www.crysys.hu

Equivalence and Similarity Checking of Inferred Protocol Messages and Grammars

Gergő Ládi

Laboratory of Cryptography and System Security

Department of Networked Systems and Services

Gergo.Ladi@CrySyS.hu

Software Verification & Validation – End-term presentation, 2018.12.06.

Outline

- Automated Protocol Reverse Engineering (APRE)
  - Aims, approaches, my research
- Equivalence and Similarity Checking – Inferred Messages
  - Work in progress
- Equivalence and Similarity Checking – Inferred Grammars
  - Future work

AUTOMATED PROTOCOL REVERSE ENGINEERING

Automated Protocol Reverse Engineering

- What is a protocol?
  Protocol specifications describe the interaction between different entities by defining message formats and message processing rules
  - Text-based
  - Binary

- Access to protocol specifications is essential for tasks such as
  - Analysing botnets
  - Building honeypots and honey systems
  - Defining rules for network intrusion detection/prevention systems
  - Testing protocol implementations
    - » Conformance testing
    - » Fuzz testing
  - …

Automated Protocol Reverse Engineering

- Unfortunately, protocol specifications are not always available
  - But it is generally possible to reconstruct these by various reverse engineering methods
- Automation is necessary
  - Manual reverse engineering is error-prone and time consuming
  - New protocols appear more frequently than what it would take to analyse them manually

Automated Protocol Reverse Engineering

- Reverse engineering approaches
  - Binary analysis
    - » Static or dynamic
    - » May not be permitted by license agreements
  - Network trace based
    - » Analyses recorded network traffic
    - » Does not work if traffic is encrypted

- Two principal goals
  - Reconstructing the message types (message formats)
  - Reconstructing the protocol state machine (protocol grammar)

Automated Protocol Reverse Engineering

- We needed a way to reverse protocols used by Industrial Control Systems
  - No binaries available -> solution must be network trace based
  - ICS protocols are usually binary -> solution must work on binary protocols
  - For starters, it was enough to reconstruct the message types

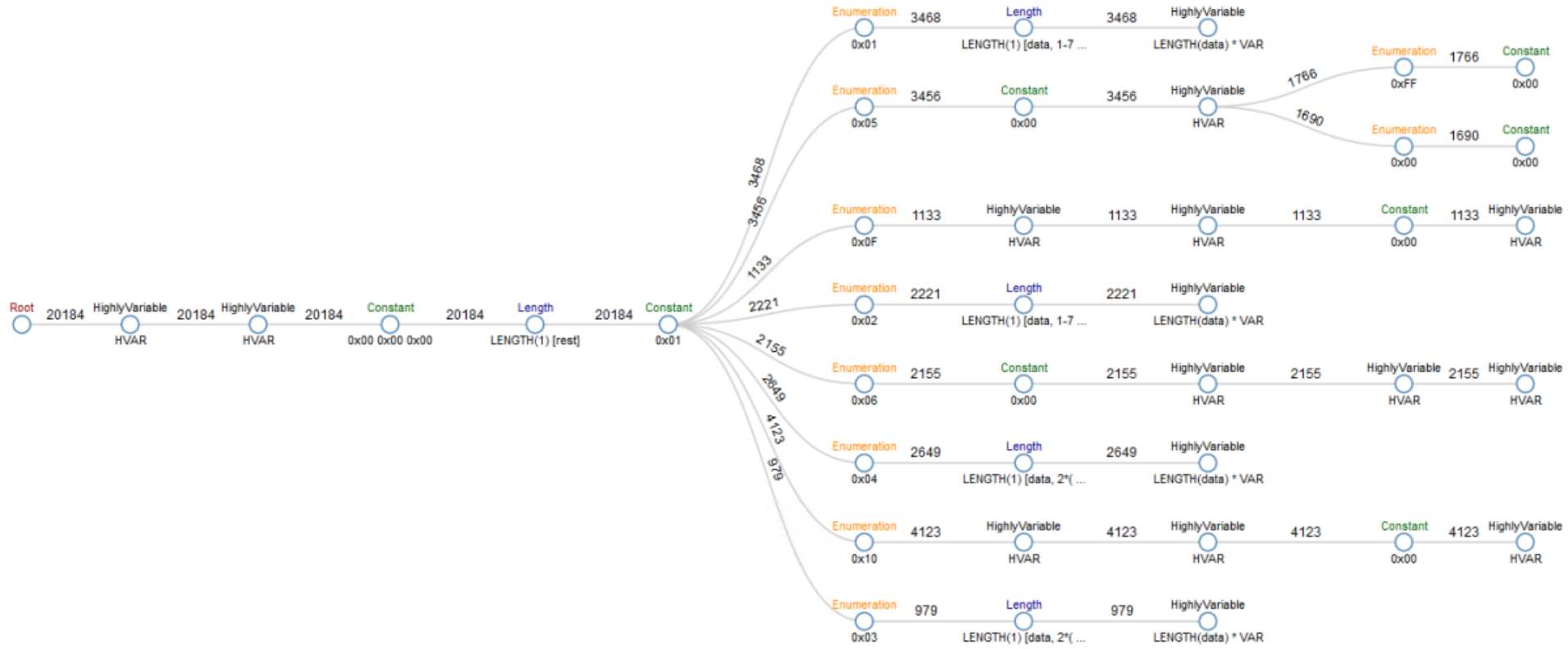
- Existing solutions (Biprominer, Discoverer, ProDecoder, …)
  - Not reliable enough
  - Not reproducible
  - Overly complex for the purpose
  - … but these helped us see common patterns

Automated Protocol Reverse Engineering

- We designed a new algorithm to infer message formats
  - G. Ládi, L. Buttyán, T. Holczer: Message Format and Field Semantics Inference for Binary Protocols Using Recorded Network Traffic (SoftCOM 2018)
- … but it's not the algorithm itself what matters in this presentation
  - It's the tests! (How do we know that the algorithm's good?)

Automated Protocol Reverse Engineering

Reversed specification (Modbus responses):



Automated Protocol Reverse Engineering

Reversed specification:

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x01 0x00 HVAR 0x00 HVAR

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x05 0x00 HVAR 0x00 0x00

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x05 0x00 HVAR 0xFF 0x00

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x0F HVAR HVAR 0x00 HVAR LENGTH(1) [data, 1-2] LENGTH(data) * HVAR

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x02 HVAR HVAR 0x00 HVAR

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x06 0x00 HVAR HVAR HVAR

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x04 0x00 HVAR 0x00 HVAR

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x10 HVAR HVAR 0x00 HVAR LENGTH(1) [data, 2*(1-10)] LENGTH(data) * HVAR

HVAR HVAR 0x00 0x00 0x00 LENGTH(1) [rest] 0x01 0x03 HVAR HVAR 0x00 HVAR

True specification:

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x01 HVAR HVAR HVAR HVAR

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x05 HVAR HVAR FLAG FLAG

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x0F HVAR HVAR HVAR HVAR LENGTH(1) [data, 1+] LENGTH(data) * HVAR

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x02 HVAR HVAR HVAR HVAR

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x06 HVAR HVAR HVAR HVAR

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x04 HVAR HVAR HVAR HVAR

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x10 HVAR HVAR HVAR HVAR LENGTH(1) [data, 2*(1+)] LENGTH(data) * HVAR

HVAR HVAR 0x00 0x00 LENGTH(2) [rest] HVAR 0x03 HVAR HVAR HVAR HVAR

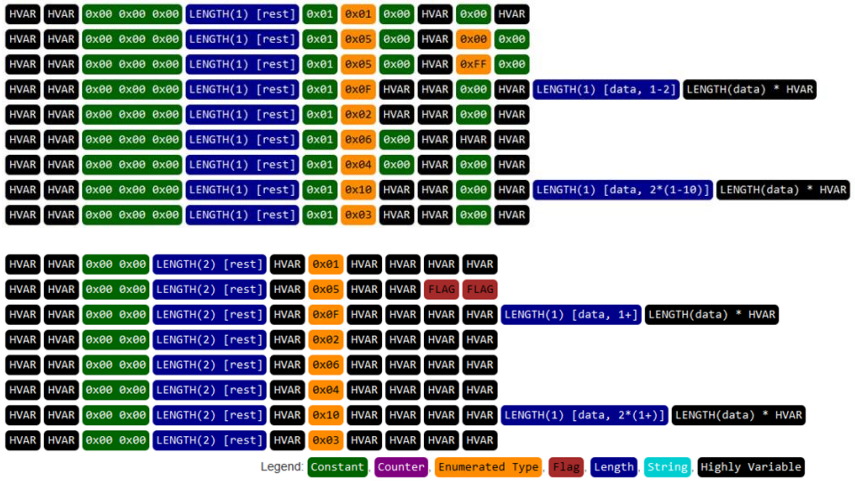
Legend: Constant, Counter, Enumerated Type, Flag, Length, String, Highly Variable

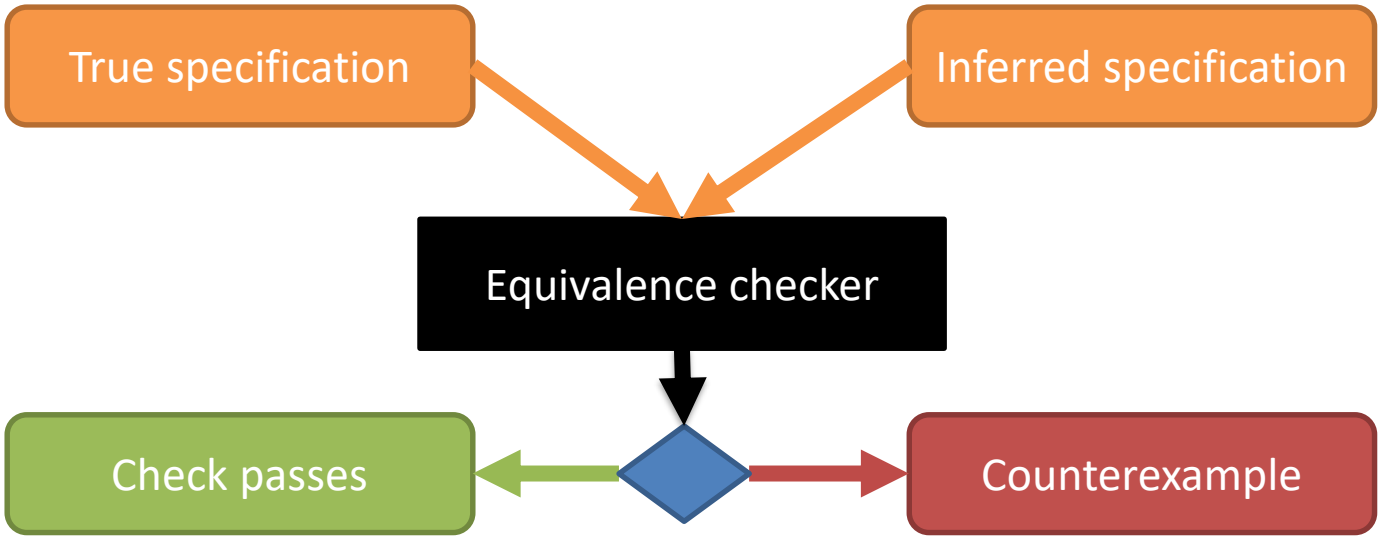
EQUIVALENCE AND SIMILARITY CHECKING – INFERRED MESSAGES

Equivalence/Similarity Checking – Messages

- How do you compare these two?

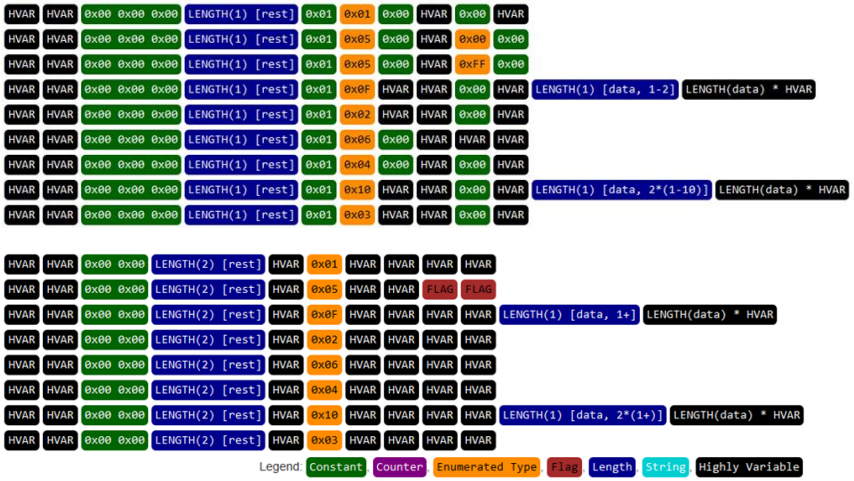
- Equivalence checking
  - Are the two specifications the same?





Equivalence Checking – Messages

- The inputs are trees
  - A tree isomorphism problem needs to be solved
    - » There exists a polinomial time algorithm that may be extended for this purpose
- This isn't really useful
  - It is very unlikely that the two trees are isomorphic
  - We need to know how similar the two are



1: **procedure** *isomorphic* (T_1, T_2)
2: assign level numbers to all nodes of T_1 and T_2
3: assign to all leaves of T_1 and T_2 the integer 0
4: let L_1 be a list of the leaves of T_1 at level 0
5: let L_2 be a list of the leaves of T_2 at level 0
6: **for all** levels i starting from 1 **do**
7: $\langle\langle$*assign integers to all nodes at level* $i\rangle\rangle$
8: **if** the roots of T_1 and T_2 are assigned the same integer **then**
9: T_1 and T_2 are isomorphic
10: **else**
11: T_1 and T_2 are not isomorphic
12: **end procedure**

Source: Gabriel Valiente: Algorithms on Trees and Graphs

Similarity Checking – Messages

- Tree Edit Distance (TED)
  - A minimal-cost sequence of edit operations that transforms a tree in such a way that the result will be isomorphic to another one
  - Edit operations
    - » Adding a node (while preserving the tree property)
    - » Deleting a node (and connect its children to its parent)
    - » Recolouring/relabeling a node
  - The costs of the edit operations may be weighted differently

Similarity Checking – Messages

- Algorithms
  - Recursive (decomposes trees into subtrees)
    - » K.-C. Tai, 1979 – $O(n^3m^3)$ [*n* and *m* are the number of nodes in trees *t1* and *t2*]
    - » K. Zhang, D. Shasha, 1989 – $O(n^2m^2)$
    - » P.N. Klein, 1998 – $O(n^2 m \log m)$
    - » ...
  - Robust algorithm (RTED) - M. Pawlik, N. Augusten, 2011
    - » Exhaustive search for the best strategy – $O(n^2)$
    - » Performing the comparison – $O(n^3)$
  - APTED
    - » By the authors of RTED
    - » Better memory complexity, better time complexity for special cases
    - » A Java implementation exists (https://github.com/DatabaseGroup/apted)
    - » This is what I'm currently experimenting with

EQUIVALENCE AND SIMILARITY CHECKING – INFERRED GRAMMARS

Equivalence Checking – Grammars

- Protocol specifications can be represented by automatons (state machines) that have their own grammar
- Automatons can be represented by graphs

- Equivalence checking
  - Are the two grammars the same?
  - The inputs are graphs
    - » A graph isomorphism problem needs to be solved
      - This is an NP problem
  - Just like equivalence checking for the messages, this isn't really useful
    - » We're unlikely to get a perfect match

Similarity Checking – Grammars

- Graph Edit Distance
  - A minimal-cost sequence of edit operations that transforms a graph in such a way that the result will be isomorphic to another one
  - Edit operations
    - » Adding a node or edge
    - » Deleting a node or edge
    - » Recolouring/relabeling a node or edge

$$GED(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(g_1,g_2)} \sum_{i=1}^{k} c(e_i)$$

- Algorithms
  - Computation is NP-complete
  - Even approximation is hard (APX-hard)
  - Some approximations exist
    - » M. Neuhaus, H. Bunke, 2007
    - » K. Riesen, 2016
  - Experimenting with these is future work for me

References

- Kuo-Chong Tai, *The Tree-to-Tree Correction Problem*, 1979.
- K. Zhang, D. Shasha, *Simple fast algorithms for the editing distance between trees and related problems,* 1989.
- P. N. Klein, *Computing the edit-distance between unrooted ordered trees,* 1998.
- M. Pawlik, N. Augsten, *RTED: A Robust Algorithm for the Tree Edit Distance*, 2011.
- M. Pawlik, N. Augsten, *Tree edit distance: Robust and memory-efficient*, 2016.
- K. Riesen, M. Neuhaus, H. Bunke, *Bipartite Graph Matching for Computing the Edit Distance of Graphs*, 2007.
- K. Riesen, H. Bunke, *Graph Edit Distance – Novel Approximation Algorithms*, 2016.

THANK YOU FOR YOUR ATTENTION!