

Reverse Engineering Binary Messages through Design Patterns

LangSec 2020

Jared Chandler
Tufts University

Kathleen Fisher
Tufts University



Automatic Reverse Engineering of Binary Messages

Who does this:

- Researchers
- Security Analysts
- Reverse Engineers

Why:

- Malware Communication Analysis
- Protocol Validation
- Old Gear with Lost Specification

Related Problems: Tags, Delimited Data, Long Distance Dependencies

Our Focus: Binary messages with *variable length*

Example of reverse engineering problem

Msg 1	2	3	67	65	84	4	66	73	82	68
Msg 2	1	5	77	79	85	83	69			
Msg 3	3	2	79	88	3	68	79	71	3	66

1. The analyst starts with messages.

Example of reverse engineering problem

Msg 1	2	3	67	65	84	4	66	73	82	68		
Msg 2	1	5	77	79	85	83	69					
Msg 3	3	2	79	88	3	68	79	71	3	66	85	71

1. The analyst starts with messages.
2. Infers some pattern in the data.



Msg 1	2	3	67	65	84	4	66	73	82	68		
Msg 2	1	5	77	79	85	83	69					
Msg 3	3	2	79	88	3	68	79	71	3	66	85	71

Example of reverse engineering problem

Msg 1	2	3	67	65	84	4	66	73	82	68		
Msg 2	1	5	77	79	85	83	69					
Msg 3	3	2	79	88	3	68	79	71	3	66	85	71

1. The analyst starts with messages.
2. Infers some pattern in the data.
3. Develops a hypothesis.

The diagram illustrates the mapping of ASCII values to letters. Above the messages, two pink boxes labeled '1' and '2' indicate specific bytes. A green arrow points from the '1' box to the second byte of Msg 1 (value 3). A yellow arrow points from the '2' box to the fourth byte of Msg 1 (value 4). Below the messages, the first four bytes are highlighted in pink and green respectively. The remaining bytes are shown in their original gray color.

Msg 1	2	3	67	65	84	4	66	73	82	68		
Msg 2	1	5	77	79	85	83	69					
Msg 3	3	2	79	88	3	68	79	71	3	66	85	71



67	65	84	
66	73	82	68
C	A	T	
B	I	R	D

Example of reverse engineering problem

Msg 1	2	3	67	65	84	4	66	73	82	68		
Msg 2	1	5	77	79	85	83	69					
Msg 3	3	2	79	88	3	68	79	71	3	66	85	71

1 2

Msg 1	2	3	67	65	84	4	66	73	82	68		
Msg 2	1	5	77	79	85	83	69					
Msg 3	3	2	79	88	3	68	79	71	3	66	85	71

Msg 1	2	3	C	A	T	4	B	I	R	D		
Msg 2	1	5	M	O	U	S	E					
Msg 3	3	2	O	X	3	D	O	G	3	B	U	G

1. The analyst starts with messages.
2. Infers some pattern in the data.
3. Develops a hypothesis.
4. Validates it on all messages.

67	65	84	
66	73	82	68
C	A	T	
B	I	R	D



What makes this problem hard for a human?

- Can take a long time to find a pattern in the data.
- Bytes at the same offset don't always come from the same **field** or **type**

Msg 1	2	3	C	A	T	4	B	I	R	D
Msg 2	1	5	M	O	U	S	E			
Msg 3	3	2	O	X	3	D	O	G	3	B

A	T	4	B
O	U	S	E
X	3	D	O

- Messages can be hundreds or thousands of bytes in size.

311 Byte Msg

```
40D513C4221EF3E2EEB96F37D3EB1C10805124771BCB9C146746E2A26CC30EB9E97BBB44821416CEF424837EEBBE8138D2B222B7D  
B07DE3FBFD791AABB867E876E2D699A0CC2A58299AB227A5822EC480A8C5F9FD7678036093DDA2575C3A762A4EA2F17D18BCC15  
385D7973B03128EFCB15CB317A5226B1B6654B01B116A56738B4B5B779F8D68334328C018C64C07A930DCD548F7C6B7A1952E26F2  
CA05340EC63BFEF513F3C1E8EB6AF00E14DC5000FE0A9CE5F876B56D7DA73352527329B60B66C552D469F3A2B12A4573B2C111557  
4FC4D30F8372A52D868DCC38D7739E94D2C0815000D3B692DCA6D82693AD93D102222D349E9EC4D101F67FC9E702B5430AFB73AB  
5361120902A82E4A6FDFF252809B36106B3C3FEC2FC8A98AFC642F1926BD4B3E72C39272004F2B8F731F8145A43D7B4D78BC
```

Our Automated Approach

Common Serialization Design Patterns

Variable Quantity Variable Length

Quantity (**Q**) $Q \cdot (L \cdot \text{BYTE}^L)^Q$ 2 5 Z E B R A 3 C A T
Length (**L**)

Type Length Value / TLV

Quantity (Q) $Q \cdot (T \cdot L \cdot \text{BYTE}^L)^Q$

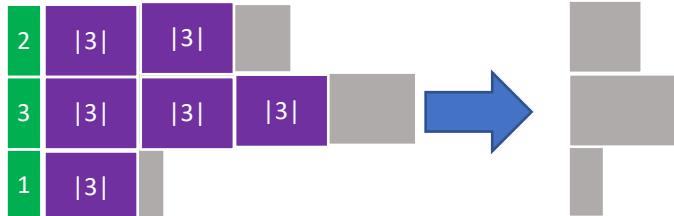
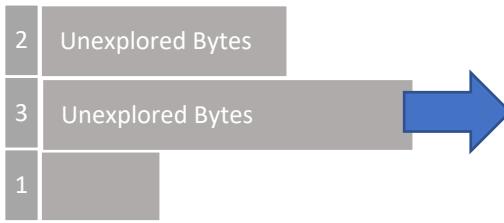
Type (T)

Length (L)

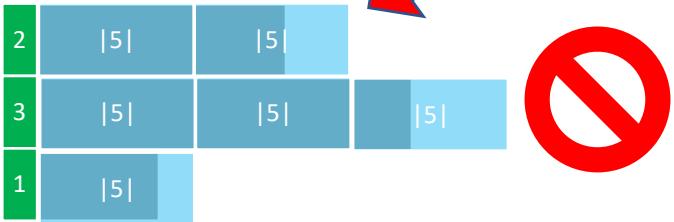
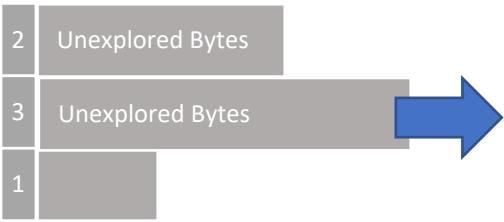
2 | T1 | 5 | Z | E | B | R | A | T2 | 3 | C | A | T

Variable Quantity Fixed Length	$Q \cdot (\text{BYTE}^K)^{[Q]}$	2	IP ADDR	IP ADDR	
Quantity (Q)		3	IP ADDR	IP ADDR	IP ADDR

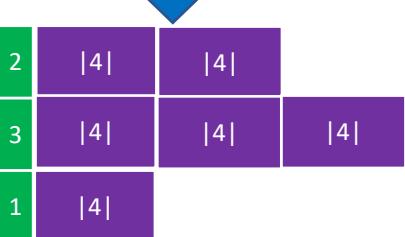
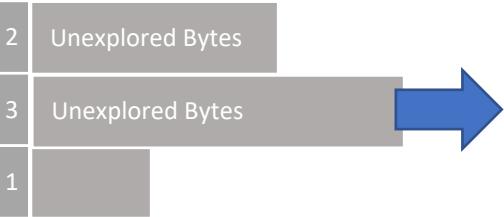
Approach for fitting design patterns to data



Apply a single design pattern
to all messages.
Recurse on leftovers.



If we index off the end of
any message, try a different
pattern.



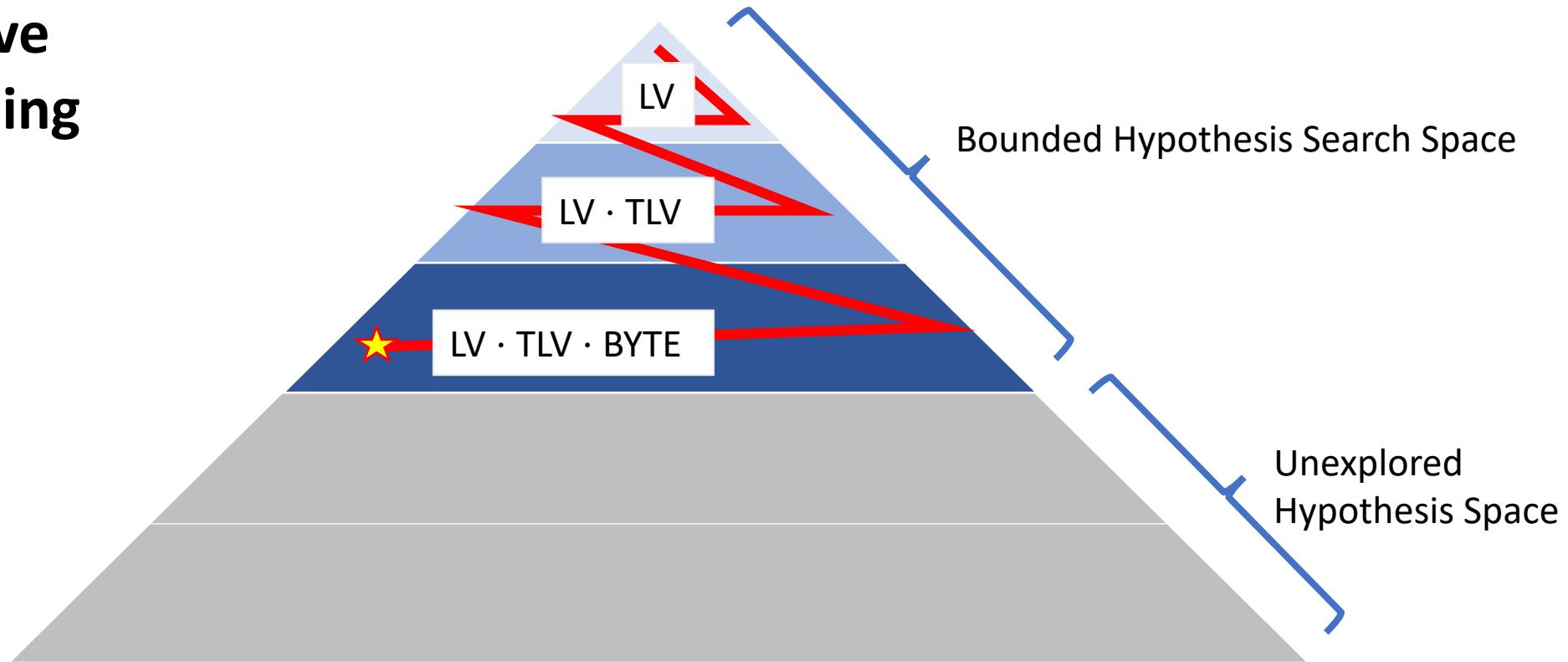
If it fits...

Success!



Hypothesis Space Exploration

**Iterative
Deepening**



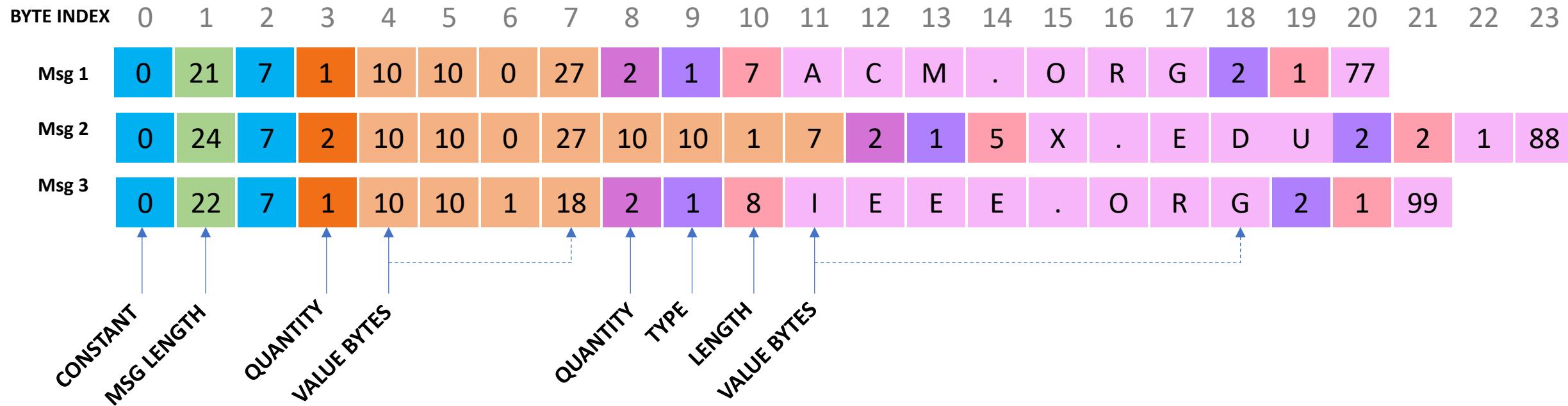
★ = Hypothesis consistent with message samples

How does our approach perform?

Experimental Condition	Test Cases	Accuracy
Patterns with random values	16500	99.9%
Patterns with values from real network traffic	1434	99.37%

1. Generated permutations of Design Patterns
(Example: LV · VQVL · VQFW₄)
2. Used each permutation to serialize values creating 100 messages.
3. Ran our inference Algorithm on each collection of 100 messages.
4. Compared Inferred Patterns with those used to Serialize.

Further Evaluation: Botnet CnC Attack Commands



Inferred Format 1: **BYTE, BYTE, BYTE, VQFL, TLV**

Inferred Format 2: **BYTE, BYTE, BYTE, VQFL, BYTE, BYTE, LV, BYTE, LV**

Next Steps

- Expand our tool box of design patterns through protocol taxonomy
- Parallelization
- Guided Search Heuristics

Thank You

Jared Chandler

jared.chandler@tufts.edu

Acknowledgements:

This material is based upon work partly supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No.HR0011-19-C-0073.

This project was sponsored in part by the Air Force Research Laboratory (AFRL) under contract number FA8750-19-C-0039. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.