Motivation

- Slammer, MSBlast, CodeRed, Nimda all exploiting *known!* Vulnerabilities whose patches are released *months!* before

- Software patching has *not* been an effective first line worm defense
Why don’t people patch?

- **Disruption:**
  - Service or machine reboot
- **Unreliability**
  - Software patching inherently hard to test
- **Irreversibility**
  - Most patches are not designed to be easily reversible
- **Accident**
  - Unaware of patch releases

Our Vision:
Shielding Before Patching

- Shield addresses the window between vulnerability disclosure and patch application.
- Shields: *vulnerability-specific, exploit-generic* network filters. Currently focus on end-host based shields.
- Patch is the ultimate fix of the vulnerability
  - Shield is removed upon patch application
Overview of Shield Usage

New Shield Policy

Incoming or Outgoing Network Traffic

Shield Vulnerability Signature (Per Vulnerability)

Shielded Traffic to Processes or Remote Hosts

• Shield lies above the transport layer.

Why apply shields instead?

• **Non-intrusive**
  - No service or machine reboot

• **Easy testability -- Reliable**
  - Configuration independent, unlike patches – much fewer number of test cases
  - Simple testing through large trace replay or existing test suites for the protocol in question


Outline

- Motivation and overview
- Vulnerability Modeling
- Shield Architecture
- Shield Language
- Analysis
- Shield prototype implementations
- Initial evaluations
- Related Work
- Concluding Remarks

Vulnerability Modeling

Shield Vulnerability Signature:
Specifies vulnerability state machine and describes how to recognize exploits in the vulnerable event
Shield Architecture: Goals

- Minimize and limit the amount of state maintained by Shield
- Enough flexibility to support any application level protocols
- Defensive design

Flexibility:
Separate Policy from Mechanism

- **Shield Mechanisms**: generic elements all application level protocols
  - All use finite state automaton for protocol operations
  - Event identification and session dispatching
  - Out-of-order datagram handling
  - Application level fragmentation handling
- **Shield Policies**: varying aspects of individual application level protocols
  - Application identification, event identification, session identification, vulnerability state machine specifications
Shield Architecture: 
Essential Data Structures

1. Per-app vulnerability state machine spec (Spec): 
   - Transformed from Shield policy 
   - Instructions for emulating vulnerability state machines in Shield at the runtime: 
     • Application identification: ports, dynamic port registration 
     • Vulnerability signature + reactions: states, events, handlers for recognizing and reacting to potential exploits 
     • Event and session identification: 
       - Location (offset, size) vector of event type and session ID in the app message. Unit: byte or "WORD" for text-based protocols 
     • Message boundaries, e.g., CRLF CRLF for HTTP and SMTP 
   - One state machine per application 
     • Multiple vulnerability state machines are merged into one 

2. Session State: current state and session context for exploit-checking
Scattered Arrivals of an Application Message

- An application message is the smallest interpretable unit by the application
- Why scattered arrivals?
  - Congestion control or application-specific message handling
- Copying: save then pass on
- What to save (parsing state): the name of the current incomplete field, the value of the current incomplete field only if the value is needed by Shield later
  - Per application message
- How to differentiate parsing state belonging to multiple sessions:
  - Safe to use socket here because only one socket should be used for delivering a complete application level message despite the M-M relationship between sockets and sessions.
- Pre-session copying: before the session info arrives
  - The parsing state is associated with the socket only
- In-session copying: after the session info arrives
  - The parsing state becomes part of the session state

Out-of-Order Application Datagrams

- Save out-of-order datagrams
- What is the max? Same as the application
- Additional info needed in Shield policy: seq num location, max number of saved datagrams
Application Level Fragmentation

- Over TCP: same treatment as scattered arrivals of a single application level message
- Over UDP: ordered copies of the fragments are treated the same as scattered arrivals
- Additional information needed in Shield policy: frag ID location

Outline

- Motivation and overview
- Vulnerability Modeling
- Shield Architecture
  - Shield Language
  - Analysis
  - Shield prototype implementations
  - Initial evaluations
  - Related Work
  - Concluding Remarks
Shield Policy Language

**Part 1: Vulnerability state machine specification and generic application level protocol info such as ports used, the locations of the event type, session ID, message boundary, etc.**

**Part 2: Handler and payload parsing specifications for run-time interpretation**

- **Handler specification:**
  - Variable types: BOOL, COUNTER, BYTES, WORDS
  - Two scopes: local or session
  - Statements: assignment, IF, special-purpose FOR-loop

- **Payload specification:**
  - Skippable fields of BYTES, WORDS, BOOL, or arrays of PAYLOAD_STRUCTs

**Coping with scattered arrivals:**

- handler continuation – part of the session state consisting of statement ID queue, parsing state
- Stream-based built-in length functions or regular expression functions: e.g., “COUNTER c = MSG_LEN(legalLimit);” c = legalLimit + 1 if msg exceeds “stopCount” number of bytes
Outline

- Motivation and overview
- Vulnerability Modeling
- Shield Architecture
- Shield Language
  - Analysis
  - Shield prototype implementations
  - Initial evaluations
  - Related Work
  - Concluding Remarks

Analysis: Scalability

- Scalability with Number of Vulnerabilities
  - # of shields doesn’t grow indefinitely – upon successful patching, the corresponding shields are removed
  - N shields for N apps ⇔ 1 shield
  - Multiple vulnerabilities of a single app can compound if they share paths on the vulnerability state machine
    - not significant because no more than 3 worm-exploitable vulnerabilities seen in a single application in 2003
  - Application throughput is at worst halved, traffic processed once in Shield and once in the application
Analysis: False Positives

- Low false positives by nature
- Two sources:
  - Misunderstanding of protocol and payload spec – can be debugged with large traffic trace or test suites
  - Differential treatment of a certain network event: could be an exploit in one runtime setting, and yet completely legal in another

Shield Prototype Implementation

- 10,702 line C++ code;
- Experimented with 15 vulnerabilities and 7 application level protocols, such as RPC, HTTP, SMTP, FTP, SMB
Outline

✓ Motivation and overview
✓ Vulnerability Modeling
✓ Shield Architecture
✓ Shield Language
✓ Analysis
✓ Shield prototype implementations
  • Initial evaluations
  • Related Work
  • Concluding Remarks

Evaluation: Shield-ability

• What are hard to shield:
  – Virus
    • *vulnerability-driven* anti-virus software would be a better alternative
  – Vulnerabilities that could be embedded in HTML scripting
  – Application-specific encrypted traffic – may be hard to get the key.
    • But for SSL/TLS, an SSL-based shield framework can potentially be built on top of SSL
### Evaluation: Shield-ability, Cont.

<table>
<thead>
<tr>
<th># of vul.</th>
<th>Nature</th>
<th>Worm-able</th>
<th>Shield-able</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Local</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>24</td>
<td>Client</td>
<td>No</td>
<td>Hard</td>
</tr>
<tr>
<td>12</td>
<td>Server input validation</td>
<td>Yes</td>
<td>Easy</td>
</tr>
<tr>
<td>3</td>
<td>Cross-site scripting</td>
<td>No</td>
<td>Hard</td>
</tr>
<tr>
<td>3</td>
<td>Server DoS</td>
<td>No</td>
<td>Hard</td>
</tr>
</tbody>
</table>

Study of 49 vulnerabilities from MS Security bulletin board in 2003

### Evaluation: Throughput

- Clients and a server use RPC/TCP. Server sends 100 MB of data back to initiating clients. Every byte is accessed by Shield on the server.
- Both have P4 2.8GHz and 512 MB of RAM, connected by 100Mbps Ethernet switch.
Evaluation: Throughput

<table>
<thead>
<tr>
<th># of clients</th>
<th>w/o Shield (Mbps)</th>
<th>w/ Shield (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>86.51</td>
<td>86.20</td>
</tr>
<tr>
<td>15</td>
<td>86.57</td>
<td>86.36</td>
</tr>
<tr>
<td>50</td>
<td>86.66</td>
<td>86.20</td>
</tr>
<tr>
<td>100</td>
<td>86.48</td>
<td>85.86</td>
</tr>
<tr>
<td>150</td>
<td>86.67</td>
<td>86.24</td>
</tr>
<tr>
<td>200</td>
<td>86.06</td>
<td>81.70</td>
</tr>
<tr>
<td>500</td>
<td>84.27</td>
<td>82.29</td>
</tr>
<tr>
<td>1000</td>
<td>66.29</td>
<td>57.56</td>
</tr>
</tbody>
</table>

Evaluation: False Positives

- Evaluate on shield for Slammer.
- Used an SSRP stress test suite obtained from a MS test group: 32 test cases for 12 message types
- No false positives observed.
Related Work

• Threats of Internet worms:
  – Own Internet, CodeRed study, Inside Slammer,
    Internet quarantine, Warhol
• Insufficiency of patches:
  – Timing patching, CodeRed study,
• Firewall
  – More coarse-grained, high-false positive solution
  – Will be much improved by fast exploit-signature
    generation schemes such as “early bird”
• NIDS (such as Bro), traffic normalizers
  – Different layers and different purposes from Shield

Concluding Remarks

• Shield: vulnerability-specific, exploit
  generic network filters for preventing
  exploits against known vulnerabilities.
• Initial prototyping and evaluation results
  are encouraging
Ongoing Work

- Gaining experience and evolving our language and architecture design
- Shield policies more difficult to write, but can be potentially easy to automate the difficult part of it
- Shield at firewall or edge router.
- Shield testing
- Vulnerabilities easier to reverse-engineer with Shield – need secure, reliable and expeditious distribution
- Apply Shield principle to anti-virus – scalability a key challenge.