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 Context-Aware Monitored Execution

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

<table>
<thead>
<tr>
<th>Method-Header</th>
<th>Request-Header</th>
<th>Entity-Header</th>
<th>Message-Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Host</td>
<td>Accept-Codes</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>Content-Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEAD</td>
<td>Accept-Encoding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Challenge

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

Application

- Network software testing
- Intrusion detection
- Fingerprint generation
- Detecting services running on non-standard ports
**Challenge**
- Multiple fields in a single message
- Non-static size of fields
- Complex relationships among protocol fields

**What we have?**
- Network traffic
  - Message between a client and server
  - Binary program
    - Application program that implement a particular protocol

**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

**Outline**
1. Background

**Idea**
A protocol implementation is programmed to recognize the protocol format and usually contains text/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.

**Code snippet in http.c**
```c
119 int read_header(int sid) {
...
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 }
```

Credit: [Figure from Zigzag's Link paper NDSS'08]
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
- Log each byte of message
- Offline protocol identify each field

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

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

2.1 Refinement (Tokenization)

2.2 Refinement (Redundant node deletion)

Why?
- overly fine granularity
- certain field may be referenced multiple times
- some field may not be referenced at all
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