# How to Dissect Unusual Protocols for Troubleshooting OT Security

nozominetworks.com/blog/how-to-dissect-unusual-protocols-for-troubleshooting-ot-security/

Bу

June 22, 2021

As OT security researchers, the Nozomi Networks Labs team continually works to understand OT and IoT device processes and their security risks. This includes understanding how assets like embedded controllers communicate with each other and their workstations. To do so, we need to reverse undocumented protocols.

Unfortunately, the process of dissection can't be standardized due to the unknown layers of complexity put in place by most vendors when they design systems. This is where the experience of security researchers can make a big difference.

This is the first of a series of articles from Nozomi Networks Labs where we'll demonstrate how to use Lua APIs to instruct Wireshark to properly dissect an undocumented protocol.

In this blog, we look at the steps involved in developing a dissector for a real-world use case, the well-known Cisco Nexus protocol. We've also posted a plug-in on GitHub to help the security community at large.

Whether you're a security researcher yourself, or you manage networks for an asset owner, this methodology and plug-in will help you troubleshoot networking issues and improve overall OT security.



It can be tedious and time-consuming to gain a complete understanding of the unknown protocols used by different security device vendors, but it's possible to use Lua APIs to more easily dissect them.

#### **Determining How to Dissect Unknown Protocols**

One of tools commonly used to kick-start the exploration process for an unknown protocol is Wireshark.

With the right set of traffic/Pcaps – generated by forcing a specific type of communication between controllers – we can start analyzing the protocol.

We begin by focusing our attention on patterns. Normally, when we start the protocol reverse engineering process, we're confronted by an unknown language. We need to identify key elements that will help us understand the communication structure step-by-step (lengths, function codes, sequence numbers, crc, etc.).

Ча¢ > > N Чо+ N ви П М г ∨ Z ви П М г ∨ Z	L2 L? s 4		нол мотнс Омрынс	< K . の ~	0 j H O X C H O O ^ A N & L 0 & O L 2 ^ P M P D 4 ^ & - 1 0 1 - 1 0 1
м: j	10	N 8 # # N 5 K × X N 8 # # N 5 K × X N 9 4 , L ← E A N 9 4 , L ← E A N 9 7 9 7 <b>R</b>	u 0 0 0 × 1, u 0 0 × 1, u 0 ×	јГн; ч∨Ү <b>н</b> {	ta XX @ VH ) o nut o 2 a

Reverse engineering of protocols begins with focusing on unknown language patterns. Plugin scripts written in the Lua language can help dissect packets and validate research findings.

While making assumptions during this phase, it helps to leverage one of Wireshark's capabilities called plugins. These are scripts written in the Lua programming language that instruct the tool to dissect each packet using our findings. Plugins also allow us to validate findings with the collected and/or live traffic.



## TIP: You can also create your own dissectors directly using the native Wireshark C language in cases where performance needs to be fine-tuned.

- Windows: %APPDATA%\Wireshark\plugins
- Unix: ~/.local/lib/wireshark/plugins
- Mac: ~/.config/wireshark/plugins

Time  Source 1 2021-05-06 08:53:44.794835 10.41.53.13				
1 2021-05-06 08:53:44.794835 10.41.53.13	Destination	Protocol	Lengtr Info	
	10.41.53.4	CIP CM	114 Unconnected Send: Identity - Get Attributes All	
2 2021-05-06 08:53;44.794897 10.41.53.13	10.41.53.4	CIP CM	114 Unconnected Send: Identity - Get Attributes All	
3 2021-05-06 08:53:44.797513 10.41.53.4	10.41.53.13	CIP	145 Success: Identity - Get Attributes All	
4 2021-05-06 08:53:44.798055 10.41.53.13	10.41.53.4	CIP CM	116 Unconnected Send: Identity - Get Attribute Single	
5 2021-05-06 08:53:44.799454 10.41.53.4	10.41.53.13	CIP	98 Attribute not supported: Identity - Get Attribute	
6 2021-05-06 08:53:44.799663 10.41.53.13	10.41.53.4	CIP CM	116 Unconnected Send: Identity - Get Attribute Single	
7 2021-05-06 08:53:44.802526 10.41.53.4	10.41.53.13	CIP	98 Attribute not supported: Identity - Get Attribute	
8 2021-05-06 08:53:44.802722 10.41.53.13	10.41.53.4	CIP CM	116 Unconnected Send: Identity - Get Attribute Single	
9 2021-05-06 08:53:44.805542 10.41.53.4	10.41.53.13	CIP	98 Attribute not supported: Identity - Get Attribute	
10 2021-05-06 08:53:44.805704 10.41.53.13	10.41.53.4	CIP CM	116 Unconnected Send: Identity - Get Attribute Single	
11 2021-05-06 08:53:44.808548 10.41.53.4	10.41.53.13	CIP	98 Attribute not supported: Identity - Get Attribute	Sing
12 2021-05-06 08:53:44.808728 10.41.53.13	10.41.53.4	CTP CM	116 Unconnected Send: Identity - Get Attribute Single	
Command Specific Data ommon Industrial Protocol Service: Get Attributes All (Response) Status: Success: [Request Path Size: 2 words] [Request Path: Connection Manager, Instance: 0x01] [P Connection Manager		0070 1d 31	00 04 02 00 00 00 00 00 00 00 03 00 81 00	6` B Et

#### Rockwell Ethernet/IP dissection

As you can imagine, the processes involved in gaining a complete understanding of an unknown communication language can be tedious and time-consuming. However, the end result is shareable knowledge (in the form of a plugin) that's highly useful to others. For example, utility operators can benefit enormously from tools like this every time they need to troubleshoot specific scenarios within a normal industrial process operation.

Rather than describe how to develop a Lua dissector from scratch (there's already a lot of easily-found documentation out there, see the tip below), we'll dive into some specific non-standard real world use cases and how to deal with them.



TIP: For those just starting their Lua dissection journey, take a look to Mika's tech blog: <u>Creating a Wireshark dissector in Lua – Part 1 (the basics).</u>

#### **Registering a New Dissector**

#### Ethertype and Recall

Our first challenge involves dealing with protocols that aren't easily accessible by Wireshark – at least not in the standard way we're used to.

Let's use Cisco Nexus as an example – a well-known protocol used between the NX-OS switch series. We'll go through the steps involved in developing the related dissector (note we also reverse engineered its inner structure).

First, the research team noticed that Wireshark doesn't support the dissection of such a protocol. This means we have to do a bit of investigation and packet analysis in order to understand its structure.

As far as Wireshark knows, we have an <u>802.1Q Virtual Lan frame</u>. This is a commonly used standard for defining VLANs (Virtual LAN) with a pretty basic structure.

The VLAN ID in place [0x8905] is non-standard. This is why the tool classifies it as an "unknown" type. Let's assume that that could be a good first indicator for a proprietary protocol, and keep it in mind for later.

											• 🖬 •
Time		Source	Destination	Protocol	Length						
		MS-NLB-PhysServ		0x8905		PRI: 0					
			Beckhoff_01:00:00	0×8905		PRI: 0			: 22		
		MS-NLB-PhysServ		0×8905		PRI: 0			: 23		
		MS-NLB-PhysServ		0×8905	864	PRI: 0	DEI:	0 ID	: 23		
5 2019-02-01	1 04:21:25.269667	MS-NLB-PhysServ	Beckhoff_01:00:00	0×8905	1345	PRI: 0	DEI:	0 ID	: 22		
			Beckhoff_01:00:00	0×8905	1345	PRI: 0	DEI:	0 ID	: 22		
7 2019-02-02	1 04:21:25.271531	MS-NLB-PhysServ	Beckhoff_01:00:00	0x8905	864	PRI: 0	DEI:	0 ID	: 23		
8 2019-02-03	1 04:21:25.271733	MS-NLB-PhysServ	Beckhoff_01:00:00	0x8905	864	PRI: 0	DEI:	0 ID	: 23		
9 2019-02-03	1 04:21:25.274713	MS-NLB-PhysServ	Beckhoff_01:00:00	0×8905	1345	PRI: 0	DEI:	0 ID	: 22		
10 2019-02-03	1 04:21:25.274991	MS-NLB-PhysServ	Beckhoff_01:00:00	0×8905	1345	PRI: 0	DEI:	0 ID	: 22		
11 2019-02-03	1 04:21:25.276513	MS-NLB-PhysServ	Beckhoff 01:00:00	0x8905	864	PRI: 0	DEI:	0 ID	: 23		
			Beckhoff_01:00:00	0x8905	864	PRI: 0			: 23		
			Beckhoff_01:00:00	0x8905	1345			0 ID	: 22		
			Beckhoff_01:00:00	0×8905	1345	PRI: 0		0 ID	: 22		
			Beckhoff_01:00:00	0×8905	864				: 23		
			Beckhoff_01:00:00	0x8905		PRI: 0			: 23		
			Beckhoff_01:00:00	0x8905							
10 2010 02 01											
thernet II, Src: MS- 2.10 Virtual LAN, P	n wire (10760 bit NLB-PhysServer- RI: 0, DEI: 0, ID	s), 1345 bytes cap : 22	otured (10760 bits) Dst	0010 0020 0030	1345	PRI: 0	DEI:	0 ID	. 22		₩^K!. 0. <3.\
rame 1: 1345 bytes o thernet II, Src: MS- 02.10 Virtual LAN, P 000	n wire (10760 bit NLB-PhysServer- RI: 0, DEI: 0, ID . = Priority: Best . = DEI: Ineligibl ) = ID: 22 H05)	s), 1345 bytes cap : 22 : Effort (default) e	otured (10760 bits) Dst	0010         0010           c         020           0030         0030           0040         0030           0050         0           0110         0           0130         0	36         68         68         48           36         10         02         08           36         10         02         08           36         06         c4         39           36         08         08         08           36         08         08         08           37         08         09         23           38         09         08         08           38         09         08         08           38         09         08         08           38         09         08         08           38         09         08         08	02 20 0 0d 10 2 10 12 0 0d 00 2 10 40 0 0d 00 2 10 40 0 00 00 0 00 00 0 00 00 0 00 00 0 00 00	d 18 4 00 0 00 d 00 d 00 0 00 0 00 0 00 0 00	24 00 00 00 24 00 00 00	c4 39 10 2 00 40 02 2 02 00 00 1 00 00 00 0 00 00 00 0 00 00 00 0 00 00	00 24 00 12 00 00 10 00 00 10 cc 00 10 00 00	0

Unknown protocol

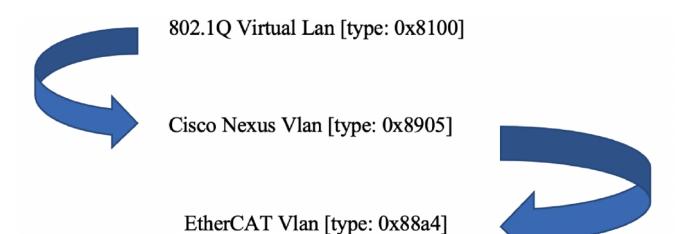
After a bit of deeper analysis on some interesting patterns in the "data" content of the packet, we determined that the structure is much simpler than expected. And, from a particular point, it is very similar to another common protocol in the industrial field. This proprietary protocol is actually a combination of a vendor specific layer, plus a well known protocol.

-`Q`-

#### HINT: Do those 0x88a4 bytes ring a bell?

Luckily, the encapsulated protocol is already known by Wireshark. This means that we have to instruct the tool to subscribe (or tag) the unknown layer to properly and accurately detect it every time it's seen over the wire.

Now, recall the discovered encapsulated dissector mentioned earlier.



Ethernet II frame structure

Let's start by writing down some code. First, we have to instruct Wireshark to link to our dissector every time it sees any ethernet packet with the unknown Type ID: 0x8905. To link properly, we can force the `DissectorTable.get()` function to point in the specified frame area by indicating that we're interested in referencing only the `ethertype` parameter [aka Type ID] and not the standard udp/tcp.port.

```
-- initialize wrapper fields
-- wrapper main function
function cisco_nexus.dissector (buf, pkt, root)
end
-- subscribe for Ethernet packets on type 0x8905.
local eth_table = DissectorTable.get("ethertype")
eth_table:add(0x8905, cisco_nexus)
```

Next, we have to initialize a second dissector table that we're going to recall at a specific offset. For this one, we'll use a different approach: Lua gives us the ability to point at a specific known dissector every time we need it by using the get\_dissector() function, and then use it through the call() function.

Let's see these functions in action.

Because the nested protocol has a defined VLAN tag within the 802.1Q IEEE standard, we can store the DissectorTable related to it in the `original\_vlan\_dissector` variable at the initialization section of our script:

```
-- load 802.1Q Virtual LAN dissector
original_vlan_dissector =
DissectorTable.get("ethertype"):get_dissector(0x8100)
```

and then call it at the right offset after the Cisco Nexus header, within the context of the main dissection function, precisely after the 4 header bytes:

-``Q`-

TIP: The first length check is done to ensure that we avoid any 0 byte packets, in case some are found.

```
-- wrapper main function
function cisco_nexus.dissector (buf, pkt, root)
-- validate packet length is adequate, otherwise quit
if buf:len() == 0 then return end
pkt.cols.protocol = cisco_nexus.name
-- create subtree for Cisco Nexus
subtree = root:add(cisco_nexus, buf())
-- subscribes ECAT dissector
original_vlan_dissector:call(buf:range(4,buf:len()-4):tvb(), pkt, subtree)
```

end

The final result is a complete dissection of the entire packet structure.

ly a display filter <	<3€/>										
Time		Source	Destination	Protocol	Length	Info					
	-02-01 04:21:25.264646			0×8905		PRI: 0					
	-02-01 04:21:25.264921			0×8905	1345	PRI: 0	DEI:	0 ]	D: 22		
	-02-01 04:21:25.266534			0×8905		PRI: 0					
	-02-01 04:21:25.266713			0×8905		PRI: 0					
	)-02-01 04:21:25.269567			0×8905		PRI: 0			D: 22		
	-02-01 04:21:25.269915			0x8905		PRI: 0			D: 22		
	0-02-01 04:21:25.271531			0x8905		PRI: 0			D: 23		
	-02-01 04:21:25.271733 -02-01 04:21:25.274713			0×8905 0×8905		PRI: 0			D: 22		
	-02-01 04:21:25.274991			0x8905		PRI: 0 PRI: 0			D: 22		
	-02-01 04:21:25.276513			0x8905 0x8905		PRI: 0			D: 22		
	-02-01 04:21:25.276739			0x8905		PRI: 0			D: 23		
	-02-01 04:21:25.279704			0×8905		PRI: 0					
	-02-01 04:21:25.279980			0×8905		PRI: 0			D: 22		
	-02-01 04:21:25.281464			0x8905		PRI: 0					
	-02-01 04:21:25.281719			0x8905		PRI: 0					
	-02-01 04:21:25.284618										
ernet II, Src .10 Virtual L 000	ytes on wire (10750 bit c: MS-NLB-PhysServer-0) LAN, PRI: 0, DEI: 0, II = Priority: Bes	ts), 1345 bytes cap L_05:34:03:cb ( D: 22 t Effort (default)	ptured (10760 bits) :03:cb), Dst	: 0020 3 0030 3 0040 0	<b>9 05</b> 57 fc 0 01 02 80 3c 82 00 00	00 00 0 33 00 9 10 12 0	4b 11 08 00 5c 01 00 00	88 a 01 0 00 0 00 1	4 21 15 0 0b 00 0 00 40 0 02 00	07 96 01 00 00 00 00 01 02 20 0d 04 0d 00 24 00 10 12 00 00	0- <····@·
nernet II, Src 2.10 Virtual L 300 0 0000 0001 Type: Unknown ta (1327 bytes	ytes on wire (10750 bit c: MS-NLB-PhysServer-0) LAN, PRI: 0, DEI: 0, II = Priority: Besi = DEI: Ineligib 1 0110 = ID: 22 (0x8905) s) edb1188a42115079601003	ts), 1345 bytes cap L_05:34:03:cb ( ): 22 t Effort (default) le	object (10760 bits) :03:cb), Dst (0)	0010         8           0020         3           0030         3           0040         6           0050         6	9         05         57         fc           10         10         28         80           12         20         80         81           12         12         80         80           14         40         44         44           16         10         28         80           18         10         10         28         40           10         10         24         40         44           10         10         28         40           10         10         28         40         40           10         10         28         80         80           10         10         28         80         80           12         32         36         80           12         32         36         10           12         32         36         10           12         32         36         10           12         32         36         10           16         09         00         60           18         89         90         90	f0 5e 4 00 00 0 33 00 5 10 12 0 02 20 0 00 10 2 10 12 0 02 20 0 00 00 5 00 00 0 00 0	4b         11           08         00           5c         01           00         00	88 a 01 0 00 0 00 1 24 0 00 0	21       15         0       00       00         0       00       40         0       02       00         0       02       00         0       02       00         0       02       00         0       02       00         0       02       00         0       04       39         0       04       39         0       04       30         0       04       30         0       04       30         0       05       06         0       06       06         0       07       01         0       06       00         0       07       08         0       07       08         0       07       08         0       07       08         0       07       08         0       08       08         0       08       08         0       08       08         0       08       08	00         00         00         01           02         20         0d         04         00           0d         00         24         00         02         00           10         12         00         00         02         00         00           02         20         2d         04         04         00         10         12         00         00           10         12         00	e 3. (

Before: Detection of an unknown protocol

ply a display filter <\$€/>												
		Destination	Protocol	Length								
1 2019-02-01 04:21:25.264646			ECAT							72 <b>,</b> 'LRD'		
2 2019-02-01 04:21:25.264921 1			ECAT							72, 'LRD'		,
3 2019-02-01 04:21:25.266534 /			ECAT							36, 'LRD'		
4 2019-02-01 04:21:25.266713 M			ECAT							06, 'LRD'		
5 2019-02-01 04:21:25.269667 1			ECAT							72, 'LRD'		
6 2019-02-01 04:21:25.269915 N			ECAT							72, 'LRD'		
7 2019-02-01 04:21:25.271531 1			ECAT							06, 'LRD'		
8 2019-02-01 04:21:25.271733 /			ECAT							06, 'LRD'		
9 2019-02-01 04:21:25.274713	15–NLB–PhysServ…	Beckhoff_01:00:00	ECAT	1345	4 Cmds,	'BRD':	len 2,	'LWR';	len 57	72, 'LRD'	: len	688, '
10 3010 03 01 01-31-35 374001 1	AC NI P. Obuc Corry	Packhoff 01.00.00	ECAY	1246	A Code	10001.	100 7	111/01.	100 51	D 11 001	100	<u> </u>
2.10 Virtual LAN, PRI: 0, DEI: 0, ID:	22				0010	89 05	57 fc	f0 5e 4	b 11	88 a4 21	15 07	96 01
isco Nexus										01 00 Ob		
802.10 Virtual LAN, PRI: 2, DEI: 0, I	D: 2833				0030					00 00 00		
010 = Priority: Exc		)			0040					00 10 02		
0 = DEI: Ineligit					0050 0060					24 00 c4 00 00 00		
1011 0001 0001 = ID: 2833					0000			10 12 0		00 10 02		
Type: EtherCAT frame (0x88a4)					0080					24 00 c4		
EtherCAT frame header					0090					00 00 00		
101 0010 0001 = Length: 0x521	1				00a0	00 00	08 00	10 40 0	2 00	00 00 00	a0 cc	00 cc
0 = Reserved: Val					00b0					00 00 cc		
0001 = Type: EtherCA					00c0					c4 38 08		
EtherCAT datagram(s): 4 Cmds, 'BRD':		572. 'LRD': len 68	8. 'LRD': le	n 3						70 01 70		
<pre>v EtherCAT datagram: Cmd: 'BRD' (7),</pre>					00c0 00f0			0f 00 0		0f 00 0f 08 09 00		
▶ Header					0100					08 09 00		
► AL Status (0x130): 0x0008, Al Sta	atus: OP				0110					08 09 00		
Working Cnt: 1					0120			00 00 c		08 09 00		00 01
<pre>v EtherCAT datagram: Cmd: 'LWR' (11),</pre>	Len: 572. Addr	0x1000000. Cnt 0			0130					00 00 00		
▼ Header					0140	c4 38	8 08 09	00 00 e	4 06	08 c0 28	00 68	00 00
Cmd : 11 (Logical Write					0150	54 ae	08 00	00 00 0	0 00	04 20 08	08 20	00 00
Index: 0x00					0160					e4 06 08		
Log Addr: 0x01000000					0170					00 00 04		
▶ Length : 572 (0x23c) - No	Roundtrin - More	Follows			0180					08 00 00		
Interrupt: 0x0000	nounacrap - nore				0190					80 38 10		
Data: 33005c010000004002200d0404	8864391812888888	00200000002400000000	1002200d18		01a0 01b0					05 00 80 03 00 18		
Working Cnt: 0	00010120000001	0010000001000000000	TOOLLOUGIO		0100					14 80 00		
EtherCAT datagram: Cmd: 'LRD' (10),	Len: 688. Addr	Ax1000800. (nt 0			01d0					42 99 14		
<ul> <li>EtherCAT datagram: Cmd: 'LRD' (10),</li> <li>EtherCAT datagram: Cmd: 'LRD' (10),</li> </ul>					01e0			00 00 0		8e 02 03		
Pad bytes: d1304373	Leni 3, Addr 0X.	LOOLOOD, CHIL U			01f0					20 00 ac		

After: Final dissection of the unknown protocol's inner structure

#### Using the Cisco Nexus Protocol to Create the Cisco Nexus Dissector Plugin

In this article, we showed how to easily manage known and unknown layer 2 frames using Lua APIs, by instructing Wireshark to properly dissect them while the analysis is being done. To illustrate this, Nozomi Networks researchers used a real-world example of a previously unknown protocol: Cisco Nexus.

During the background analysis, the team found that the proprietary protocol encapsulated a well-defined communication structure. We then leveraged this knowledge to test the following combinations of DissectorTable functions:

- .get(v1)
- .get(v1):get\_dissector(v2)
- .get(v1):get\_dissector(v2):call(v3,v4,v5)

The outcome of our research is a plugin called the Cisco Nexus Dissector. We've posted it in <u>GitHub</u> to help asset owners troubleshoot activities within their own networks. The global security community can also use it to further their analysis and research projects.

Next month, Nozomi Networks Labs will investigate how to create a plugin for another unknown protocol found in a commonly-used industrial communications equipment. Stay tuned!

#### **Related Content**



### TRITON: The First ICS Cyber Attack on Safety Instrument Systems

Understanding the Malware, Its Communications and Its OT Payload

Alessandro Di Pinto, Younes Dragoni, Andrea Carcano Black Hat USA 2018 - Research Paper

RESEARCH REPORT

#### TRITON: The First ICS Cyber Attack on Safety Instrument Systems

#### Understanding the Malware, Its Communications and Its OT Payload

#### Learn about:

- How the TRITON cyberattack unfolded and why it's so important
- Understanding TRITON through reverse engineering
- The implications of TRITON on industrial control systems security
- Two free community tools that help you defend against TRITON

#### Download Deleted Lin

#### **Related Links**

- Research Report: Nozomi Networks Labs OT/IoT Security Report
- Github.com: <u>Nozomi Networks Dissectors: Cisco Nexus</u>
- Blog: Demonstrating the Link Between Functional Safety and ICS Security
- Blog: <u>Colonial Pipeline Ransomware Attack: Revealing How DarkSide Works</u>
- Blog: Black Hat: Understanding TRITON, The First SIS Cyber Attack
- Blog: <u>The Clever Use of Post Dissectors to Analyze Layer 2 Protocols</u>
- Webpage: Nozomi Networks Labs
- Data Sheet: <u>Threat Intelligence</u>



Security Researcher

Younes Dragoni is a member of the World Economic Forum's Global Shaper Community, a worldwide network of young people actively shaping our future through solution building, policy-making and lasting change. His fascination with computer security, and desire to be on the offensive side, began many years ago. Now, as Security Researcher with Nozomi Networks, Younes thrives on hunting down vulnerabilities in automation devices (ICS/SCADA) and examining malicious software to understand the nature of threats to industrial operations.