Automatic Network Protocol Analysis

Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel and Engin Kirda

{gilbert,pmilani}@seclab.tuwien.ac.at
chris@cs.ucsb.edu
engin.kirda@eurecom.fr
Reverse Engineering Network Protocols

- Find out what application-layer “language” is spoken by a server implementation
  - Message formats
  - Protocol state machine
- Slow manual process
- Do it automatically!
Reverse Engineering Network Protocols: Security Applications

- Black-box fuzzing
- Deep packet inspection
- Intrusion detection
- Reveal differences in server implementations
  - server fingerprinting
  - testing/auditing
Reverse Engineering Network Protocols: Sources of Information

- Network traces
  - limited information (no semantics)
- Server binaries
  - static analysis
  - dynamic analysis
Our approach

• Mostly dynamic analysis (+ static analysis)
• Use dynamic taint analysis to observe the data flow
• Observe how the program processes (parses) input messages
• Analyze individual messages
• Generalize to a message format for messages of a given type (i.e. HTTP get, NFS lookup..)
• Classification of messages into types is currently done manually
Dynamic taint analysis environment

Execution trace

Execution traces for individual messages

Tree of fields

Message format

alignment/generalization

Automatic Network Protocol Analysis
Dynamic Taint Analysis

- Run unmodified binary in a monitored environment (based on qemu, valgrind, ptrace..)
- Assign a unique label to each byte of network input
- Propagate the labels in shadow memory
  - for each instruction, assign labels of input to output destinations
  - also track address dependencies (example: lookup table-based toupper() function)
Label Input:

<table>
<thead>
<tr>
<th>G</th>
<th>E</th>
<th>T</th>
<th>/</th>
<th>H</th>
<th>T</th>
<th>T</th>
<th>P</th>
<th>/</th>
<th>1</th>
<th>0</th>
<th>\r</th>
<th>\n</th>
<th>\r</th>
<th>\n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

Propagate Labels:

EAX

<table>
<thead>
<tr>
<th>c</th>
<th>G</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

BL

<table>
<thead>
<tr>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

| push %esi |
| push %ebx |
| mov (%eax),%bl |
| sub $0x1,%ecx |

Tainted data affects program flow:

Is (something derived from) byte 0 equal to '\n'?

| cmp $0x0a,%bl |
| je 93 |
Message Format Analysis

• **Structure-forming semantics**
  – enough information to parse a message out of a network data flow
  – variation between messages

• **Additional semantics**
  – keywords, file names, session ids,..
Structure-Forming Semantics

- Length fields
  - and corresponding target fields, padding
- Delimiter fields
  - and corresponding scope fields
- Hierarchical structure
Detecting Length Fields (1/2)

- Length fields are used to control a loop over input data
- Leverage static analysis to detect loops
- Look for loops where an exit condition tests the same taint labels on every iteration
- Need at least 2 iterations
Detecting Length Fields (2/2)

• The tricky part is detecting the target field!
• Look at labels touched inside length loop
• Remove labels touched in all iterations
• May need to merge multiple loops (example: memcpy uses 4-byte mov instructions, but may need to move 1-3 bytes individually)
• Some bytes may be unused
Detecting Delimiters

• Delimiter is one or more bytes that separate a field or message
  – Observation: all bytes in the scope of the delimiter are compared against a part of the delimiter

• Delimiter field detection
  – Create a list of taint labels used for comparisons for each byte value, merge consecutive labels into intervals

• Intervals indicate delimiter scope,
  – nesting gives us a hierarchical structure
  – recursive analysis to “break up” message
Automatic Network Protocol Analysis

Initial Intervals

- "\r" [0,25]
- " " [0,23]
- "." [4,15]
- "/" [4,9]
Additional Semantics

- Protocol keywords
- File names
- Echoed fields (session id, cookie, ..)
- Pointers (to somewhere else in packet)
- Unused fields
Detecting Keywords

• A keyword is a sequence of (1 or 2 byte) characters which is tested against a constant value
  – adjacent characters being successfully compared to non tainted values are merged into a string
  – take delimiters into account
• Ideally, we would want to check it is being tested against values which are hard coded in binary
  – trace taint from entire binary
• Currently, we just check the string (of at least 3 bytes) is present in the binary
Generalization (1/3)

- Message alignment
- Based on Needleman-Wunsch
- Extended to a hierarchy of fields
Generalization (2/3)

- Needleman-Wunsch
- Dynamic programming algorithm for string alignment
- Computes alignment which minimizes edit distances
- Also provides edit path between the strings
- Scoring function (for match, mismatch, gap)
Generalization (3/3)

- Hierarchical Needleman-Wunsch
- Operate on a tree of fields, not on a string of bytes
- To align two inner nodes (complex fields) recursively call NW on the sequence of child nodes
- To align two leaf nodes, take into account field semantics
  - a length field only matches another length field
  - a keyword only matches same exact keyword
  - ...
- Simple scoring function: +1 for match, -1 for mismatch or gap
Generalization: More Semantics

• Sets of keywords (i.e. *keep-alive* OR *close*..)
• Length field semantics
  – encoding: endianess
  – compute target field length \( T \) from length \( L: T = A \times L + C \)
• Pointer field semantics
  – encoding: endianess
  – offset: relative or absolute
  – offset value is \( A \times L + C \)
• Repetitions
  – generalize \( a? a? \) to \( a^* \)
Evaluation

• 7 servers (apache, lighttpd, iacd, sendmail, bind, nfsd, samba)
• 6 protocols (http, irc, smtp, dns, nfs, smb)
• 14 message types (  
  – http get  
  – irc nick, user  
  – smtp mail, helo, quit,  
  – dns IPv4 A query  
  – rpc/nfs lookup, setattr, create, write  
  – smb/cifs negotiate protocol request, session setup andX request, tree connect andX request)
DNS A IPV4 query

Session ID
2 bytes

B000100000000

0000010001

Sequence

Length
1 byte

Target
A=1,C=0

B: any byte
T: any printable ascii byte
0001: constant byte values in hex
HTTP GET line

Scope "" (space)

GET  ""

Scope "."

Scope "/"

Filename

'S'

Sequence

'?'

Sequence

Delimiter

Keyword

T

T

T
Parsing

- The message format allows us to produce a parser
- Successfully parses real-world messages of the same type
  - all structural information was successfully recovered
- Rejects negative examples
  - different message types from the same protocol
  - hand-crafted negative examples
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Length</th>
<th>Target</th>
<th>Padding</th>
<th>Pointer</th>
<th>Delimiter</th>
<th>Keyword</th>
<th>File</th>
<th>Repetition</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4/5</td>
<td>6/6</td>
<td>1/1</td>
<td>1/2</td>
<td>12/14 (86%)</td>
</tr>
<tr>
<td>lighttpd</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4/5</td>
<td>7/7</td>
<td>1/1</td>
<td>1/2</td>
<td>13/15 (87%)</td>
</tr>
<tr>
<td>ircnick</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/1</td>
<td>1/1</td>
<td>0</td>
<td>0</td>
<td>2/2 (100%)</td>
</tr>
<tr>
<td>ircuser</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2/2</td>
<td>1/1</td>
<td>0</td>
<td>0</td>
<td>3/3 (100%)</td>
</tr>
<tr>
<td>smtpheho</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>1/1</td>
<td>0</td>
<td>0</td>
<td>2/3 (67%)</td>
</tr>
<tr>
<td>smtpquit</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/1</td>
<td>1/1</td>
<td>0</td>
<td>0</td>
<td>2/2 (100%)</td>
</tr>
<tr>
<td>smtpmail</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3/5</td>
<td>3/3</td>
<td>0</td>
<td>0</td>
<td>6/8 (75%)</td>
</tr>
<tr>
<td>dnsquery</td>
<td>1/1</td>
<td>1/1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/1</td>
<td>3/3 (100%)</td>
</tr>
<tr>
<td>nfslookup</td>
<td>4/5</td>
<td>4/4</td>
<td>2/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/1</td>
<td>0</td>
<td>11/11 (92%)</td>
</tr>
<tr>
<td>nfsgetattr</td>
<td>3/4</td>
<td>3/3</td>
<td>1/1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7/8 (88%)</td>
</tr>
<tr>
<td>nfscreate</td>
<td>4/5</td>
<td>4/4</td>
<td>2/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10/11 (91%)</td>
</tr>
<tr>
<td>nfswrite</td>
<td>4/6</td>
<td>4/4</td>
<td>2/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10/12 (83%)</td>
</tr>
<tr>
<td>smbnegotiate</td>
<td>2/2</td>
<td>2/2</td>
<td>1/1</td>
<td>0</td>
<td>1/1</td>
<td>10/10</td>
<td>0</td>
<td>0/1</td>
<td>16/17 (94%)</td>
</tr>
<tr>
<td>smbtree</td>
<td>2/3</td>
<td>2/2</td>
<td>0</td>
<td>1/1</td>
<td>2/2</td>
<td>3/3</td>
<td>0</td>
<td>0</td>
<td>10/11 (91%)</td>
</tr>
<tr>
<td>smbsession</td>
<td>8/9</td>
<td>8/8</td>
<td>0</td>
<td>7/7</td>
<td>2/2</td>
<td>2/2</td>
<td>0</td>
<td>0</td>
<td>27/28 (96%)</td>
</tr>
</tbody>
</table>

**Table 2. Field detection results: correctly identified fields / total fields in message format.**
Related Work

• Network traces
  – M. Beddoe. The Protocol Informatics Project. Toorcon 2004

• Static and dynamic analysis

• Dynamic taint analysis
Conclusions

• Reverse engineer application layer network protocols
• Recover a message format
• Validate format by parsing real world messages
• Tested on common servers and protocols
Questions?