Examining Self-Modifying Code

Drew Ivarson, Union College CS Department
Advisors: Prof. Anderson, Prof. Spinelli

Overview

- Background
- Motivation
- My contributions

Examining Self-Modifying Code

- What code am I talking about?
- How do I examine it?
- How is it self-modifying?

Examining Self-Modifying Code

- The Code found in executable files

Binary	=	Assembly instructions
01010001020101	=	INC 0x1
0200		MOVB 0x1 0x2
		INC 0x1
		JMP 0X0

Examining Self-Modifying Code

- Dynamic Analysis
  - run the program
  - evaluate the results of each instruction being executed
- Static Analysis
  - not running the program!
  - quickly cover all possible traversals

Examining Self-Modifying Code

non-self-modifying:

0x0: movb 0x7 reg1
0x3: inc reg1
0x5: jmp 0x0
0x7: inc reg1

Self-Modifying? cont.

non-self-modifying:

0x0: movb 0x7 reg1

0x3: inc reg1

0x5: jmp 0x0

0x7: inc reg1

self-modifying:

0x0: movb 0x7 0x6

0x3: inc reg1

0x5: jmp 0x0 → jmp 0x7

0x7: inc reg1

Intro Summary

Examining Self-Modifying Code:

1. Binary Files - binary (assembly) code
2. Static Analysis - not running it
3. Self-Modifying: writing to instruction memory instead of data memory

1260, a self-modifying virus

http://www.informit.com/articles/article.aspx?p=366890&seqNum=5

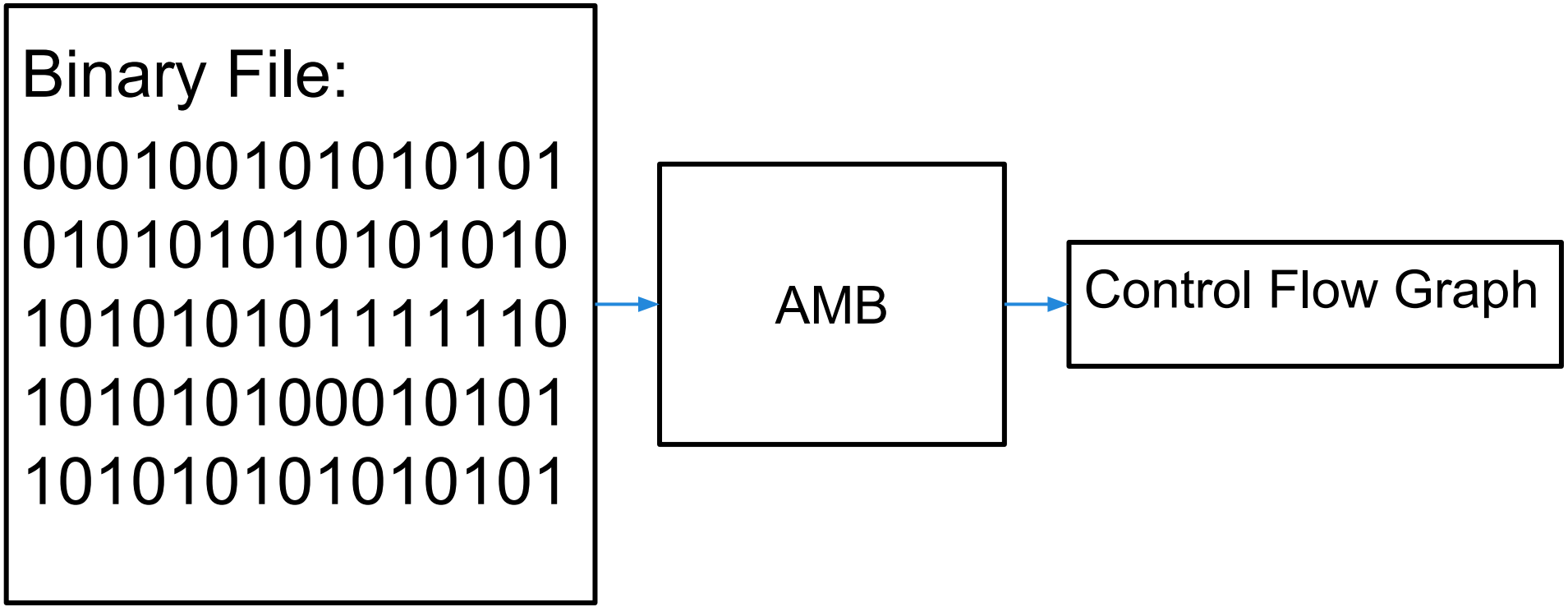
- Before Running:
  - Do register math
  - Read from memory
- While Running:
  - Do register math
  - Read from memory
  - SEND PERSONAL INFORMATION TO SOME IP
- After Running:
  - Same as before...

A Model for Self-Modifying Code

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.8328&rep=rep1&type=pdf

- Answer to static analysis problem
- AMB algorithm and data structure
- A *model*, so it hasn't been implemented!

Input and Output to AMB

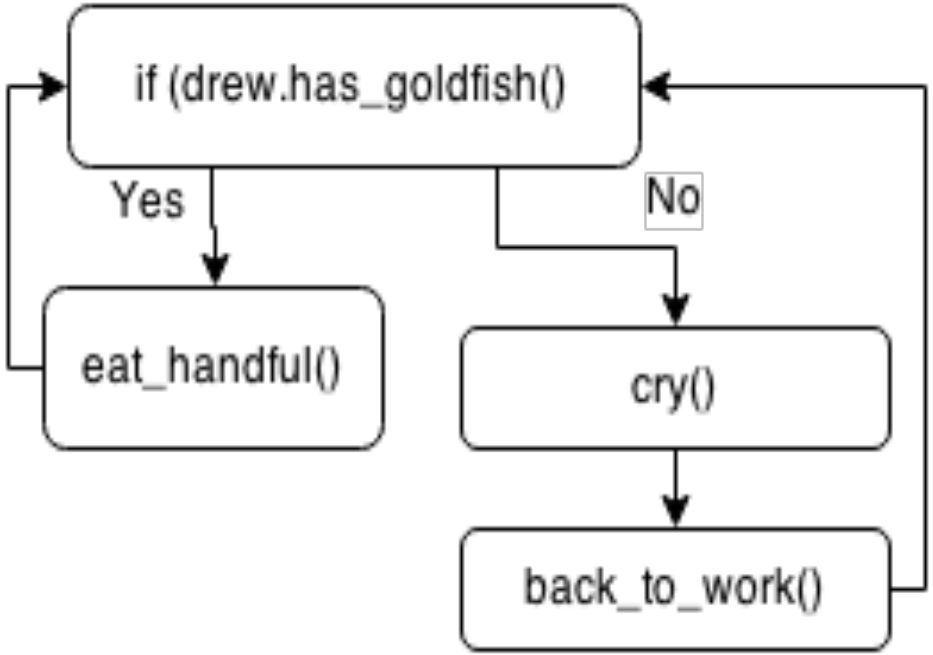


Control Flow Graphs (CFGs)

- Graph to show program control flow, ie, function calls, conditional statements
- A picture of a traversal through a program

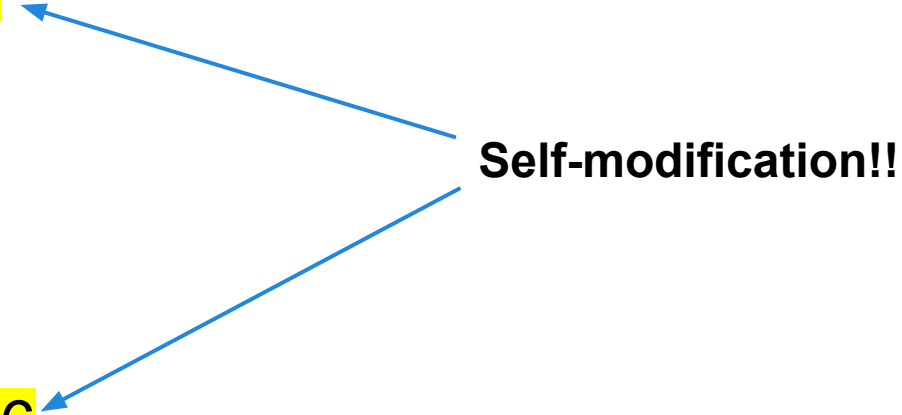
CFG Example

```
while (true)
    if (Drew.has_goldfish())
        eat_handful();
    else
        cry();
        back_to_work();
```



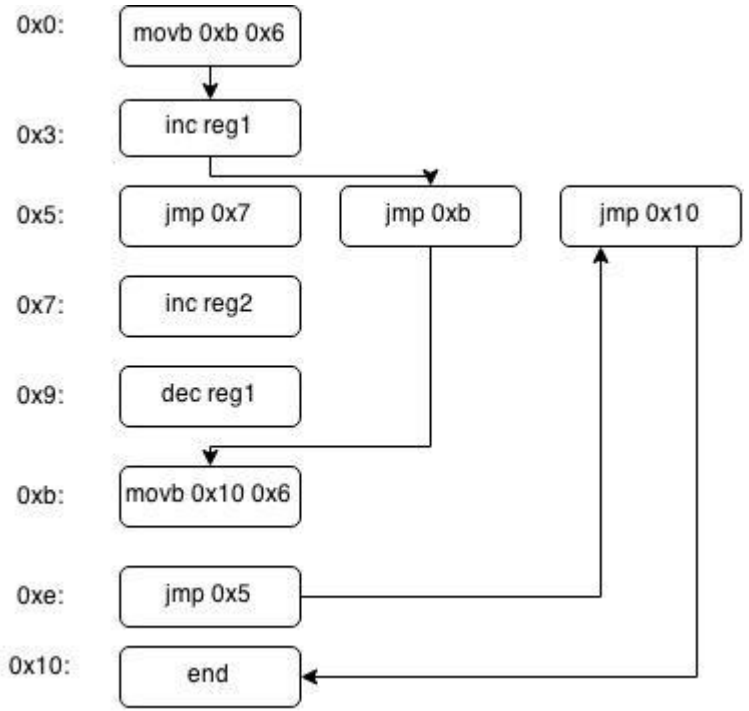
A more *assembled* example

0x0: movb 0xb 0x6
0x3: inc reg1
0x5: jmp 0x7
0x7: inc reg2
0x9: dec reg1
0xb: movb 0x10 0x6
0xe: jmp 0x5
0x10: end



A CFG of our new example

0x0: movb 0xb 0x6
0x3: inc reg1
0x5: jmp 0x7
0x7: inc reg2
0x9: dec reg1
0xb: movb 0x10 0x6
0xe: jmp 0x5
0x10: end



AMB Algorithm

- Conservative Estimate

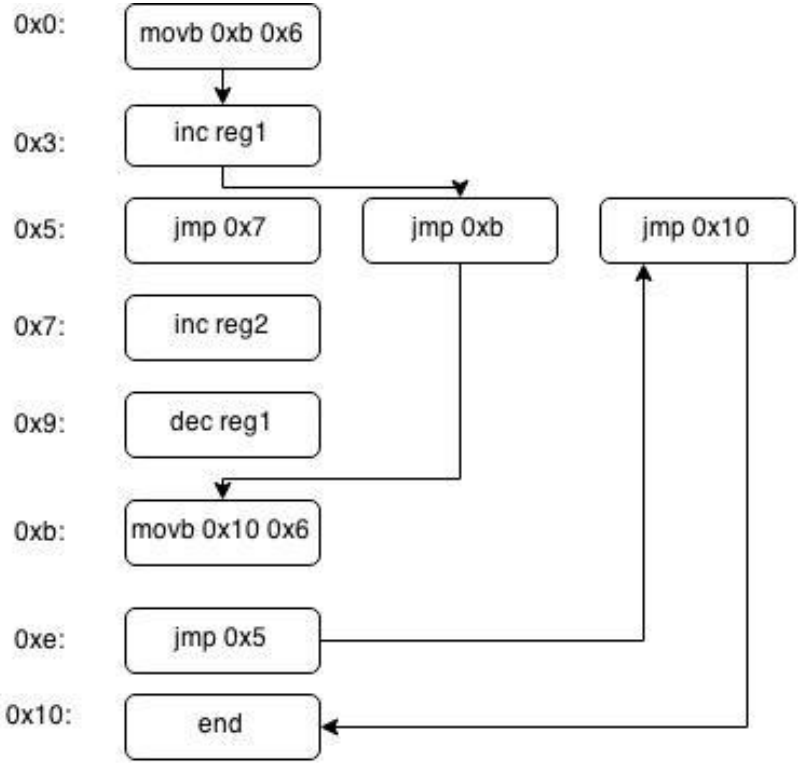
while (state of instruction memory is changing)
    recurse over the program given the current state of memory
        store results of instructions that write to memory, and the results of instructions that change the control flow

Summary of The Model

```
0x0: movb 0xb 0x6
0x3: inc reg1
0x5: jmp 0x7
0x7: inc reg2
0x9: dec reg1
0xb: movb 0x10 0x6
0xe: jmp 0x5
0x10: end
```

AMB Algorithm

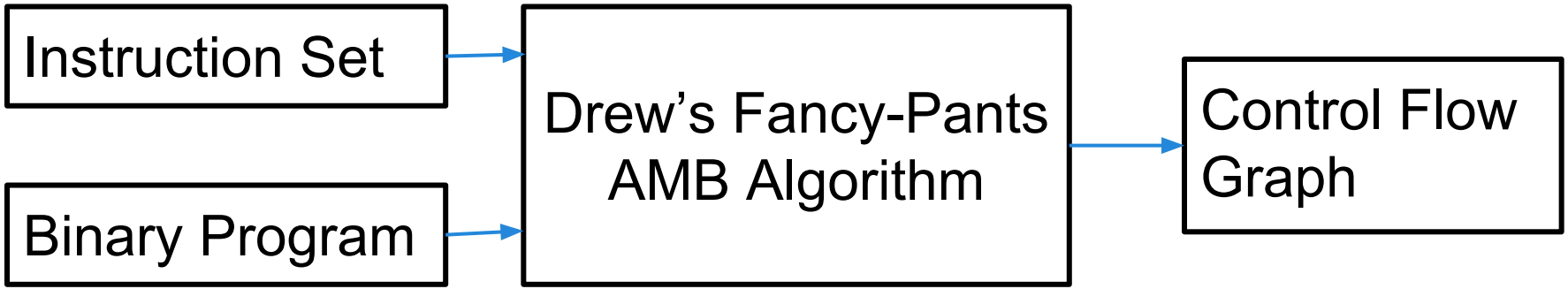
- CodeBytes
- Instructions



My Contribution

- Implement this algorithm
- Bring it from a model to reality
  - User-defined instruction sets
  - User-written test programs
  - Graphical output

Input and Output of My Research



User-Defined Instruction Sets

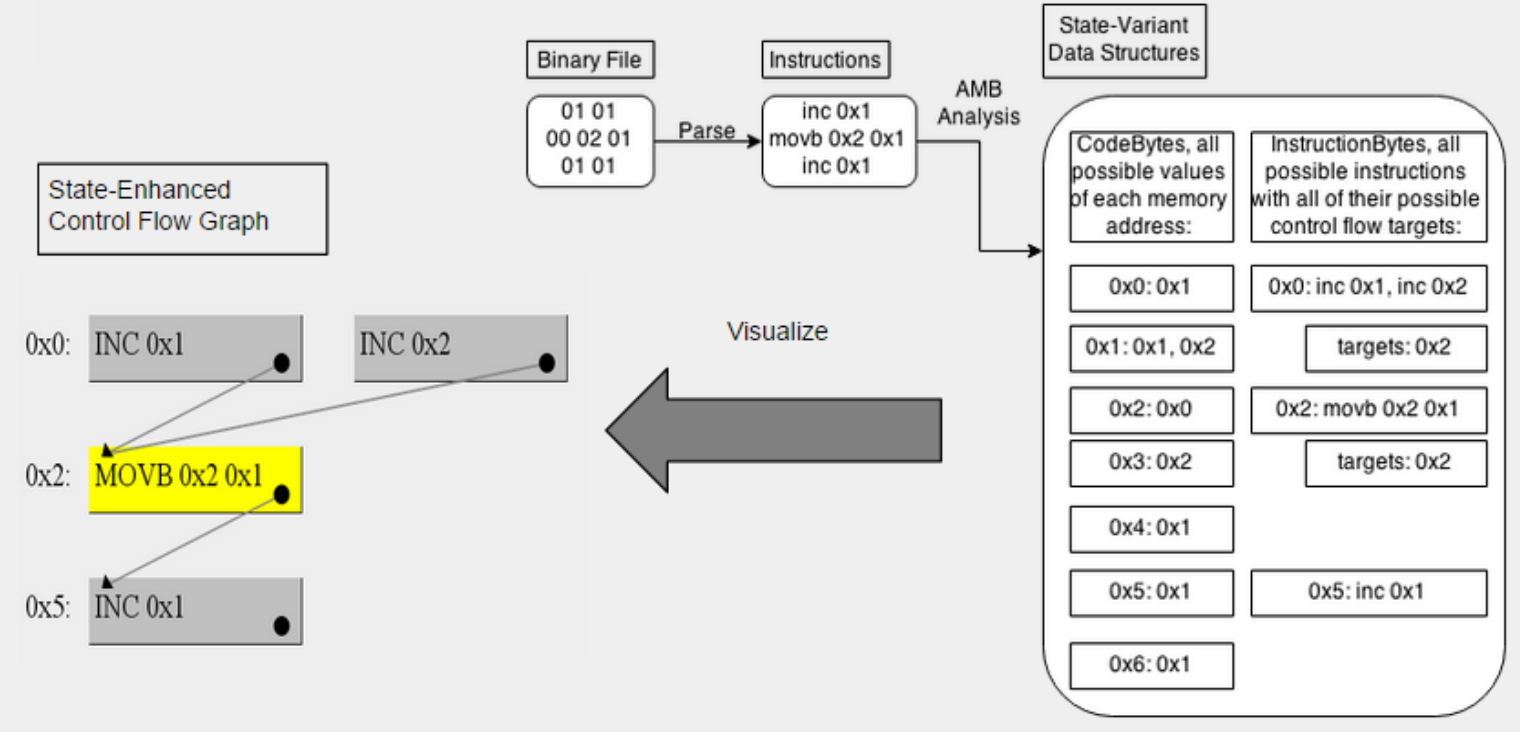
- Abstract Syntax
  - Writes
  - Gotos
  - Skips

Opcode, length,
name, abstract syntax

Example:

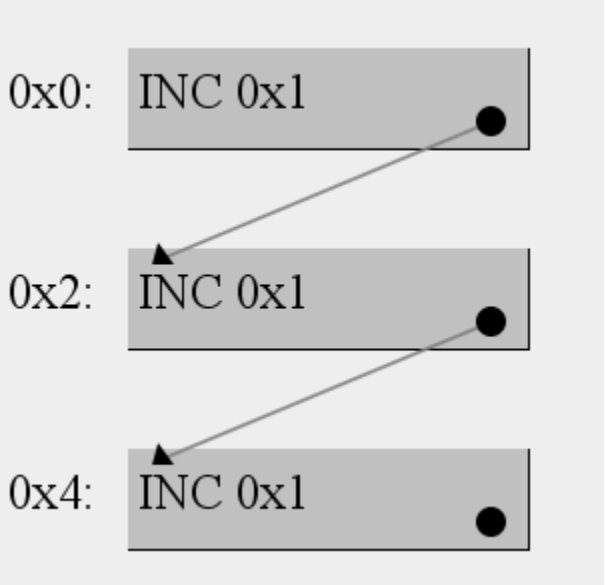
00	3	MOVB	WRITE
01	2	INC	SKIP
02	2	JMP	GOTO

Implementation

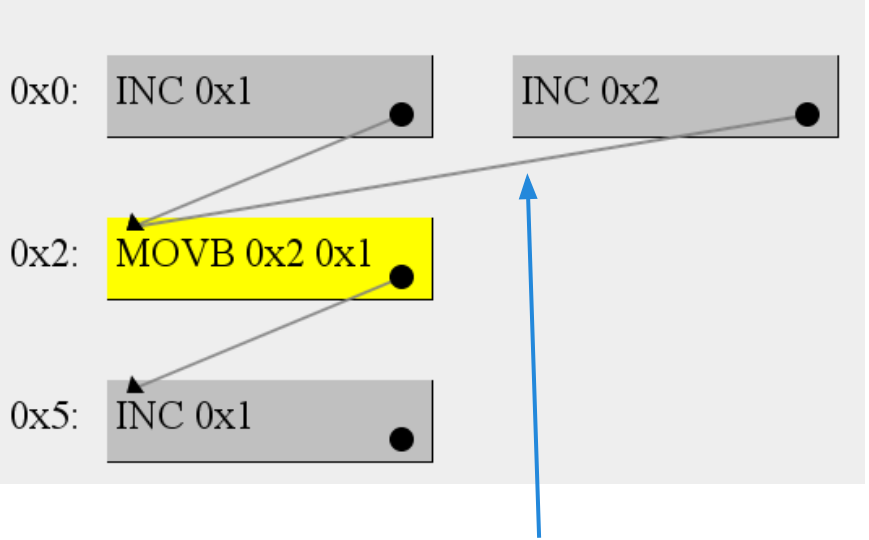


Results

Simple, no modification program:

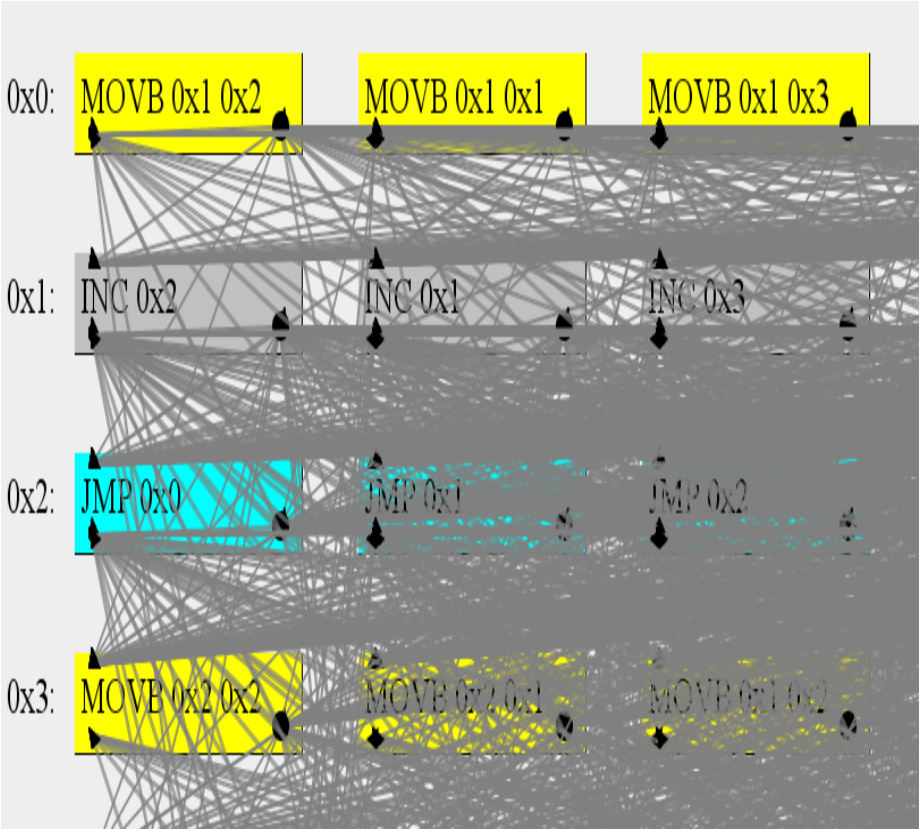


Simple, self-modifying program:



Already an impossible edge!

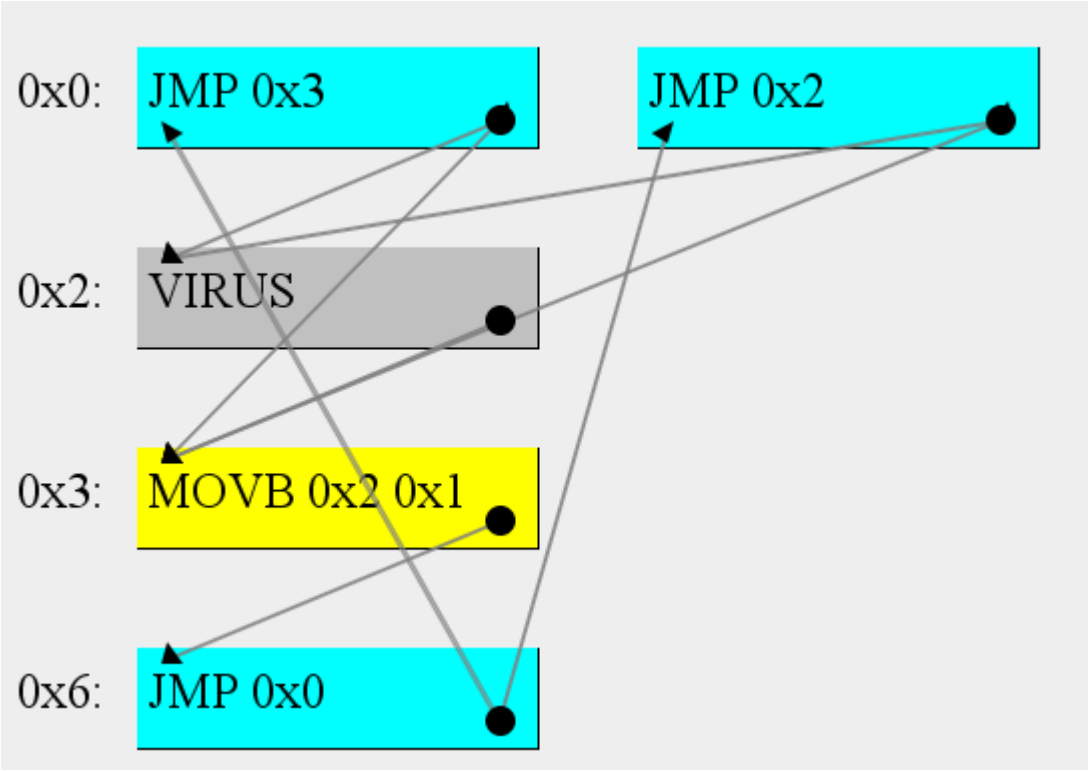
Results (cont).



This is a 10 line program with no jumps!

Algorithm computes over a million edges!

One More Result



Before running the algorithm, VIRUS looks like an unreachable line.

Conclusion

- Some optimization required
  - Too many edges and nodes
  - Remove unreachable code
- Detected VIRUS
- Generated graphs based on user-defined instruction sets and user-written programs
- Did not conquer polymorphic code engines

Future Work

- Expand to full instruction sets (like an actual assembly language)
- Top priority: algorithm optimization

Questions