Malware Analysis

Arun Lakhotia
University of Louisiana at Lafayette, USA

Presented at ISSISP 2017, CNRS Gif Sur Yvette

Introduction

Professor of Computer Science

Founder, CEO

Geolocation

Plan of talk

- Malware detection in practice
- Binary Analysis
- Challenges in Binary Analysis

Malware detection

In Practice

What is Malware?

"Software that **steals** your data.
Software that **destroys** your data.
Software that **abuses** your machine."

@Pinkflawd

Types of malware

- Ransomware
- Botnets
- Password Stealers
- Remote-Access-Trojans (RATs)
- Click-jackers (stealing ad clicks)
- Banking Trojans
- SCADA disruptors

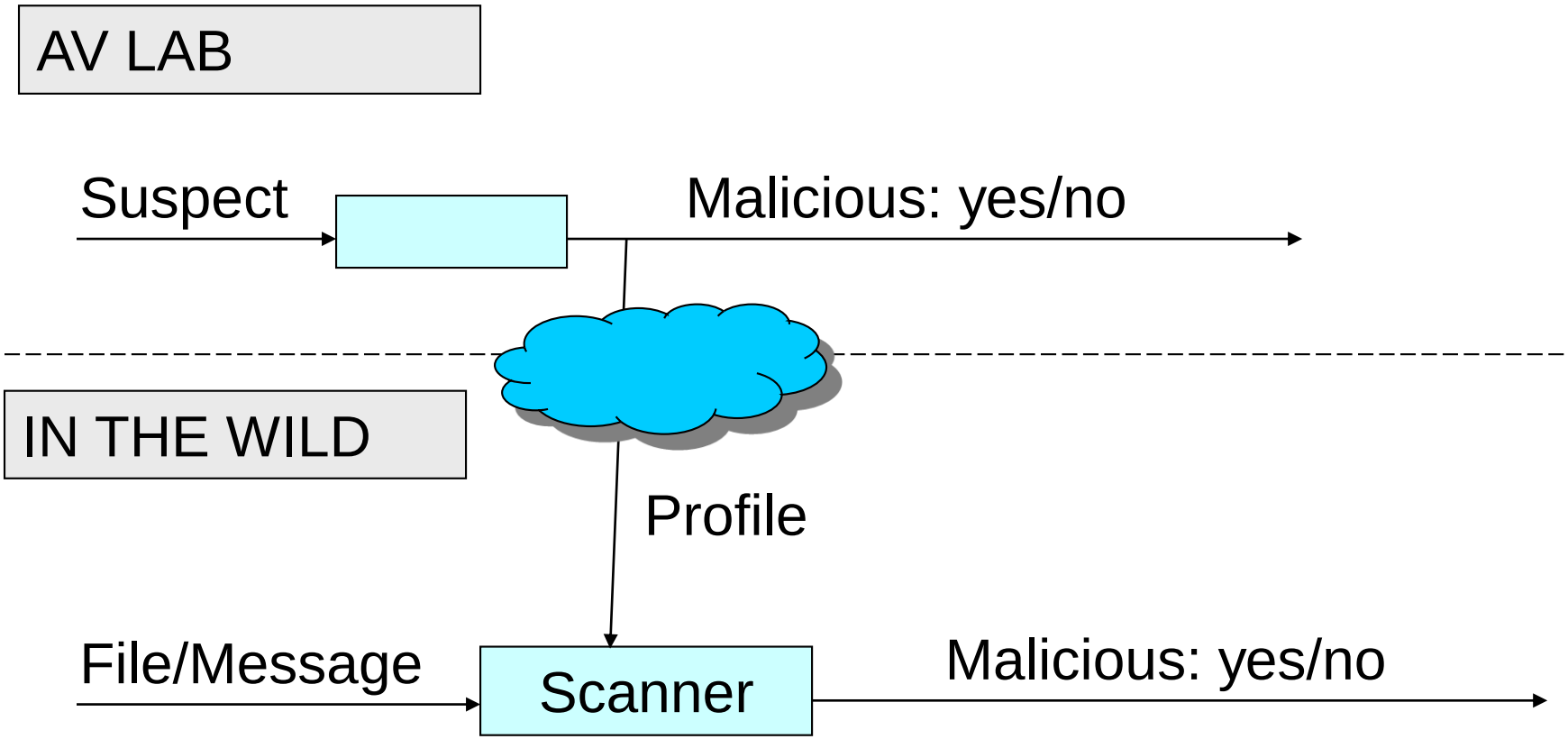
How to determine something is malware?

- Run it
- Observe if it
  - steals or destroys your data
  - abuses your machine

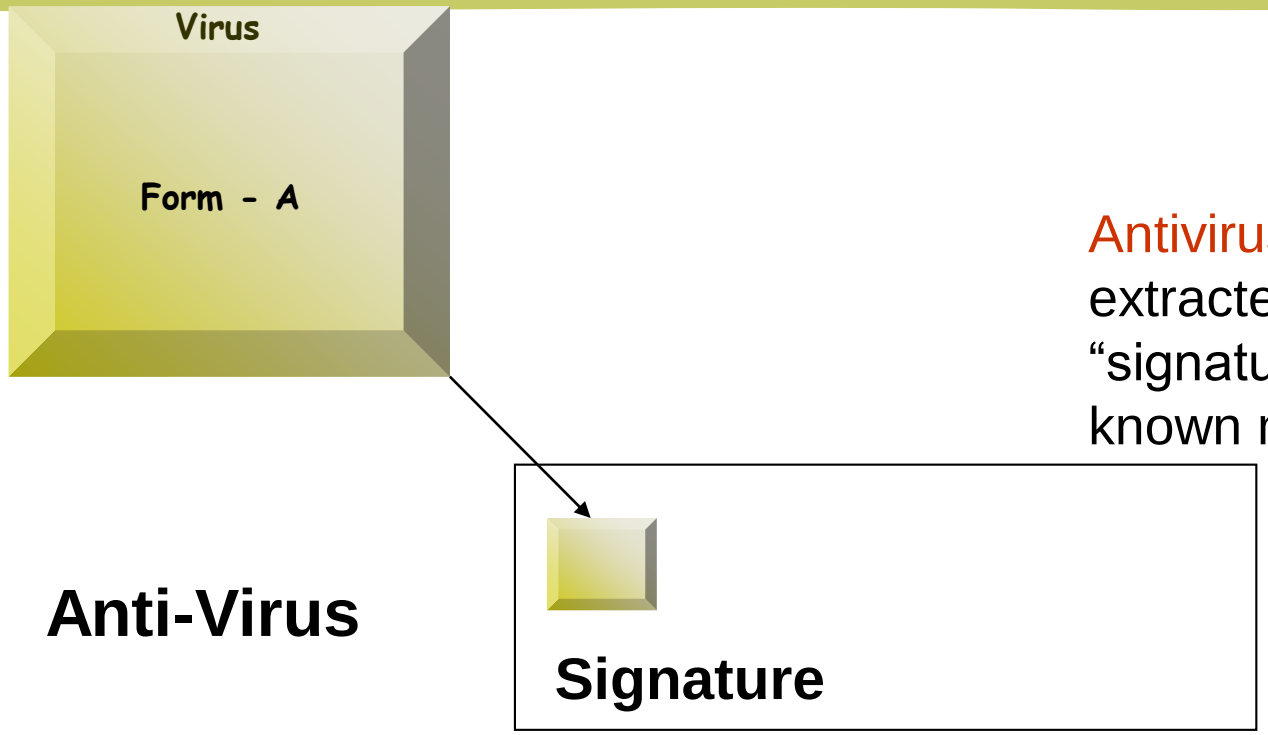
Determining malware in practice

- Individually testing each program on every machine for maliciousness is not feasible
- In reality:
  - Someone observes some unexpected activity
  - Traces activity to a program
  - Passes it on to a security expert
  - Expert analyzes to confirm
  - Creates a 'profile' of the program
  - Uses the 'profile' to detect other occurrences of the malware

Malware Detection Process (Theory)



Virus (Malware) Identification



Antivirus scanners use extracted patterns, or "signatures" to identify known malware.

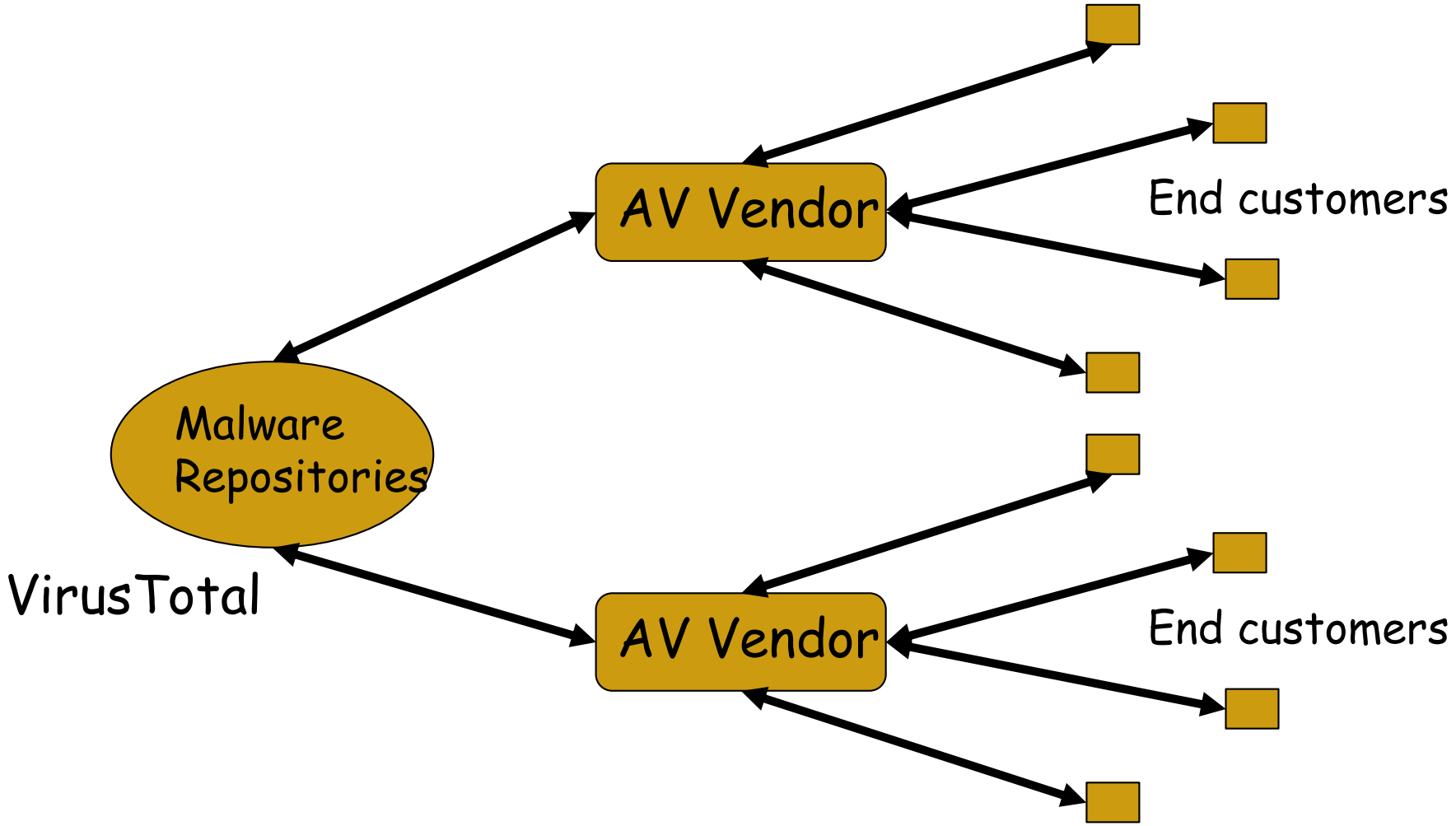
Static Signature

- Hex strings from virus variants
  - 67 33 74 20 73 38 6D 35 20 76 37 61
  - 67 36 74 20 73 32 6D 37 20 76 38 61
  - 67 39 74 20 73 37 6D 33 20 76 36 61
- Hex string for detecting virus
  - 67 ?? 74 20 73 ?? 6D ?? 20 76 ?? 61
  - ?? = wildcard

Dynamic Signature

- Monitor a running program to detect malicious behavior
- Examples
  - Analyze audit trails
  - Look at patterns of system calls
- Allows examination of only selected testcases

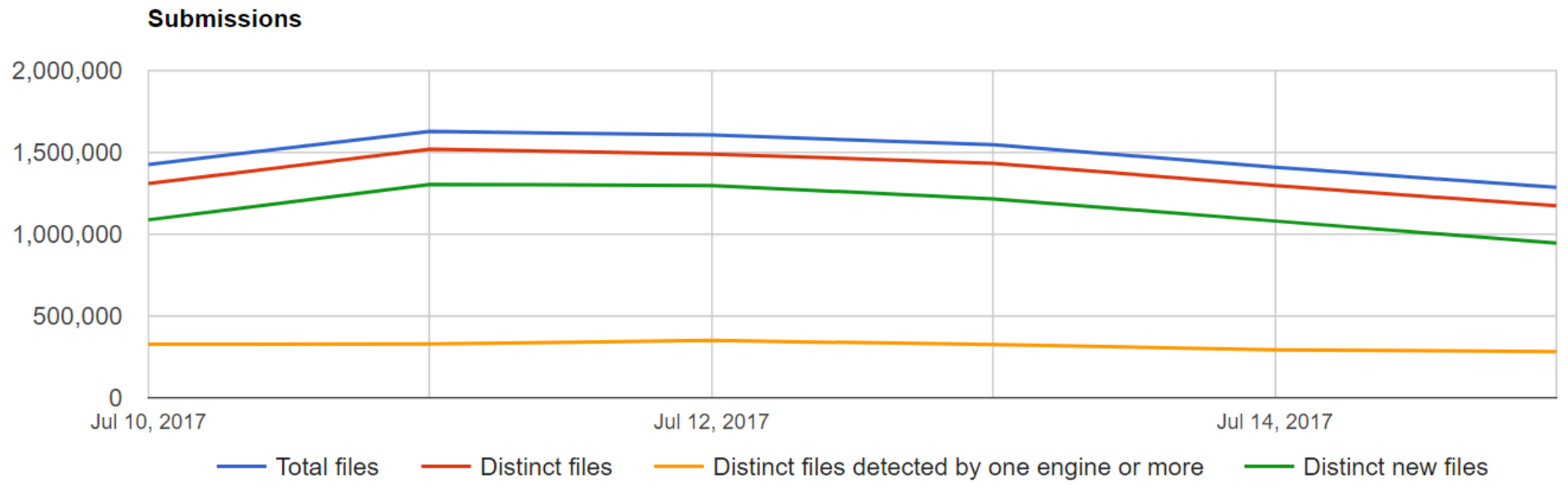
Malware detection ecosystem has a lot of sharing



Suspect files, daily volume

virustotal

File statistics during last 7 days



Multiple-Scanner Report

virustotal

SHA256: 6acb8b73cedb6c5a721c2cca33571e7cac44714bf8014dd666b5bdc2bc2c57b1

File name: LocalWeatherRadar.exe

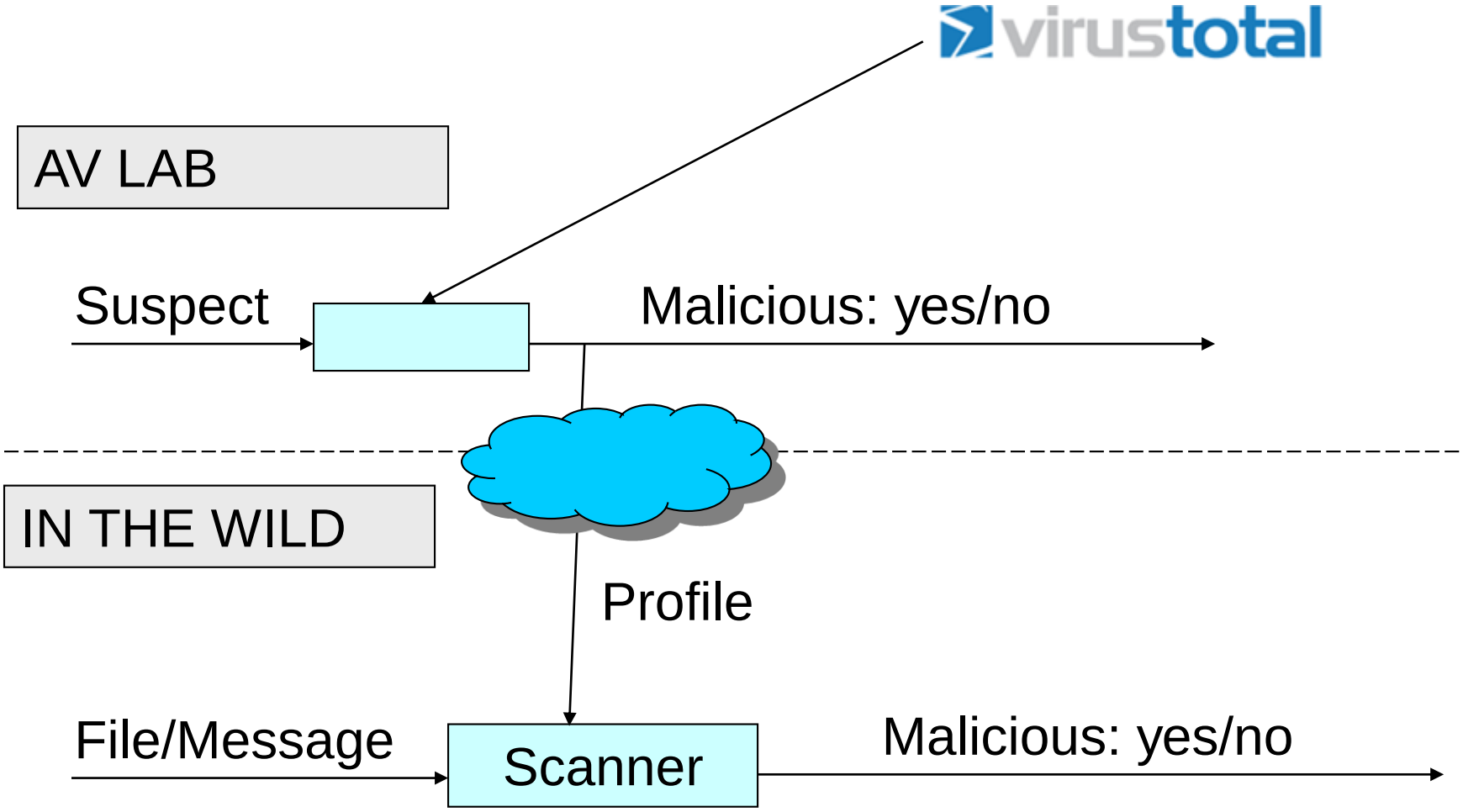
Detection ratio: 44 / 64

Analysis date: 2017-07-16 21:40:26 UTC (1 day, 12 hours ago)

Analysis | File detail | Additional information | Comments 1 | Votes | Behavioural information

Antivirus	Result	Update
Ad-Aware	Gen:Variant.Zusy.238549	20170716
AhnLab-V3	PUP/Win32.Helper.R200762	20170716
ALYac	Gen:Variant.Zusy.238549	20170716
Antiy-AVL	Trojan/Win32.StartPage	20170716
Arcabit	Trojan.Zusy.D3A3D5	20170716
Avast	Win32:Malware-gen	20170716
AVG	Win32:Malware-gen	20170716
Avira (no cloud)	ADWARE/Agent.lckrn	20170716
AVware	Trojan.Win32.GenericIBT	20170716
Baidu	Win32.Adware.Agent.t	20170714
BitDefender	Gen:Variant.Zusy.238549	20170716
Bkav	W32.HfsAdware.58CA	20170716
CAT-QuickHeal	Trojan.Dynamer.S709405	20170715
Comodo	Application.Win32.Widgi.B	20170716

Malware Detection Process (Practice)



Malware Definition: In practice

- X is a malware:
  - if it creates a huge hue and cry
  - if P out of S AV scanners (on VT) say it is malware
  - if some customer report it as suspect and a security analyst confirms

How to perform Community Voting

- Use Hash(X) instead of X.
- Hash(X) is malware if:
  - if P out of S AV scanners (on VT) say it is malware

Community Voting is very rigid.

Cannot check for unseen malware.

Other challenges related to Malware

- Determine the objective of a malware
- Determine the actors/creators
- Disrupt botnets

BINARY ANALYSIS

Learn about you

- Binary Analysis:
  - Level of knowledge: Level 1-5 (low-high)
  - How much do you care? Level 1-5

Binary Analysis – Why?

- Debugging and Patching
- Legacy Migration
- Software Protection
  - Protecting IP
- Software Cracking
- Malicious Detection
  - Binary with undesired/unknown behavior

Binary Analysis Tools

STATIC

- Hex editor
- PE/ELF editors
- Disassembler
- Decompiler
- Data/control flow
- Abstract interpreter
- Specialized checkers
  - Buffer overflow
  - Theorem provers

DYNAMIC

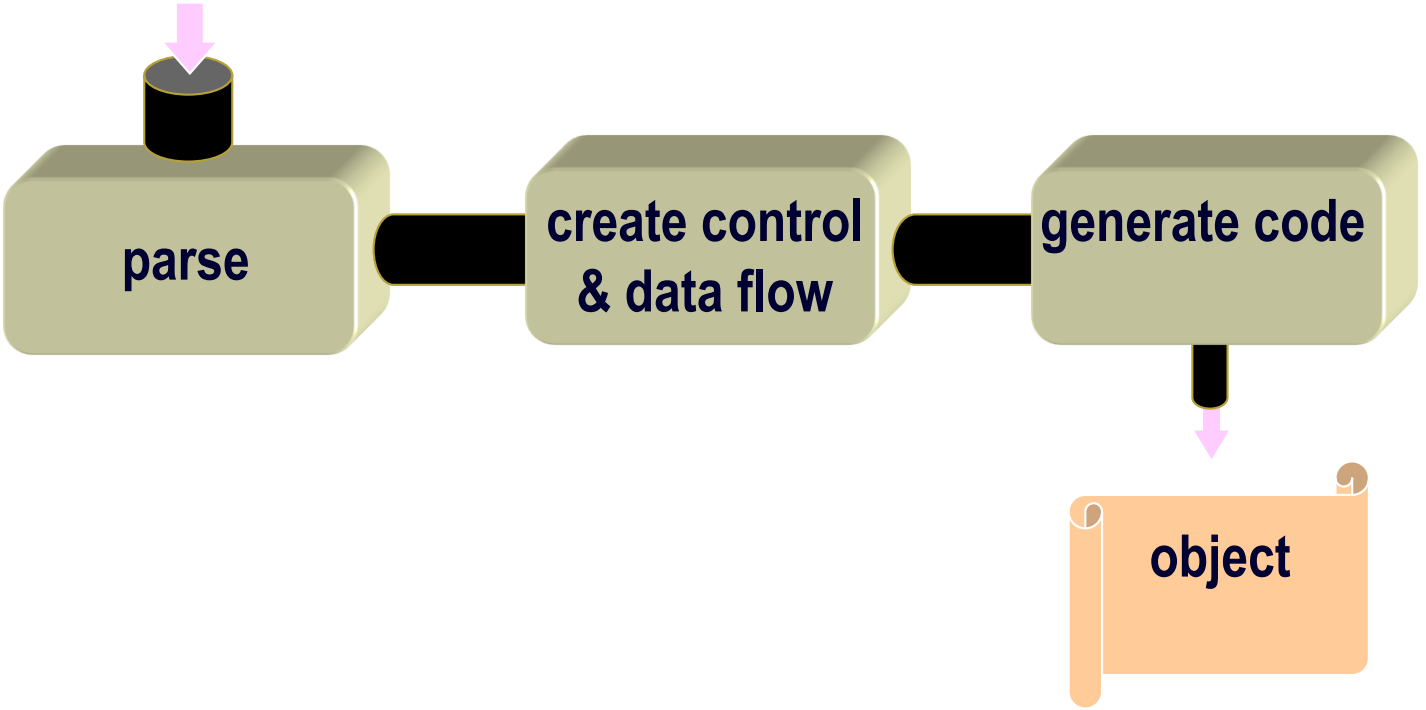
- Debugger
- Emulator
- Run-time monitors
- Network monitors
- Fuzzers

- MIXED – CONCOLIC
  - Combination of dynamic and static

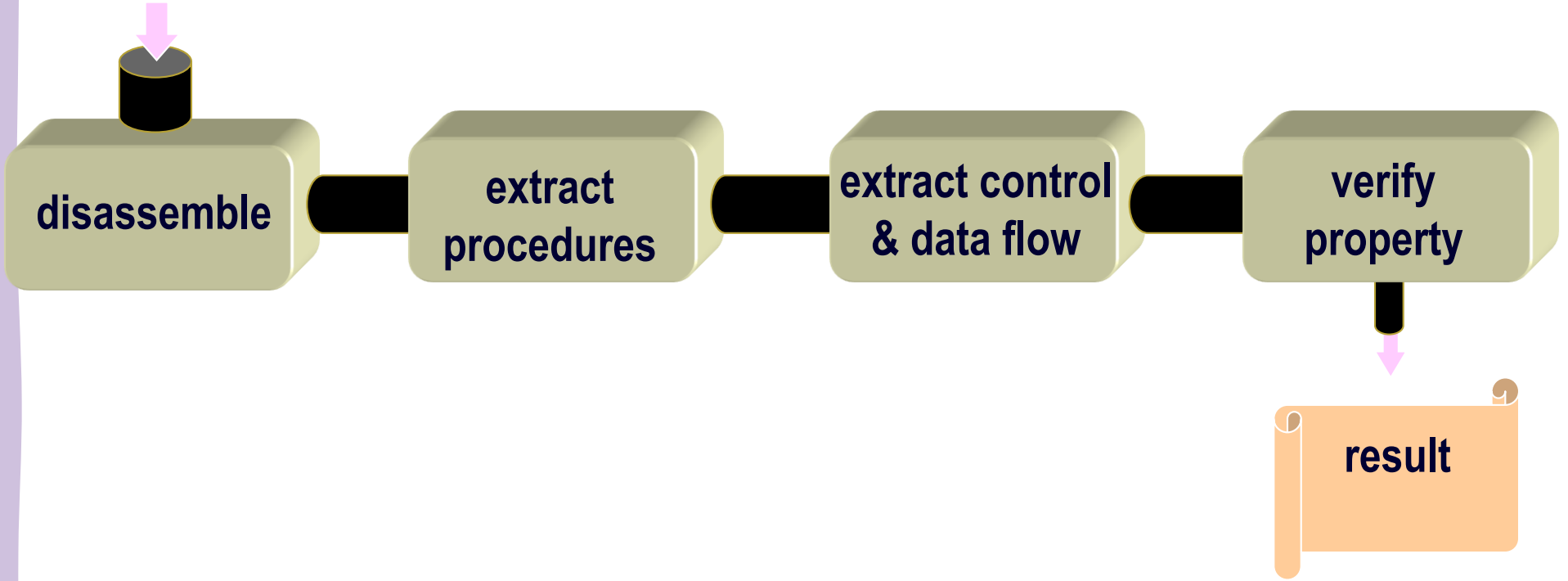
History of analysis tools

- 50+ years of program analysis (PA)
  - compilers, security analysis, …
- 25+ for reverse engineering (RE)
  - design recovery, reengineering, evolution, …
- Fundamental theories, algorithms, methods
  - program decomposition, abstraction
  - disassembly, flow graphs
  - liveness, dependence, dominance, …
  - clustering, abstraction, visualization, comparison

Compiler processing



Binary analysis, adapted from source



Decomposing binaries

High Level Program

```
main() {
   Max(0xA, 0xB);
   Max(0xC, 0xD);
}

Max(int x, int y) {
    if (x > y) return 1;
    return 0;
}
```

Partition into procedures?

Disassembled Binary

```
L01:  PUSH   0xA
L02:  PUSH   0xB
L03:  CALL   L08
L04:  PUSH   0xC
L05:  PUSH   0xD
L06:  CALL   L08
L07:  RET
L08:  MOV    eax, [esp+4]
L09:  MOV    ebx, [esp+8]
L10:  CMP    eax, ebx
L12:  JG             L14
L13:  MOV    eax, 0
L14:  RET
L15:  MOV    eax, 1
L16:  RET
```

Procedures are encapsulated

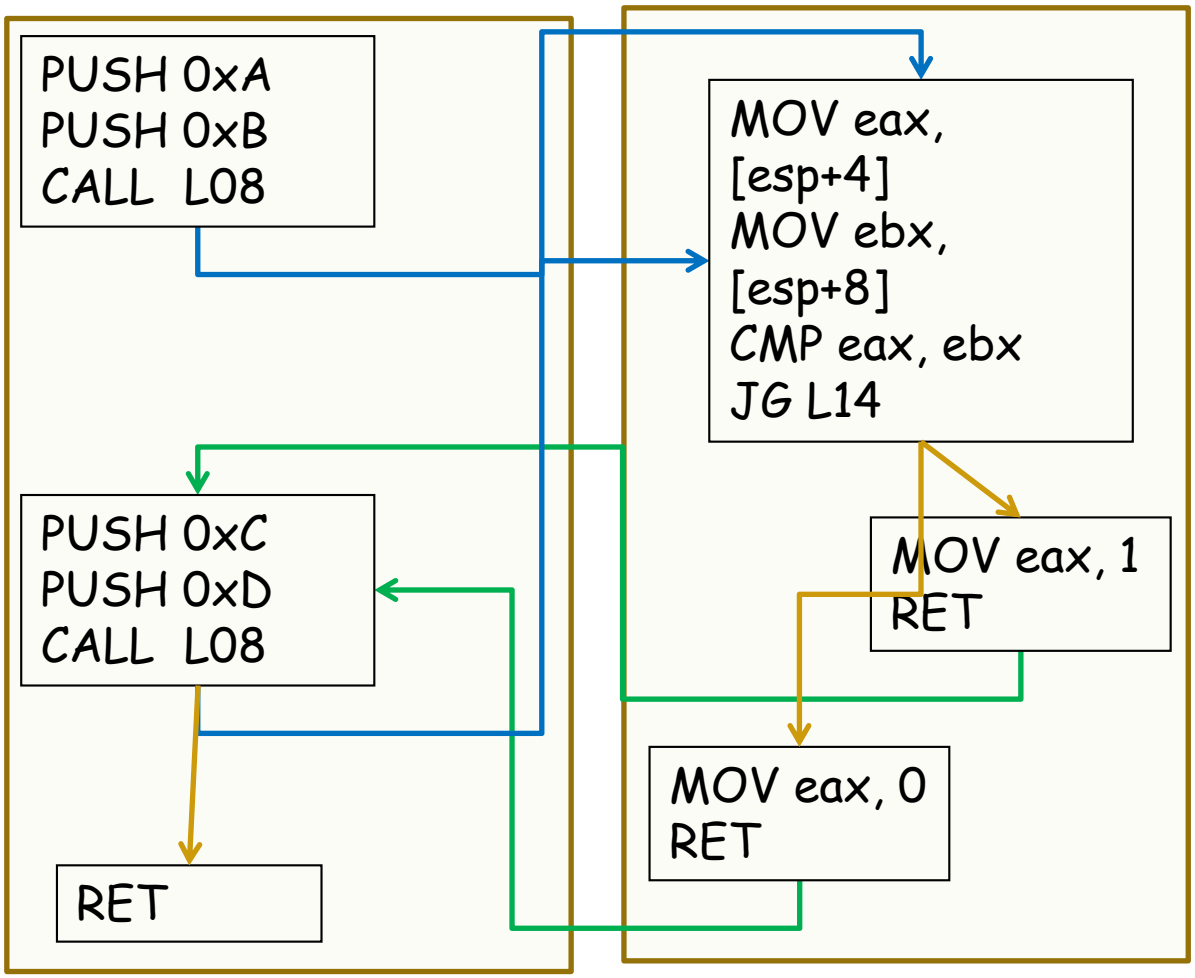
No syntactic boundary for procedures

Analysis of Binary

Disassembled Program

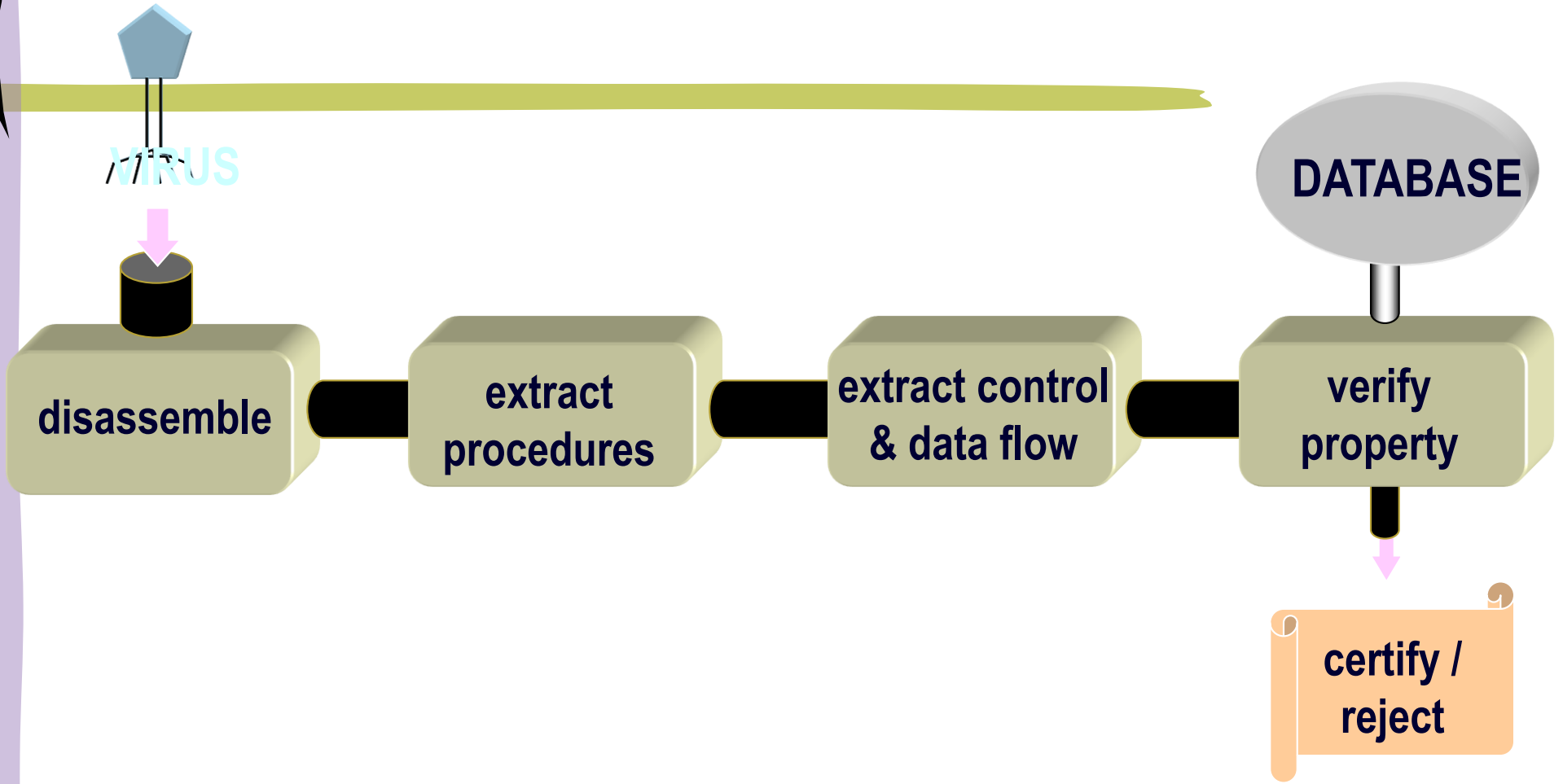
```
L01:  PUSH   0xA
L02:  PUSH   0xB
L03:  CALL   L08
L04:  PUSH   0xC
L05:  PUSH   0xD
L06:  CALL   L08
L07:  RET
L08:  MOV    eax, [esp+4]
L09:  MOV    ebx, [esp+8]
L10:  CMP    eax, ebx
L12:  JG              L14
L13:  MOV    eax, 0
L14:  RET
L15:  MOV    eax, 1
L16:  RET
```

Interprocedural CFG

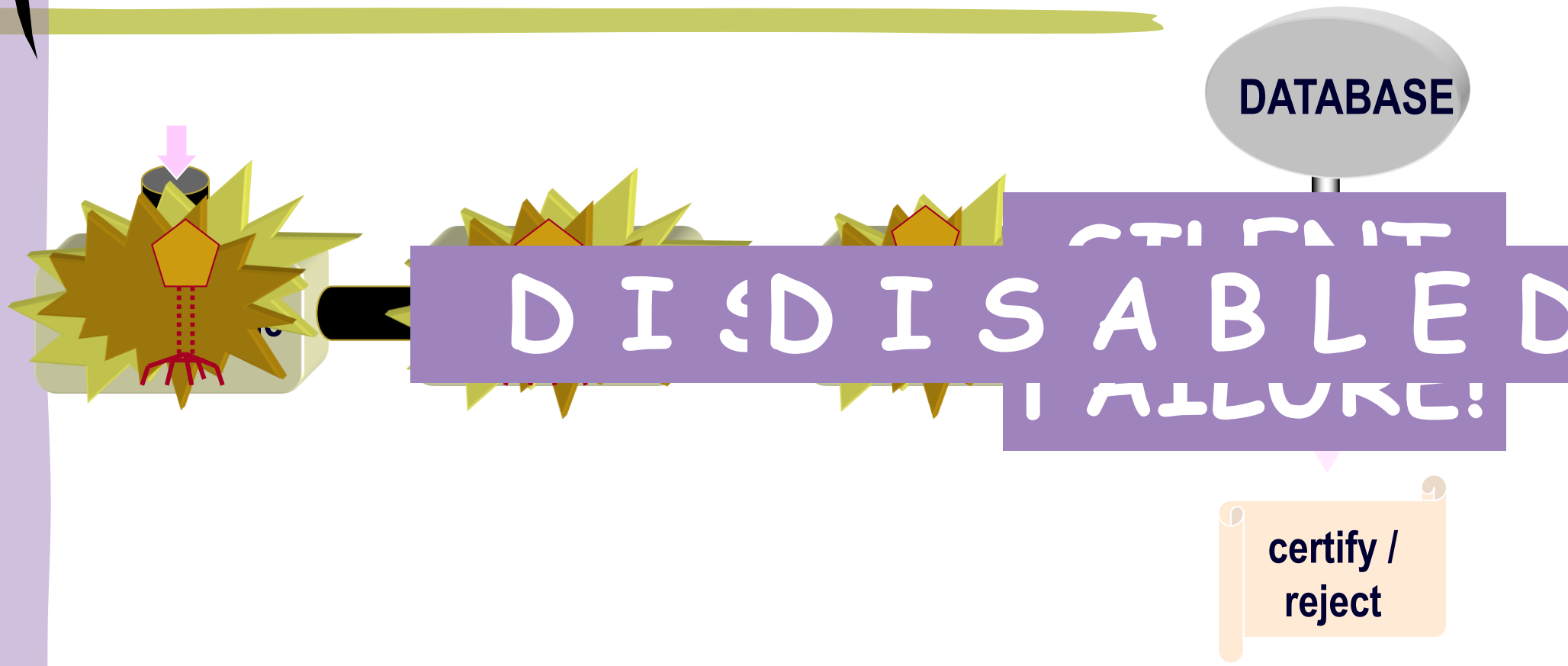


Binary Analysis - Challenges

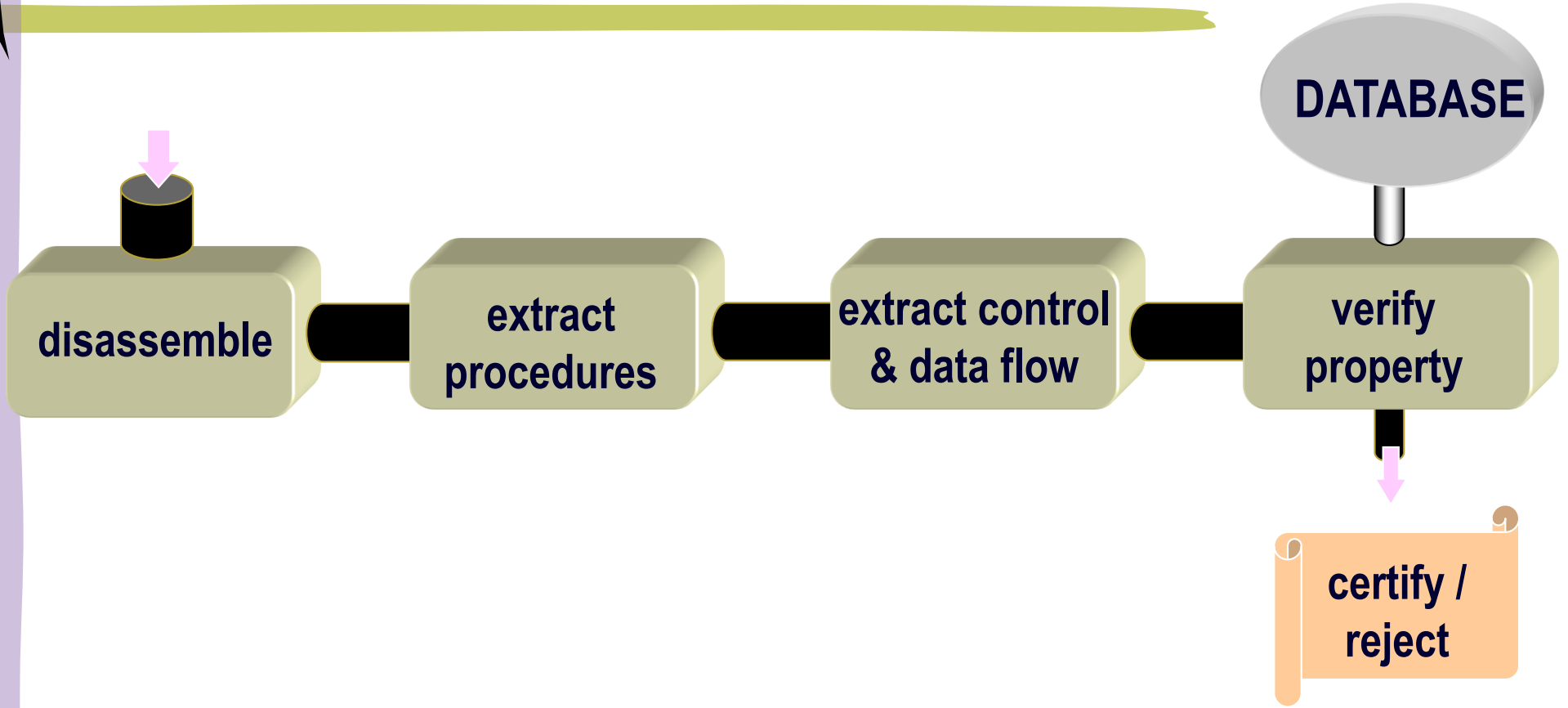
Typical analysis pipelines



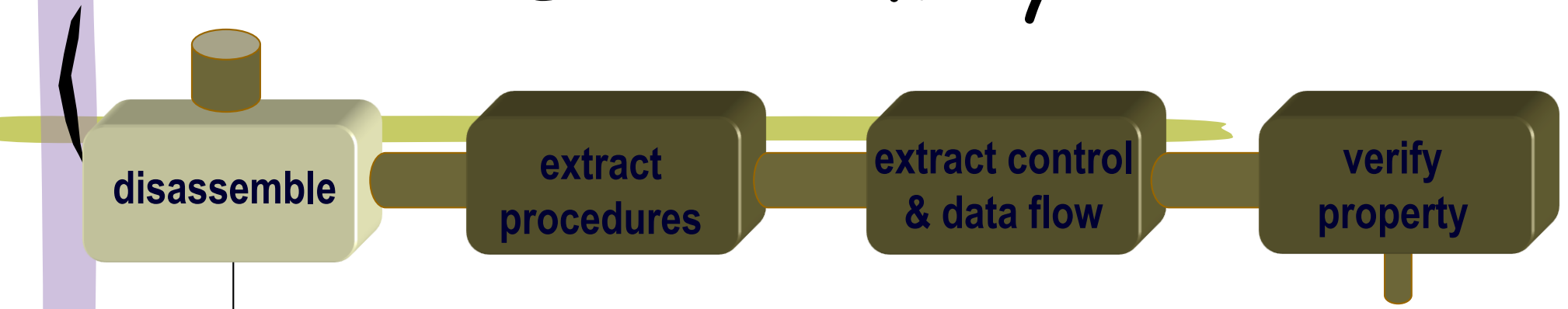
Problem: Not hardened



Typical analysis pipeline



Attack: Disassembly



decode machine instructions (byte seq)

	ORIG BYTES	ASSEMBLY
401063:	5d	pop %ebp
401064:	c3	ret
401065:	55	push %ebp
401066:	89 e5	mov %esp,%ebp
401068:	83 ec 08	sub $0x8,%esp
40106b:	eb 05	jmp 0x401072
40106d:	e8 ee ff ff ff e8	mov 0x4010660fff,%esi
401073:	e9 e9 ff ff ff 77	jmp 0x4010677
401078:	45 fc 00 00 00 00	inc %ebp,0xfffffffc(%ebp)
40107e:	81 7d fc e7 03 00 00	cmpl $0x3e7,0xfffffffc(%ebp)

bad disassembly (no jump target)

jump over junk malicious func

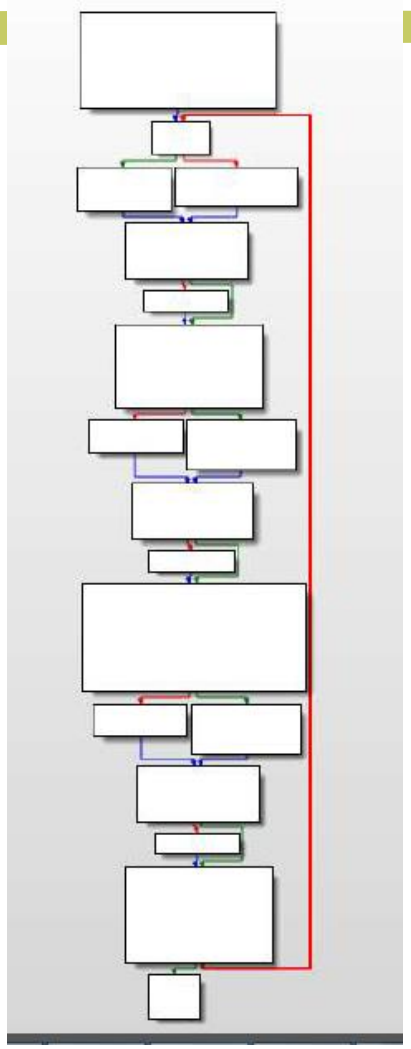
M/o/vfusctor (by Chris Domas)

```
4004e9:  mov    DWORD PTR [rbp-0x8],0x0
4004f2:  push   600004
4004f8:  call   printf
4004fa:  pop    eax
4004fc:  add    DWORD PTR [rbp-0x8],0x1
400500:  cmp    DWORD PTR [rbp-0x8],0x100
400507:  jle    4004f2 <main+0xb>
```

```
80515bc:  mov    eax,ds:0x835d81a
80515c1:  mov    ebx,DWORD PTR [eax+0x835d6fc]
80515c7:  mov    edx,DWORD PTR ds:0x835d7da
80515cd:  mov    eax,0x0
80515d2:  mov    al,BYTE PTR [ebx+edx*1]
80515d5:  mov    al,BYTE PTR [eax+0x835dc7e]
80515db:  mov    BYTE PTR [ebx+edx*1],al
80515de:  mov    eax,ds:0x835d81a
80515e3:  mov    ebx,DWORD PTR [eax+0x835d6fc]
80515e9:  mov    edx,DWORD PTR ds:0x835d7da
80515ef:  mov    eax,0x0
80515f4:  mov    al,BYTE PTR [ebx+edx*1]
```

```
nov [dword 0x80a0451],edx
nov eax,0x0
nov ax,[0x80a0451]
nov byte [eax+0x80e17bc],0x0
nov al,[eax+0x80e17bc]
nov [0x80a0451],al
nov eax,[0x80a0556]
nov edx,[eax+0x80a058e]
nov eax,[0x80a0451]
nov eax,[eax+edx]
nov [0x80a044d],eax
nov eax,[0x80a044d]
nov eax,[eax+0x80a054e]
nov dword [eax],0x139
nov eax,[0x80a044d]
nov eax,[eax+0x80a055e]
nov dword [eax],0x0
nov eax,[0x80a044d]
nov eax,[eax+0x80a056e]
nov dword [eax],0x4
nov eax,[0x80a0556]
nov eax,[eax+0x80a05a6]
nov [0x80a0451],eax
nov eax,0x0
nov ax,[0x80a0546]
nov byte [eax+0x80e17bc],0x0
nov al,[eax+0x80e17bc]
nov [0x80a044d],al
nov eax,[0x80a044d]
nov edx,[eax+0x80a058e]
nov eax,[0x80a0451]
nov eax,[eax+edx]
nov [0x80a044d],eax
nov eax,[0x80a0566]
nov eax,[eax+0x80a05a6]
nov [0x80a0451],eax
nov eax,[0x80a044d]
nov edx,[eax+0x80a058e]

mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]

mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0

mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx

mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a051e]
mov eax,[ebx]
mov edx,0x0
mov dx,[eax+eax+0x80c0bba]
mov [ebx],edx
mov eax,[0x80a0556]
mov ebx,[eax+0x80a0438]
mov edx,[dword 0x80a0516]
mov eax,0x0
mov al,[ebx+edx]
mov al,[eax+0x80a09ba]
mov edx,[eax+0x80a058e]
mov eax,[0x80a0451]
```

Attack: Defeat CFG Construction



Transform code to data

```
unsigned char and[2][2]={ { 0, 0 }, {0, 1} };
unsigned char or[2][2]={ { 0, 1 }, {1, 1} };
unsigned char not[2]={ 1, 0 };
```

```
and[1][0]
or[0][1]
not[1]
```

Defeat signatures: Packer, with encryption

Packer - Limitation

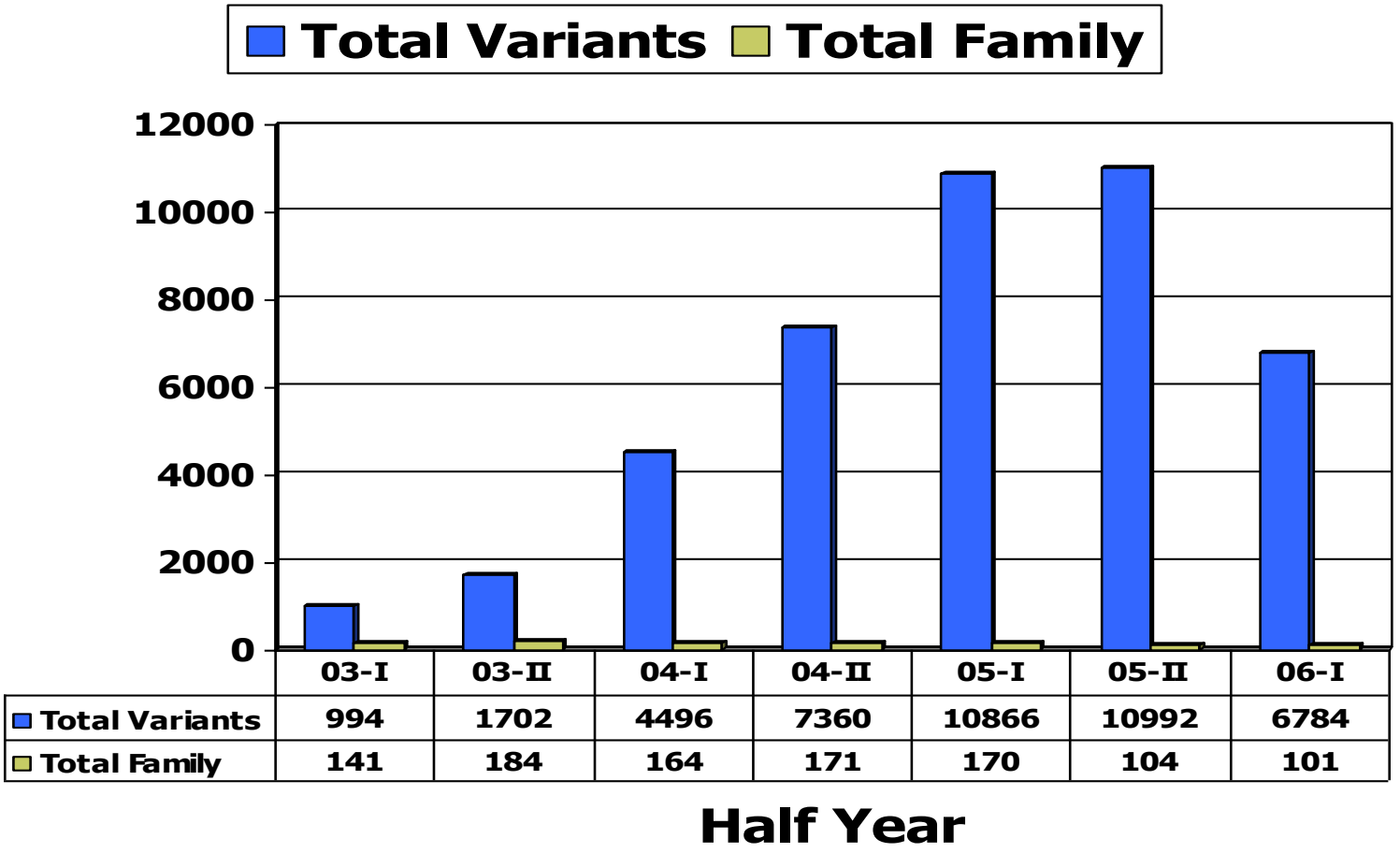
- Original code in clear text at some point

Protectors – Virtual Machine

- Slip a VM under the program

Variants vs Family

Source: Symantec Corp 2006



	03-I	03-II	04-I	04-II	05-I	05-II	06-I
Total Variants	994	1702	4496	7360	10866	10992	6784
Total Family	141	184	164	171	170	104	101

Theoretical Challenge: Undecidability

- Waiting for page to load
  - Do you hit 'reload' or do you wait?
- Halting Problem
  - Write a program that answers:
    - Will the program P halt for any input?
  - No program can correctly answer this question for all programs
- Virus (malware) detection problem
  - Write a program that answers:
    - Is program P a virus?
  - Problem is undecidable (Cohen 1984)

Implications of Undecidability

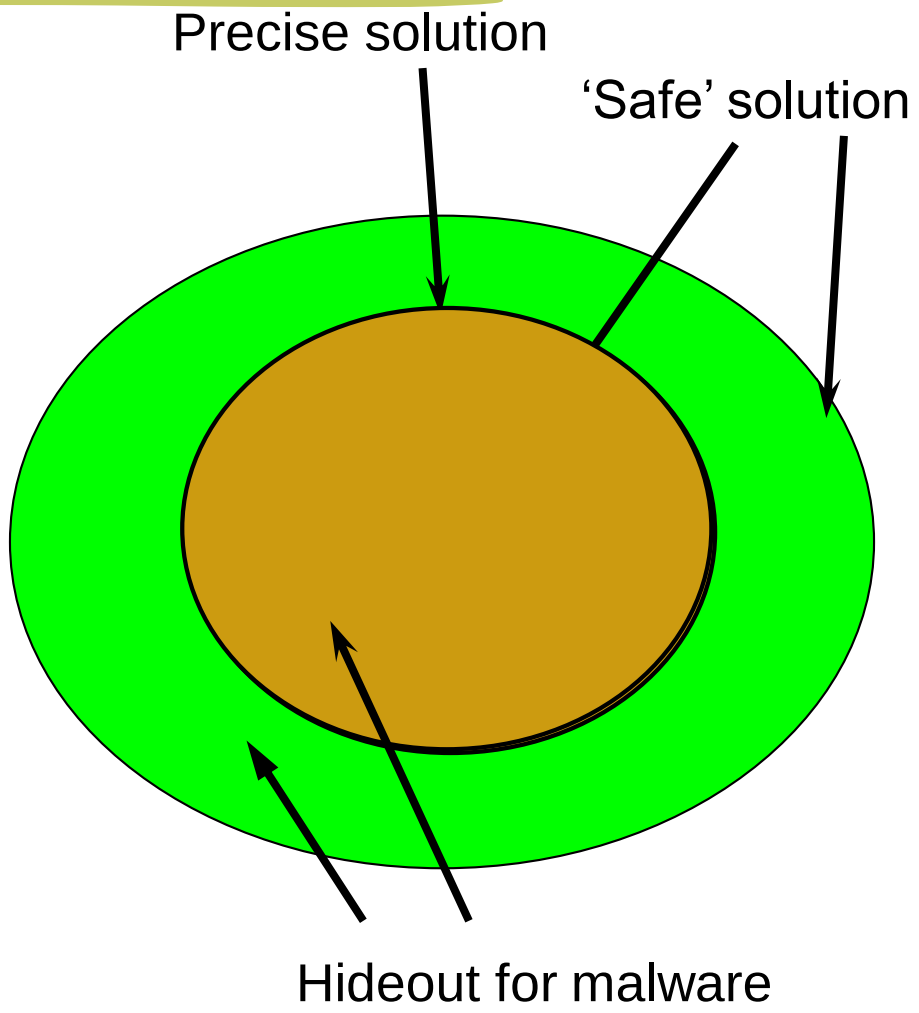
Analysis problems are undecidable

Precise solutions cannot be computed

Solutions are approximated

Play ‘safe’: over approximate or under approximate

Catch: ‘Safe’ solutions leave hideouts for malware



Obfuscation also has limits

- Obfuscation increases:
  - Code size
  - Runtime
- Cannot be applied ad-infinitum

- Research challenge:
  - How to take advantage of limits of obfuscation?

Summary

- Malware Detection Ecosystem
- Binary Analysis – Areas and Issues
- Binary Analysis Challenges
- Anti-AV Techniques
  - Transform, Hide