Lecture 38: Reverse Engineering

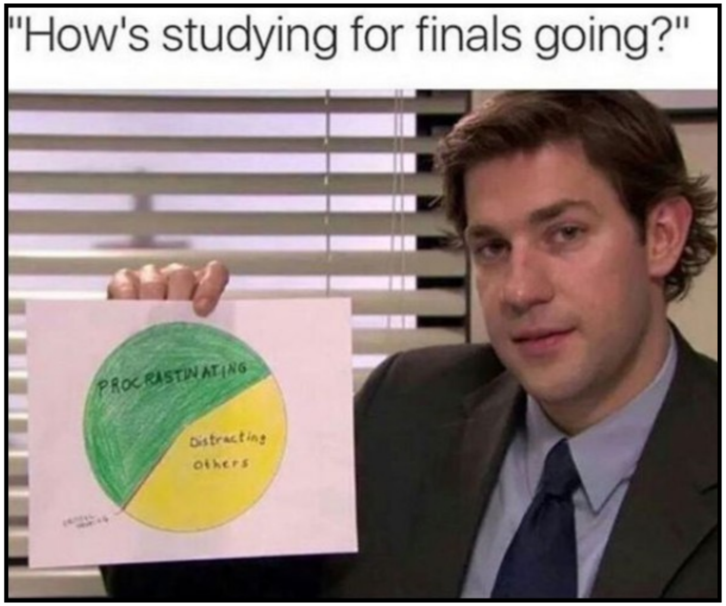
Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Goals for Today

- Learning Objectives:
  - Understand the objectives and methods of reverse engineering
  - Review several examples of important RE results in the literature
- Announcements, etc:
  - Forensics CP2 due **December 6th**
  - Final Exam — 7pm December 13th

Reminder: Please put away devices at the start of class

Final Details



- December 13th, 7-10pm
  - <u>Here</u>, 1404 Siebel
- Multiple choice + short answer
- **Closed book.**
- *<u>No electronic devices permitted (or necessary)!</u>*
- **Content**: <u>All</u> lectures, and MP3, MP4, MP5.
- Sample exams are not available for the final; feel free to re-review midterm sample exams

Final Details

- Changes from midterm:
  - Multiple Choice will now be scantron. Only one answer is correct per question. No points deducted for guessing.
  - More multiple choice questions, ~30+ as opposed to 20.
  - More short answer questions
  - MP questions will account for a smaller percentage of overall score.

Reverse Engineering

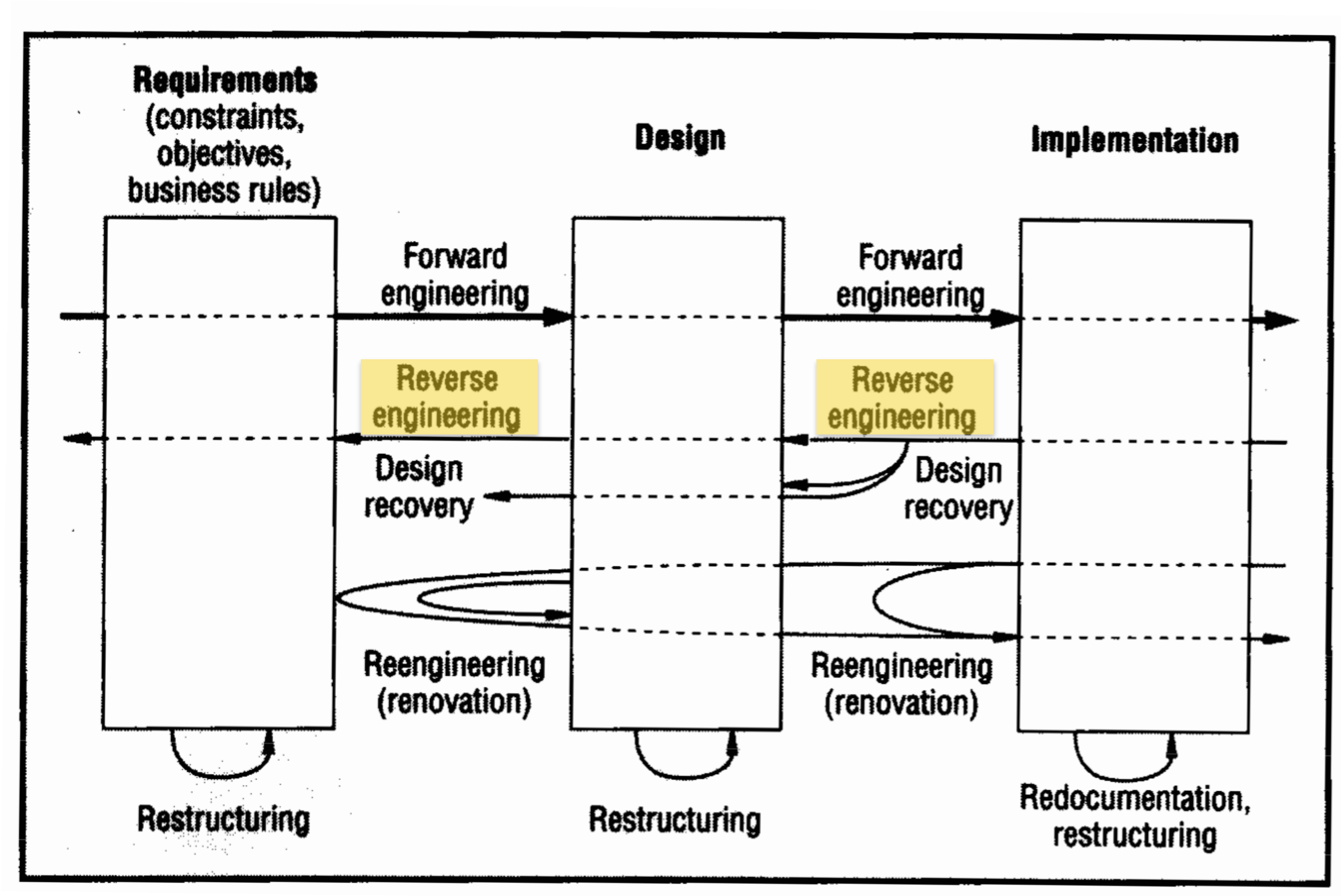
- *"The process of analyzing a subject system to identify the system's components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction."*
- Term originates in hardware, attempting to decipher design documents from finished products.
- Term can be applied to a variety of sociotechnical products today.

[Chikofsky and Cross, IEEE'90]

This semester…

- Arguably, we've actually already covered reverse-engineering a lot:
  - Binary Exploitation almost always requires RE
  - Software Testing lecture discussed RE
  - Malware analysis to determine malware payloads
  - During cloud lecture, we discussed the reverse engineering of an Amazon data center policy
  - …
- But if you want more, I'll give you more! :)

Reverse Engineering



[Chikofsky and Cross, IEEE'90]

Recap

- Reverse Engineering (RC), Reverse Code Engineering (RCE)
- reverse engineering -- process of discovering the technological principles of a [insert noun] through analysis of its structure, function, and operation.
- The development cycle … backwards

Why Reverse Engineer?

- Malware analysis
- Vulnerability or exploit research
- Check for copyright/patent violations
- Interoperability (e.g. understanding a file/protocol format)
- Copy protection removal

RE Techniques (Binary)

- Static Code Analysis (structure)
  - Disassemblers
- Dynamic Code Analysis (operation)
  - Tracing / Hooking
  - Debuggers

Disassembly

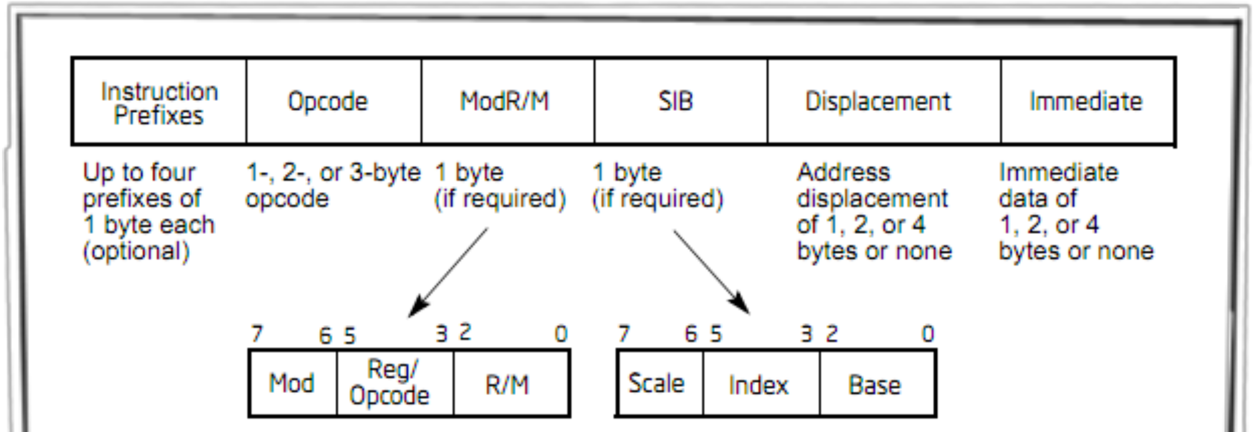
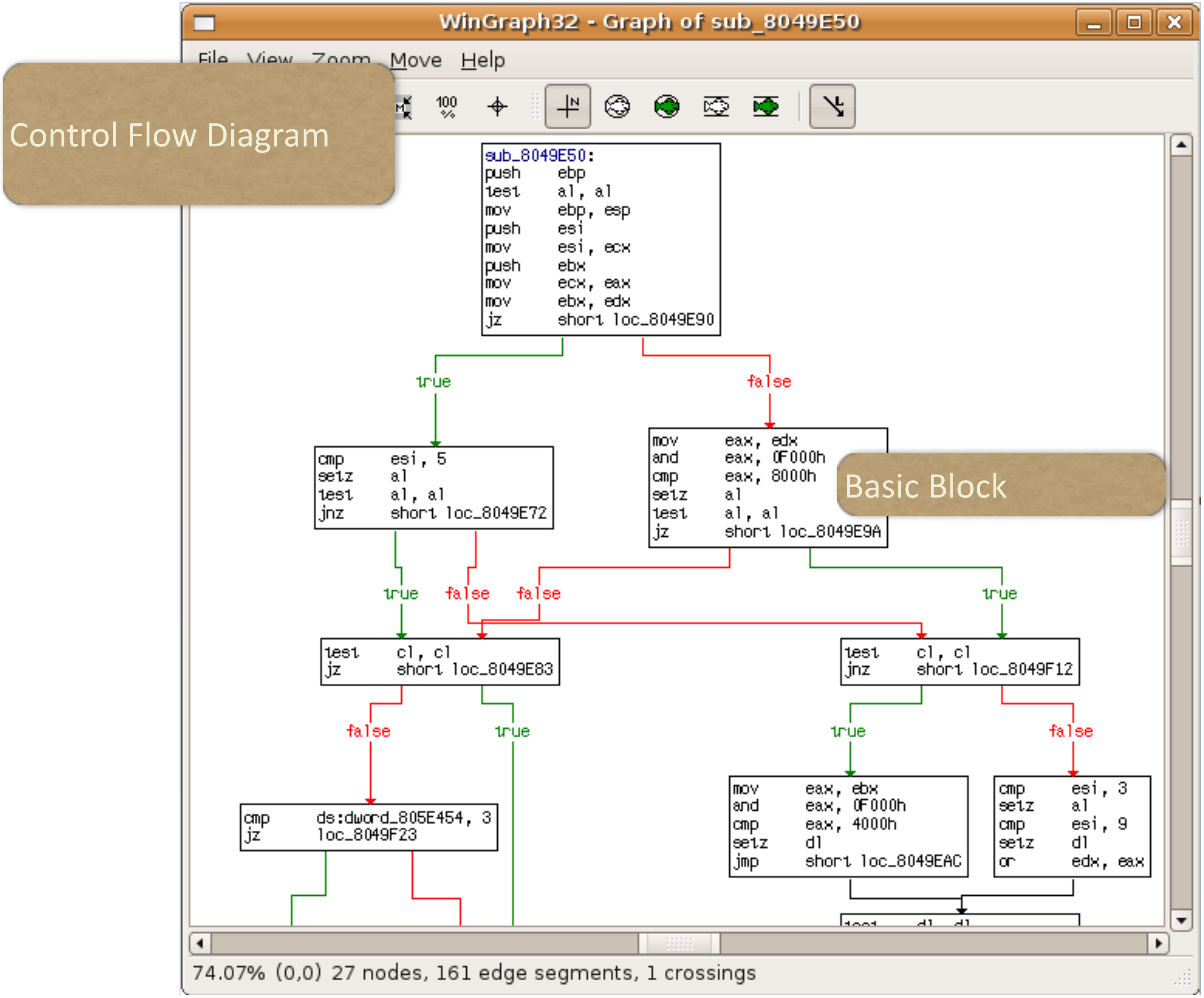


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Bits	Hex Bytes	Instructions (human-readable)
11101011 00000110	0xEB 0x06	JMP +6
01010000	0x50	PUSH EAX

Disassembly

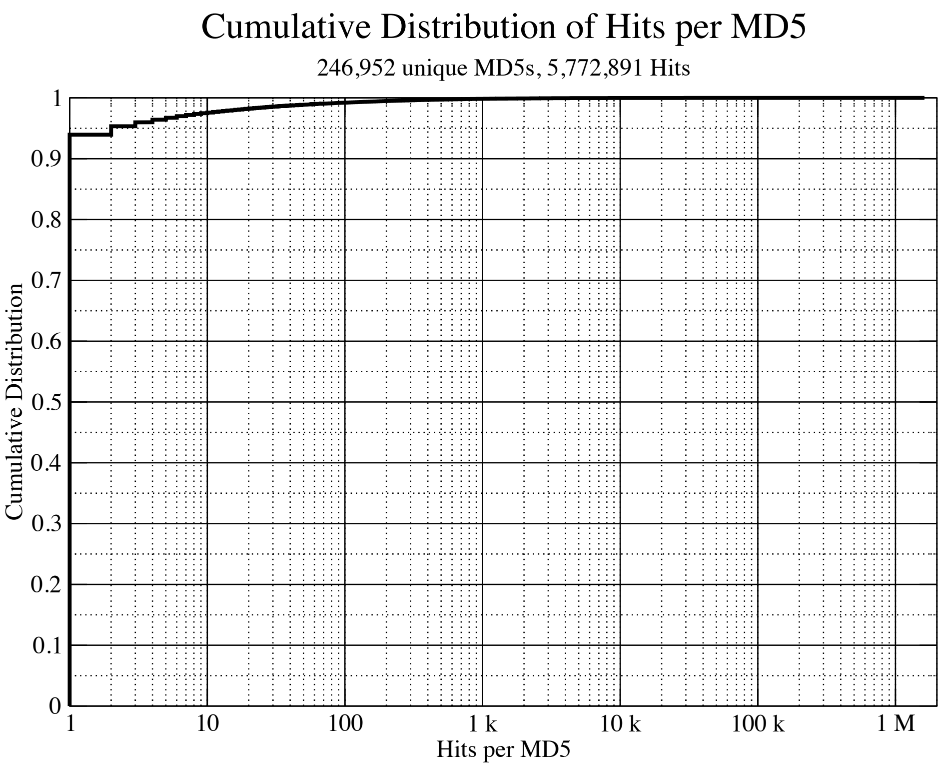


Challenges to Disassembly

- Static analysis isn't perfect at recovering original code
- "Benign" Optimizations
  - Constant folding
  - Dead code elimination
  - Inline expansion
  - etc...
- Intentional Obfuscation (note: not just for attackers!)
  - Packing
  - No-op instructions

E.G.: Malware Packing

- Packing is used by software developers specifically to avoid reverse engineering… just ask malware authors!



Packer identification
98,801 malware samples

PEiD	Count
UPX	11244
Upack	6079
PECompact	4672
Nullsoft	2295
Themida	1688
FSG	1633
tElock	1398
NsPack	1375
ASpack	1283
WinUpack	1234

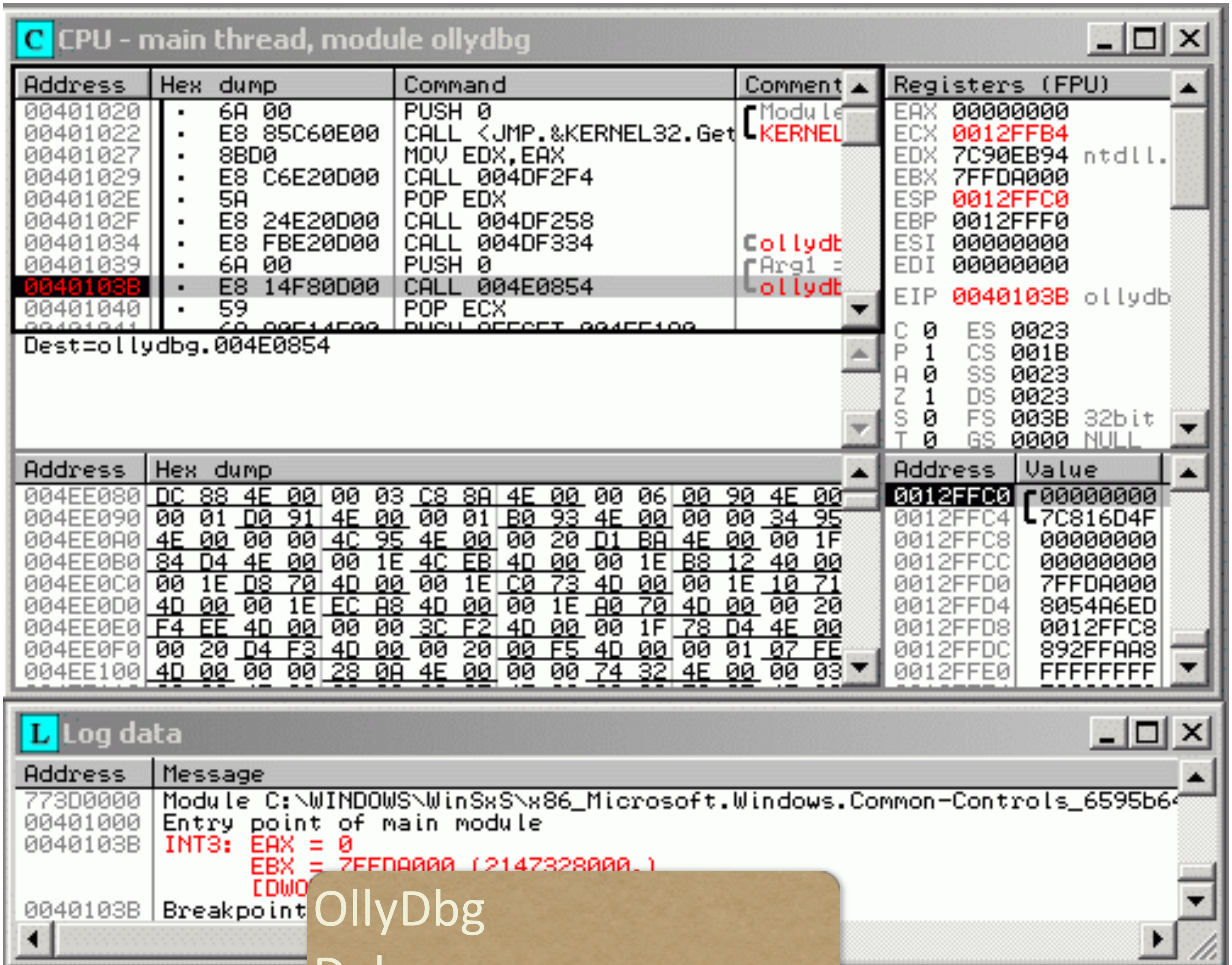
SigBuster	Count
Allaple	22050
UPX	11324
PECompact	5278
FSG	5080
Upack	3639
Themida	1679
NsPack	1645
ASpack	1505
tElock	1332
Nullsoft	1058

Identified: 59,070 (60%)
Top 10: 33.3%

Debugging

- Trace every instruction a program executes -- single step
- Or, let program execute normally until an exception
- At every step or exception, can observe / modify:
- Instructions, stack, heap, and register set
- May inject exceptions at arbitrary code locations
- INT 3 instruction generates a breakpoint exception

Debugging



OllyDbg Debugger

Debugging Benefits

- Sometimes easier to just see what code does
- A workaround to packing routines:
  - just let the code unpack itself and debug as normal
  - … unless the code tests for a debugging/VM environment!
- Most debuggers have in-built disassemblers anyway
- Can always combine static and dynamic analysis

Debugging Challenges

- We are now executing potentially malicous code
    - use an isolated virtual machine
- Anti-Debugging
  - detect debugger and [exit | crash | modify behavior]
  - IsDebuggerPresent(), INT3 scanning, timing, VM-detection, pop ss trick, etc., etc., etc.
  - Anti-Anti-Debugging can be tedious

angr

- angr: a Python framework for binary analysis
- Based on Valgrind VEX intermediate representation (IR)
  - In Valgrind, used to transform machine code to IR, instrument the IR, then compile back to machine code.
- Uses Symbolic Execution to simulate the effects of statements, expressions, operations, etc. in the IR code.
- Symbolic Execution allows us to learn constraints of the program without actually executing!

Symbolic Execution

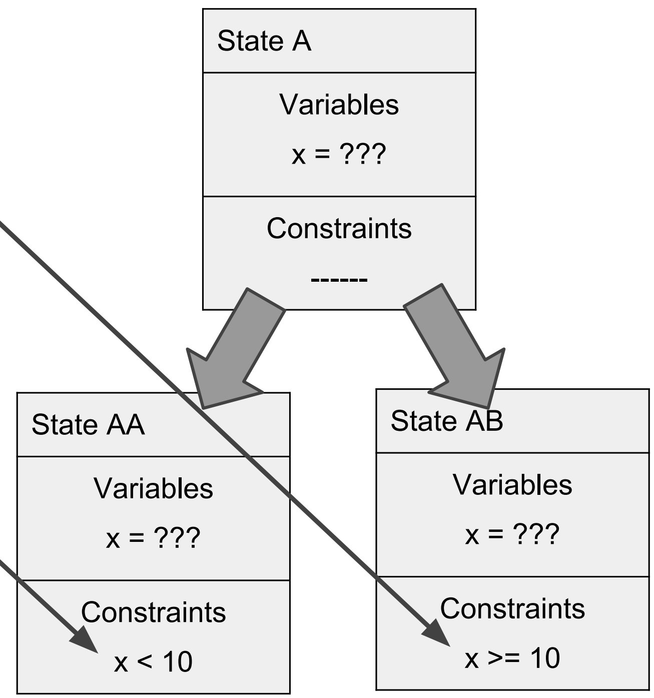
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

State A
Variables x = ???
Constraints ------

[Shellphish Lab, UCSB]

Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



[Shellphish Lab, UCSB]

Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

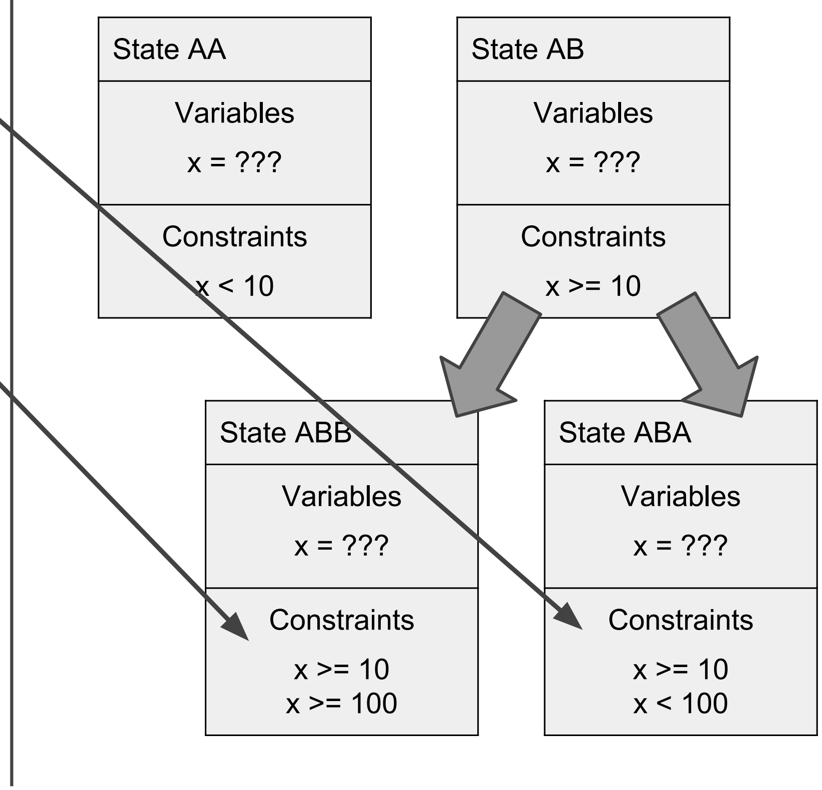
State AA
Variables x = ???
Constraints x < 10

State AB
Variables x = ???
Constraints x >= 10

[Shellphish Lab, UCSB]

Symbolic Execution

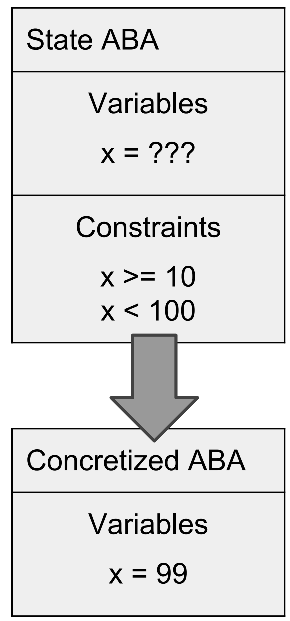
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



[Shellphish Lab, UCSB]

Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



[Shellphish Lab, UCSB]

Symbolic Execution

- Advantages
  - Semantically Aware
  - Targetable
- Disadvantages
  - path explosion
  - constraint solving

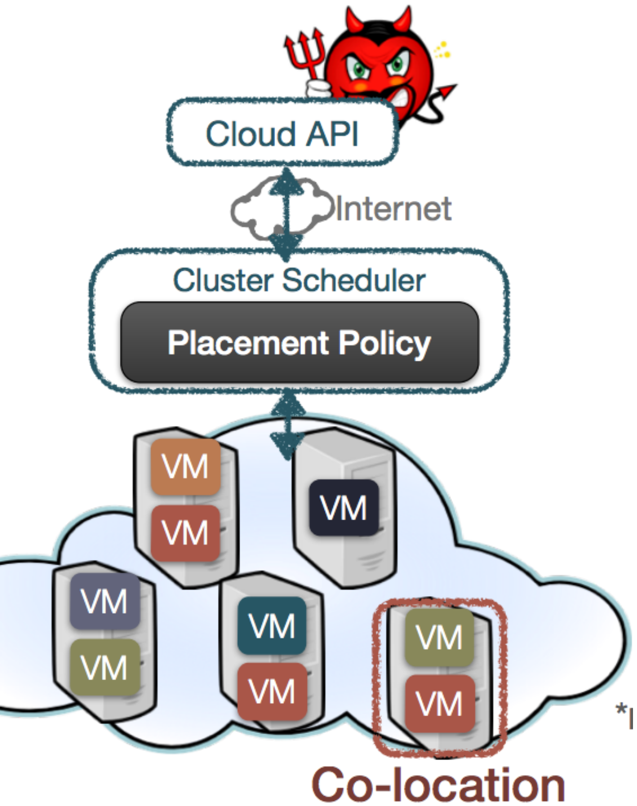
angr analyses

Analysis	Description
CFGFast/CFGAccurate	Control flow graph recovery
Disassembly	Linear disassembly rendering routine
DDG	Data dependence analysis
VFG	Value-flow graph recovery, performs
BackwardSlicing	Backward program slicing based on
BoyScout*	Determines architecture of binary blobs
GirlScout*	Determines base addresses of binary

Other Reverse Engineering

- What else can we reverse engineer?
  - A policy?
  - A machine learning model??
  - Other stuff???

Ex: Cloud Co-Residency Attacks



If a truly random placement policy was used…

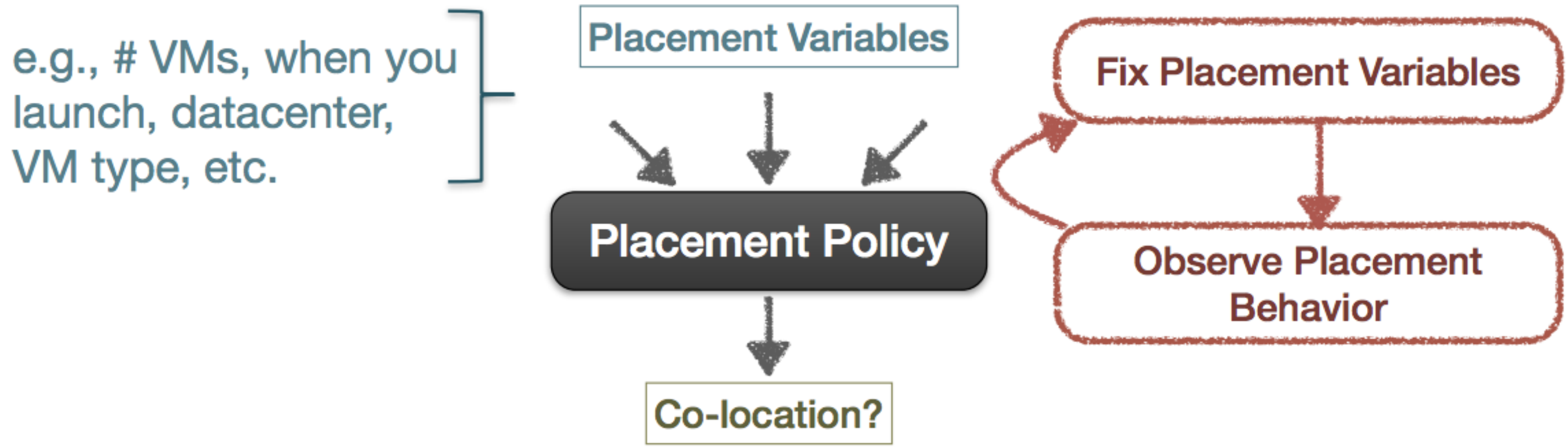
- N = 50,000 machines
- v victim VMs and a attacker VMs
- Probability of Collision:

$$P_c = 1 - (1 - \frac{v}{N})^a$$

v	a = ln(1– P_c)/ln(1 –v/N); P_c= 0.5
10	3466
20	1733
30	1155

[Varadarajan et al., Security'15]

Reverse Engineering Amazon's VM Placement Policy

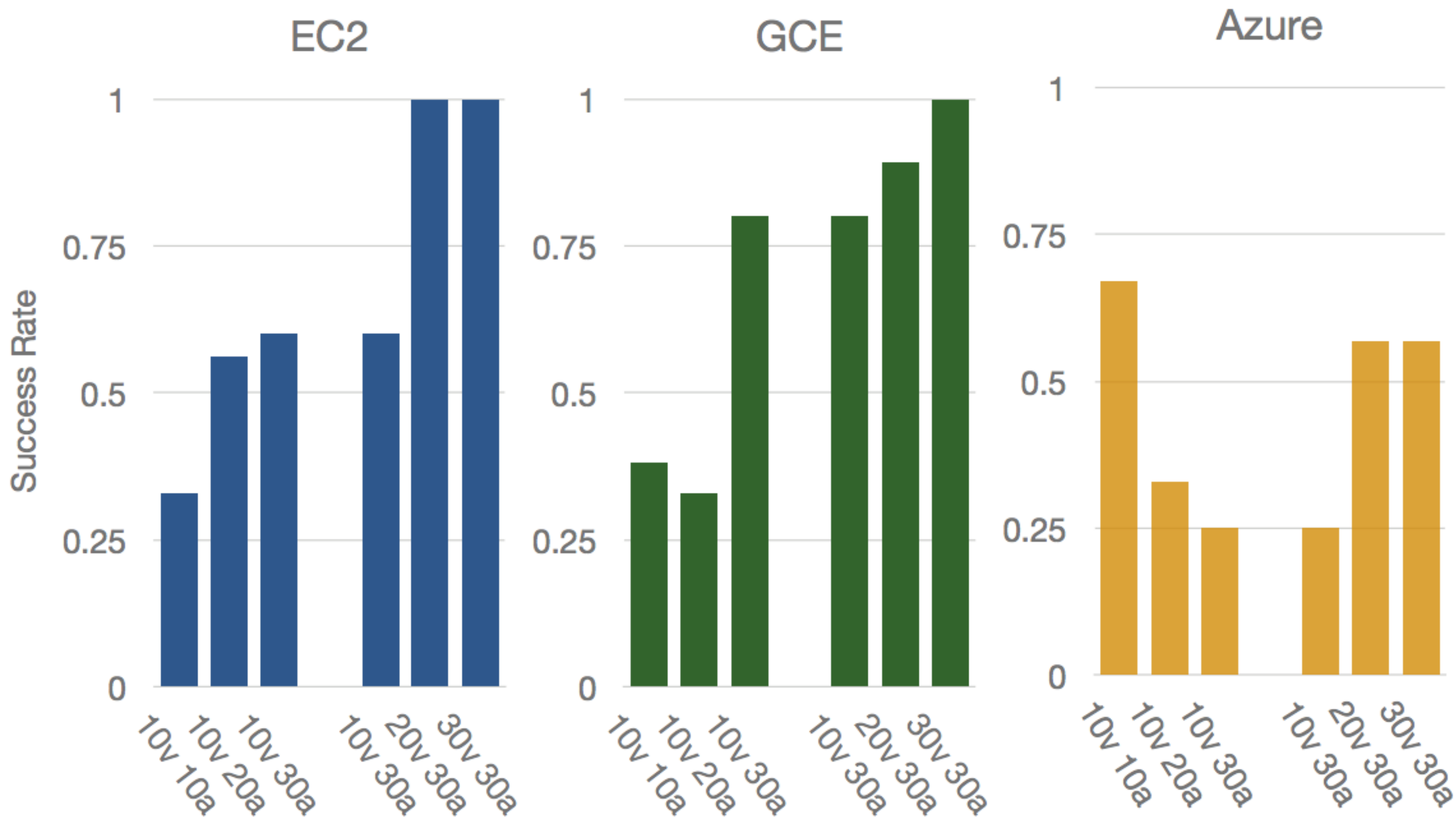


- 6 placement variables: # victim & attacker VMs, delay b/w launches, time of day, day of week, datacenter, cloud provider Small instance type
- 9 samples per strategy with 3 runs per time of day and 2 days of week (weekday/weekend).

Google Compute Engine amazon web services EC2 Microsoft Azure

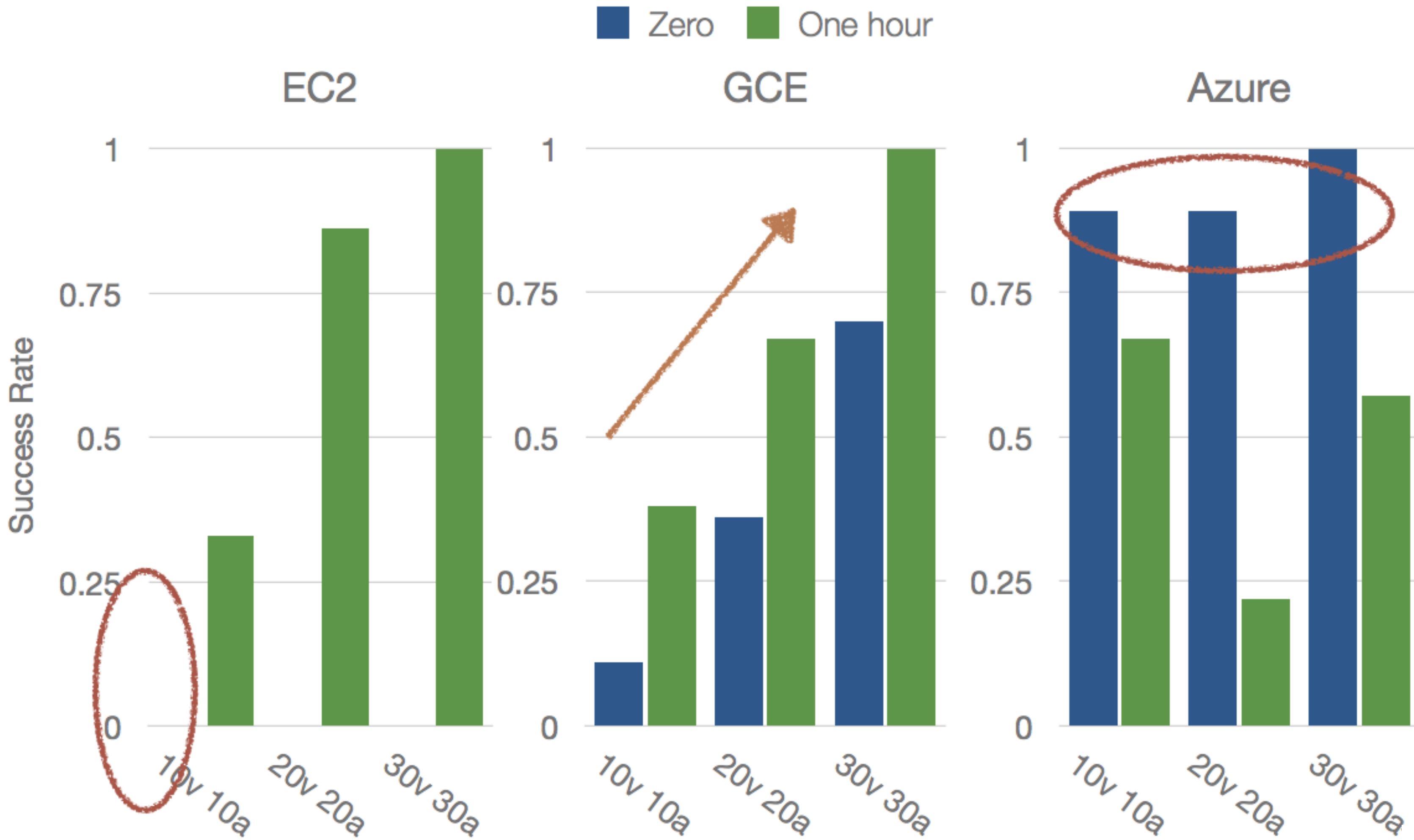
[Varadarajan et al., Security'15]

Reverse Engineering Amazon's VM Placement Policy



Co-location is possible with as low as 10 VMs and always achieves co-location with 30 VMs

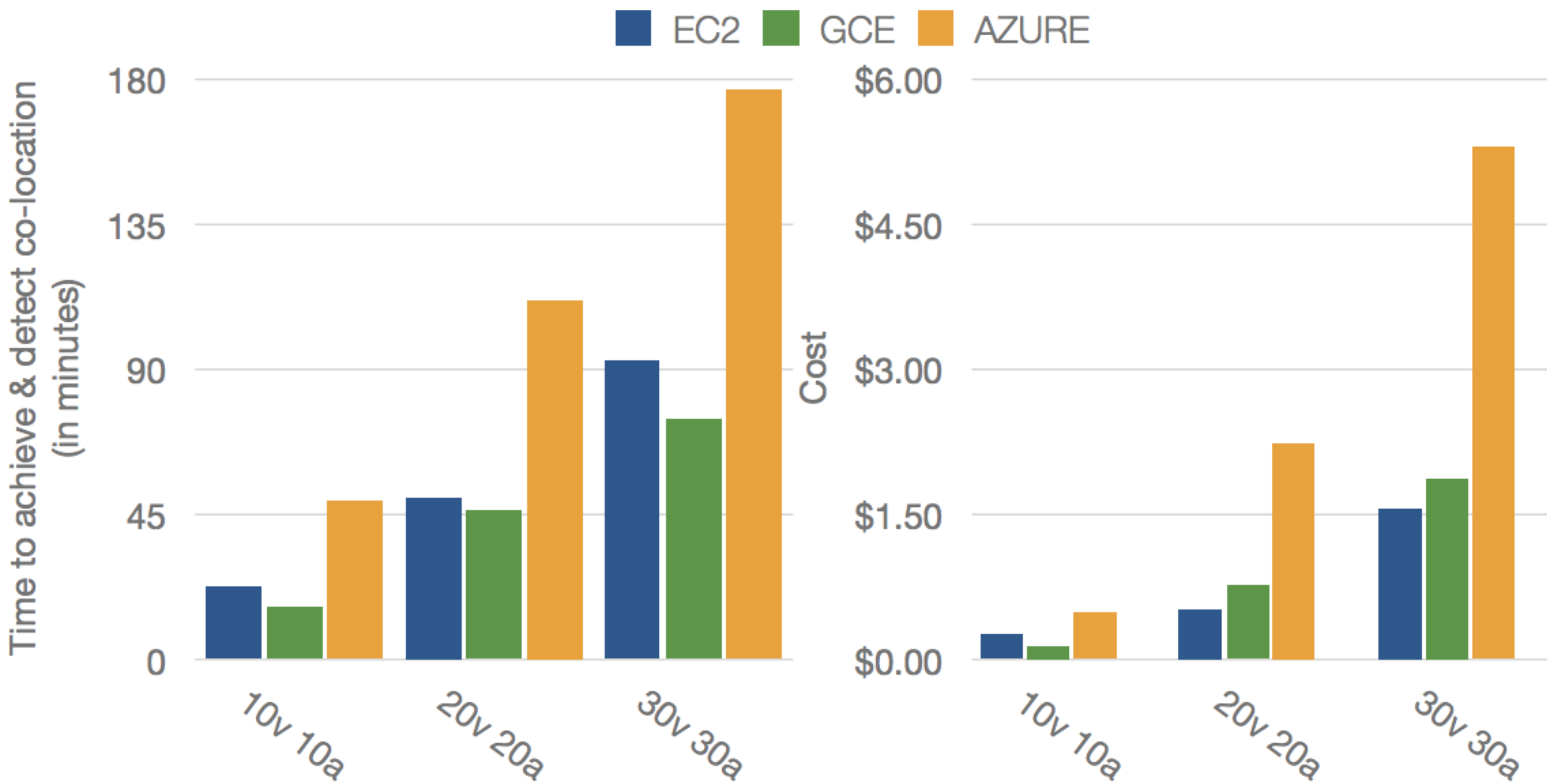
[Varadarajan et al., Security'15]



Different clouds have wildly different temporal placement strategies

[Varadarajan et al., Security'15]

Reverse Engineering Amazon's VM Placement Policy

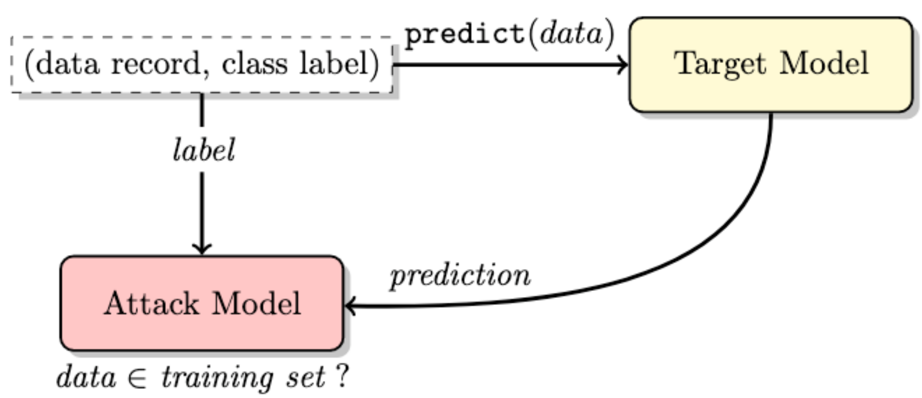


Successful co-location as affordable as 14 cents.

[Varadarajan et al., Security'15]

Ex: Membership Inference Attacks

- Attacker queries the target model with a data record and obtains the model's prediction on that record.
- The prediction is a vector of probabilities, one per class, that the record belongs to a certain class.
- This prediction vector, along with the label of the target record, is passed to the attack model, which infers whether the record was in or out of the target model's training dataset.



[Shokri et al., Oakland'17]

Ex: Model Inversion Attacks

- Also exploits confidence values revealed alongside predictions.
- Perturb features of some input image (e.g., transform pixel values in facial recognition), record change in confidence values.
- Use gradient descent to identify local minima (in this case, highest obtained confidence values).
- Can be used to launch a reconstruction attack on the original training data

Training Data

Inverted Model

[Fredrikson et al., CCS'15]

ICES Evaluation

Please write "**A. Bates CS 461 AL4**" at the top of your ICES form.

You can doodle it in just under "INSTRUCTOR AND COURSE EVALUATION SYSTEM."