Rongkuan MA, Hao ZHENG, Jingyi WANG, Mufeng WANG, Qiang WEI, Qingxian WANG, 2022. Automatic protocol reverse engineering for industrial control systems with dynamic taint analysis. *Frontiers of Information Technology & Electronic Engineering*, 23(3):351-360. https://doi.org/10.1631/FITEE.2000709

Automatic protocol reverse engineering for industrial control systems with dynamic taint analysis

Key words: Industrial control system (ICS); ICS protocol reverse engineering; Dynamic taint analysis; Protocol format

Corresponding author: Qiang WEI
E-mail: weiqiang66@126.com
ORCID: https://orcid.org/0000-0002-7207-6691

Motivation

1. Formats of industrial control system (ICS) protocols are usually undocumented in practice, which hinders advancement in ICS cybersecurity research, such as intrusion detection, investigation forensics, and fuzz testing.

2. Previous works based on program analysis have limitations in analyzing a binary-based protocol program. Their methodologies usually perform well on text-based protocols, and are often too coarse-grained and not accurate enough.

Motivation example

1. Our key insight is that an individual field in a message is typically handled in an individual basic block (BBL) group.

2. We provide our methodology based on the intuition that an individual field in a received message is handled by an individual BBL group in a binary application, where a BBL group is defined as the continuous basic block before a subroutine is called in a function's execution trace.

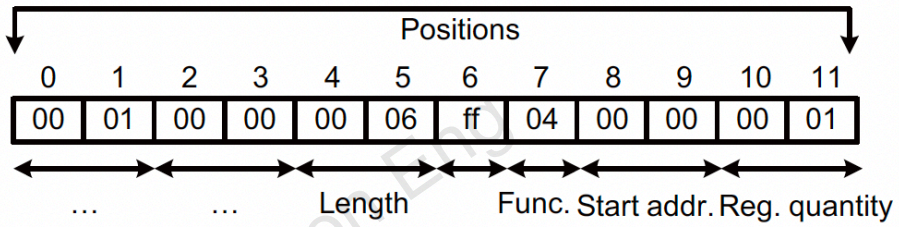


Fig. 1 An example of the Modbus protocol read register query message

When the program processes the input data, the destination memory is tainted with the original offset position

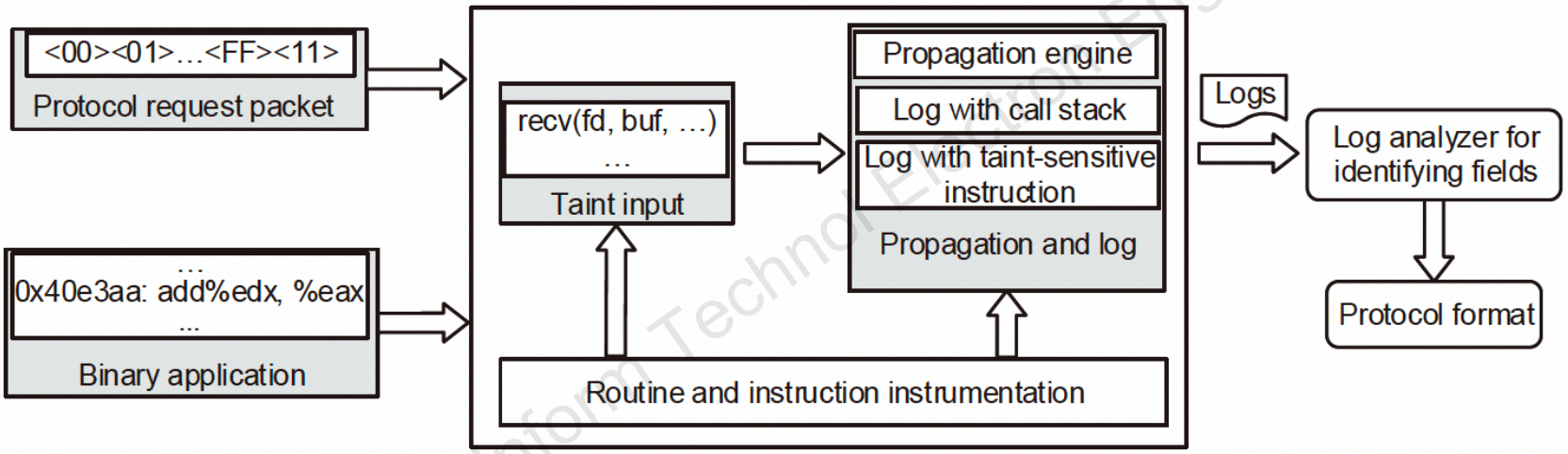
```
641 int modbus_reply(modbus_t *ctx, const uint8_t *req, int req_length, modbus_mapping_t *mb_mapping)
643 {
644     int offset = ctx->backend->header_length;
645     int slave = req[offset - 1];
646     int function = req[offset];
647     uint16_t address = (req[offset + 1] << 8) + req[offset + 2];

652       if (ctx->backend->filter_request(ctx, slave) == 1) {
653         /* Filtered */
654         return 0;
655       }
...
659     sft.t_id = ctx->backend->prepare_response_tid(req, &req_length);
661     switch (function) {
662     case _FC_READ_COILS: {
663         int nb = (req[offset + 3] << 8) + req[offset + 4];

665         if ((address + nb) > mb_mapping->nb_bits) {
            ...
673         } else {
674             rsp_length = ctx->backend->build_response_basis(&sft, rsp);
675             rsp[rsp_length++] = (nb / 8) + ((nb % 8) ? 1 : 0);
676             rsp_length = response_io_status(address, nb, mb_mapping->tab_bits, rsp, rsp_length);
```

Fig. 2 Code snippet in modbus.c of libmodbus

Framework



Overview of the industrial control system protocol reverse engineering framework (ICSPRF)

Methods

Algorithm 1 Dynamic taint analysis framework

```
addr, len ← instrumenting taint introduction function;
if len is not empty then
    Taintinit(addr, len);
end if
if Img ∈ selected images then
    left_op, write_op ← instrumenting instructions in Img;
else
    left_op, write_op ← instrumenting data movement function called in Img;
end if
if isTainted(left_op)||isTainted(write_op) then
    Run taint, unmark, or merge according to our taint propagation policy;
end if
```

Table 1 Policy I: taint propagation policy of data movement operations

Operation type	Policy	Example(s)
reg_l ← reg_r	$T(reg\_w) = T(reg\_r)$	mov, esi, edi
reg_l ← mem_r	$T(reg\_w) = T(mem\_r)$	mov eax, dword ptr [rbp]
reg_l ← imm_r	$T(reg\_w) = false$	mov eax, 0x1
mem_l ← reg_r	$T(mem\_w) = T(reg\_r)$	push rbp
mem_l ← mem_r	$T(mem\_w) = T(mem\_r)$	push qword ptr [rbp-0x4]
mem_l ← imm_r	$T(mem\_w) = false$	mov qword ptr [rsp], 0x0
dst ← src	$T(dst) = T(src)$	memcpy(dst, src, size), memmove(dst, src, size)

Table 2 Policy II: taint propagation policy of arithmetic and logic operation instructions

Instruction	Examples	Policy
xor	xor edi, edi	Untaint(edi)
sub	sub esi, esi	Untaint(esi)
and	and eax, 0x0	Untaint(eax)
shl/shr	shl reg, 0x8	ShiftT(reg)
or/add	add eax, edx	MergeT(eax, edx)
and	and eax, 0xff	PartialSaveT(eax)

There are two stages during the dynamic taint analysis of a program: taint initialization and taint propagation. As shown in Algorithm 1, at the taint initialization stage, ICSPRF intercepts the system calls by routine-grained instrumentation, taints the received data, and records every byte with its position offset in the entire message. The taint propagation stage is focused on the application-level program images that process the received message.

Methods

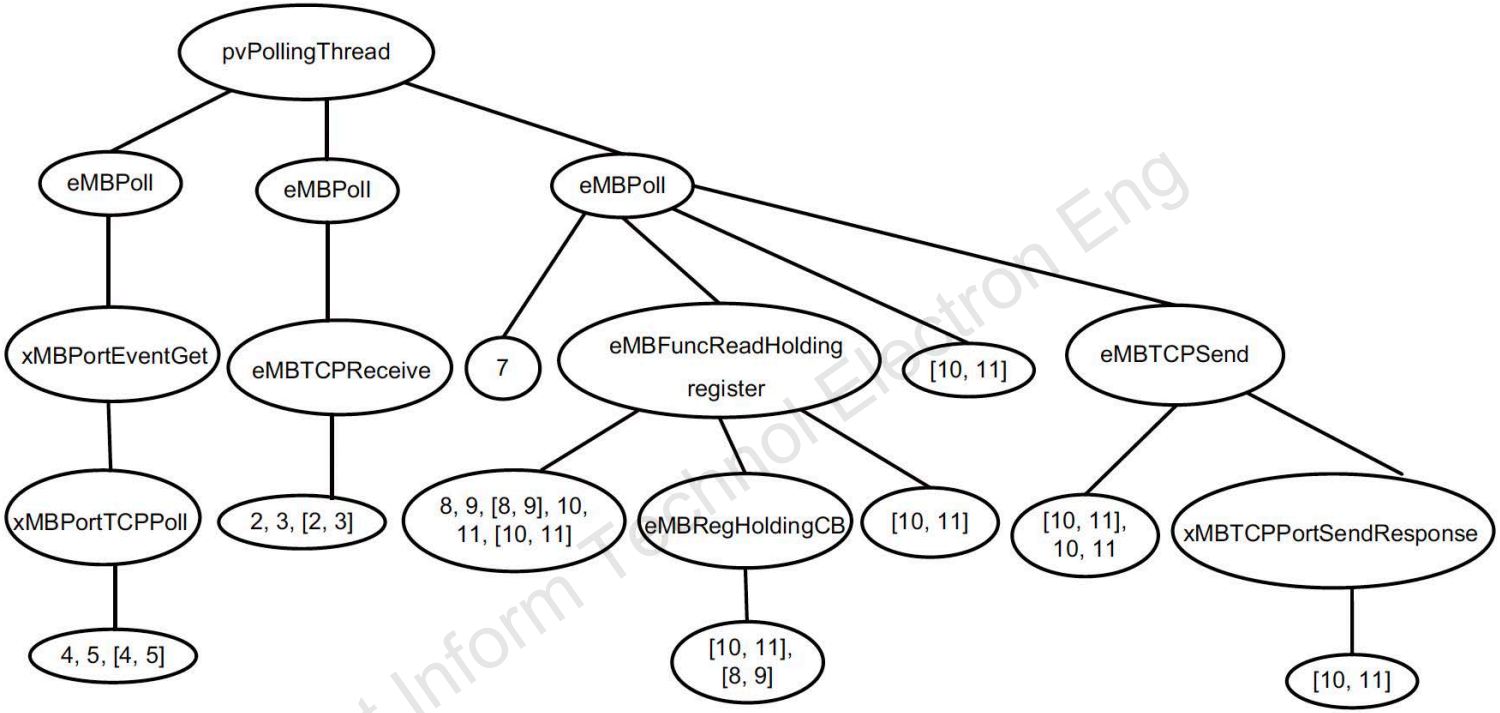
At the stage of offline log analysis, we aim to identify the borders of the field chunks in a protocol message. First, we provide an algorithm to construct a hierarchical tree that demonstrates the call relationship and sequential relationship among all BBL groups that process tainted bytes. From this tree, we can infer individual fields based on several principles.

Algorithm 2 Hierarchical tree generation

Require: the log items logs.
Ensure: a hierarchical tree.

```
root ← FunctionNode();
node ← root;
data ← DataNode();
for each log ∈ logs do
  if log.type == FUNCTION then
    if data is not empty then
      Add data → node;
      data ← DataNode();
    end if
    name ← log.name;
    if log.keyword == enter then
      newnode ← FunctionNode(name);
      Add newnode → node;
      node ← newnode;
    end if
    if log.keyword == exit then
      node ← node.parent;
    end if
  end if
  if log.type == INSTRUCTION then
    Add log.offsets → data;
  end if
end for
return root;
```

Major results



As shown in Fig. 4, the bytes at offsets 0, 1, and 6 of the received message are not processed in any BBL group, the byte at offset 7 is processed in a single BBL group, and the bytes at offsets [2, 3], [4, 5], [8, 9], and [10, 11] are processed together in individual BBL groups separately. From this tree, we can conclude that the field chunks in the received message can be split as [0, 1] [2, 3] [4, 5] [6] [7] [8, 9] [10, 11].

Major results

Table 4 Identified field chunk comparison between Wireshark and ICSPRF

Project	Message type	Length	Wireshark	ICSPRF					AutoFormat	
			Size	|SF|	R_e	|SF$_o$|	|SF$_c$|	$\overline{R}_e$	$\overline{R}_e(A)$	$\overline{R}_e(B)$
libmodbus	Read registers	12	7	6/7	86%	0	1	94.3%	88.5%	80.0%
	Write and read registers	21	11	10/11	91%	**0**	1			
	Write coils	14	7	6/7	86%	**0**	1			
freemodbus	Read registers	12	7	7/7	100%	0	0			
	Write and read registers	21	11	11/11	100%	**0**	0			
	Write coils	14	7	7/7	100%	**0**	0			
IEC104	U-format	6	4	4/4	100%	0	0			
IEC60870	U-format	6	4	4/4	100%	0	0			
DNP3	Read class	18	11	11/11	100%	0	0			
Snap7	Read SZL	33	22	18/22	82%	0	3			
	List blocks	31	22	20/22	91%	0	1			
	Write variables	36	26	25/26	96%	1	1			

$\overline{R}_e(A)$ and $\overline{R}_e(B)$ of AutoFormat represent the match ratios for all the evaluated protocols and for the binary-based protocols, respectively. The units of length and size are both byte

Table 4 shows the detailed results. For all the evaluated projects, the match ratio R_e is no less than 82% and we obtain R_e=94.3% with small |SF$_o$| and |SF$_c$|. In contrast, for the same metric, AutoFormat achieves a lower accuracy than ICSPRF (88.5% for all evaluated protocols and 80.0% for binary-based protocols). For certain examples, e.g., freemodbus, IEC104, IEC60870, and DNP3, we even obtain R_e=100% with zero |SF$_o$| and |SF$_c$|.

Conclusions

ICSPRF, the proposed framework, is based on the insight that a protocol program has chunk-feature behaviors in its execution contexts. By instrumenting and monitoring program execution, we can obtain the taint propagation process among BBL groups and use it to infer protocol fields. The experimental results showed that ICSPRF achieved higher accuracy (on average, a 94.3% match ratio for the binary-based ICS protocol) in protocol field identification; in contrast, AutoFormat achieved an 88.5% match ratio for all evaluated protocols and only 80.0% for binary-based protocols.

Rongkuan MA received his PhD degree in the State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China, in 2021. He is a system security researcher. His research interests include program analysis and embedded system security, iCPS security, and Web security.

Hao ZHENG received the BS and MS degrees in Zhejiang University, China, in 2016 and 2019, respectively. He is now an algorithm engineer in the industry.

Jingyi WANG is currently a tenure-track assistant professor at the College of Control Science and Engineering, Zhejiang University, China. He received his BS degree in Information Engineering from Xi'an Jiaotong University in 2013 and PhD degree from Singapore University of Technology and Design in 2018. He was a research fellow at the School of Computing, National University of Singapore during 2019–2020 and at Information Systems Technology and Design Pillar, Singapore University of Technology and Design during 2018–2019. His research interests include formal methods, software engineering, cyber-security, and machine learning.

Mufeng WANG is a research fellow at the College of Control Science and Engineering, Zhejiang University. He received his BS degree from West Anhui University in 2013, MS degree from Guangxi Normal University in 2016, and PhD degree from South China University of Technology in 2019. His research interests include analysis and synthesis of security for industrial control systems and cyber-physical systems.

Qiang WEI received his PhD degree in Computer Science and Technology from the China National Digital Switching System Engineering and Technological Research Center, Zhengzhou, China. He is currently a professor with the State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include network security, industrial Internet security, and vulnerability discovery.

Qingxian WANG received his MS degree from the Department of Computer Science and Technology, Peking University, China. He is currently a professor at the School of Cyber Science and Engineering, Zhengzhou University, China. His research interests include network security, trusted computing, and vulnerability discovery.