Protocol Reverse Engineering:
Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution

Yangchun Fu

September 16th, 2011
Outline

1. Background

2. Automatic Protocol Format Reverse Engineering through Contex-Aware Monitored Execution
Outline

1. Background

What is protocol reverse engineering?

Protocol reverse engineering, the process of extracting the application-level protocol used by an implementation without access to the protocol specification.

- Extract message format
What is protocol reverse engineering?

Protocol reverse engineering, the process of extracting the application-level protocol used by an implementation without access to the protocol specification

- Extract message format
- Extract protocol state machine
What is protocol reverse engineering?

Protocol reverse engineering, the process of extracting the application-level protocol used by an implementation without access to the protocol specification

- Extract message format
- Extract protocol state machine

**BNF structure of HTTP request message**

Request = Request-Line
  *(( general-header
        | request-header
        | entity-header) CRLF) CRLF
  [ message-body]
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
Application

1. Network software testing
Application

1. Network software testing
2. Intrusion detection
Application

1. Network software testing
2. Intrusion detection
3. Fingerprint generation
Application

1. Network software testing
2. Intrusion detection
3. Fingerprint generation
4. Detecting services running on non-standard ports
Challenge

- No protocol specification: many closed application protocols
Challenge

- No protocol specification: many closed application protocols
- Automatic: Samba, ICQ took years
Challenge

- Multiple fields in a single message
Challenge

- Multiple fields in a single message
- Non-static size of fields
Challenge

- Multiple fields in a single message
- Non-static size of fields
- Complex relationships among protocol fields
Network traffic

Message between a client and server

... 0040 cd 46 47 45 54 20 2f 0050 20 48 54 50 2f 31 0060 41 67 65 74 3a 20 57 0070 2e 3a 20 57 4f 0080 2e 3a 20 4f 29 0090 2f 2a 0d 0a 48 6f 74 3a 20 31 32 39 2e 31 00a0 34 2e 38 38 2e 37 00b0 6f 6e 3a 20 4b 65 65 70 2d 00c0 0a 0d 0a
What we have?

Network traffic

Message between a client and server

... 0040 cd 46 47 45 54 20 2f 6e 65 77 73 2e 68 74 6d 6c .FGET /news.html
0050 20 48 54 50 2f 31 2e 30 0d 0a 55 73 65 72 2d HTTP/1.0..User-
0060 41 67 65 6e 74 3a 20 57 67 65 74 2f 31 2e 31 30 Agent: Wget/1.10
0070 2e 32 20 28 52 65 64 20 48 61 74 20 6d 6f 64 69 .2 (Red Hat modi
0080 66 69 65 64 29 0d 0a 41 63 65 70 3a 20 2a fied)..Accept: *
0090 2f 2a 0d 0a 48 6f 73 74 3a 20 31 32 39 2e 31 37 /*..Host: 129.17
00a0 34 2e 38 38 2e 37 31 0d 0a 43 6f 6e 6e 65 63 74 4.88.71..Connect
00b0 69 6f 6e 3a 20 4b 65 65 70 2d 41 6c 69 76 65 0d ion: Keep-Alive.
00c0 0a 0d 0a

Binary program

Application program that implement a particular protocol
**Network Trace**

- Extract protocol information purely from network traffic
- Use only syntactic information, no semantic limits the accuracy
- Can’t work for encrypted network traffic
Approach?

Network Trace
- Extract protocol information purely from network traffic
- Use only syntactic information, no semantic limits the accuracy
- Can’t work for encrypted network traffic

Binary analysis
- Leverage the ability of a program binary implement the protocol
1. Background

2. Automatic Protocol Format Reverse Engineering through Contex-Aware Monitored Execution
A protocol implementation is programmed to recognize the protocol format and usually contains protocol field-specific execution context, and we can actually leverage such context to infer the hierarchical structure of protocol fields, and even get their BNF structures.
Idea

Code snippet in http.c

```c
119 int read_header(int sid) {
    ...,
137   if (sscanf(line, "%^\[ ]%[^ ]\n[^ ]", conn[sid].dat->in_RequestMethod,
                conn[sid].dat->in_RequestURI, conn[sid].dat->in_Protocol)!=3)
    ...,
147   while (strlen(line)>0) {
        ...
154   if (strncasecmp(line, "Cookie: ", 8)==0)
155     strncpy(conn[sid].dat->in_Cookie, (char *)&line+8,
                sizeof(conn[sid].dat->in_Cookie)-1);
156   if (strncasecmp(line, "Host: ", 6)==0)
157     strncpy(conn[sid].dat->in_Host, (char *)&line+6,
                sizeof(conn[sid].dat->in_Host)-1);
    ...,
160   if (strncasecmp(line, "User-Agent: ", 12)==0)
161     strncpy(conn[sid].dat->in_UserAgent, (char *)&line+12,
                sizeof(conn[sid].dat->in_UserAgent)-1);
162   }
    ...
187 }
```
Goal

Identify the boundary of each field in the protocol message, but also structure the identified fields so that the overall message skeleton and the relations among message fields can be uncovered.
System design

- Mark received data and keep track of propagation

Credit: [Figure from Zhiqiang Lin’s paper, NDSS 2008]
System design

Mark received data and keep track of propagation
Log each byte of message
System design

- Mark received data and keep track of propagation
- Log each byte of message
- Offline protocol identify each field

Credit: [Figure from Zhiqiang Lin’s paper, NDSS 2008]
Context-Aware Execution Monitor

- Intercept network-related system calls, mark message received, annotate every message byte with its offset.
- Record the run-time call stack and the address of instruction that access a marked memory.
Log file structure

Each record save in the form of <o,c, s, l>

- o is offset of the reference memory in the entire message
- c is the content
- s is the run-time call stack
- l is the location of instruction
Context-Aware Execution Monitor

GET /news.html

Context-Aware Execution Monitor

log
Protocol Field Identifier

- Analyze the log file
- Build protocol field tree from the logged data
- Refine the tree using three heuristics
- Output the result
1. Build protocol field tree

<table>
<thead>
<tr>
<th>Offset</th>
<th>Content</th>
<th>Stack Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'G'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x4BA56A2-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_rgetline_core-&gt;ap_get_brigade-&gt;0x2D2CE-&gt;ap_get_brigade-&gt;0x2D667-&gt;apr_brigade_split_line-&gt;memchr</td>
</tr>
<tr>
<td>1</td>
<td>'E'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x4BA56A2-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_rgetline_core-&gt;ap_get_brigade-&gt;0x2D2CE-&gt;ap_get_brigade-&gt;0x2D667-&gt;apr_brigade_split_line-&gt;memchr</td>
</tr>
<tr>
<td>2</td>
<td>'T'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x4BA56A2-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_rgetline_core-&gt;ap_get_brigade-&gt;0x2D2CE-&gt;ap_get_brigade-&gt;0x2D667-&gt;apr_brigade_split_line-&gt;memchr</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>'\n'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x26187-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_rgetline_core</td>
</tr>
<tr>
<td>23</td>
<td>'\r'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x26322-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_rgetline_core</td>
</tr>
<tr>
<td>0</td>
<td>'G'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x1F7F3-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_getword_white</td>
</tr>
<tr>
<td>1</td>
<td>'E'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x1F7F3-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_getword_white</td>
</tr>
<tr>
<td>2</td>
<td>'T'</td>
<td>main-&gt;ap_mpm_run-&gt;0x15C57-&gt;0x15B38-&gt;0x15941-&gt;ap_process_connection 0x1F7F3-&gt;ap_run_process_connection-&gt;0xF5A8-&gt;ap_read_request-&gt;ap_getword_white</td>
</tr>
</tbody>
</table>

GET/news.html HTTP/1.0 \r\n \r \n GET
1. Build protocol field tree

```
GET /news.html HTTP/1.0
```

Diagram:
- Root node
  - GET
  - \r
  - \n
Refinement

Why?

- overly fine granularity
- certain field may be referenced multiple time
- some field may not be referenced at all
Refinement
2.1 Refinement (Tokenization)

Merging two neighbor node if possible
2.2 Refinement (Redundant node deletion)

parent node with one child can be remove

GET /news.html HTTP/1.0

GET /news.html HTTP/1.0

GET /news.html

GET /new.html

HTTP/1.0

\r

\n
HTTP/1.0
2.3 Refinement (Node Insertion)

insert new node with missing offset
Identify parallel field

Idea

Parallel fields share certain execution history
- Collect execution history for each node
- Child node of same parent with similar history are considered parallel field
Conclusion

limitation

- Dynamic trace dependency
- Byte granularity
- Do not correlate multiple messages
- Handle obfuscated binaries