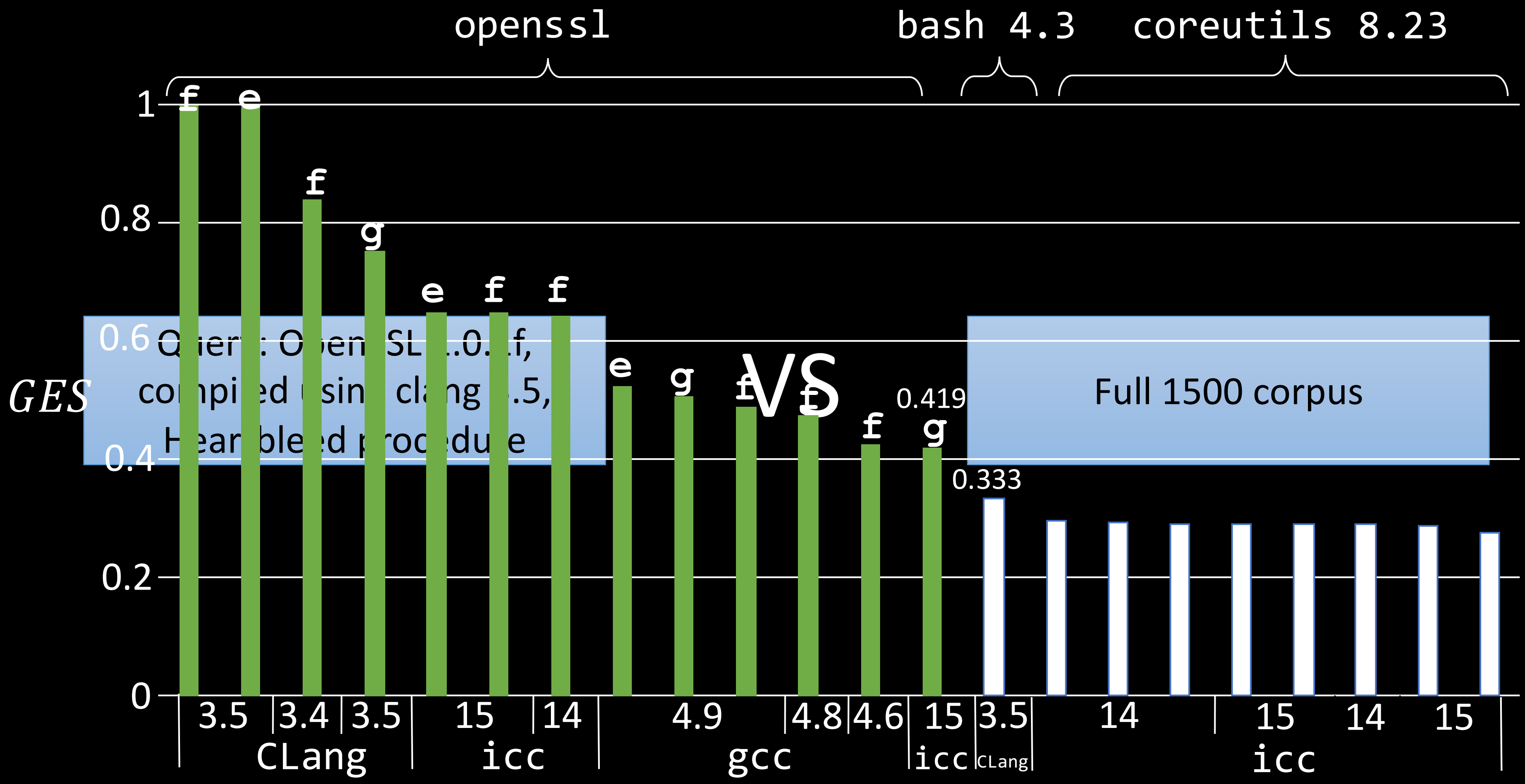
CORPUS

Evaluation - Vulnerabilities

- Corpus
  - Real-world code packages
    - *open-ssl, bash, qemu, wget, ws-snmp, ffmpeg, coreutils*
  - Spanning across product versions
    - e.g. *openssl-1.0.1{e,f,g}*
  - Compiled with *clang 3.{4,5}*, *gcc 4.{6,8,9}* and *icc {14,15}*
- 1500 procedures picked at random

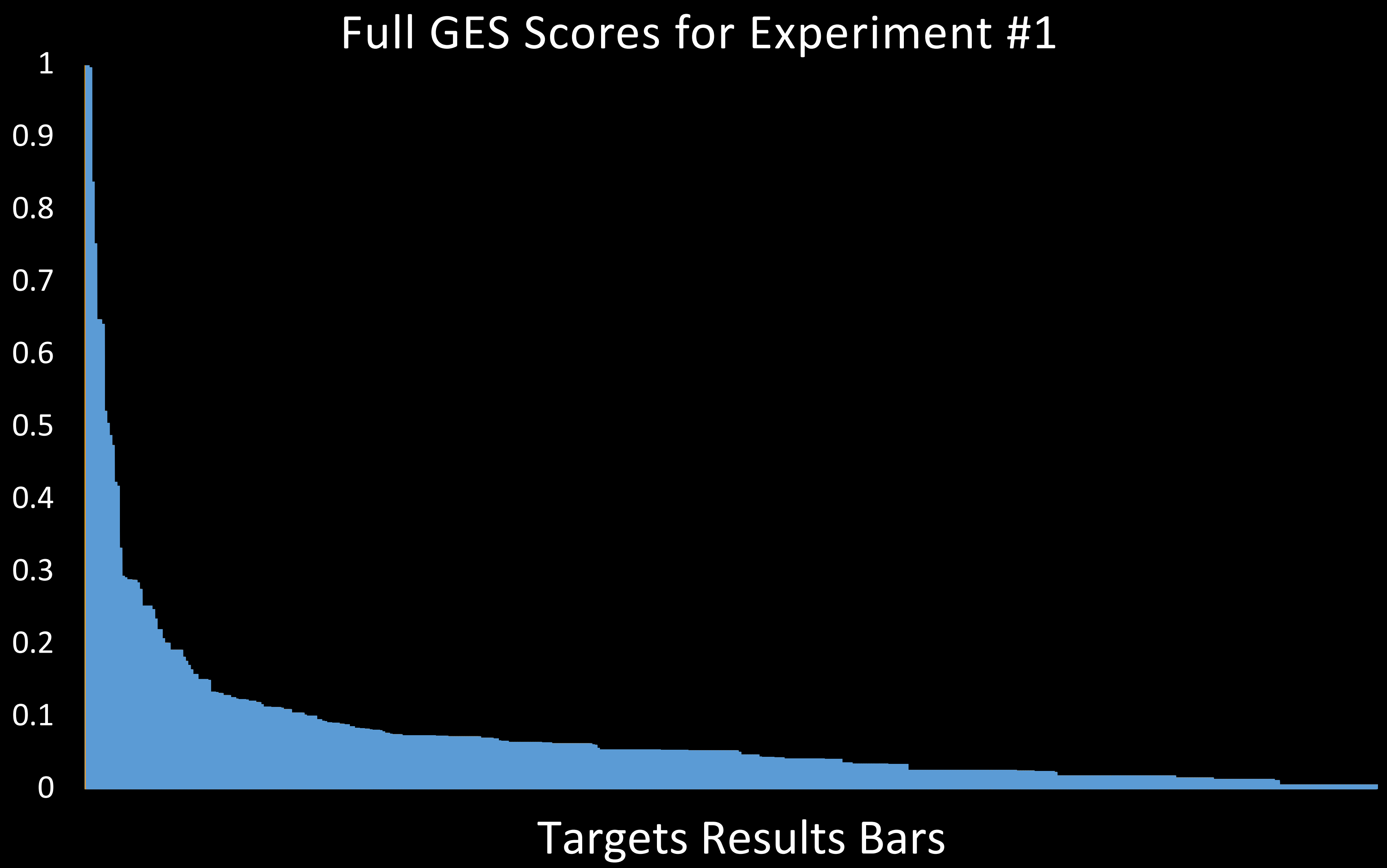
- Queries
  - Focused on vulnerabilities (for motivation's sake)

Results - *Finding Heartbleed*



Compiler version (top), and vendor (bottom)

Results - *Finding Heartbleed*



Results - Vulnerabilities

	Vulnerability	False Positives	False positives rate
1	Heartbleed	0	0
2	Shellshock	3	0.002
3	Venom	0	0
4	Clobberin' Time	19	0.0126
5	Shellshock #2	0	0
6	ws-snmp	1	0.0006
7	wget	0	0
8	ffmpeg	0	0

- Low FP rate
  - Crucial to the vulnerability search scenario
- Previous methods fail at cross-{version,compiler} scenario or produce too many FPs (see paper)

Evaluation – All vs All

- Verified with randomly picked procedures
  - For example – when ff_rev34_decode@ffmpeg-2.4.6 is selected

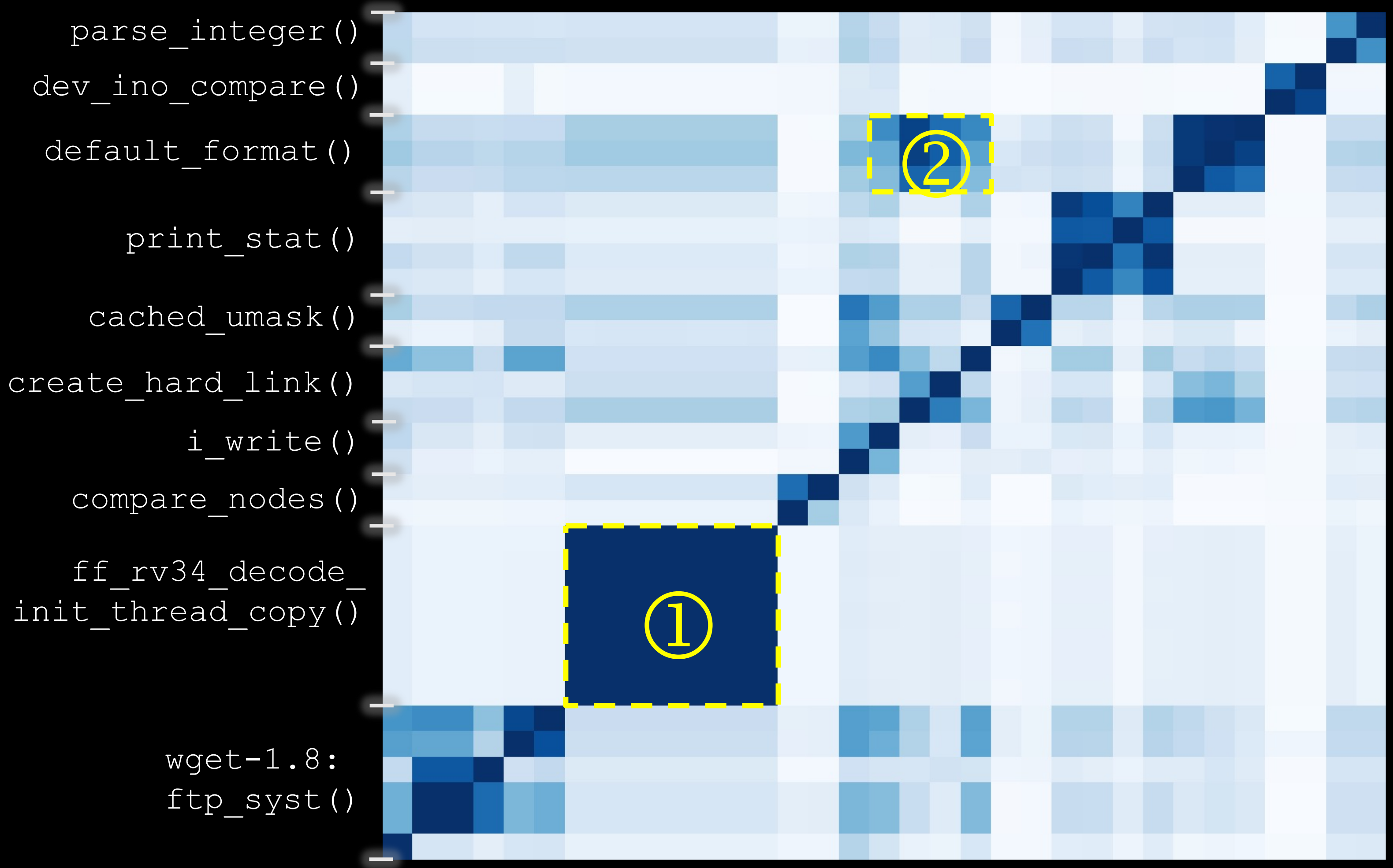
	icc	gcc	clang
clang			1.0
gcc		1.0	
icc	1.0		

Evaluation – All vs All

- Verified with randomly picked procedures
  - For example – when ff_rev34_decode@ffmpeg-2.4.6 is selected

	icc	gcc	clang
clang	1.0	1.0	1.0
gcc	1.0	1.0	1.0
icc	1.0	1.0	1.0

Results – All vs All



All v. All comparison

www.binsim.com

(code+demo)

Summary

- Clear motivation
  - Finding vulnerable code, detecting clones, etc.
- Challenging scenario
  - Finding similarity cross-{compiler, version} in stripped binaries
- Applied to real-world code

- Take home:
  - A semantic approach, yet feasible
  - Accuracy achieved with statistical framework