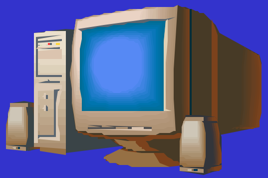
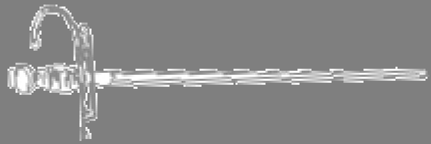
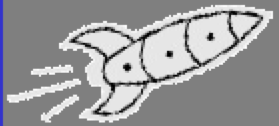
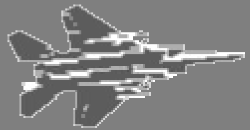
Static Analysis of Binaries for Malicious Code Detection

Mihai Christodorescu, Somesh Jha
{mihai,jha}@cs.wisc.edu
University of Wisconsin, Madison

Arms Race



 Vanilla virus	Signatures
 Register renaming	Regex signatures
 Packing/encryption	Emulation/heuristics
 Code reordering	?
 Code integration	?

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Dismal State of the Art

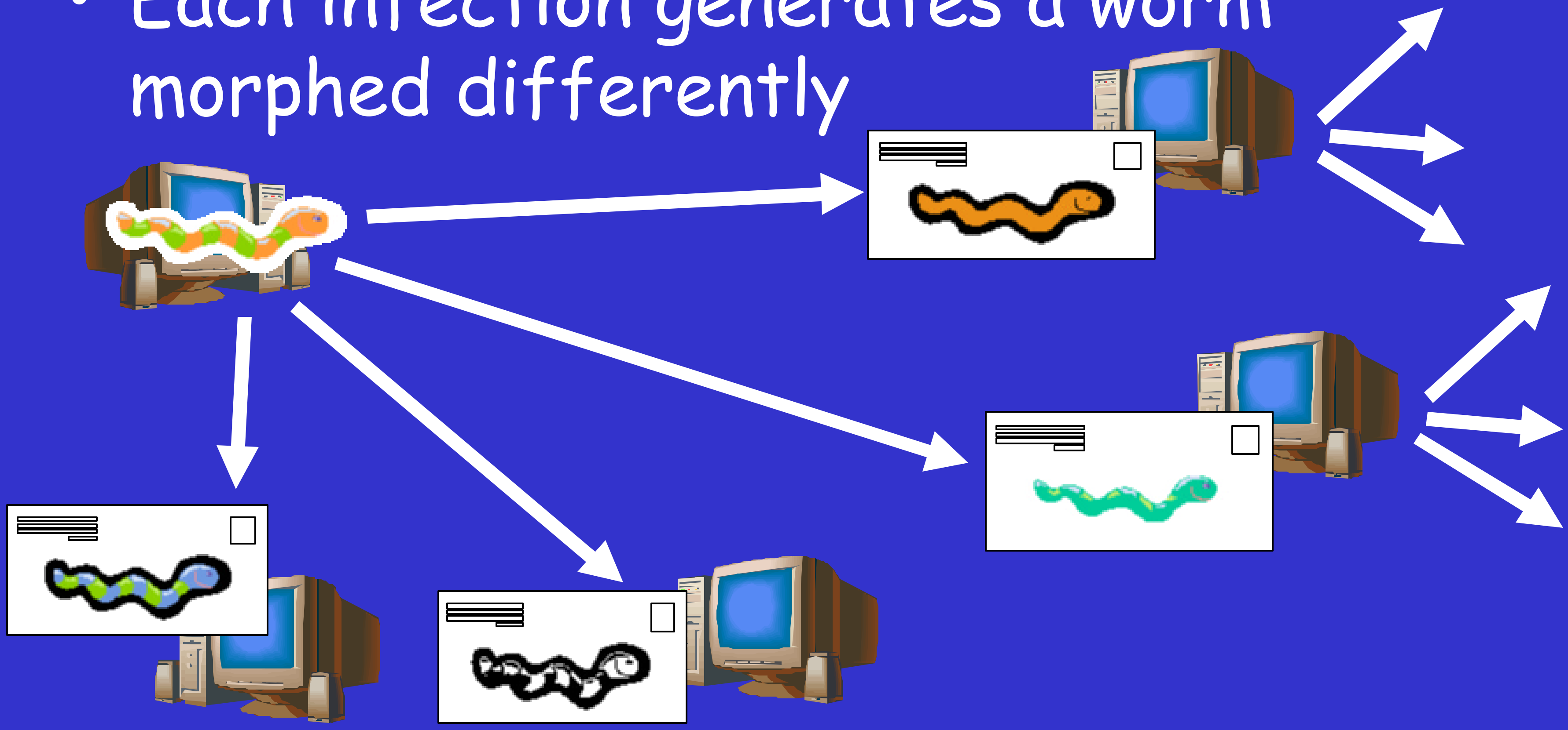
Commercial antivirus tools vs. morphed versions of known viruses

	Norton AntiVirus	McAfee VirusScan	Command Software Systems
Chernobyl-1.4	× Not detected	× Not detected	× Not detected
f0sf0r0	× Not detected	× Not detected	× Not detected
Hare	× Not detected	× Not detected	× Not detected
z0mbie-6.b	× Not detected	× Not detected	× Not detected

Obfuscations used in morphing: NOP insertion, code reordering

Worst-Case Scenario

- Each infection generates a worm morphed differently



Clear Danger

- Unlimited variants can be cheaply generated
  - Practically undetectable
- Obfuscations: part of the virus propagation step

◆ Threat of highly mobile, highly morphing malicious code

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
        pop     ecx
        jecxz   SFModMark
        mov     esi, ecx
        mov     eax, 0d601h
        pop     edx
        pop     ecx
        call    edi
        jmp     Loop
```

Morphed Virus Code:

```
Loop:
        pop     ecx
        nop
        jecxz   SFModMark
        xor     ebx, ebx
        beqz    N1
N1:     mov     esi, ecx
        nop
        mov     eax, 0d601h
        pop     edx
        pop     ecx
        nop
        call    edi
        xor     ebx, ebx
        beqz    N2
N2:     jmp     Loop
```


Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
          pop       ecx
          jecxz     SFModMark
          mov       esi, ecx
          mov       eax, 0d601h
          pop       edx
          pop       ecx
          call      edi
          jmp       Loop
```

Morphed Virus Code:

```
Loop:
          pop       ecx
          nop

          call      edi
          xor       ebx, ebx
          beqz      N2
N2:       jmp       Loop

          nop
          mov       eax, 0d601h
          pop       edx
          pop       ecx
          nop

          jecxz     SFModMark
          xor       ebx, ebx
          beqz      N1
N1:       mov       esi, ecx
```


Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

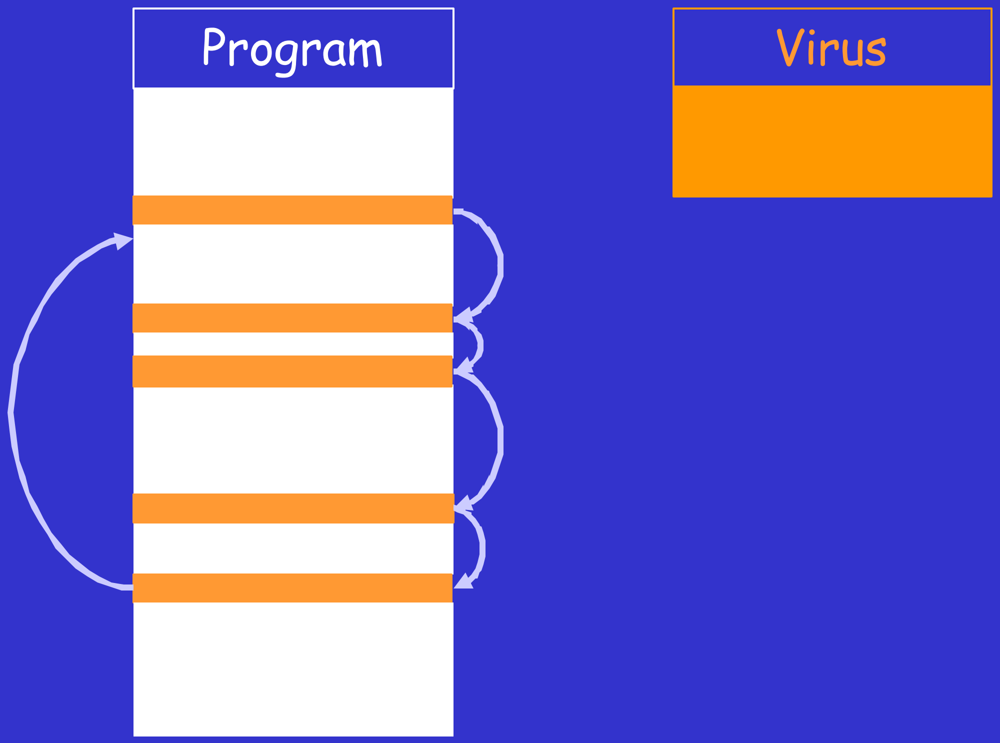
```
Loop:
         pop      ecx
         jecxz    SFModMark
         mov      esi, ecx
         mov      eax, 0d601h
         pop      edx
         pop      ecx
         call     edi
         jmp      Loop
```

Morphed Virus Code:

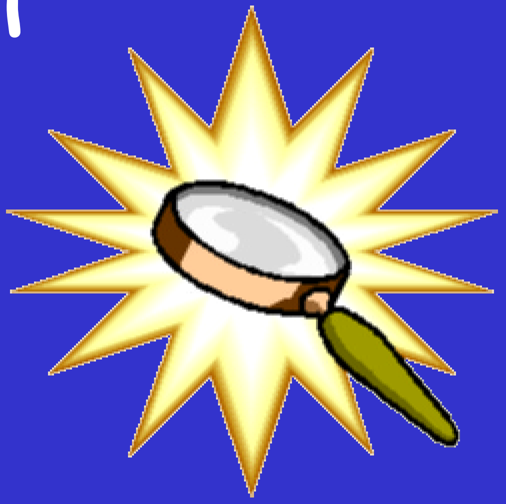
```
Loop:
         pop      ecx
         nop
         jmp L1
L3:      call     edi
         xor      ebx, ebx
         beqz     N2
N2:      jmp      Loop
         jmp L4
L2:      nop
         mov      eax, 0d601h
         pop      edx
         pop      ecx
         nop
         jmp L3
L1:      jecxz    SFModMark
         xor      ebx, ebx
         beqz     N1
N1:      mov      esi, ecx
         jmp L2
L4:
```

Code Integration

- Integration of virus and program



Our Solution



Better virus scanner:

- Analyze the program semantic structure
  - Control flow
  - Data flow
- Build on existing static analyses

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Overview

- Threats
- Current detection limitations
- Detector design and architecture
- Sample detection
- Performance
- Future work and conclusions

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Design Goals

- Static analysis
  - Provides safe results: identifies possible malicious sequences
  - Immune to anti-emulation techniques
- Identify malicious intent
  - Same behavior can be achieved through many implementations

Static Analysis of Binaries

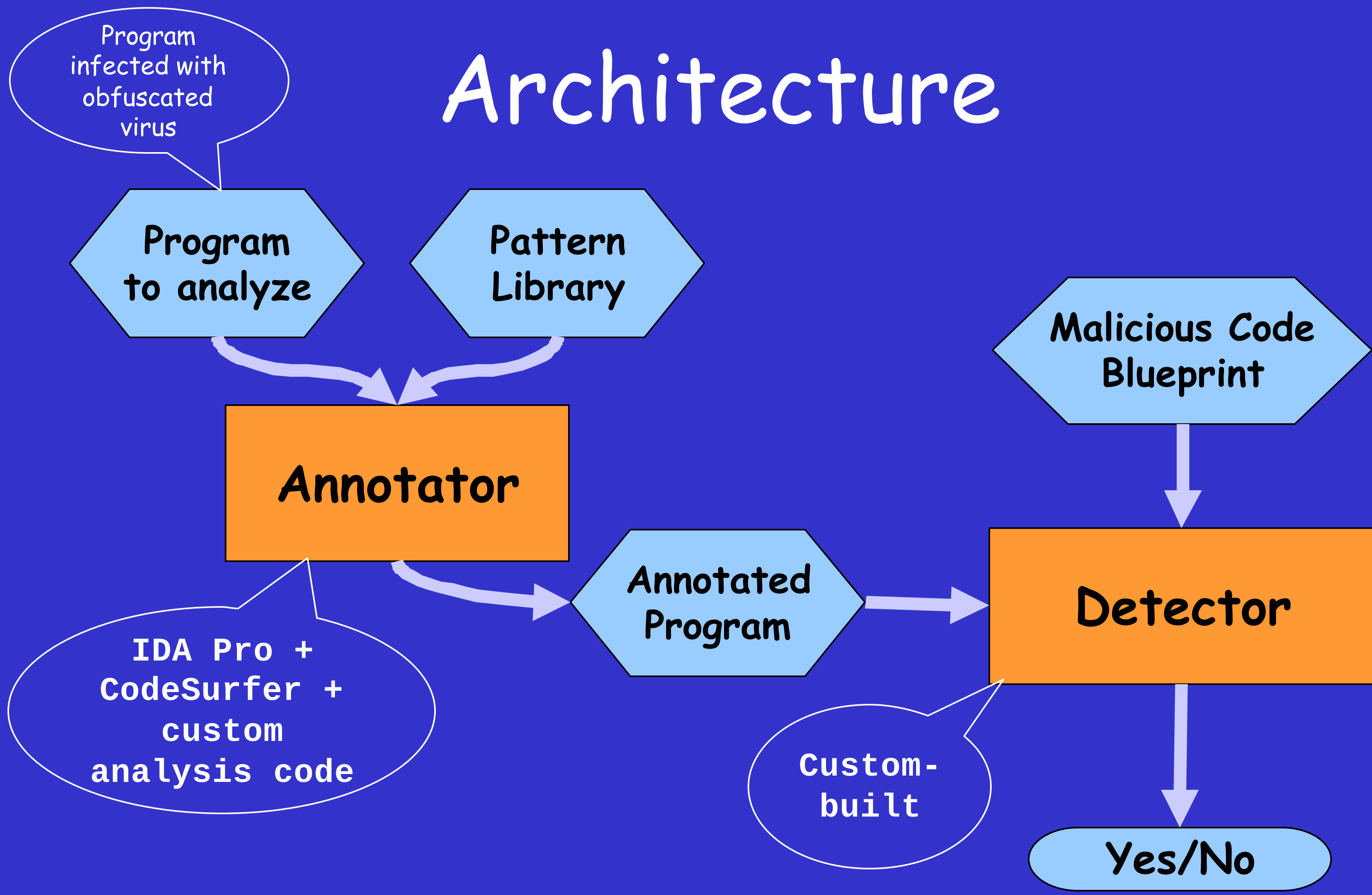
- Detection is as good as the static analyses available
  - More predicates ◆ better detection
  - Better predicates ◆ fewer false alarms

Example: pointer analysis (P.A.)

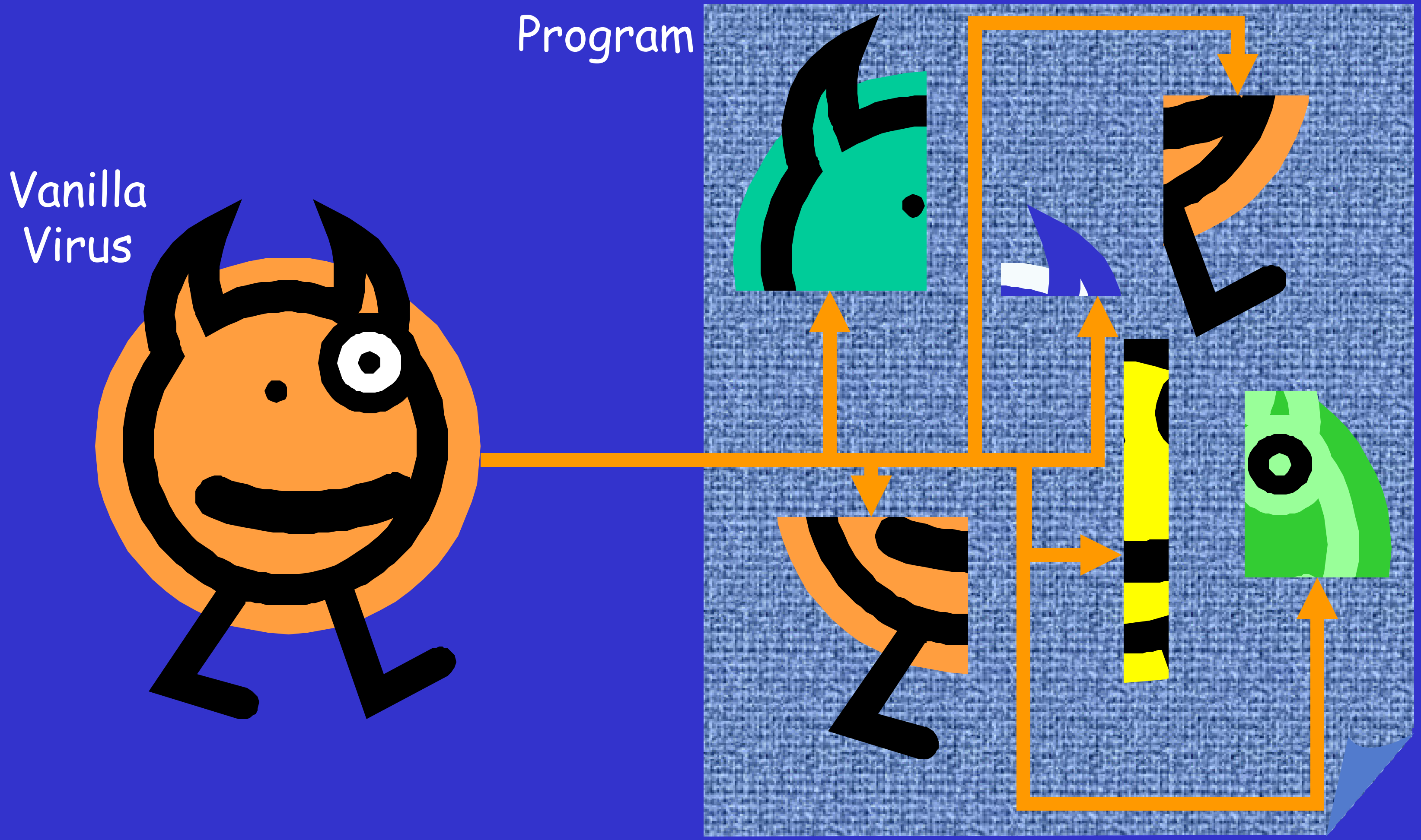
- No P.A.: it is safe to assume all pointers point to all memory locations
- With P.A.: reduced cost to attain safety

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

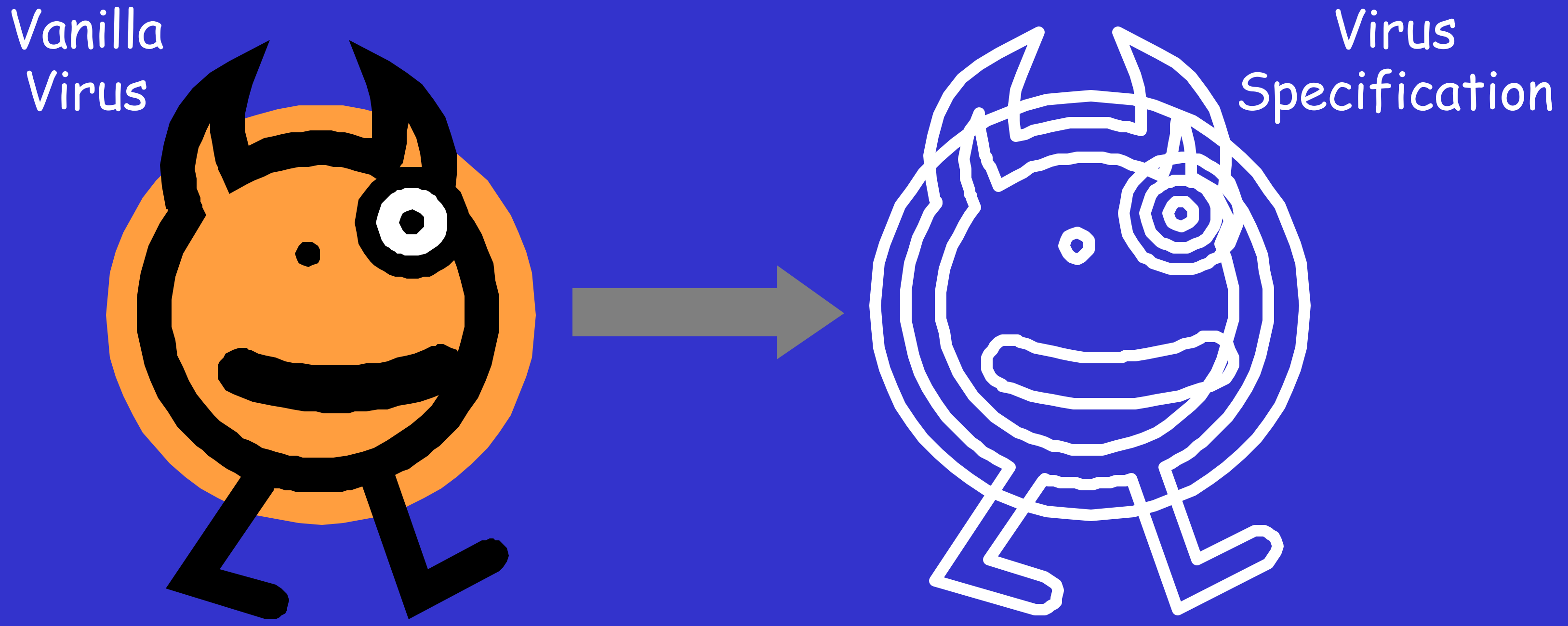
Architecture



Infection:



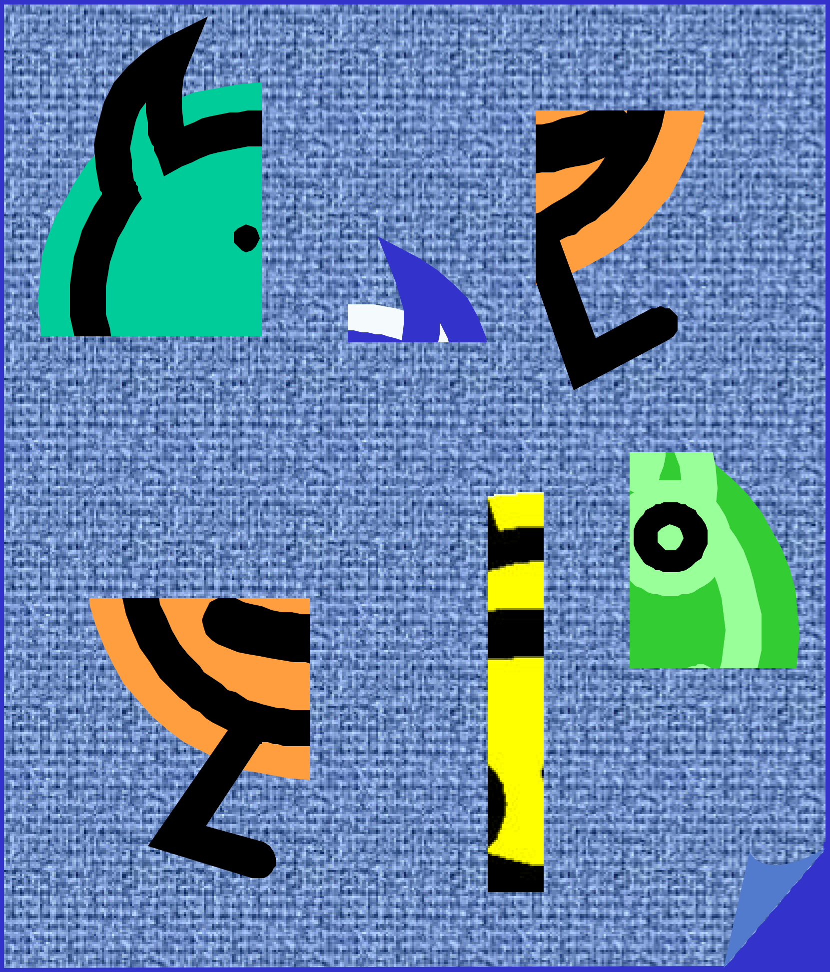
Detection: 1) Virus Blueprint



Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Detection: 2) Deobfuscation

Program

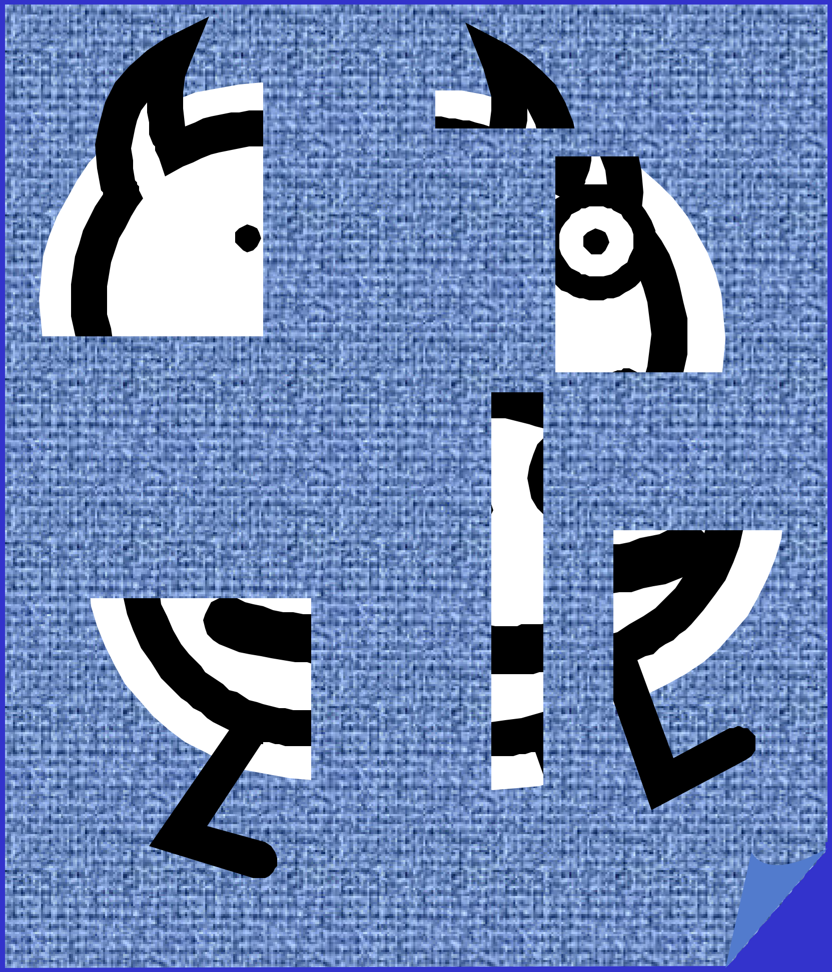


1. Detect code reordering

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Detection: 2) Deobfuscation

Program

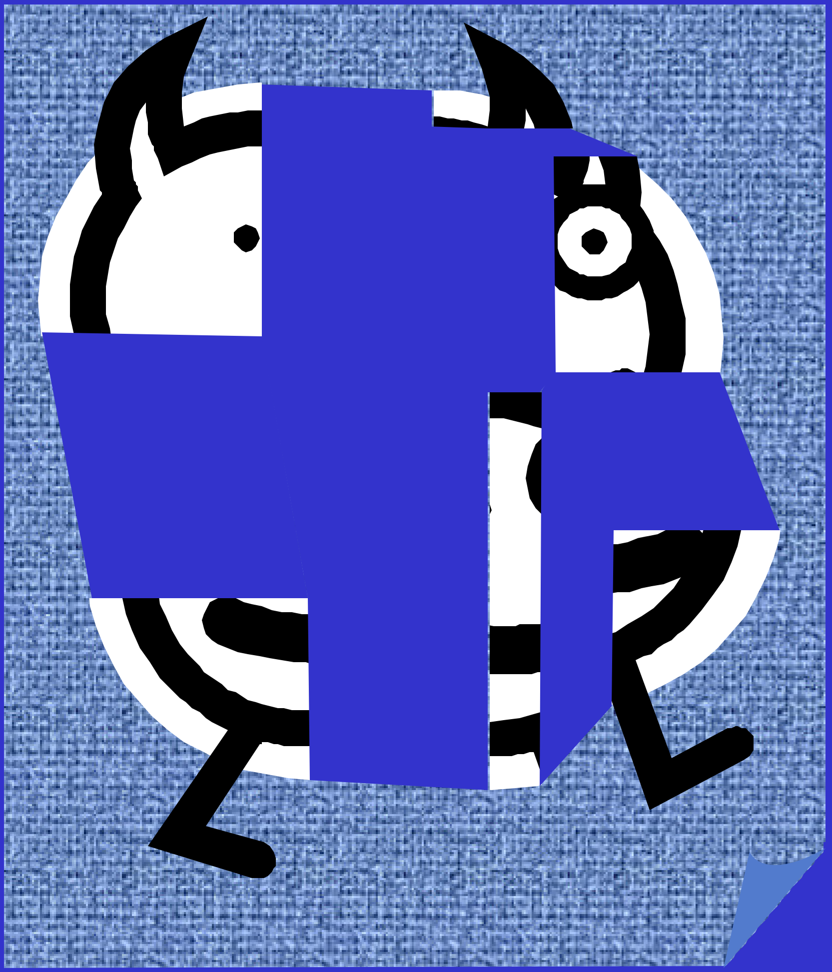


1. Detect code reordering
2. Detect register renaming

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Detection: 2) Deobfuscation

Program



1. Detect code reordering
2. Detect register renaming
3. Detect irrelevant code

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

What is irrelevant code?

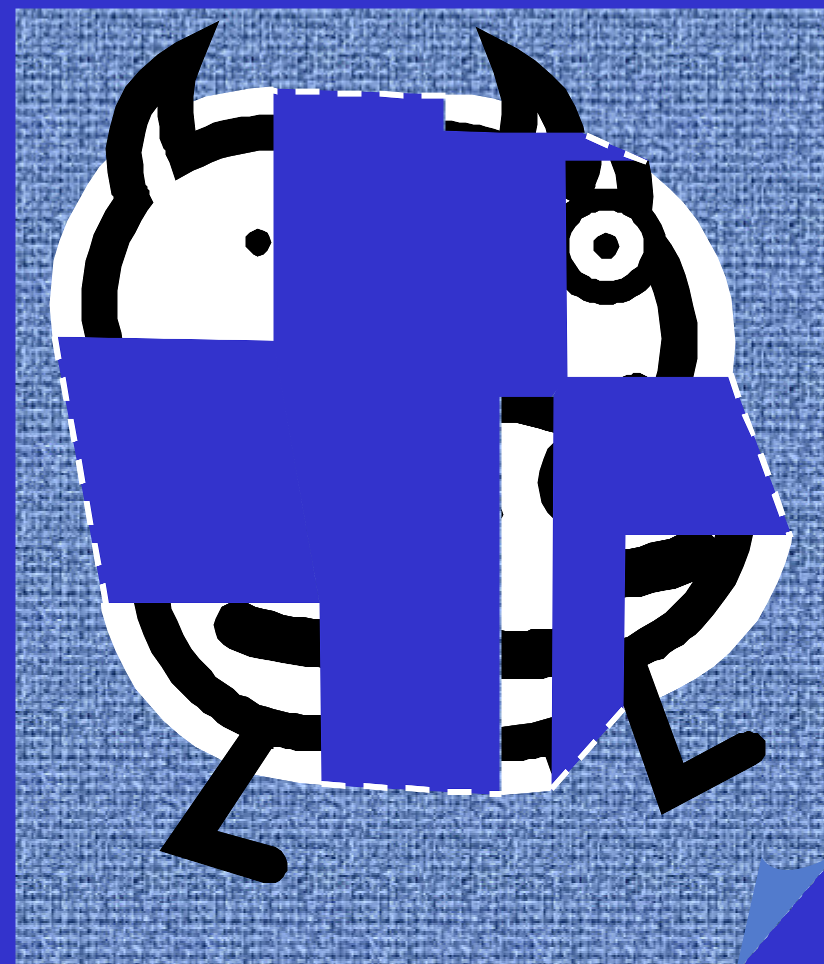
- Code does not change program behavior:
  - NOPs
  - Jumps/branches that do not change the control flow
  - Code that modifies dead registers
  - Code that do not modify the program state
    - e.g.: add ebx, 1
sub ebx, 1
- Theorem provers can be used to find irrelevant code

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Detection: 3) Matching

Annotated Program

~



Virus Specification

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Detection in Theory

- ☹ General detection problem is undecidable:
  Cohen Computer viruses: Theory and experiments (Computers and Security 1987)
  Chess, White An undetectable computer virus (VBC'00)
- ☹ Static analysis is undecidable as well:
  Landi Undecidability of static analysis (LOPLAS'92)
- ☺ (Computationally-bound) obfuscation is impossible
  Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, Yang On the (im)possibility of obfuscating programs (CRYPTO'01)

Detection in Practice

- Our approach is geared to common obfuscations in the wild
- Detection algorithm is matched against current obfuscation threats
  - Can handle more variants than signatures

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Building block: Patterns

Two components:

1. sequence of instructions
2. predicate controlling pattern application

- Predicates use static analysis results

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Defeating Garbage Insertion

```
<instruction A>
<instruction B>
```

```
<instruction A>
add ebx, 1
sub ebx, 1
nop
<instruction B>
```

Pattern:

```
instr 1
…
instr N
where
Delta( state pre 1, state post N ) = 0
```


Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Defeating Register Renaming

- Use uninterpreted symbols

Program 1:

```
         mov  ebp, [ebx]
         nop
         mov  bp, [ebx-04h]
         test ebx
         beqz next
next:    lea  esi, MyHook - @1[ecx]
```

Program 2:

```
         mov  eax, [ecx]
         nop
         mov  ax, [ecx-04h]
         test edx
         beqz next
next:    lea  ebi, MyHook - @1[ebx]
```

Virus Spec: with Uninterpreted Symbols:

```
         mov  ebp, [ebx]
```

```
         mov  X, [Y]
```

- Matches with Program 1
- Matches both Programs 1 and 2

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Defeating Code Reordering

```
<instruction A>
<instruction B>
```

```
        jmp L_1
L_2:    <instruction B>
        jmp L_3
L_1:    <instruction A>
        jmp L_2
L_3:    …
```


Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Defeating Code Reordering

Construct CFG:

```
jmp L_1
   ↓
<instruction A>
   ↓
jmp L_2
   ↓
<instruction B>
   ↓
jmp L_3
```

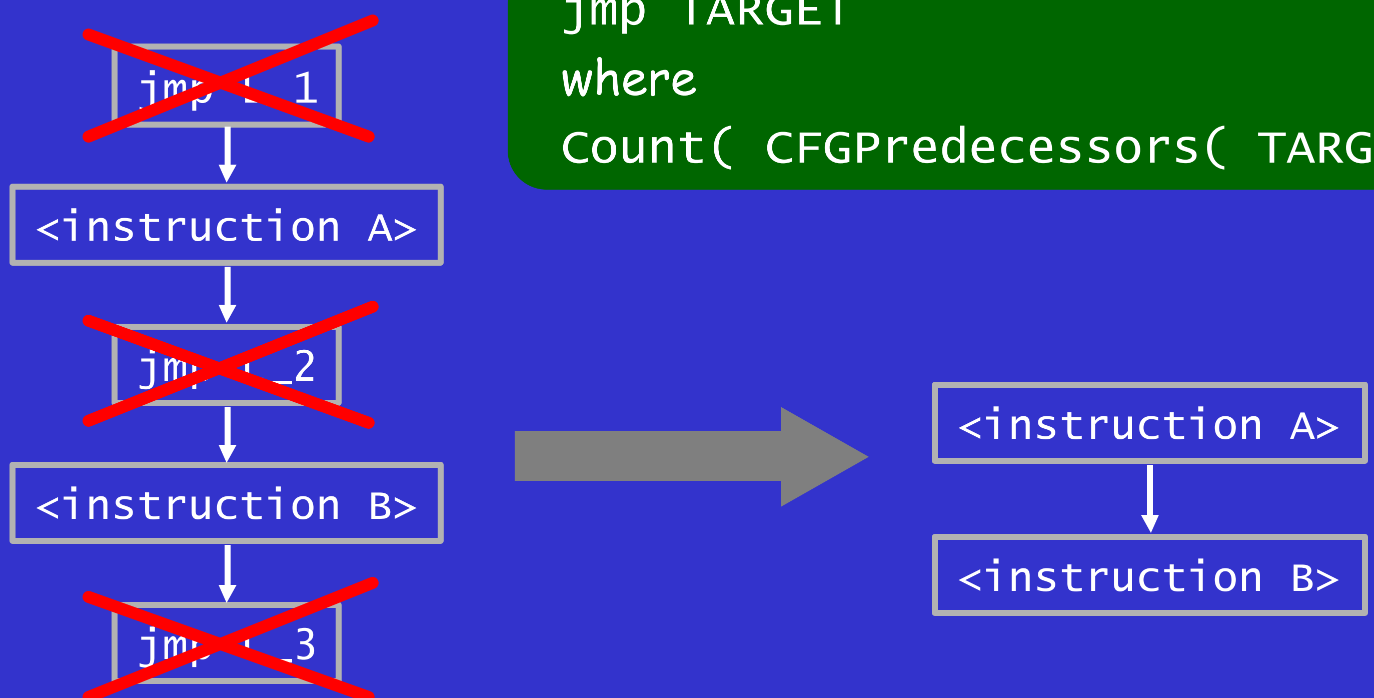
```
        jmp L_1
L_2:    <instruction B>
        jmp L_3
L_1:    <instruction A>
        jmp L_2
L_3:    …
```


Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Defeating Code Reordering

Pattern:

```
jmp TARGET
where
Count( CFGPredecessors( TARGET ) ) = 1
```



Prototype Implementation

- The detection tool can handle:
  - ✓ NOP-insertion
  - ✓ Code reordering (irrelevant jumps and branches)
  - ✓ Register renaming
- Work in progress to detect:
  - Malicious code split across procedures (need inter-procedural analysis)
  - Obfuscations using complex data structures (need integration with pointer analyses)

Testing Setup

Goals:

- Measure true negatives and false positives
  - Scan a representative collection of benign programs
- Measure true positives and false negatives
  - Scan a set of viruses obfuscated with various parameters
- Measure performance

Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Results

Effectiveness:

False positive rate: 0

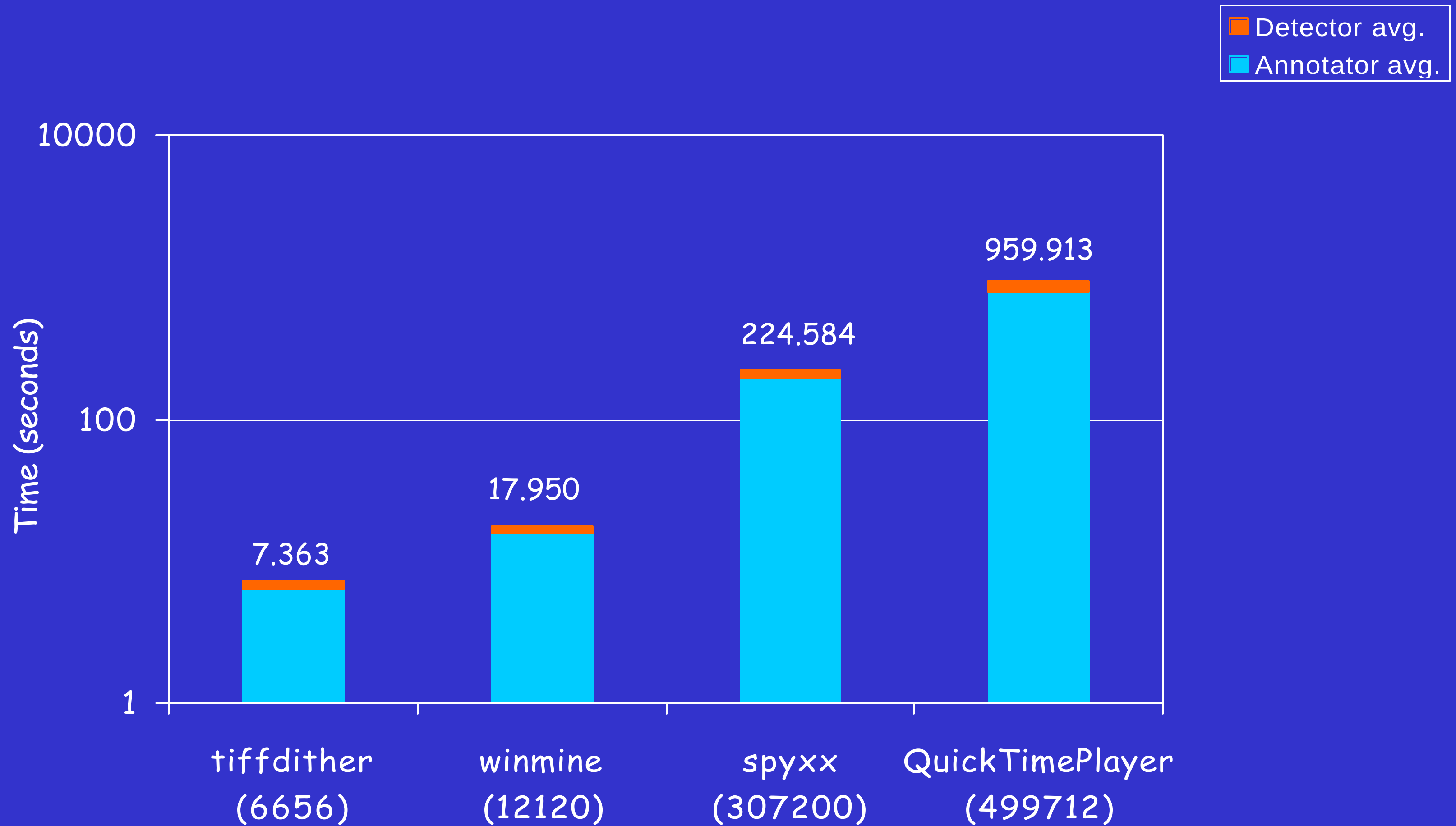
All benign programs passed the scans.

False negative rate: 0

All obfuscated viruses were detected.

But there are obfuscations we cannot yet detect.

Performance

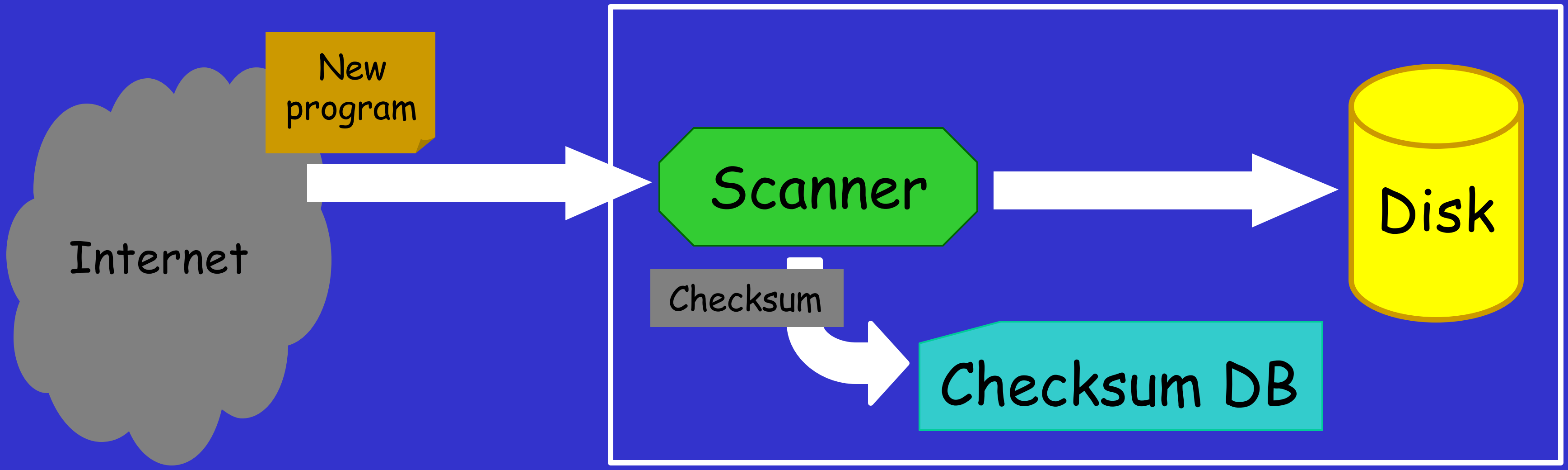


Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Performance Implications

- Combine with other techniques to amortize cost

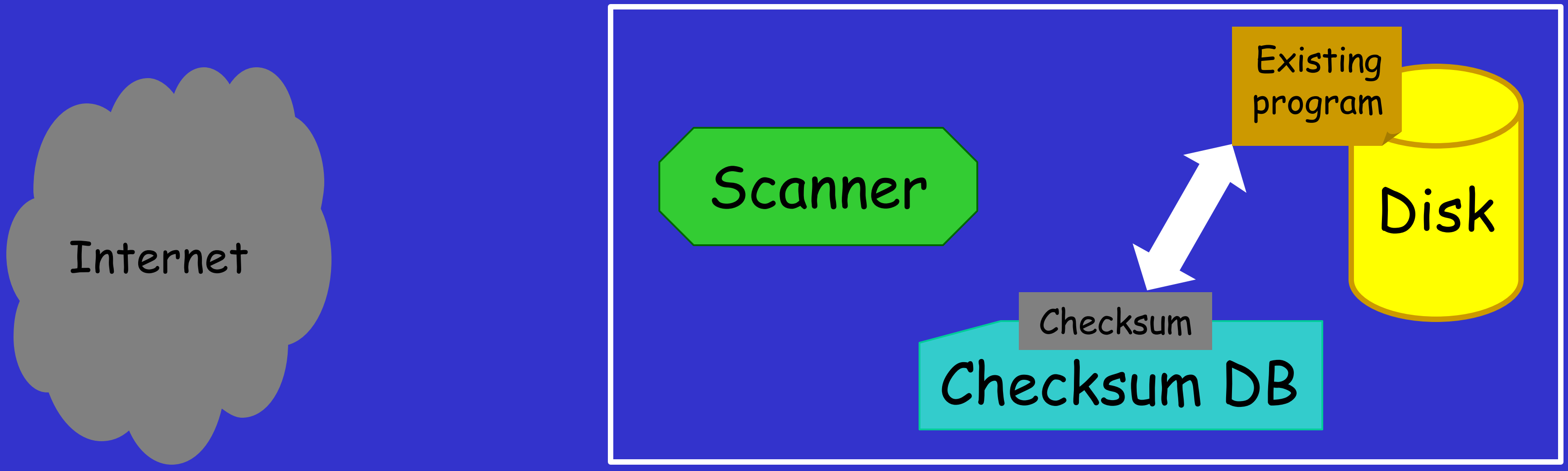
E.g.: Secure checksum database



Performance Implications

- Combine with other techniques to amortize cost

E.g.: Secure checksum database



Mihai Christodorescu mihai@cs.wisc.edu
WiSA Project http://www.cs.wisc.edu/wisa

Future Directions

- New languages
  - Scripts: Visual Basic (in progress), ASP, JavaScript
  - Multi-language malicious code
- Attack diversity
  - Beyond virus patterns: worms, trojans
- Irrelevant sequence detection
  - Decision procedures
  - Theorem provers

Conclusions

Viruses can self-modify as they propagate.

Current virus scanners cannot detect such malware.

Our semantic analysis can defeat obfuscations and detect viruses.

Related Work

- Metacompilation:
  Ashcraft, Engler Using programmer-written compiler extensions to catch security holes (Oakland'02)
- Theorem proving for security properties:
  Chess Improving computer security using extended static checking (Oakland'02)
- Model checking programs for security properties:
  Chen, Wagner MOPS: an infrastructure for examining security properties of software (CCS'02)
- Malicious code filter:
  Lo, Levitt, Olsson MCF: a malicious code filter (Computers and Society 1995)
- Inline reference monitors
  Erlingsson, Schneider IRM enforcement of Java stack inspection (Oakland'00)

Static Analysis of Binaries for Malicious Code Detection

Mihai Christodorescu, Somesh Jha
{mihai,jha}@cs.wisc.edu
University of Wisconsin, Madison

WiSA Project

http://www.cs.wisc.edu/wisa