Mining Specifications of Malicious Behavior

Mihai Christodorescu IBM Research (work done at University of Wisconsin)

Somesh Jha University of Wisconsin

Christopher Kruegel UCSB

Why Understand Malicious Behavior?

- Forensics
  - Understand what a malware does

- Malware Detector
  - Move to behavior-based detectors
  - These need detectors need a high-level specification of malware

Wide Spectrum of Detectors

- Static detectors:

- Dynamic/hybrid detectors, host IDS:

……

Semantics-aware malware detection [Christodorescu et al. 2005]

Model checking based malware detection [Kinder et al. 2005]

Behavior-based spyware detection [Kirda et al. 2006]

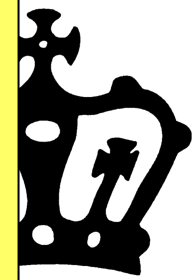
Shadow Honeypots [Anagnostakis et al. 2005]

Misuse Detection

Distinct techniques, fundamentally similar...

Sample specification:

- Creates an email with itself attached, and
- Collects email addresses, and
- Sends emails



They all require high-quality specifications of malicious behavior.

Key Definitions

Variants : New strains of viruses that borrow code, to varying degrees, directly from other known viruses.

Source: Symantec Security Response Glossary

Virus family: a set of variants with a common code base.

Signature-Based Detection

```
lea     eax, [ebp+Data]
push    offset aServices_exe
push    eax
call    _strcat
pop     ecx
lea     eax, [ebp+Data]
pop     ecx
push    edi
push    eax
lea     eax, [ebp+ExistingFileName]
push    eax
call    ds:CopyFileA
```

```
8D 85 D8 FE FF FF
68 78 8E 40 00
50
E8 69 06 00 00
59
8D 85 D8 FE FF FF
59
57
50
8D 85 D4 FD FF FF
50
FF 15 C0 60 40 00
```

Signature

- Signatures (aka scan-strings) are the most common malware detection mechanism.

Current Signature Management

McAfee: release daily updates

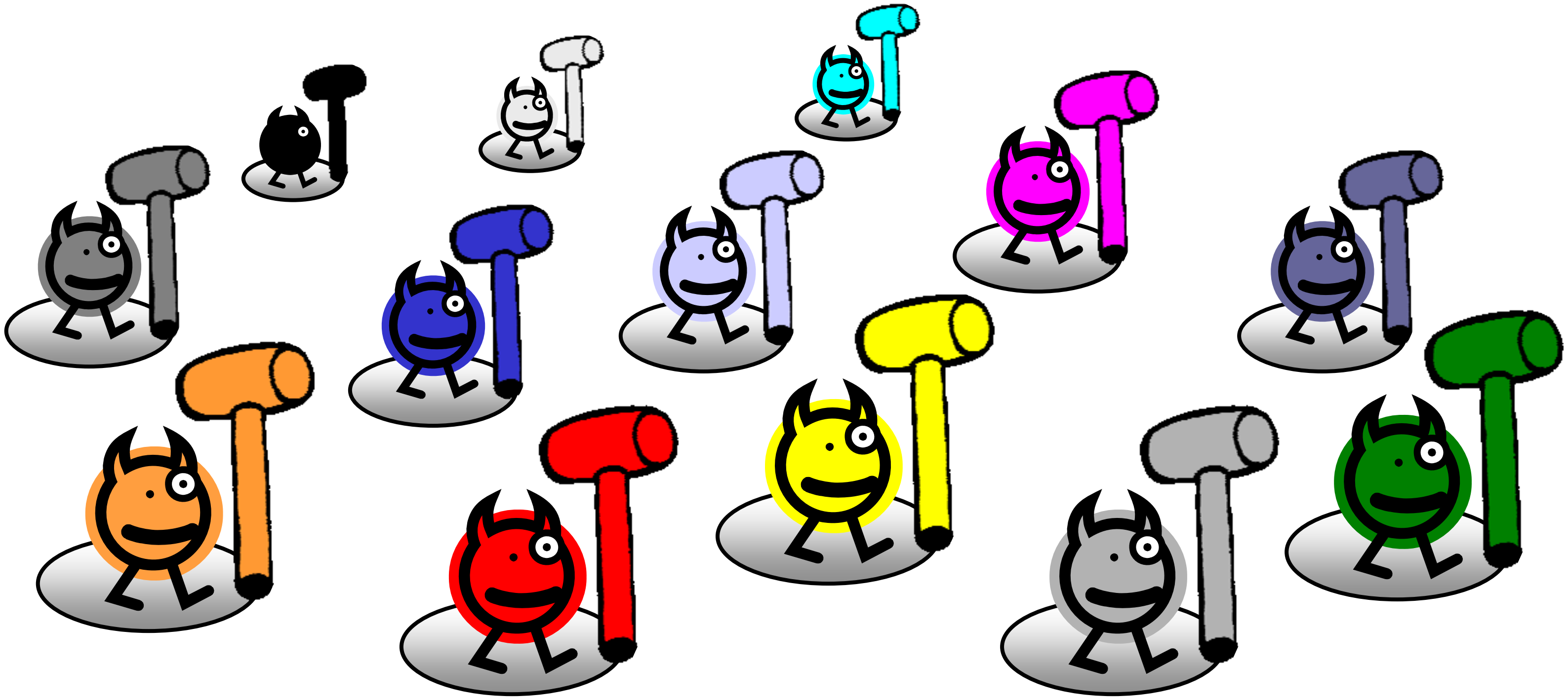
- Trying to move to hourly “beta” updates

DAT File #	Date	Threats Detected	New Threats Added	Threats Updated
4578	Sep. 09	147,382	22	188
4579	Sep. 12	147,828	27	231
4580	Sep. 13	148,000	11	236
4581	Sep. 14	148,368	42	140
4582	Sep. 15	148,721	16	203
4583	Sep. 16	149,050	18	117

Source: McAfee DAT Readme

Signature Detection Does Not Scale

One signature for one malware instance.



Goals for Better Detection

- Make the malware writer's job as hard as possible.
- Detect malware families,
  not individual malware instances.
- Move away from syntactic signatures.

Threat Model

- Malware writers craft their programs so to avoid detection.

Two common evasion techniques:

- Program Obfuscation
  (Preserves malicious behavior)

- Program Evolution
  (Enhances malicious behavior)

Obfuscations for Evasion

Nop insertion
Register renaming
Junk insertion
Instruction reordering
Encryption
Compression
Reversing of branch conditions
Equivalent instruction substitution
Basic block reordering
…

Evasion Through Junk Insertion

```
lea     eax, [ebp+Data]
nop
push    offset aServices_exe
nop
nop
push    eax
call    _strcat
nop
nop
nop
pop     ecx
lea     eax, [ebp+Data]
pop     ecx
push    edi
push    eax
nop
lea     eax, [ebp+ExistingFileName]
push    eax
call    ds:CopyFileA
```

```
8D 85 D8 FE FF FF
68 78 8E 40 00
50
E8 69 06 00 00
59
8D 85 D8 FE FF FF
59
57
50
8D 85 D4 FD FF FF
50
FF 15 C0 60 40 00
```

Signature

Evasion Through Reordering

```
        lea     eax, [ebp+Data]
        jmp label_one

label_two:
        lea     eax, [ebp+Data]
        ...
        push    eax
        call    ds:CopyFileA
        jmp label_three

label_one:
        ...
        call    _strcat
        ...
        jmp label_two

label_three: ...
```

```
8D 85 D8 FE FF FF
90*
68 78 8E 40 00
90*
50
90*
E8 69 06 00 00
90*
59
90*
.
.
.
90*
50
90*
FF 15 C0 60 40 00
```

Regex Signature

Evasion Through Encryption

```
        lea      esi, data_area
        mov      ecx, 37
again:
        xor byte ptr [esi+ecx], 0x01
        loop     again
        jmp      data_area
        .
        .
        .
data_area:
        db       8C 84 D9 FF ...
        .
        .
        .
        db       FE 14 C1 61 ...
```

```
8D 85 D8 FE FF FF
90*
68 78 8E 40 00
90*
50
90*
E8 69 06 00 00
90*
59
90*
.
.
.
90*
50
90*
FF 15 C0 60 40 00
```

Regex Signature

Evasion Through Evolution

- Malware writers are good at software engineering:
  - Modular designs
  - High-level languages
  - Sharing of exploits, payloads, and evasion techniques

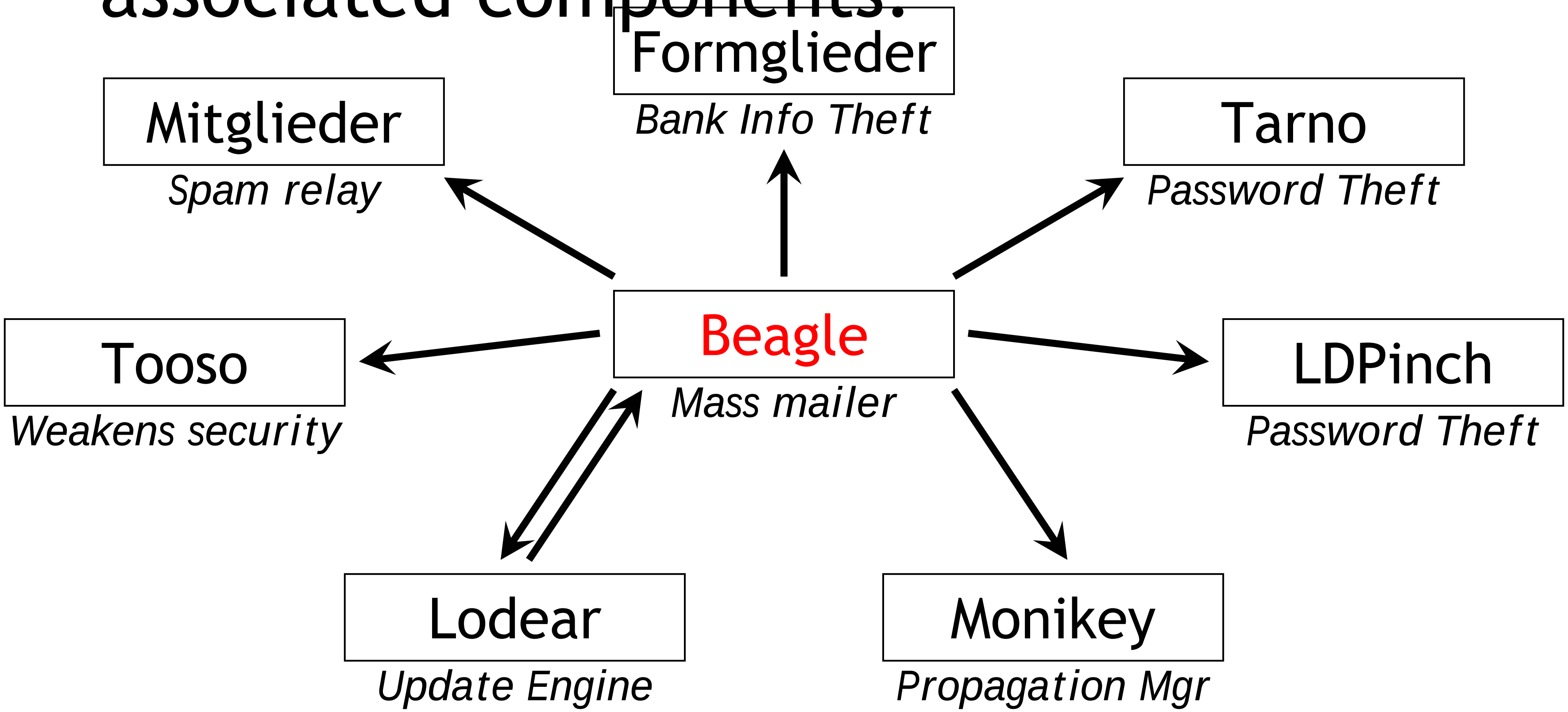
Example:

Beagle e-mail virus gained additional functionality with each version.

Beagle Evolution

Source: J. Gordon, infectionvectors.com

- More than 100 variants, not counting associated components.



Describing Malicious Behavior

[Christodorescu et al., Oakland 2005]

- Informal description:

  “Mass-mailing virus”

- A more precision description:

  “A program that:

  sends messages containing copies of itself,

  using the SMTP protocol,

  in a large number over a short period of time.”

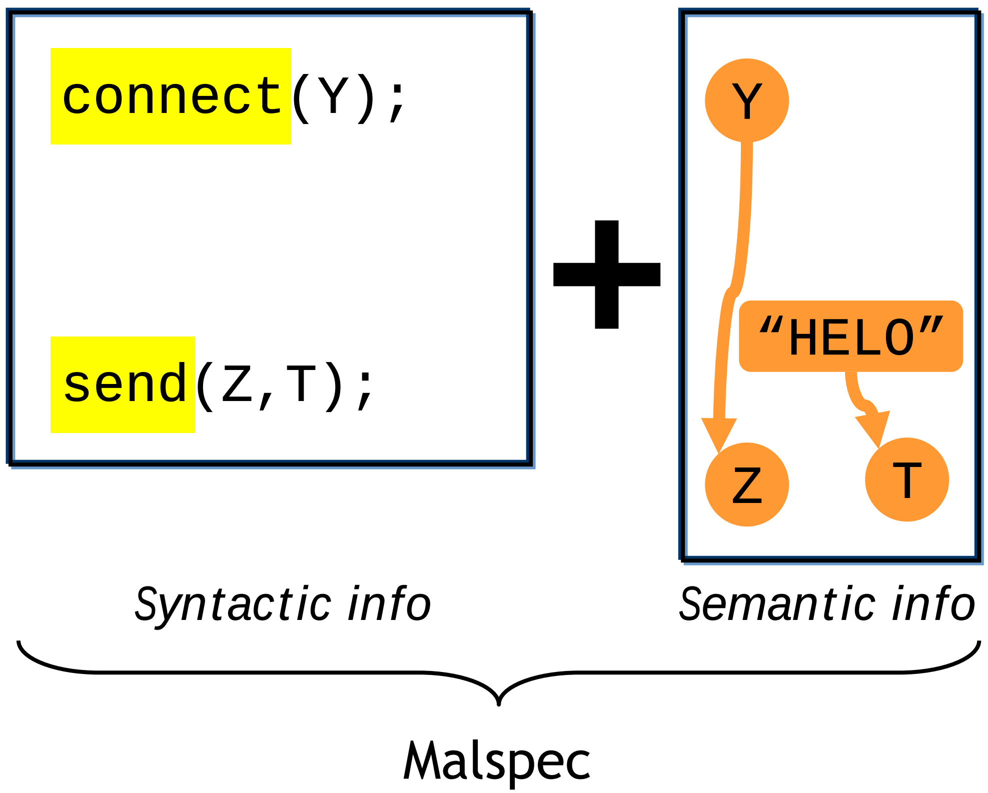
Malspec

- A specification of behavior.

```
push    10h
push    eax
push    edi
call    connect
...     ; compose SMTP
        ; command "HELO ..."
push    eax
push    ecx
push    edi
call    send
```

Malware Instance
(Netsky.B)

=



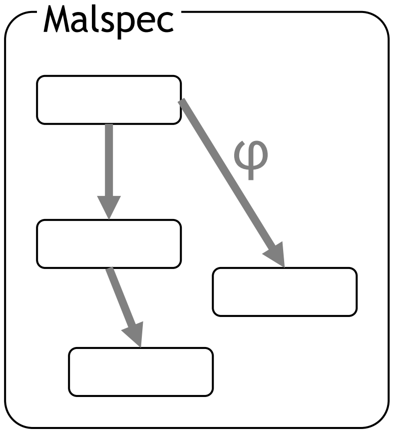
Obfuscation Preserves Behavior

```
push    10h
push    eax
push    edi
call    connect
...     ; compose SMTP
        ; command "HELO ..."
push    eax
push    ecx
push    edi
call    send
```

```
push    10h
nop
push    eax
xor     eax, ebx
xor     eax, ebx
push    edi
call    connect
...     ; compose SMTP
        ; command "HELO ..."
push    eax
push    eax
pop     eax
push    ecx
push    edi
call    send
```

- Junk insertion + code reordering.

Detection Using Malspecs

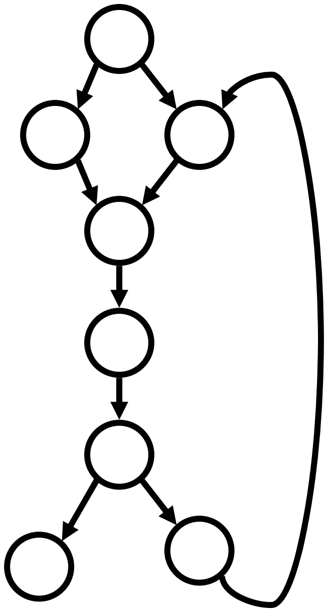


Static detection:

Given an executable binary, check whether it satisfies the malspec.

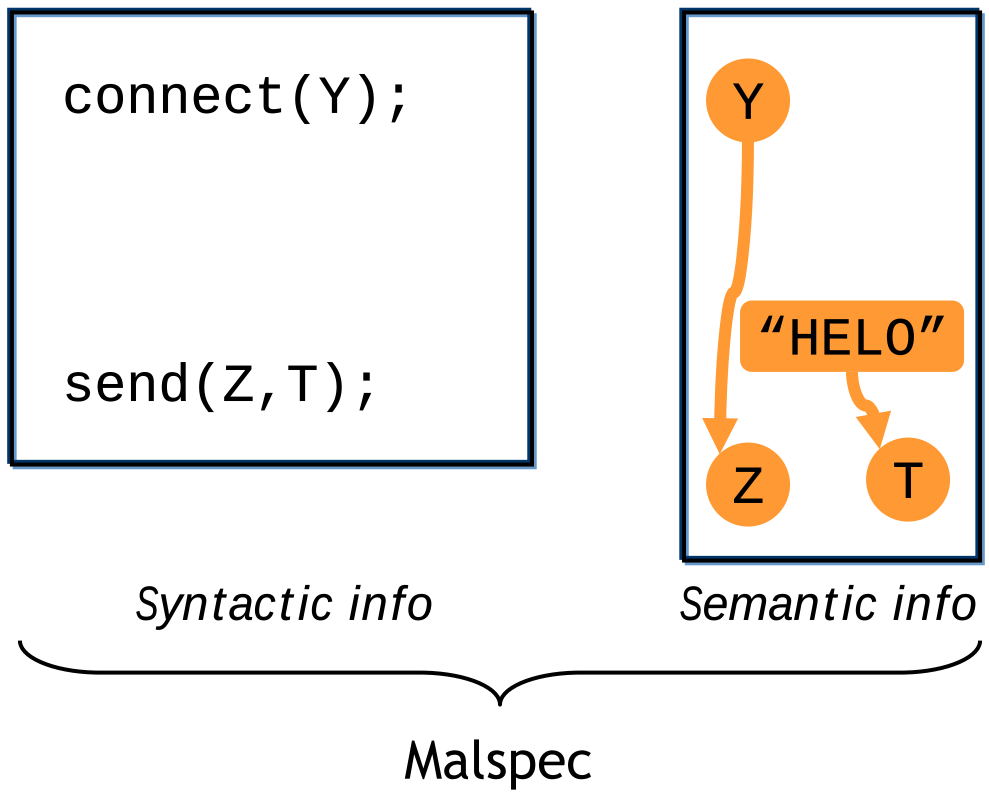
Just like model checking, but...

- Malicious code allows no assumptions to be made
- Real-time constraints



A Behavior-Based Detector

- Match the syntactic constructs, then check the semantic information.



Check the Semantic Info

Program (Netsky.O):

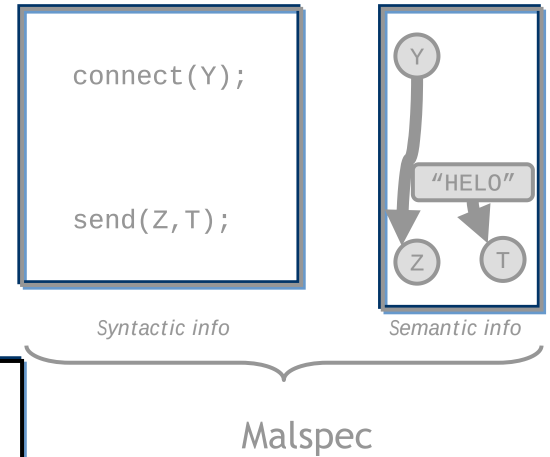
```
push    10h
push    eax
push    [ebp+s]
call    connect
...
push    ebx
lea     eax, [ebp+s]
push    eax
call    send_email
```

send_email()

```
...          ; compose SMTP
             ; command “HELO ..."
lea          eax, [ebp+arg1]
push         eax
lea          eax, [ebp+buffer]
push         eax
call         SMTP_send_and_rcv
```

SMTP_send_and_rcv()

```
push    eax
push    [ebp+arg1]
mov     eax, [ebp+arg2]
push    [eax]
call    send
```



Check with the Oracle

- Assume we have an oracle that can validate value predicates.

Does
eax before == ebx after
for the code sequence:

```
push eax
call foo
mov ebx, [ebp+4]
```

?

Yes.

Check the Semantic Info

Program (Netsky.O):

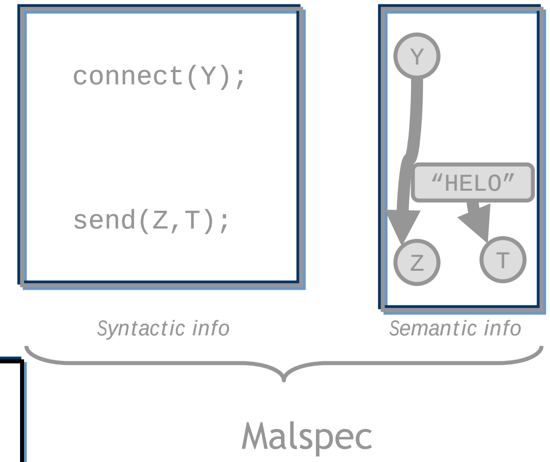
```
    push   10h
    push   eax
    push   [ebp+s]
A:  call   connect
    ...
    push   ebx
    lea    eax, [ebp+s]
    push   eax
    call   send_email
```

send_email()

```
...      ; compose SMTP
         ; command "HELO ..."
lea      eax, [ebp+arg1]
push     eax
lea      eax, [ebp+buffer]
push     eax
call     SMTP_send_and_rcv
```

SMTP_send_and_rcv()

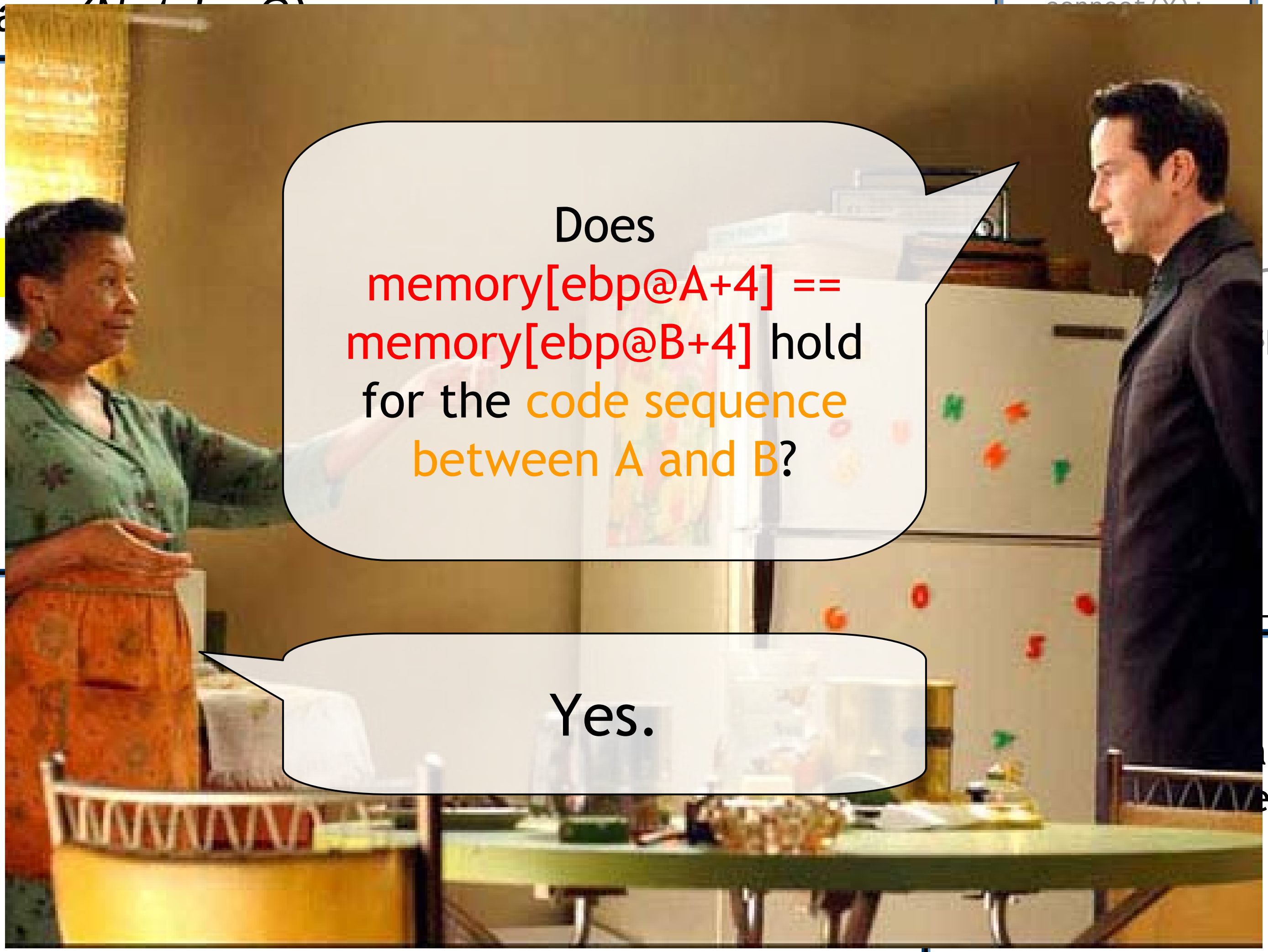
```
    push   eax
    push   [ebp+arg1]
    mov    eax, [ebp+arg2]
    push   [eax]
B:  call   send
```

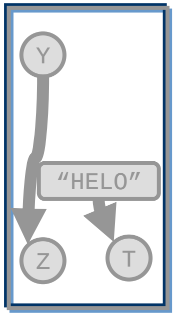


Query the Oracle

Progra

```
     push
     push
     push
A:   call
     …
     push
     lea
     push
     call
```





spec

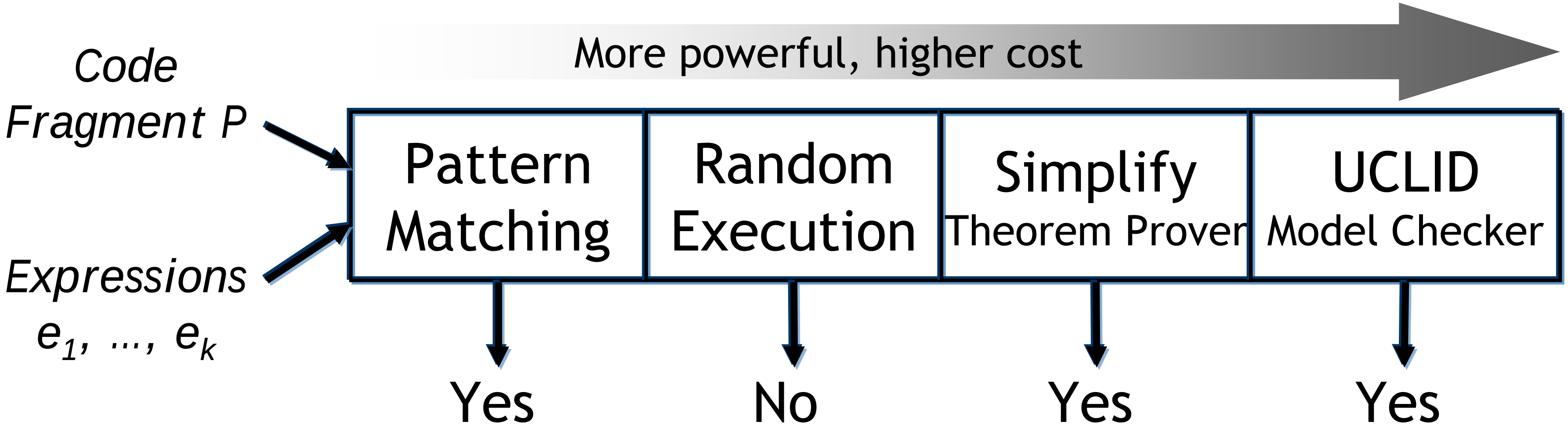
_and_rcv()

rg1]
ebp+arg2]

A Recipe for an Oracle

- Instance of program verification problem:

  Does program P respect property ϕ?



Evaluation of Malspecs

Netsky.B → Decryption sig

Netsky.B → Mass-mailing sig

→ Prototype detector

Netsky.C	✓
Netsky.D	✓
Netsky.O	✓
Netsky.P	✓
Netsky.T	✓
Netsky.W	✓

McAfee uses individual signatures for each worm.

Malspecs provide forward detection.

Additional Information

- Papers
  - M. Christodorescu and S. Jha, Testing Malware Detectors, *International Sympoisum on Testing and Analysis (ISSTA)*, 2004
  - M. Christodorescu, S. Seshia, S. Jha, D. Song, and R. Bryant, *Semantics-Aware Malware Detection, IEEE Symposium on Security and Privacy (Oakland)*, 2005.
- Website
  - http://www.cs.wisc.edu/wisa/

Problem 2: Spec. Imprecision

Too general = false positives

→ Angry users

→ Infected machines

Too specific = false negatives

Our Automatic Solution

MiniMal: a technique for mining malicious specifications

- (Mostly) automatic
- Flexible specification language
- Fast
- Performs well (compared to a human expert)

Specification Language

What's In a Specification?

Requirements for obfuscation resilience:

1. Describe only relevant operations
2. Capture dependencies where present
3. Preserve independence of operations

Specifying Malicious Operations

- We chose system calls
  - Compatible with specifications for behavior-based detectors
  - Define interface between trusted OS and untrusted programs

- Mining algorithm is not restricted to the system-call interface.

Specifying Malicious Constraints

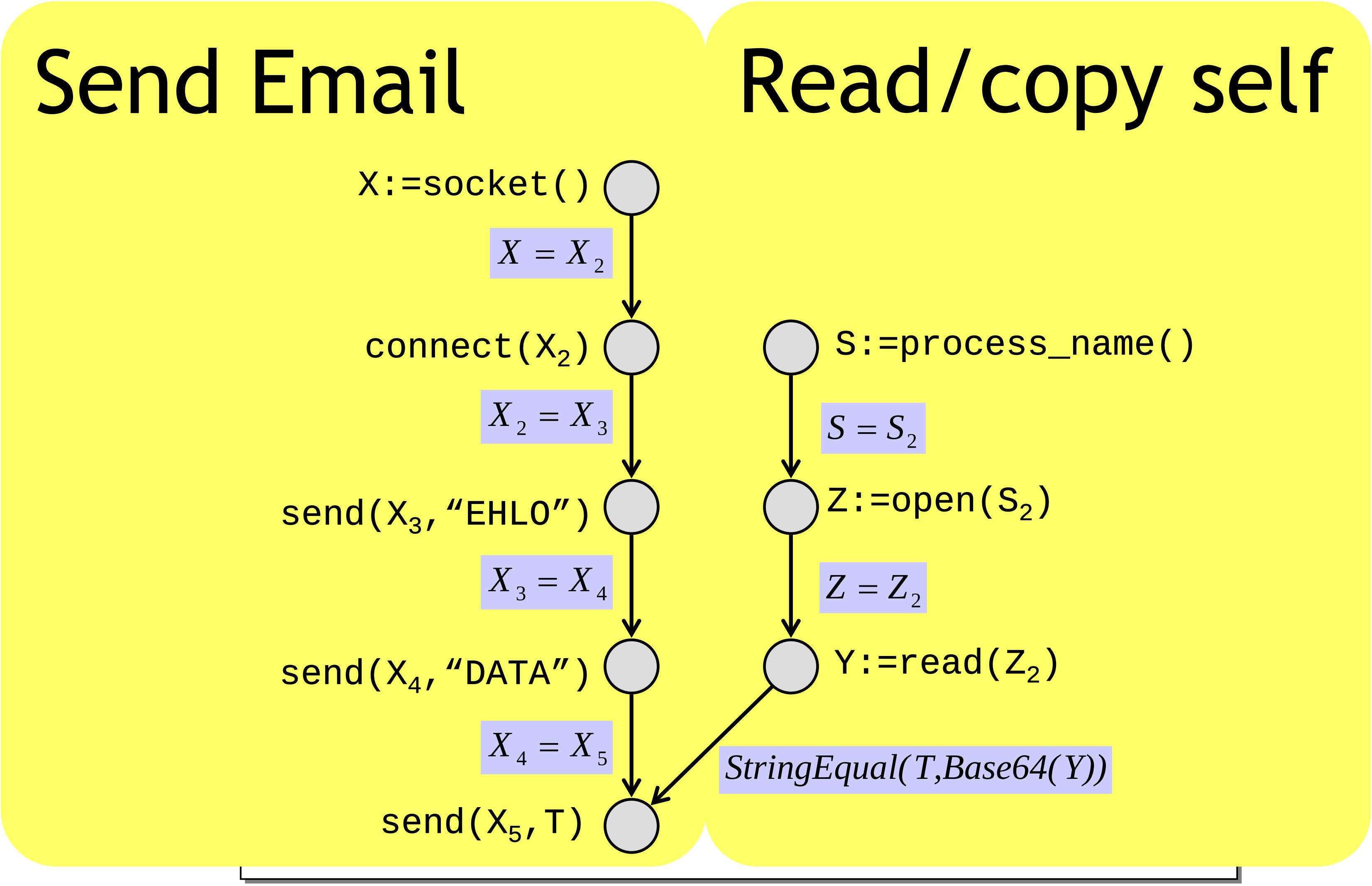
- Program operations are insufficient to distinguish malicious from benign.

- We need to capture relations between operations:

F=open("file") ; read(F,buf) ; send(S,buf)

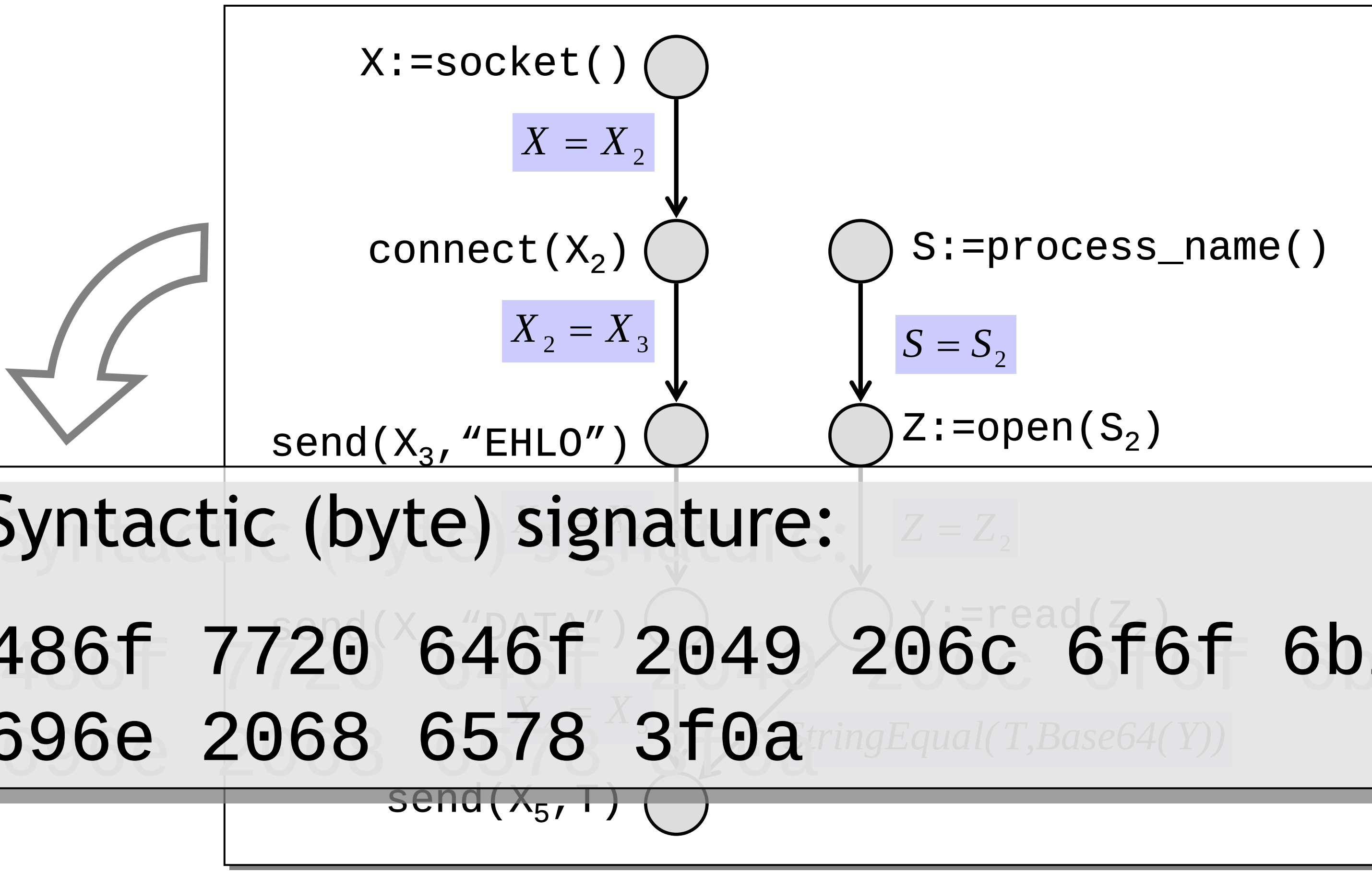
Constraints = logical formulas over system-call arguments

A Sample Specification (Malspec)



A Sample Specification (Malspec)

- Rich specification can be “dumbed down”



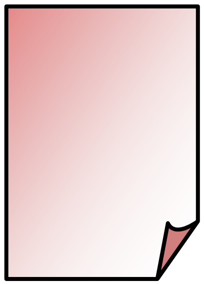
Syntactic (byte) signature:

486f 7720 646f 2049 206c 6f6f 6b20
696e 2068 6578 3f0a

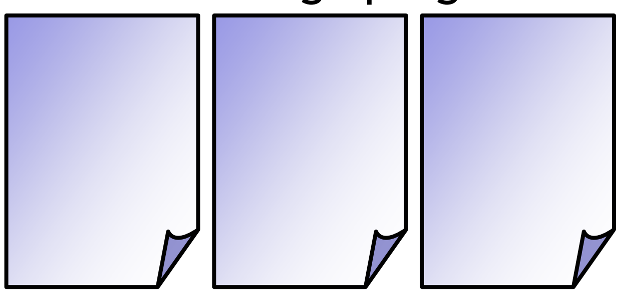
Mining Algorithm

The Specification Mining Problem

Known malware

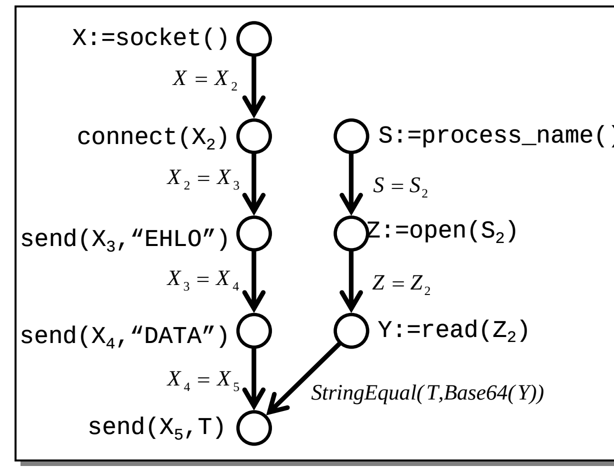


Known benign programs

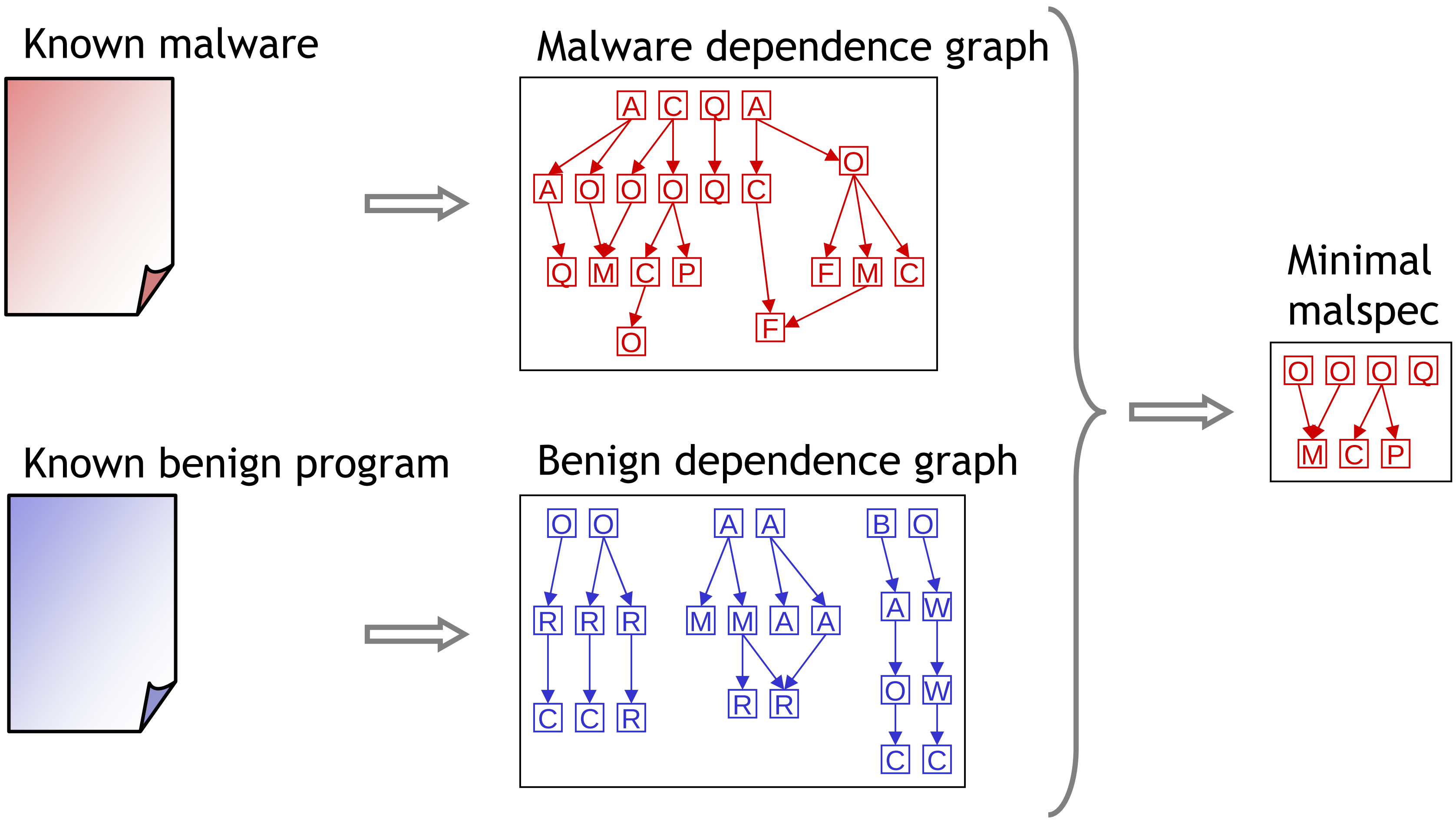


MiniMal

Specification of malicious behavior



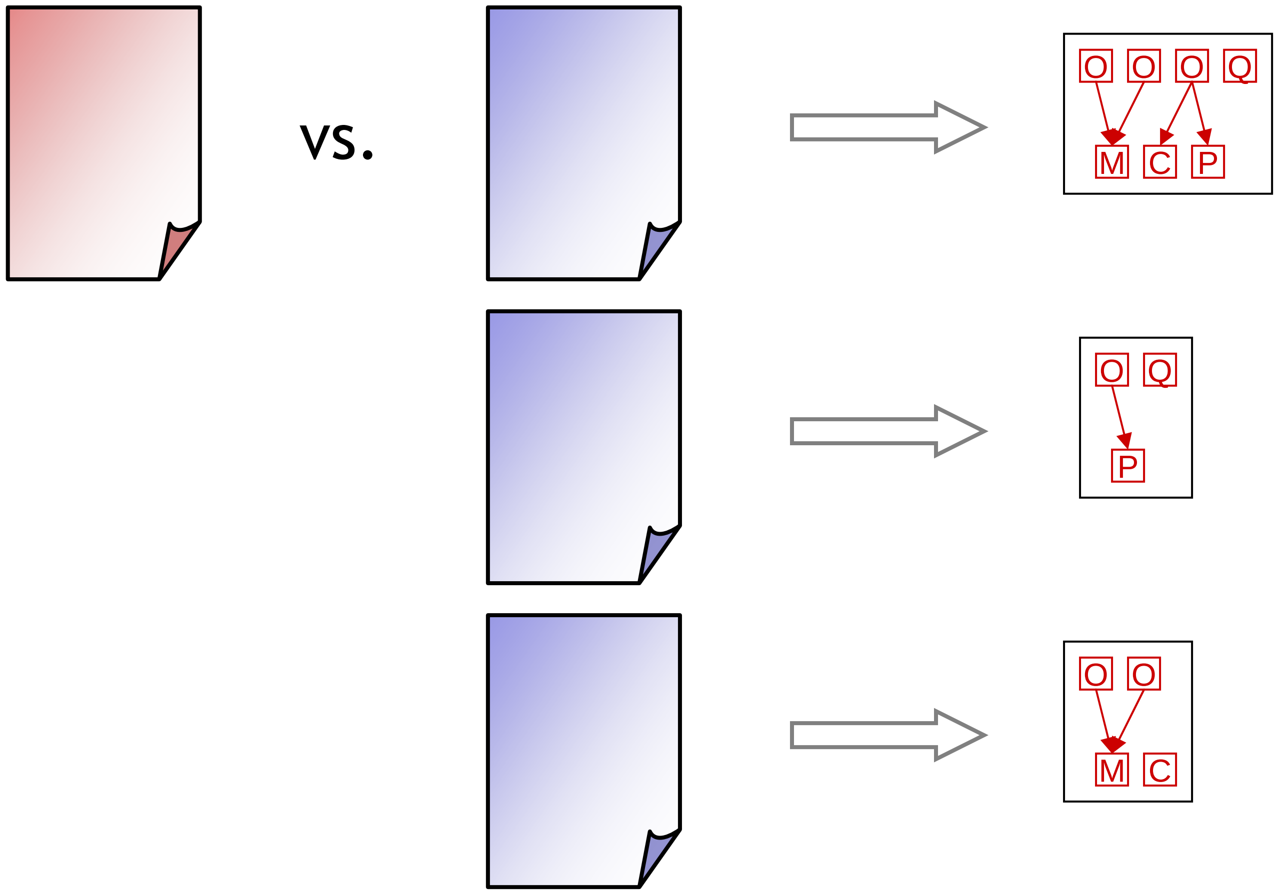
The Basic Mining Operation



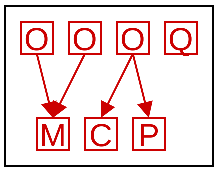
Step 1 Compute dependence graphs

Step 2 Compute graph difference

Multi-Program Mining



Maximal union of malspecs:



System-Call Dependence Graph

- We use a dynamic analysis to construct the dependence graph
  - Static analysis too imprecise on binary code

- Steps:
  1. Collect system-call trace
  2. Infer dependencies between system calls
  3. Construct (an underapproximation of the) dependence graph

Step 1

Discovering Dependences

NtOpenKey(372, 0x20019, {24, 356,
"ActiveComputerName", 0x40, 0, 0})

NtQueryValueKey(372, "ComputerName", Full, {
TitleIdx=0, Type=1,
ComputerName", Data="Z..."
6)

Def-Use Dependences

NtClose(372)

Substring Dependences

Step 1

Discovering Local Constraints

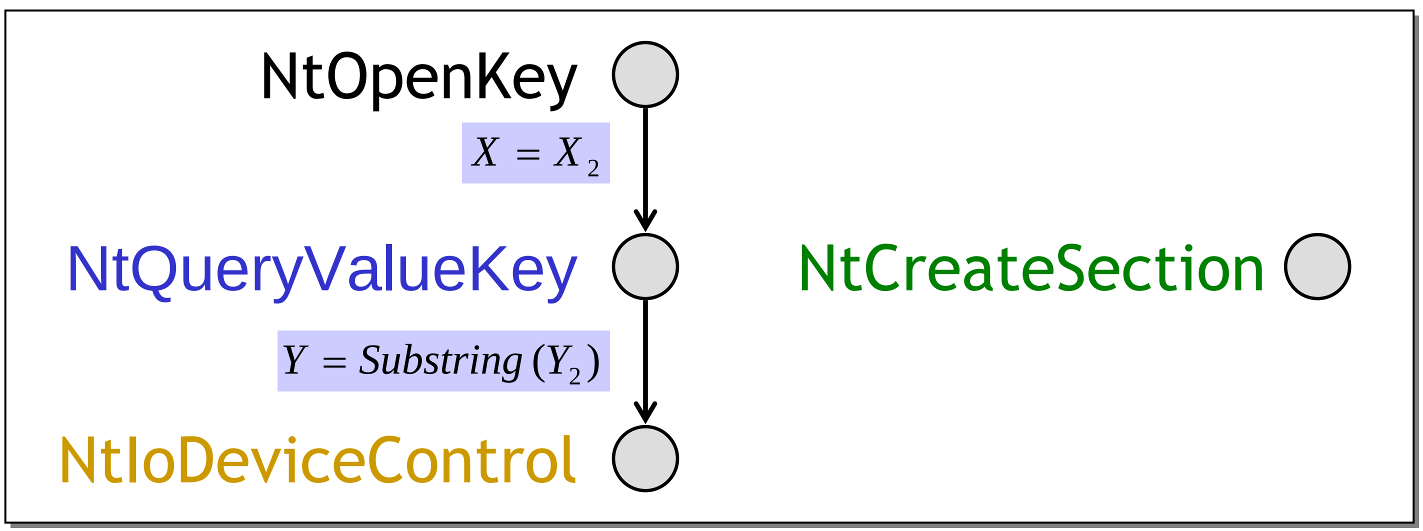
- Access to well-defined resources:
  - Windows registry
  - Access to self
  - System files/directories

NtCreateFile (…, { …, "I-Worm.Mydoom.l.exe", … }, …)

Step 1

Dependence Graph Example

NtOpenKey(372, …)
NtQueryValueKey(372, …, { …, Data="Z…" }, …)
NtCreateSection(…)
NtIoDeviceControl(…, OutBuffer="…Z…" …)



Step 2

Graph Differencing

Problem:

Find the smallest subgraph of malicious operations that does not appear in any benign graph.

Solution:

Minimal Contrast Subgraph

[Ting, Bailey “Mining Minimal Contrast Subgraph Patterns”, SDM 2006]

Minimal Contrast Subgraphs

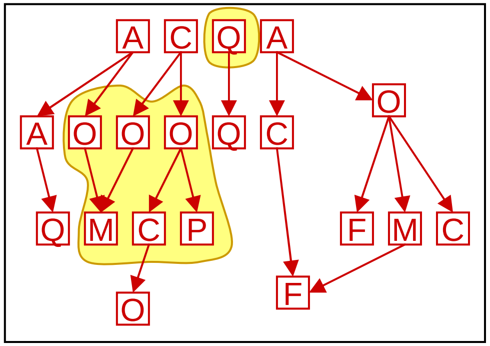
- Idea:
  Minimal contrast subgraphs and
  maximal common edge sets
  are duals.

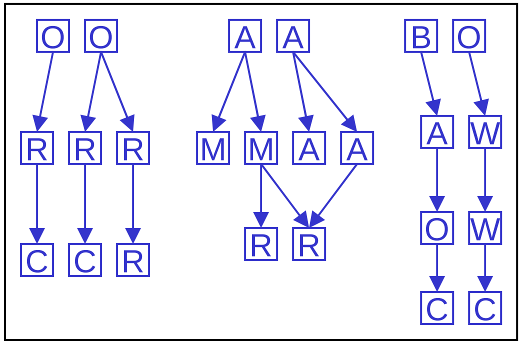
- Finding maximal common edge set:
  - Consider all edge sets (order by size)
  - Eliminate edge-set candidates as early as possible

Step 2

Mining Contrast Subgraphs

Malware dependence graph





Benign dependence graph

- Size of graphs: N = 100K-1.5M nodes
- Worst-case complexity: O(N!)

Heuristics Reduce Problem Size

- Normalize dependence graph
  - Replace system-call sequences with shorter equivalents
- Eliminate disconnected subgraphs
- Eliminate trivial subgraphs

[see paper for details]

Evaluation

Evaluating MINIMAL

- Goals:
  - Compare MINIMAL malspecs with those from human expert
  - Use mined malspecs with behavior-based detector

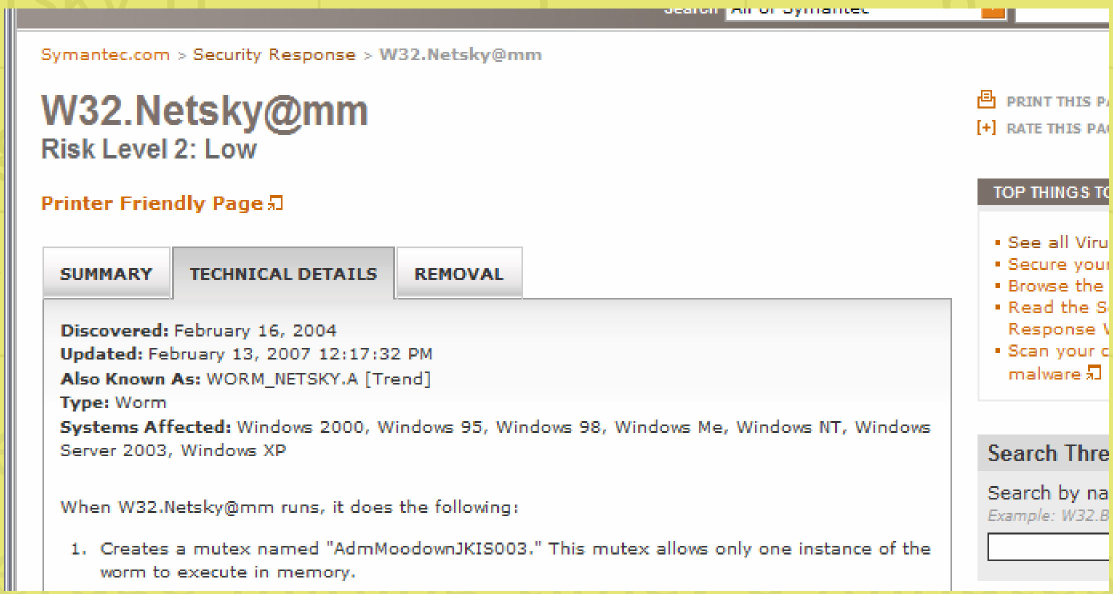
Experimental Setup

- Trace collection in Windows 2000:
  - Malware samples run with no user input (cf. expected execution model)
  - Benign samples run with normal user input
  - Execution for 1 or 2 minutes
- 16 malware samples:
  - Netsky, MyDoom, Beagle
- 6 benign programs:
  - Firefox, Thunderbird, installers

MINIMAL vs. Human Expert

	MINIMAL malspecs	

Behavioral features as given by Symantec:



Symantec.com > Security Response > W32.Netsky@mm

W32.Netsky@mm

Risk Level 2: Low

Printer Friendly Page

SUMMARY | TECHNICAL DETAILS | REMOVAL

Discovered: February 16, 2004
Updated: February 13, 2007 12:17:32 PM
Also Known As: WORM_NETSKY.A [Trend]
Type: Worm
Systems Affected: Windows 2000, Windows 95, Windows 98, Windows Me, Windows NT, Windows Server 2003, Windows XP

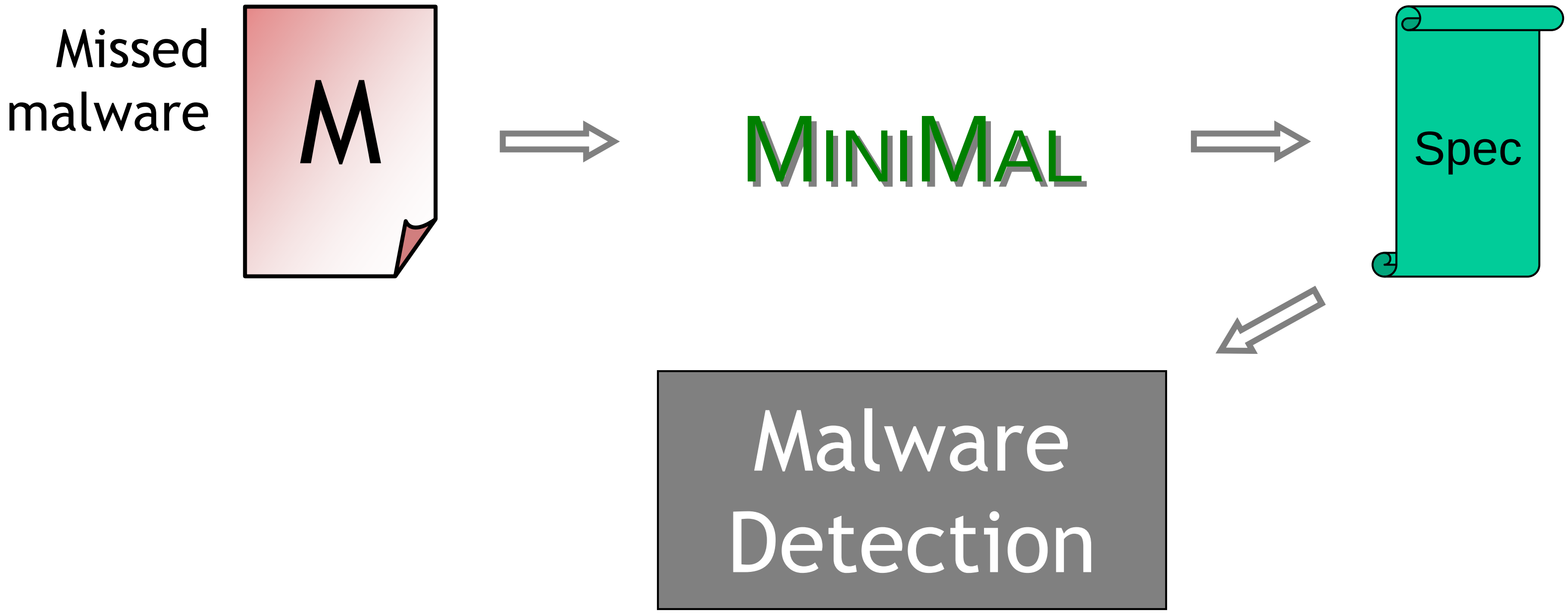
When W32.Netsky@mm runs, it does the following:

1. Creates a mutex named "AdmMoodownJKIS003." This mutex allows only one instance of the worm to execute in memory.

Mined Malspecs for Netsky.A

	MINIMAL malspecs
Create mutex	✓
Self-installation	✓
Modify boot sequence	✓
Terminate antivirus	✓
Email self as ZIP file	
Copy self to network drive	

MINIMAL Specs in Detection



- Using mined malspecs in semantics-aware malware detection:

  Netsky.A malspec → Netsky.D, E, F, …

~~Limitations of~~ Future Work MINIMAL

- Sensitive to test environment
  - Malicious behavior might not be observed during tracing.
- Underapproximation of dependence graph
  - Complex constraints are not discovered.
- Sensitive to test-set selection
  - Not all differences are malicious behaviors.

Questions?

Mining Specifications of Malicious Behavior

Somesh Jha

jha@cs.wisc.edu