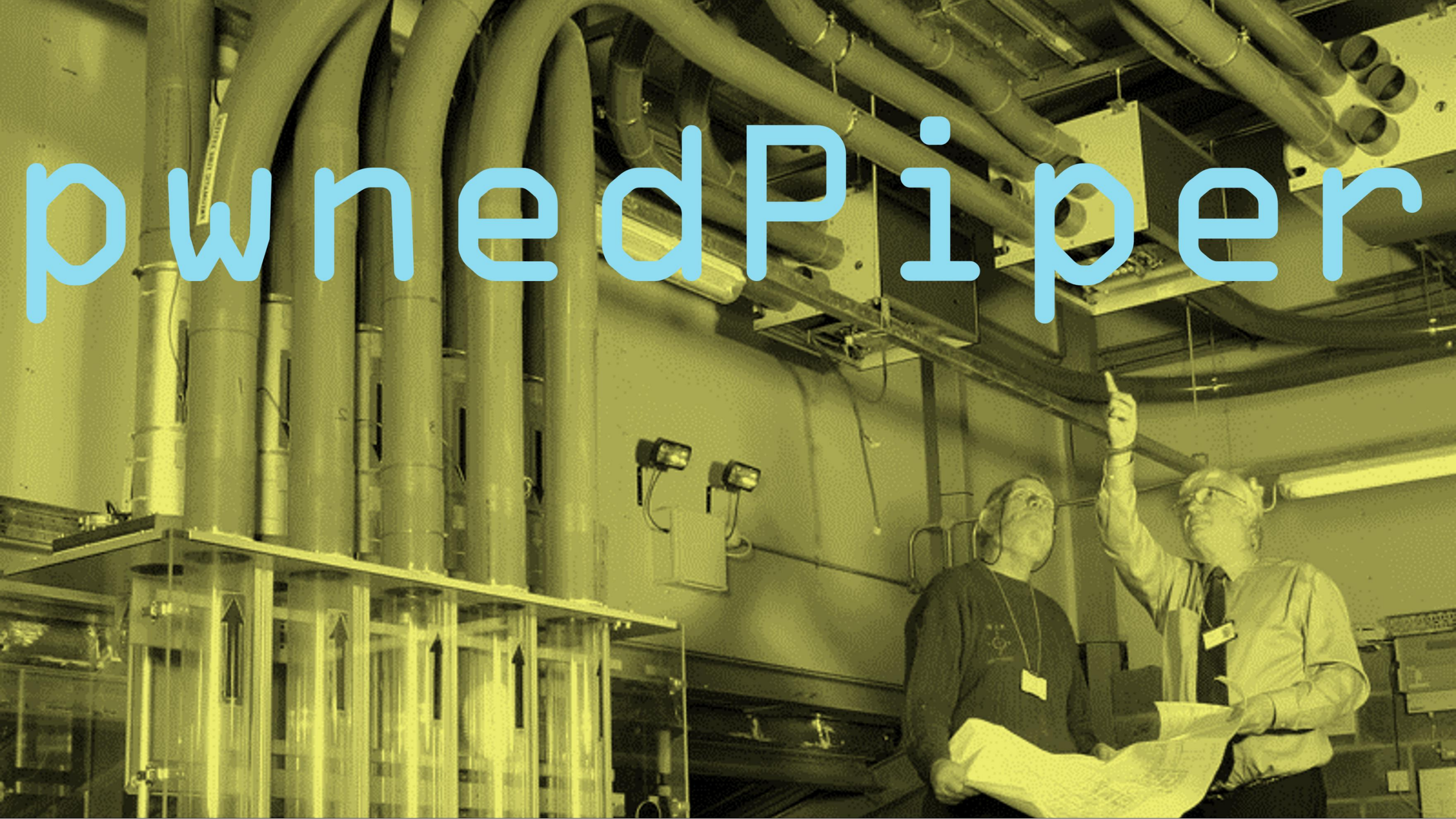
A Hole In The Tube

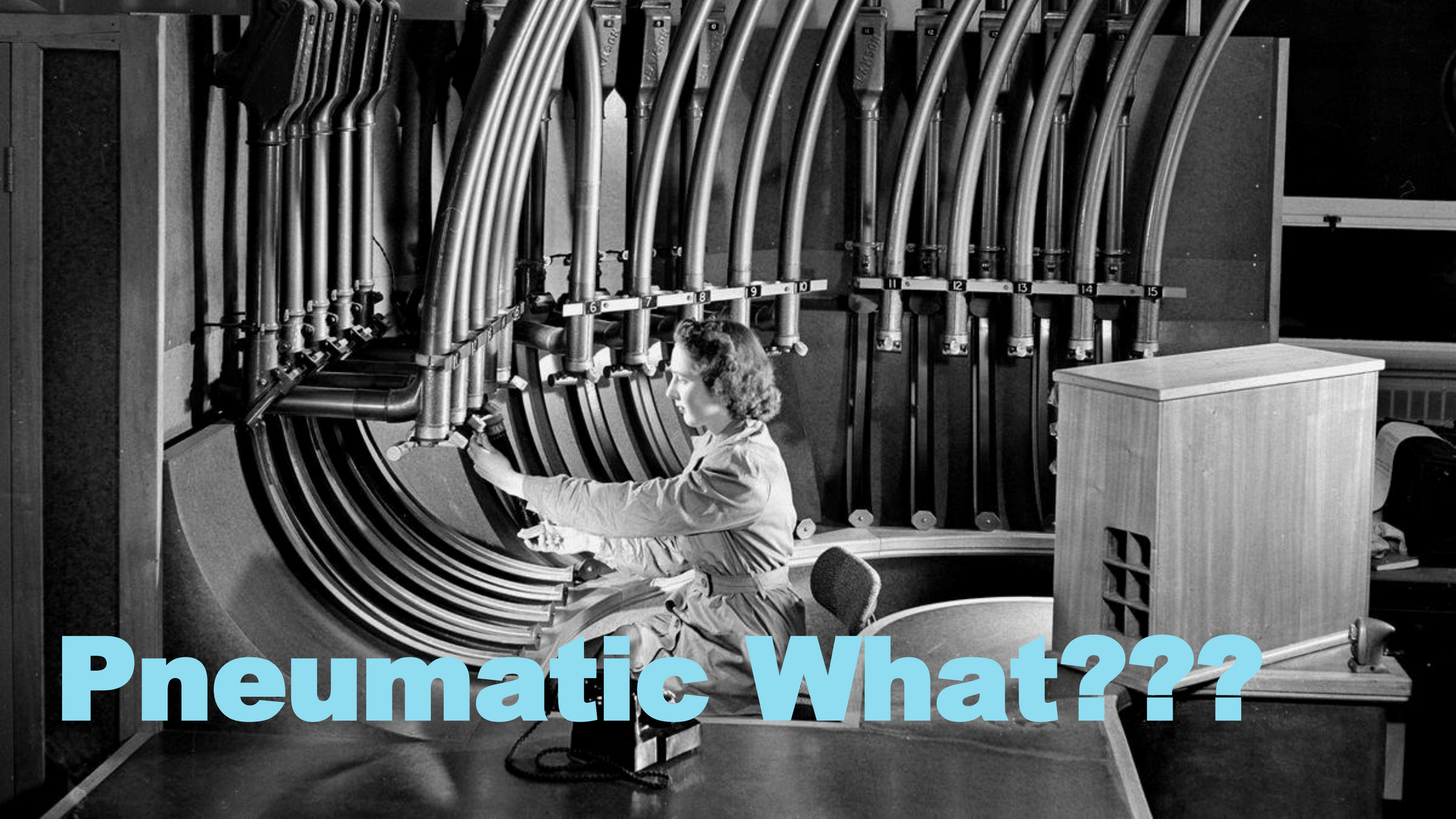
Uncovering Vulnerabilities in Critical Infrastructure of Healthcare Facilities

Ben Seri – VP Research

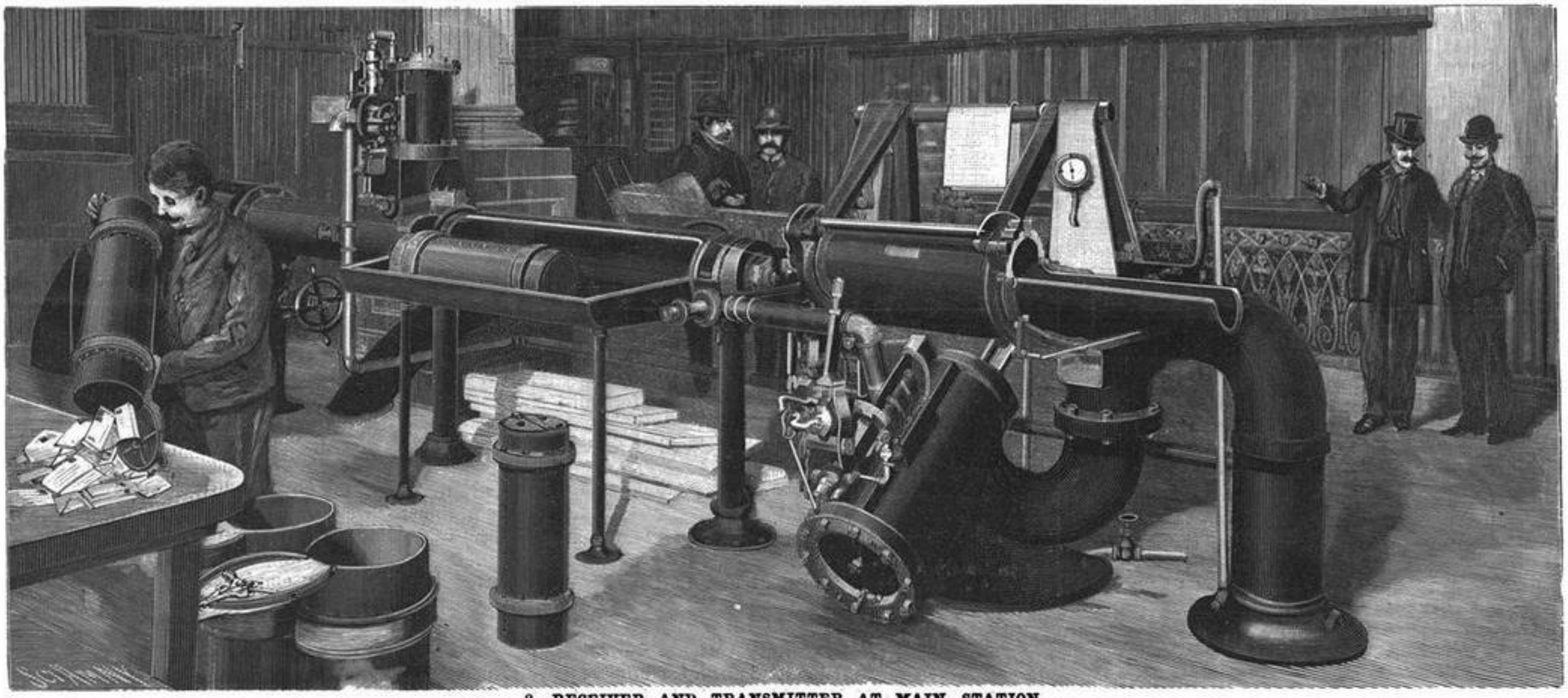
Barak Hadad – Security Researcher

ARMIS™


pwnedPiper

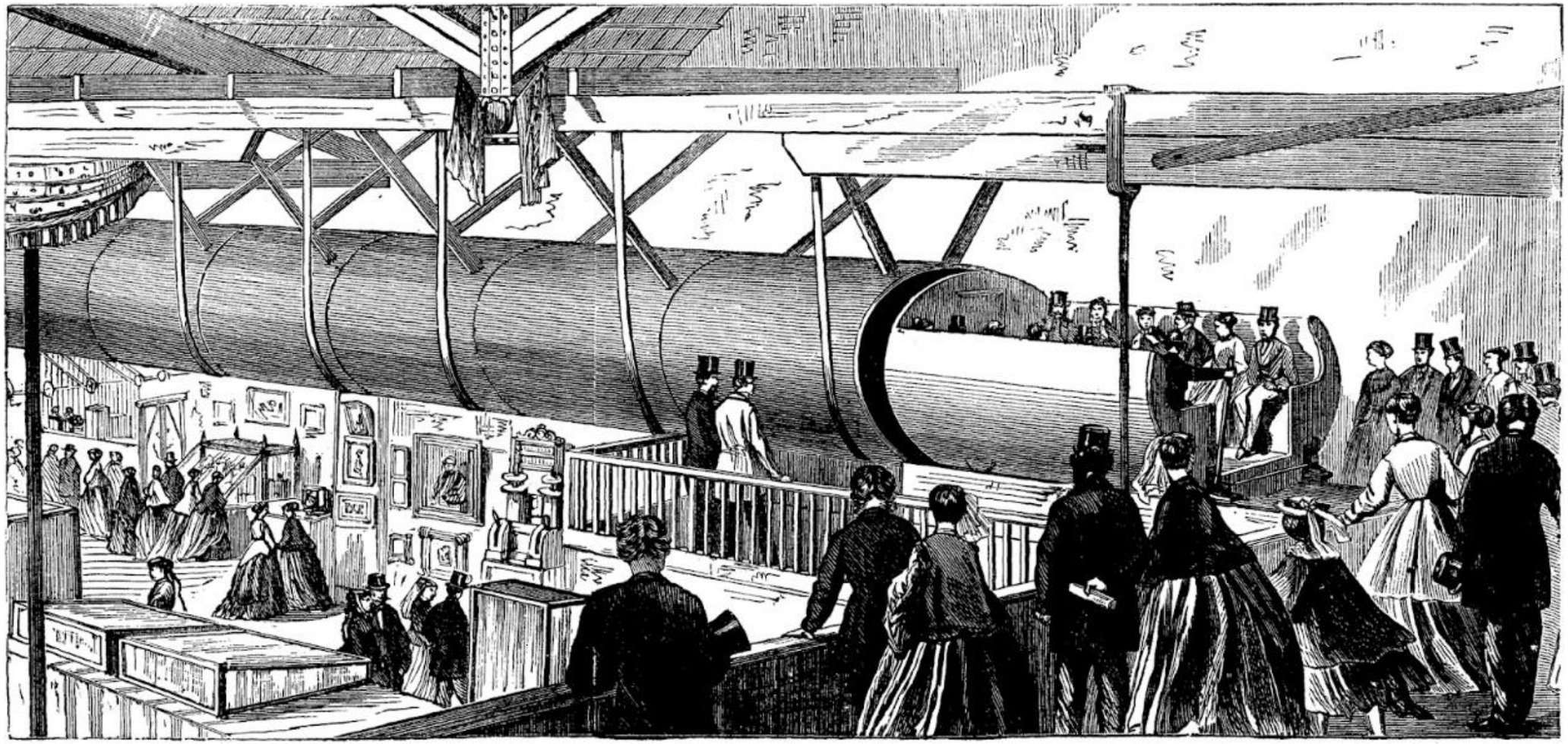

Pneumatic What???

Pneumatic Tubes are NOT new



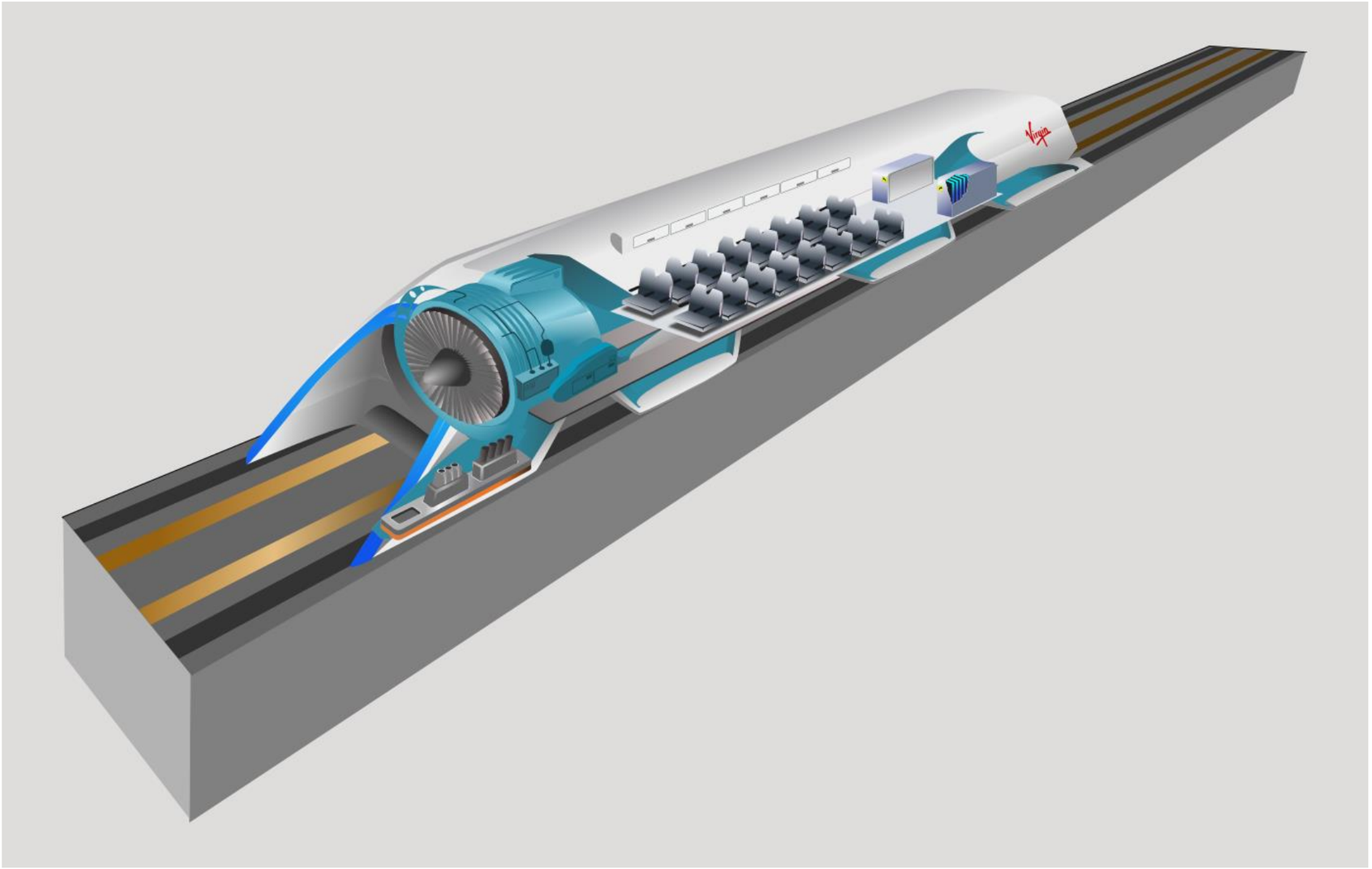
3.—RECEIVER AND TRANSMITTER AT MAIN STATION.

Pneumatic Tubes are NOT new

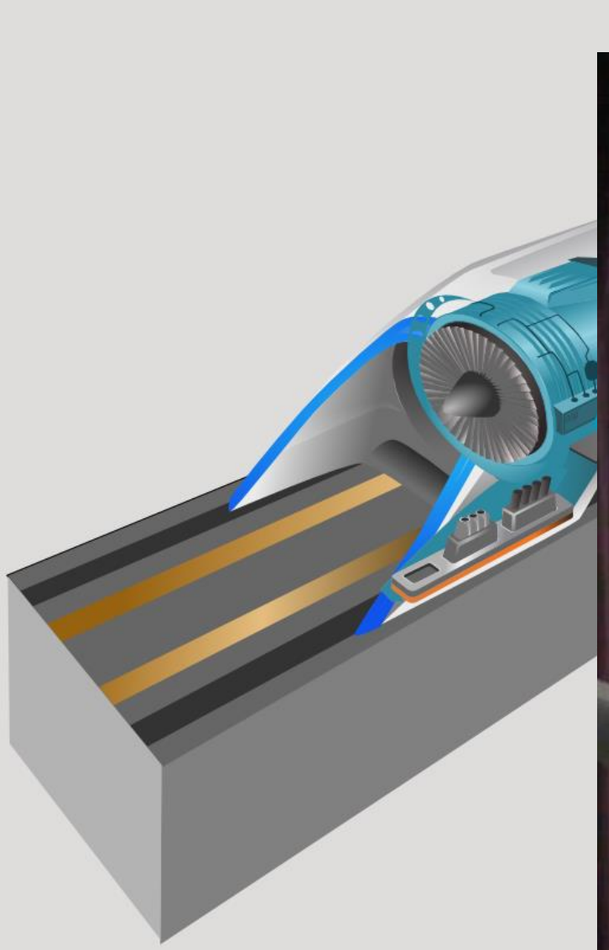


Pneumatic Tubes are new?

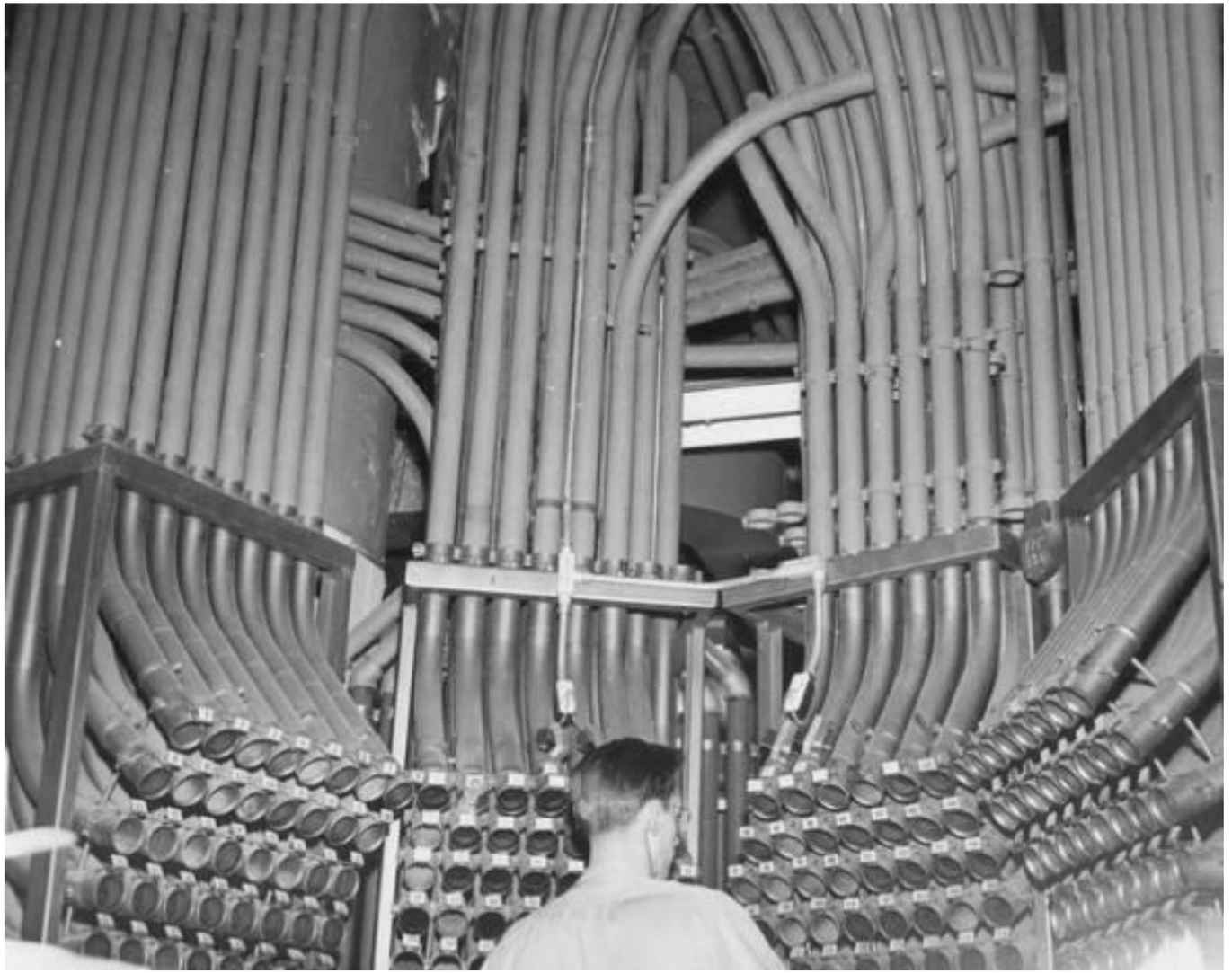
Hyperloop



Pneumatic Tubes are new?

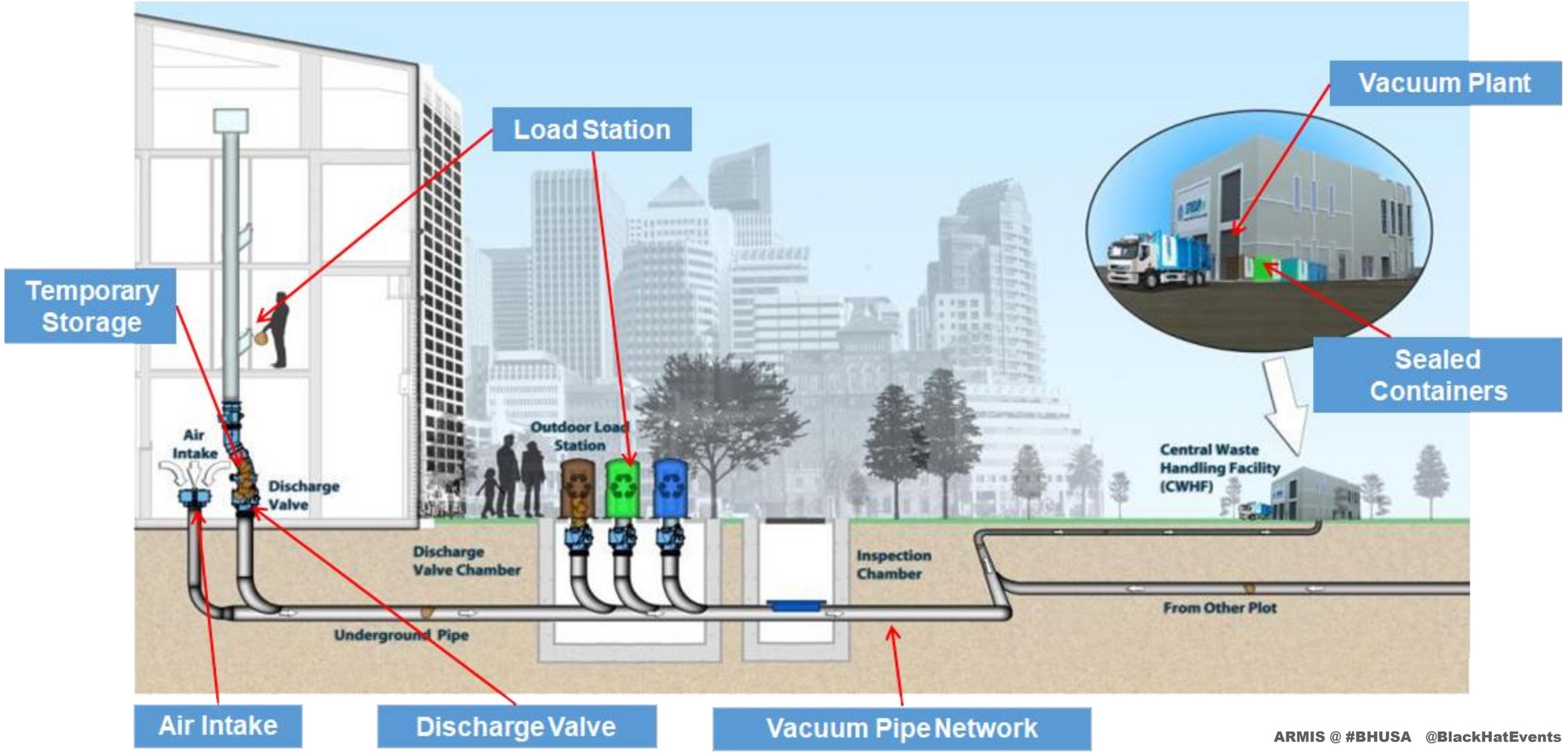


Inter-office messaging

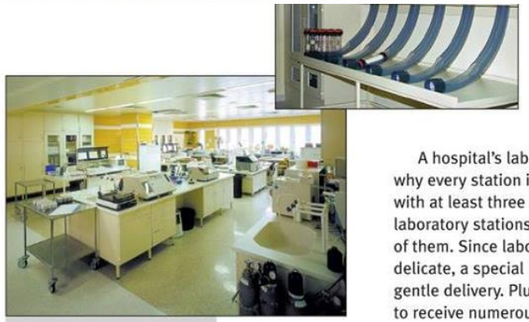


Inter planetary messaging ?

Pneumatic Tubes - The Future of Waste?



In Hospitals, PTS is a critical infrastructure



THE **LABORATORY STATION**

A hospital's laboratory is a busy place. This is why every station in the laboratory is provided with at least three dispatch magazines, large laboratory stations even require six, nine or more of them. Since laboratory samples are often delicate, a special mechanism guarantees a gentle delivery. Plus, conveyor belts are installed to receive numerous carriers. The empty carriers are returned automatically without manual destination selection.

Automatic stations with varying applications

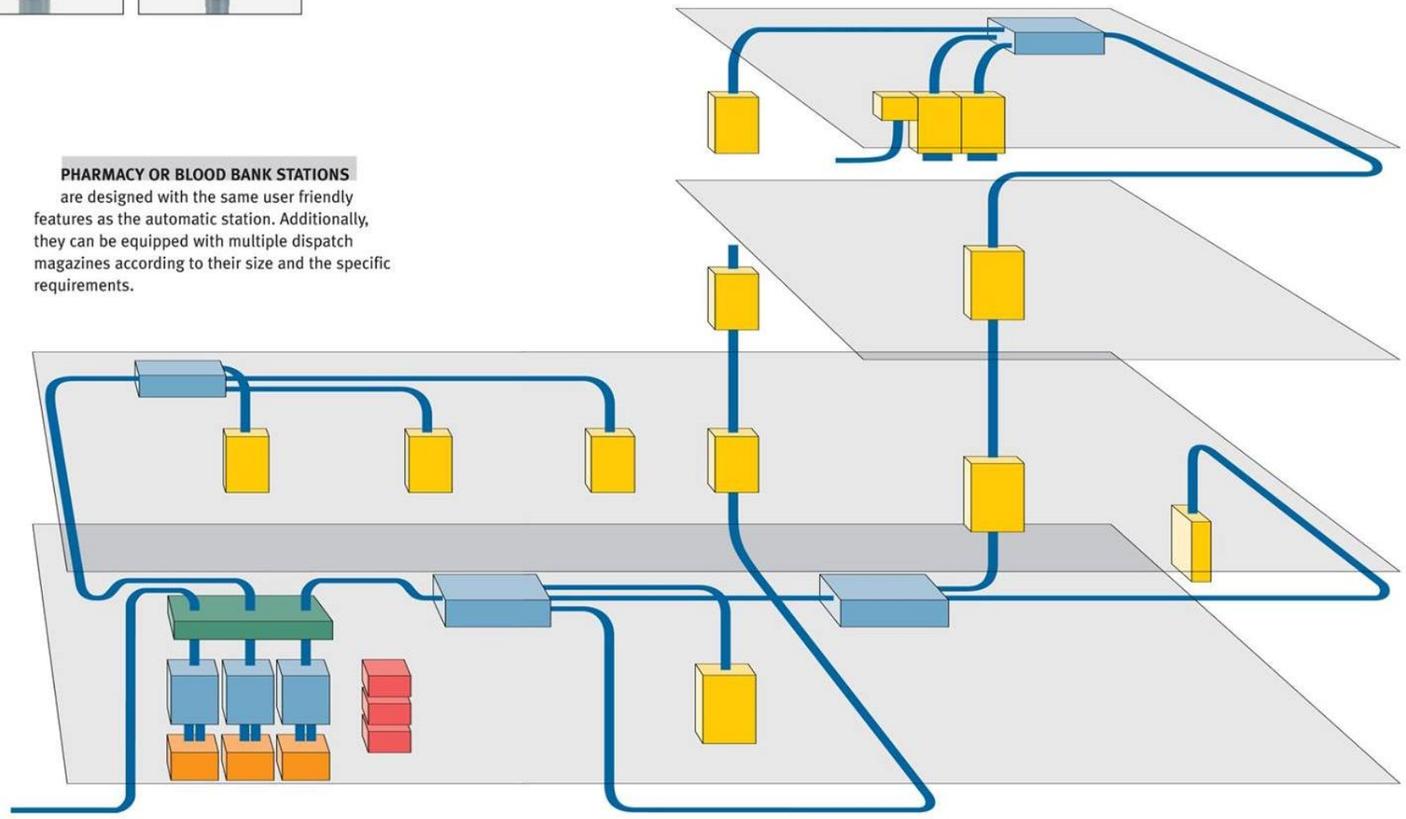
Nurse's checkpoint with automatic station

THE **AUTOMATIC STATION**

shows a series of user friendly features such as dispatch magazine, destination selection key, index of names and plaintext display. In order to protect the transport load, for instance when transporting laboratory samples, an air cushion softly slows the carrier down. Its arrival in the station is announced automatically by a signal. After removing the contents, the automatic destination selection system facilitates the return of the empty carrier.

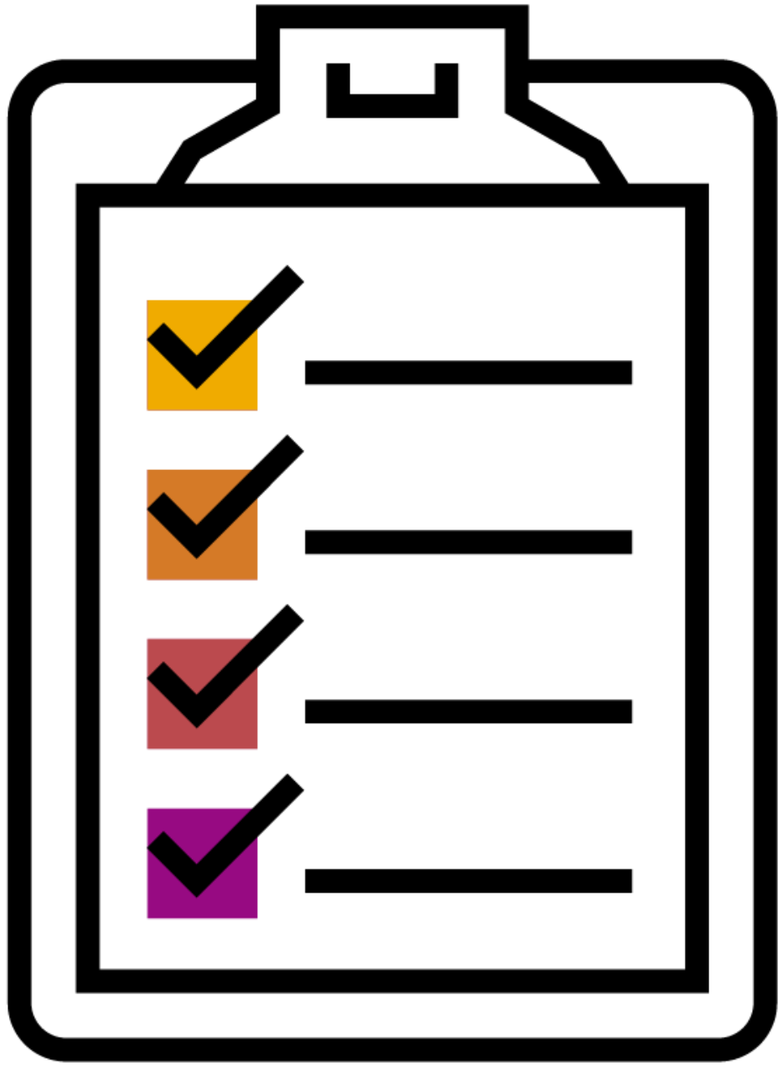
PHARMACY OR BLOOD BANK STATIONS

are designed with the same user friendly features as the automatic station. Additionally, they can be equipped with multiple dispatch magazines according to their size and the specific requirements.



Agenda

- PTS system architecture & components
- Swisslog TransLogic Devices
- Vulnerabilities
- Exploitation
- Demo!
- Final Thoughts



pwnedPiper – Overview

- 9 vulnerabilities discovered in Swisslog's Translogic Pneumatic Tube System
- Critical vulnerabilities were found in the Nexus Station – A prominent PTS station by Swisslog:
  - Hardcoded Passwords, Privilege Escalation, Heap & Stack overflows (can lead to RCE), DoS, and non-secure firmware upgrade mechanism
- All vulnerabilities can be triggered via unauthenticated network packets, without any user-interaction
- Disclosed to Swisslog on May 1, 2021, working together to patch & test

swisslog

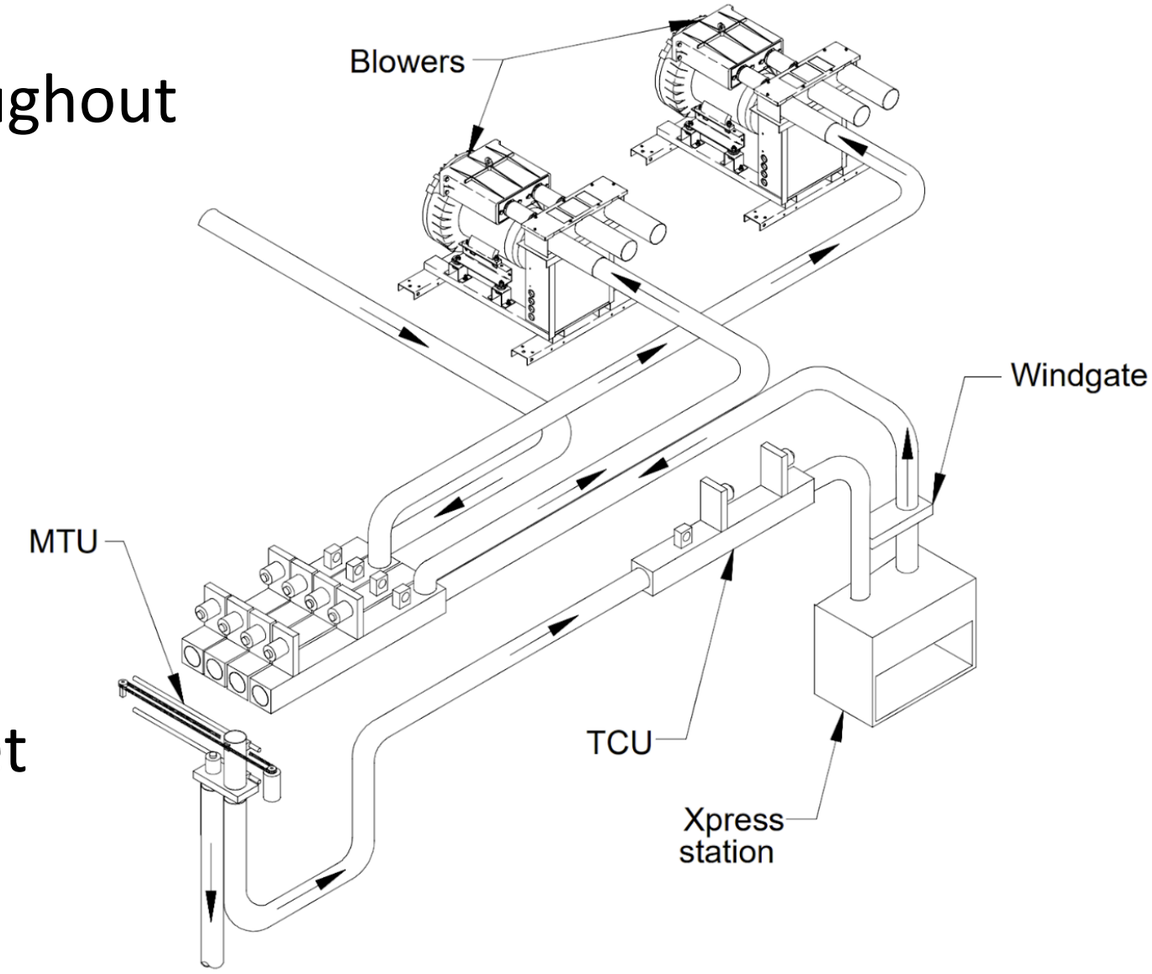
Swisslog TransLogic - The Leading PTS Sytem

- TransLogic is installed in more than 2,300 hospitals in North America and over 3,000 worldwide.
- The majority of hospitals in North America use Swisslog TransLogic as their PTS solution
- TransLogic is one of the most advanced PTS systems in the market, supports high-load, advanced features, reliability and even physical-security features

swisslog

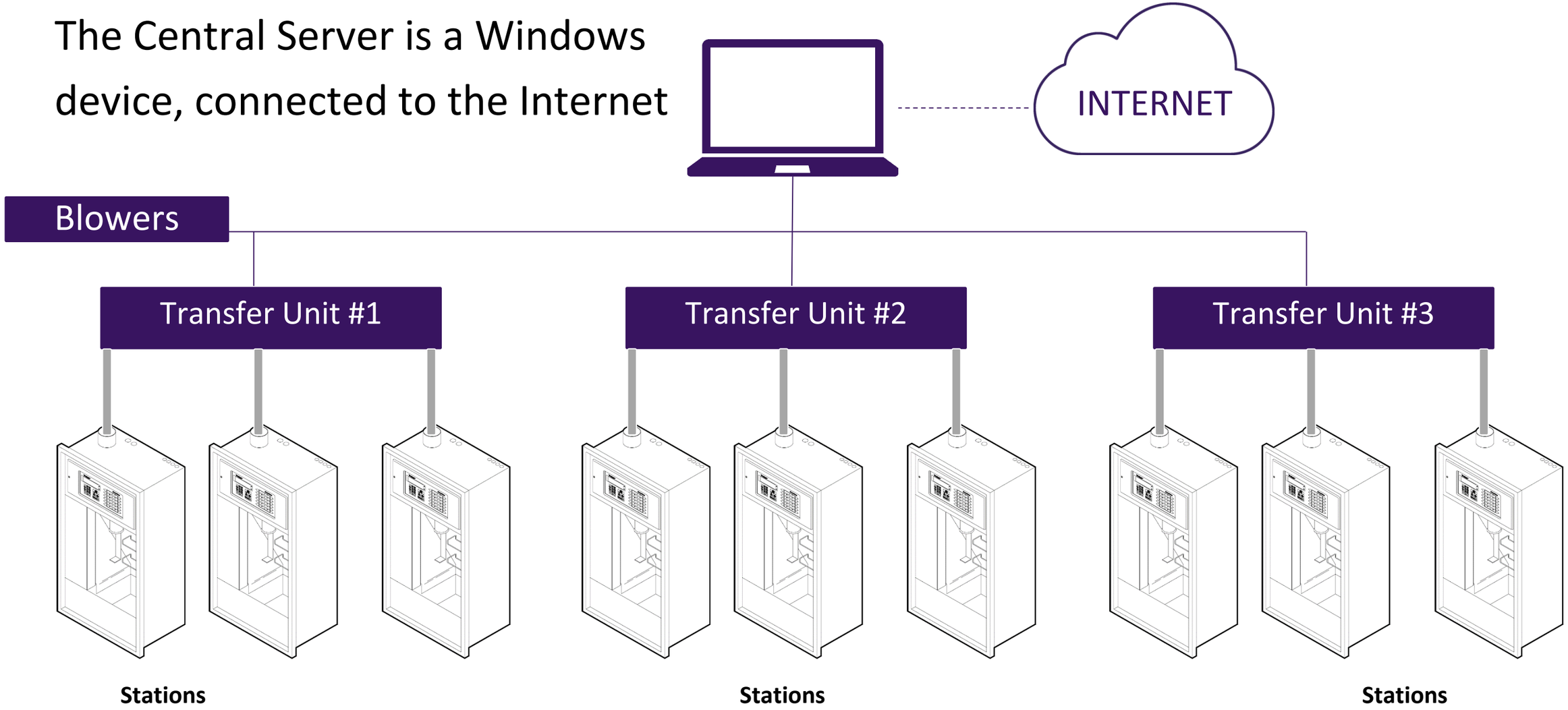
PTS systems are complex analog networks

- PTS systems transfer physical carriers throughout hospitals using a complex network of:
  - Tubes
  - Blowers
  - Transfer Units (Routers)
  - Stations
- The entire system is managed over Ethernet by a central server



Pneumatic Tube System – IP-connected

The Central Server is a Windows device, connected to the Internet

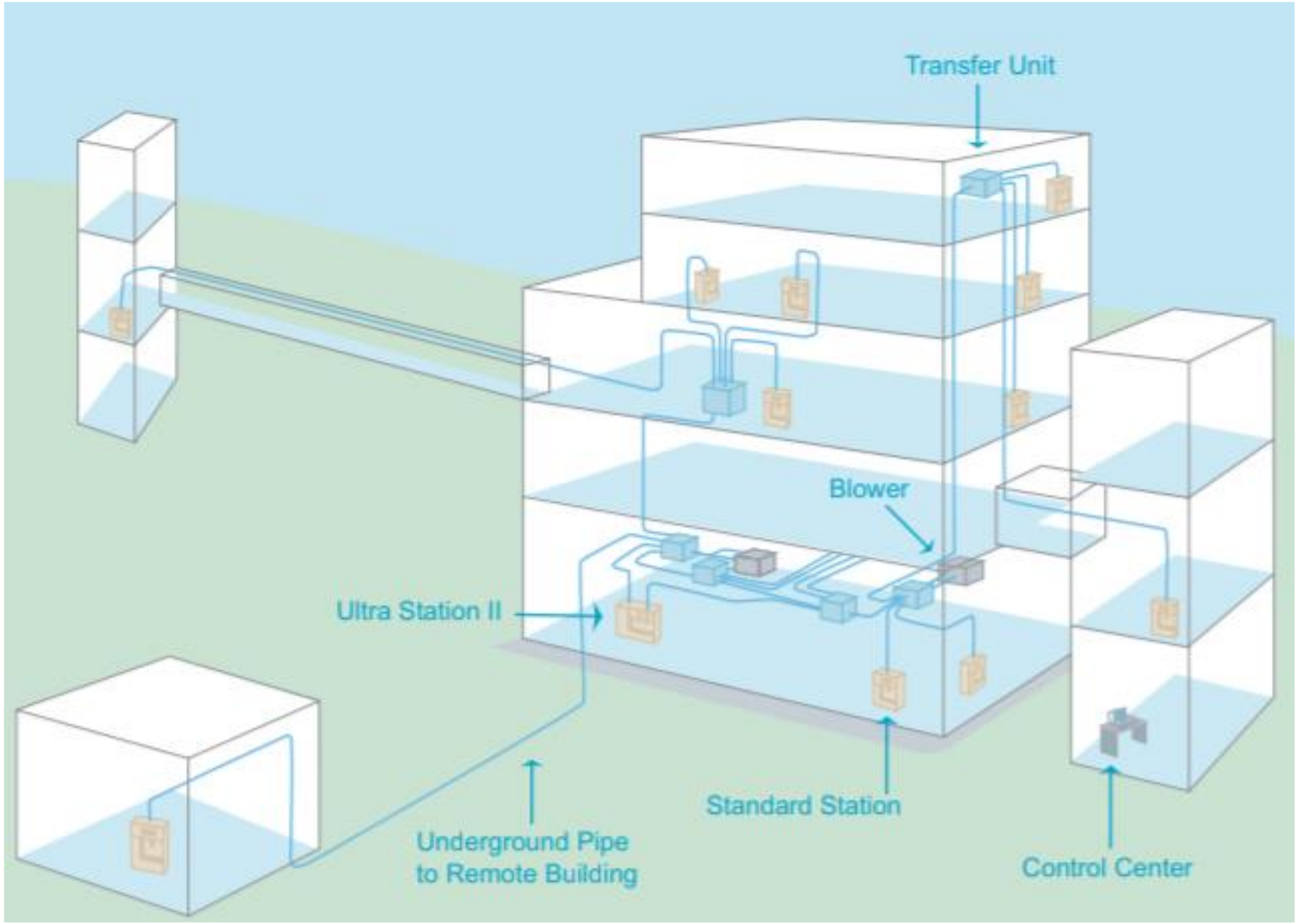


Potential Affect of cyber attacks on PTS systems

- Takeover of PTS stations can result in various attacks

- DoS of the PTS network
- Information leak of PII (staff records, RFID credentials, etc.)
- Sophisticated Ransomware\Sabotage attacks:
  - Re-routing carriers can derail hospital operations significantly

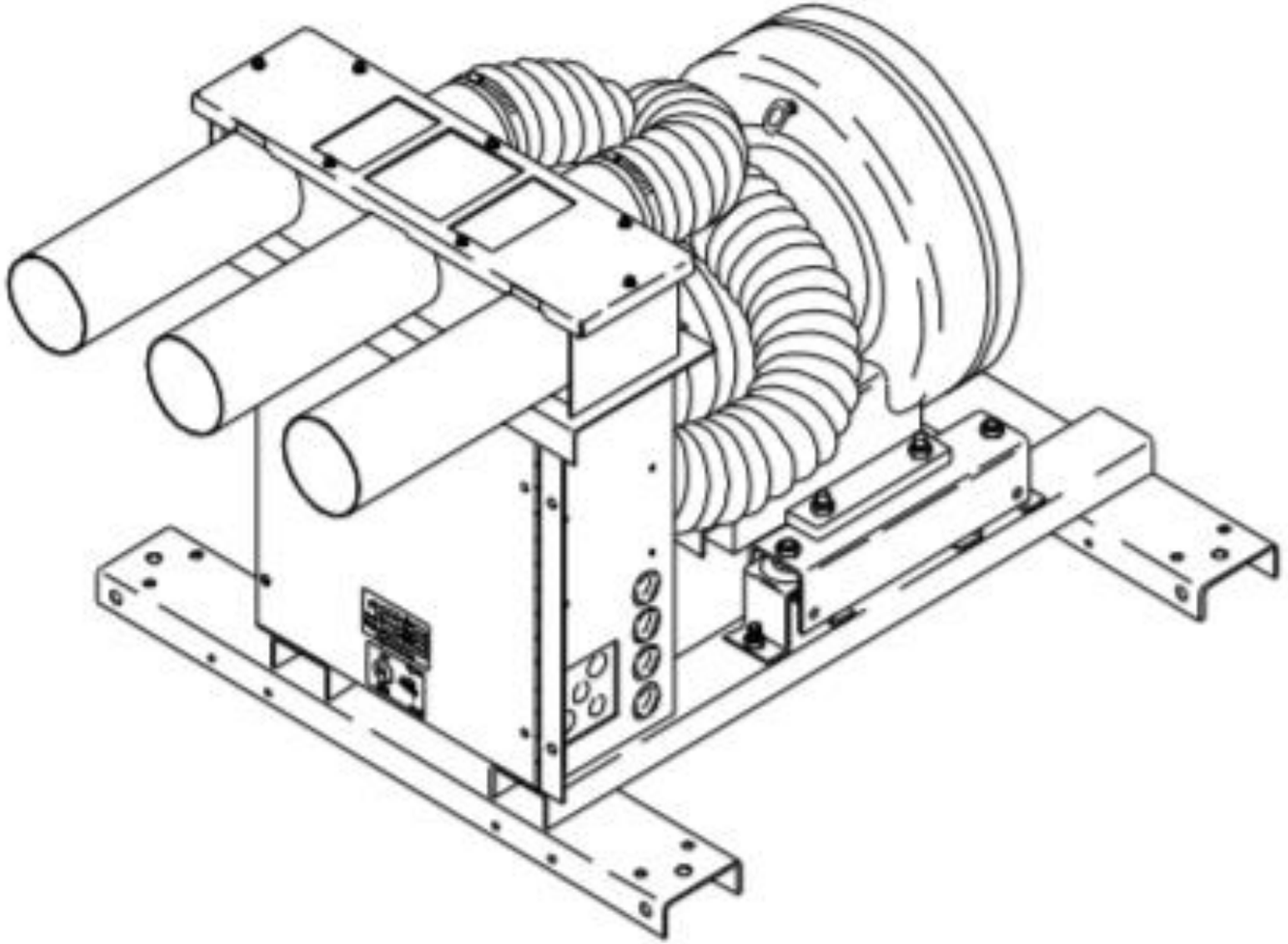
Design and structure of the PTS system



Stations

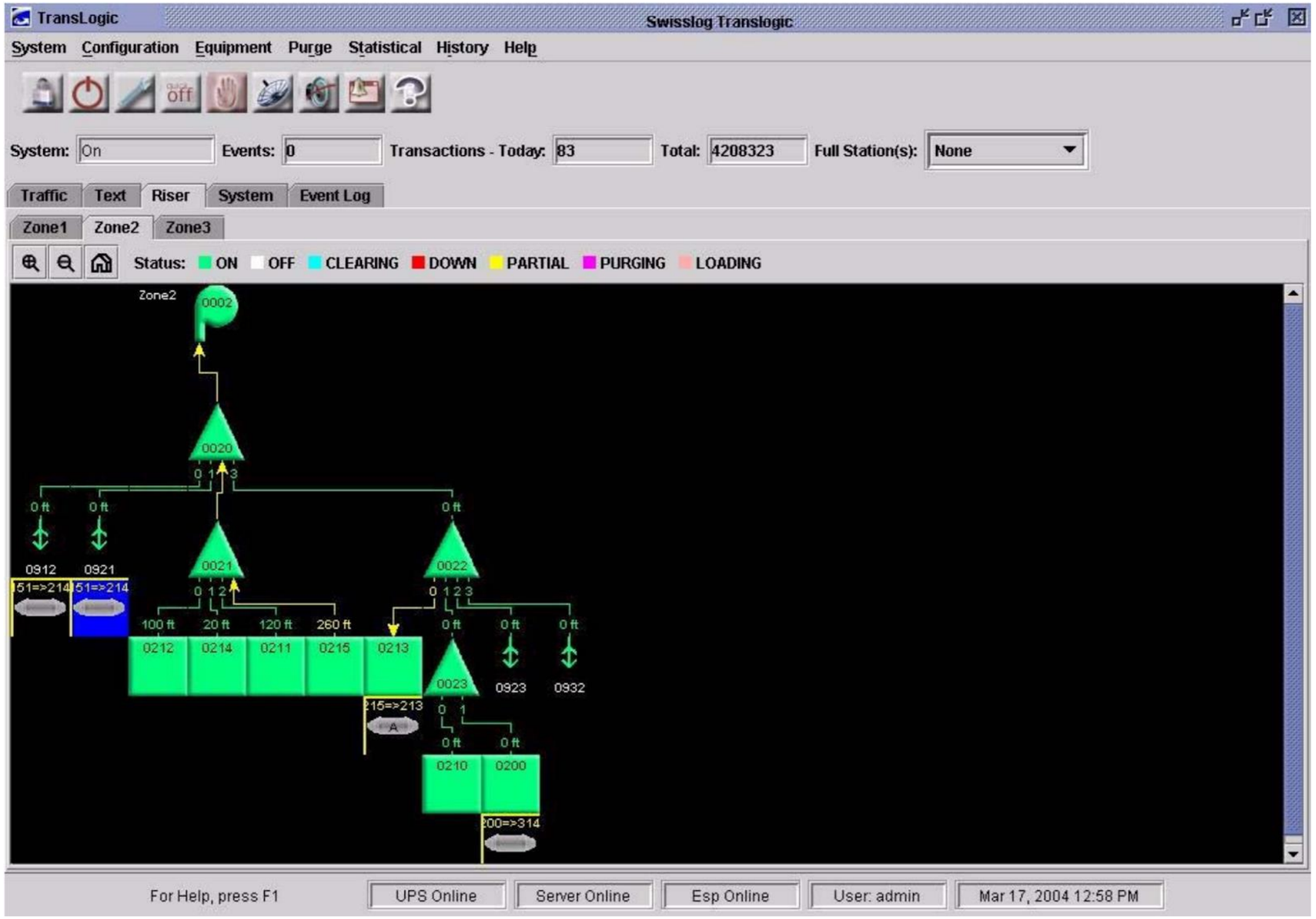

swisslog
Send Keypad
Speed Dials
Features
Directory
Remote Indicators
400
1 2 3
4 5 6
7 8 9
0
Send Normal
Send Secure
Send Stat
Send Empty
STATION
400
Return surplus carriers

Blowers

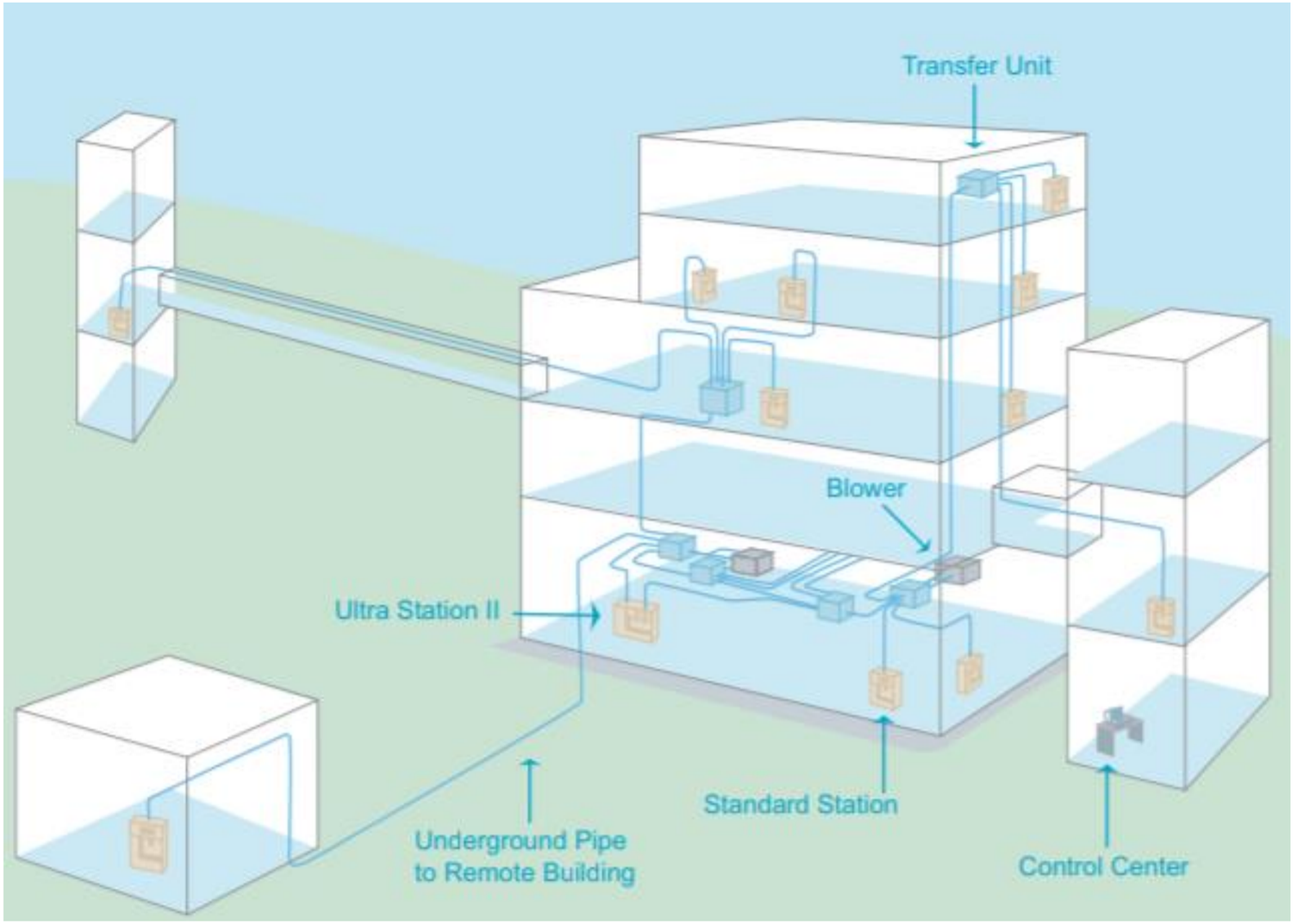


Diverters

Central Management Server (SCC)



Design and structure of the PTS system



Swisslog Translogic PTS – A "Next-gen" PTS with advanced features

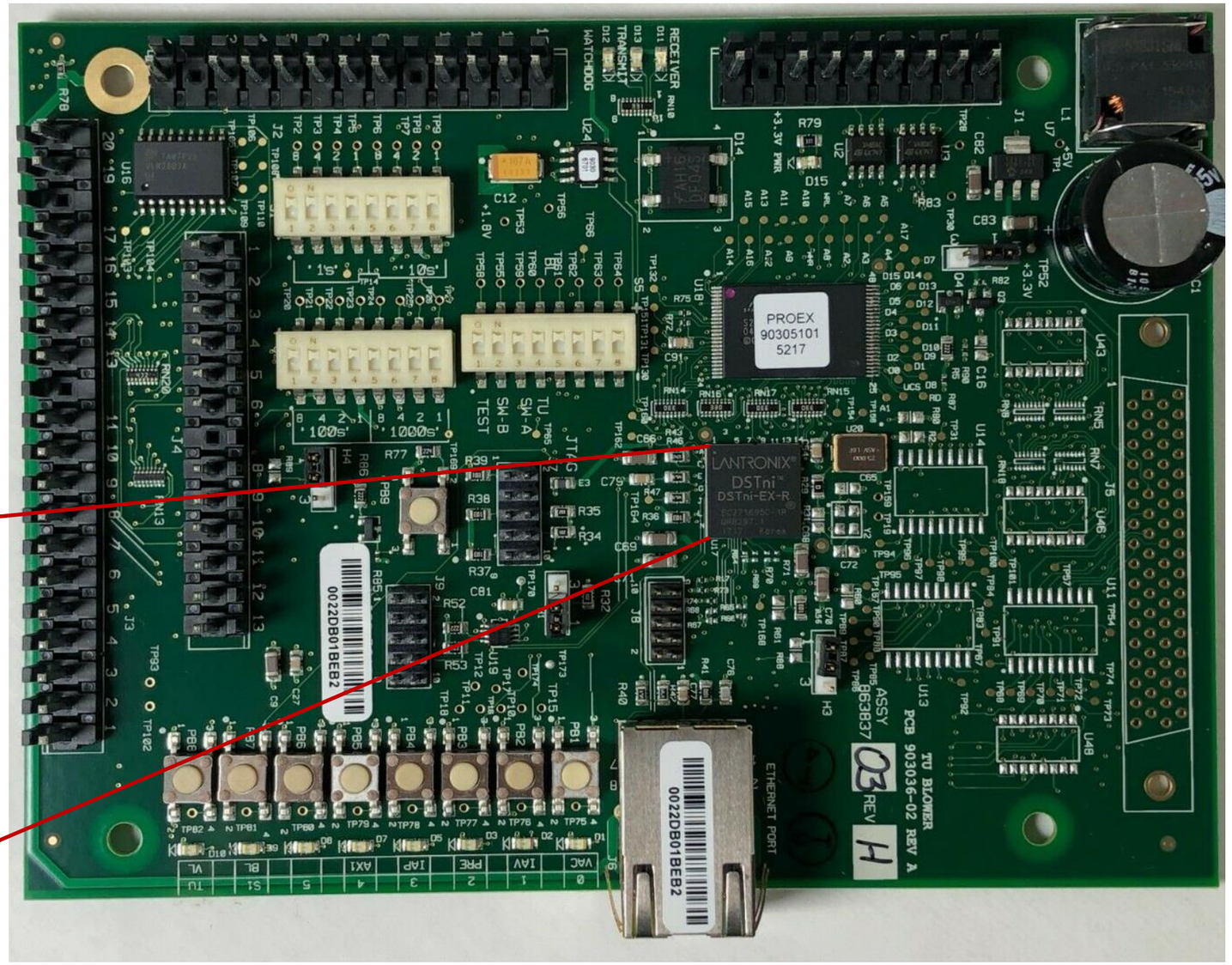
- Secure transfers, with RFID and/or password-protected carriers
- Slow-speed transfers, for sensitive cargo
- Internet connected Alert system, for user notifications via email/text/etc
- Remote system monitoring, for offloading the maintenance of the system to the Swisslog Cloud

TransLogic Legacy Stations

- CTS 30 Station
- IQ Station
- Supports serial connection (RS-422) or Ethernet (in newer models)

IQ Station

- Has Ethernet connection
- Uses 8086 16-bit MCU (DSTni-Ex)
- Firmware is non-encrypted and unsigned…



IQ Station

- Firmware upgrade requires a physical switch change:

Table 2.1 : S1 DIP Switch Modes

Mode	Function
Normal operation mode	Sets the station for normal operation. This is the default mode.
Reset ID mode	Clears all user-defined settings, including station ID, speed dials, audio level, and display contrast level. Resets the administration password to "1234" and enables the default user functions.
Download mode	Prepares the station to download program files when using the remote download kit or to download program files from the system control center.
Reset ID/download mode	Clears all user-defined settings and sets the station to download mode.

Nexus Station

- High-end station with touchscreen and RFID
- IP-connected, runs Linux **v2.6**
- 32Bit ARM CPU
- Two main processes:
- HMI3 – ELF containing the low-level operation of the station
- HMI3.jar – Responsible for the GUI and high level operations

HMI3

- Not PIC so no ASLR for the main elf
- No stack canaries, and no DEP for the bss (?)
- Compiled with debug symbols

```
arch     arm
baddr    0x8000
binsz    137191
bintype  elf
bits     32
canary   false
class    ELF32
compiler GCC: (4.4.4_09.06.2010) 4.4.4
crypto   false
endian   little
havecode true
intrp    /lib/ld-linux.so.3
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  ARM
maxopsz  16
minopsz  1
nx       false
os       linux
pcalign  0
pic      false
relocs   true
relro    partial
rpath    NONE
sanitiz  false
static   false
stripped false
subsys   linux
va       true
```

Swisslog PTS protocol

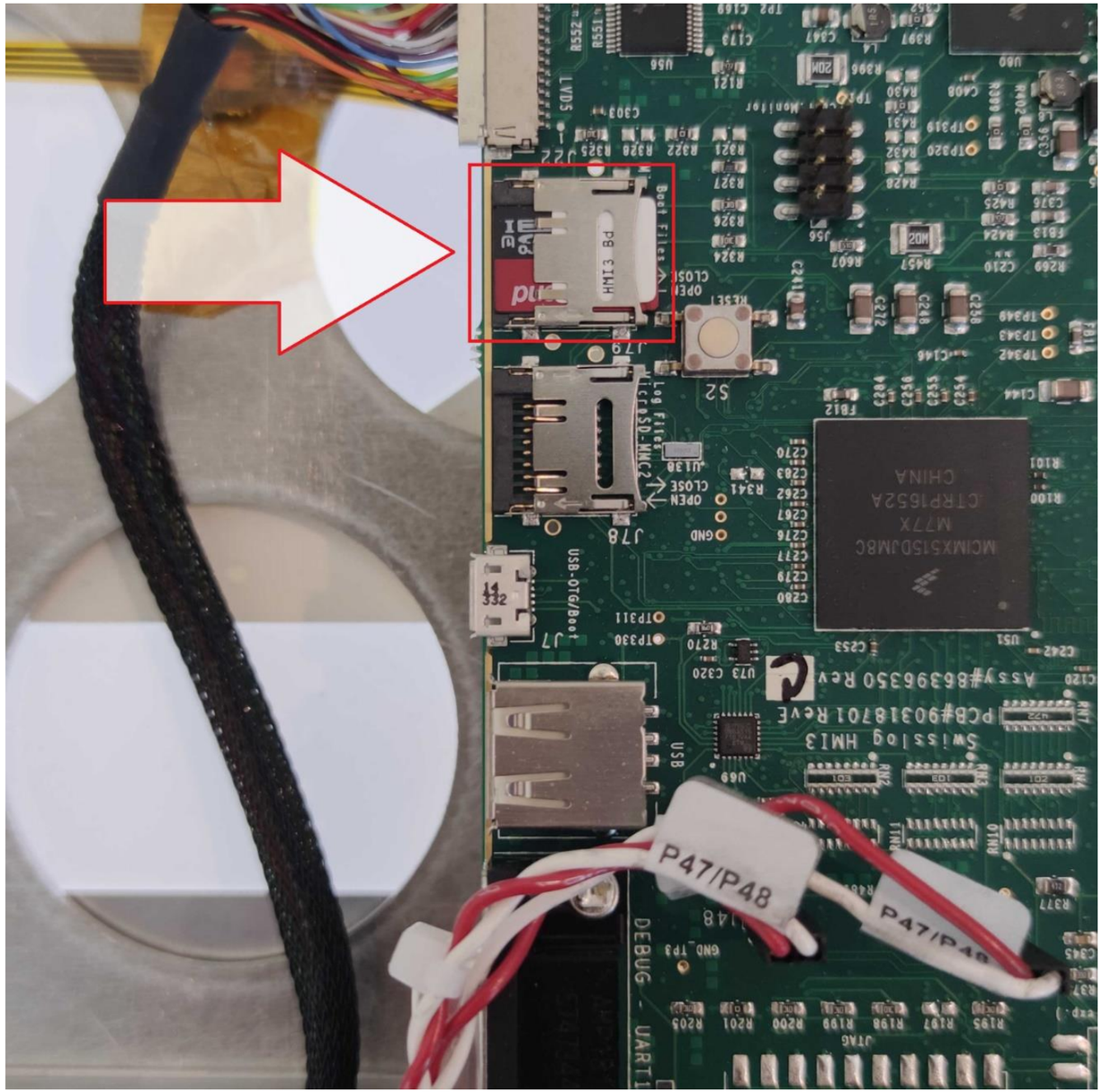
- 20 Bytes header
- Marker – “TLPU”

```
> User Datagram Protocol, Src Port: 65168, Dst Port: 12345
  <Wireshark Lua fake item>
v Swisslog PTS Protocol
    Magic: 0x544c5055
    SequenceNum: 0x00000c9d
    swisslog.op: Query (0x00000001)
  > Query
    data: 3001
```

```
0000  00 22 db 01 b7 7f 00 24  9b 30 3b 04 08 00 45 00   ·"·····$ ·0;···E·
0010  00 32 4b 12 40 00 7f 11  66 fc c0 a8 64 33 c0 a8   ·2K·@··· f···d3··
0020  64 28 fe 90 30 39 00 1e  a5 f7 54 4c 50 55 00 00   d(··09·· ··TLPU··
0030  0c 9d 00 00 00 01 00 00  00 03 00 00 00 00 30 01   ········ ......0·
```

Physical Attack Surface

An SD card containing the **non-encrypted, unsigned** firmware



Network Attack Surface

From the manual:

Network security

Most, if not all, site networks have access to the internet and/or outside networks that increase the possibility of a security breach or virus. Because the SCC has internet and network access, it should be provided with appropriate virus and security protection that falls within the requirements specified in this section.

The rest of the system is not vulnerable to attacks because the equipment uses a language that only the SCC can understand, thereby eliminating any network security concerns for the other PTS devices.

Security by obscurity is no security at all!

Internet Attack Surface

- The central management server connects outbound to the Internet.
- This connection allows various features such as alert notifications via the Alert System and remote monitoring and maintenance.
- Any vulnerability found in its proprietary code can lead an attack from the Internet to control the entire PTS system

Discovered Vulnerabilities

#1 & #2 Hard-coded passwords (yeah, that old trick)

```
ben@HerrBuntu:~/projects/research/swisslog/sd/etc$ cat shadow | grep "\$1"
root:$1                                    :0:0:99999:7:::
user:$1                                    :11851:0:99999:7:::
```

CENSORED


```
BN5


freescale login: user
Password:
user@freescale ~$ ls
hmi
user@freescale ~$ ls -l
drwxr-xr-x    5 user     user          4096 Aug 20  2019 hmi
```

#3 Privilege escalation

/home/user/hmi/run

- user writeable
- Executed by *root* (!)

```
user@freescale ~$ ps | grep run
 2143 root       3296 S    hald-runner
 2181 root       2272 S    /bin/bash /root/run-ccp
 2182 root       2272 S    /bin/bash /home/user/hmi/run
 2254 user       1764 S    grep run
user@freescale ~$ ls -l /home/user/hmi/run
-rwxr-xr-x    1 user     user           480 Jan  1  1970 /home/user/hmi/run
user@freescale ~$
```

#3 Privilege escalation

- Connect to the telnet server using the user “user” with the hardcoded password
- Edit “/home/user/hmi/run” to do whatever
- Reboot using the memory corruption vulnerability on the next slide
- …
- Profit!

```
user@freescale ~$ ps | grep run
 2143 root      3296 S    hald-runner
 2181 root      2272 S    /bin/bash /root/run-ccp
 2182 root      2272 S    /bin/bash /home/user/hmi/run
 2254 user      1764 S    grep run
user@freescale ~$ ls -l /home/user/hmi/run
-rwxr-xr-x    1 user     user           480 Jan  1  1970 /home/user/hmi/run
user@freescale ~$
```

#4 Underflow in udpRXThread (RCE)

```
void __noreturn udpRxThread()
{
        ...
        rec_len = recvfrom(udp_socket, buf, 370u, 0, &addr, &addr_len);
        ...
        q_buf_1 = Q_remove_block(freeQ);
        ...
        q_buf_1->data_len = rec_len - 20;
        q_buf_1->should_process_using_hmi = 0;
        memcpy(q_buf_1->data, &buf[20], rec_len - 20);
```

Bad memcpy (CVE-2020-6096)

[Bug libc/25620] Signed comparison vulnerability in the ARMv7 memcpy() (CVE-2020-6096)

fweimer at redhat dot com sourceware-bugzilla@sourceware.org
Wed Jul 8 12:22:20 GMT 2020

- Previous message (by thread): [Bug build/26217] Build of glibc 2.11.1 configure fails make too old but is GNU make 4.2.1
- Next message (by thread): [Bug libc/25620] Signed comparison vulnerability in the ARMv7 memcpy() (CVE-2020-6096)
- **Messages sorted by:** [date] [thread] [subject] [author]

---

https://sourceware.org/bugzilla/show_bug.cgi?id=25620

Florian Weimer <fweimer at redhat dot com> changed:

Bad memcpy (CVE-2020-6096)

```
_BYTE *__fastcall memcpy(_BYTE *dst, char *src, uint len)
{
  ...
  if ( (int)len >= 16 )
  {
    ...
  }
  if ( len & 8 )
  {
    ...
  }
  if ( len & 4 )
  {
    ...
  }
  ...
  return dst;
}
```

#4 Underflow in udpRXThread (RCE)

```
void __noreturn udpRxThread()
{
        ...
        rec_len = recvfrom(udp_socket, buf, 370u, 0, &addr, &addr_len);
        ...
        q_buf_1 = Q_remove_block(freeQ);
        ...
        q_buf_1->data_len = rec_len - 20;
        q_buf_1->should_process_using_hmi = 0;
        memcpy(q_buf_1->data, &buf[20], rec_len - 20);
```

```
1 void *__fastcall setHmiBuffer(q_buffer *a1, q_buffer *a2)
2 {
3   a1->data_len = a2->data_len;
4   return memcpy(a1->data, a2->data, (unsigned __int16)a2->data_len);
5 }
```

#5 Overflow in sccProcessMsg (RCE)

```
int __fastcall sccProcessMsg(q_buffer *a1)
{
  ...
  q_buffer *q_buff; // [sp+1Ch] [bp-8h]
  ...
  if ( a1->data[0]== 0x90 )
  {
    q_buff = Q_remove_block((q_buffer *)&freeQ);
    // if data_len is 0, it copies MAX_USHORT bytes
    q_buff->data_len = a1->data_len - 1;
    memcpy(q_buff->data, &a1->data[1], (uint16)q_buff->data_len)
    sendHmiMsg(q_buff);
    return 3;
  }
```

#6 GUI socket DOS in tcpServerThread

```
while ( 1 )
{
  hmiCommStatus = 0;
  printLog(4u, 0, "Waiting for TCP connection...");
  c_socket = accept(fd, &addr, &addr_len);
  if ( c_socket < 0 )
  {
    perror("<1>accept()");
    v3 = __errno_location();
    printLog(8u, *v3, "<1>TCP accept().");
  }
  printLog(4u, 0, "CCP accepted TCP socket.");
```

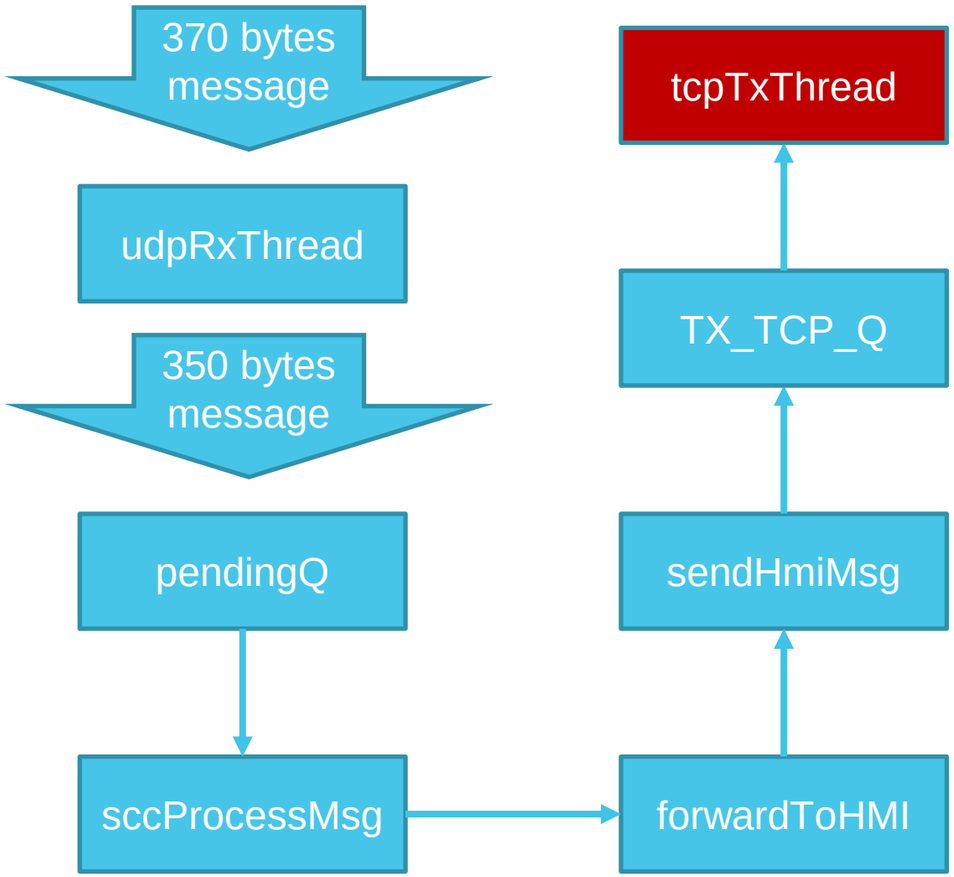
#6 GUI socket DOS in tcpServerThread

```
root@freescale ~$ tail -f ccp.log
01/01/1970 00:00:34.277 INFO: Worker count: 20
01/01/1970 00:00:34.278 INFO: Op-mode = APPLICATION
01/01/1970 00:00:34.292 INFO: Dipswitch: 0xff
01/01/1970 00:00:34.295 INFO: Main starting.
01/01/1970 00:00:34.300 INFO: TCP socket bind.
01/01/1970 00:00:34.301 INFO: Waiting for TCP connection...
01/01/1970 00:00:34.611 INFO: RFID not found.
01/01/1970 00:01:01.003 INFO: CCP accepted TCP socket.
01/01/1970 00:01:01.004 INFO: HMI comm good.
```

#7 Overflow in hmiProcessMsg (RCE)

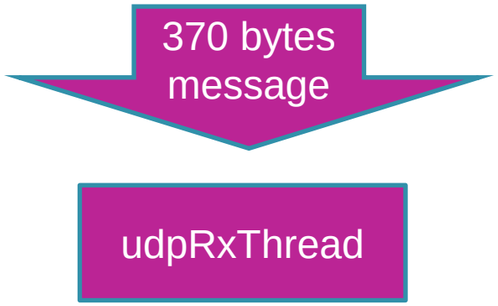
```
int __fastcall hmiProcessMsg(q_buffer *a1)
{
  q_buffer *v5; // [sp+14h] [bp-8h]
  ...
  If ( a1->data[0] == 0x33)
  {
    v5 = Q_remove_block((q_buffer *)&freeQ);
    // Overflow when a1->data_len == 0, data len is an unsigned short
    v5->data_len = a1->data_len - 1;
    memcpy(v5->data, &a1->data[1], (uint16)v5->data_len);
```

#8 Off-by-three stack overflow in tcpTxThread (RCE)



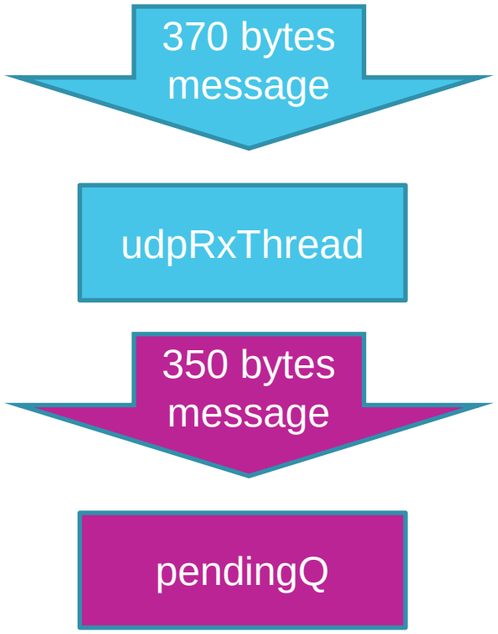

```
void tcpTxThread()
{
  char buf[352]; // [sp+18h] [bp-17Ch]
  q_buffer *buffer_to_send; // [sp+178h] [bp-1Ch]
  ...
  while ( 1 )
  {
    while ( 1 )
    {
      ...
      buffer_to_send = Q_remove_block((q_buffer *)&TX_TCP_Q);
      ...
    }
    buf[0] = 2;
    if ( buffer_to_send->op == 2 )
      ...
    else
      ...
      buf[2] = buffer_to_send->data_len;
      memcpy(&buf[3], buffer_to_send->data,
             (unsigned __int16)buffer_to_send->data_len);
      addCRC((int)&buf[1], buffer_to_send->data_len + 2);
      ...
  }
}
```

#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
while ( 1 )
{
  rec_len = recvfrom(udp_socket, buf, 370u, 0, &addr, &addr_len);
```

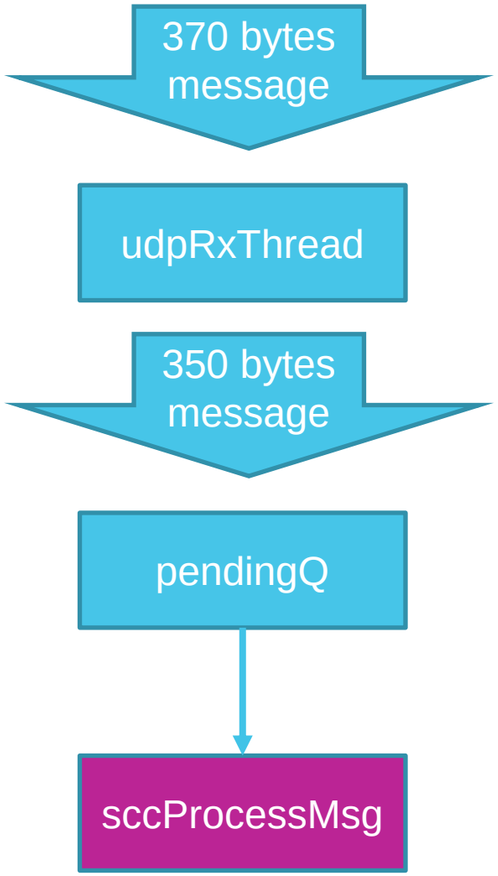
#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
q_buf_1->data_len = rec_len - 20;
q_buf_1->should_process_using_hmi = 0;
// Underflow when rec_len<20
memcpy(q_buf_1->data, &buf[20], rec_len - 20);
```

```
Q_add_block(q_buf_1, (q_buffer *)&pendingQ);
pthread_cond_broadcast(&WB_flag_cv);
```

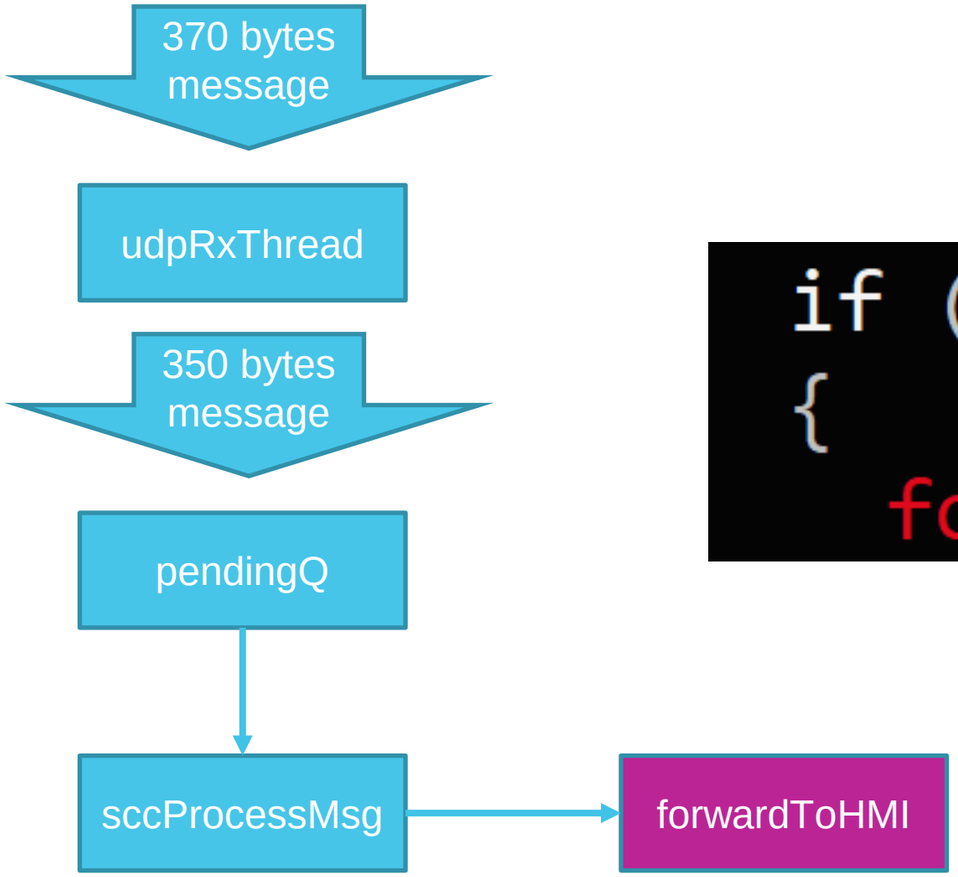
#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
q_buf = Q_remove_block((q_buffer *)&pendingQ);
```

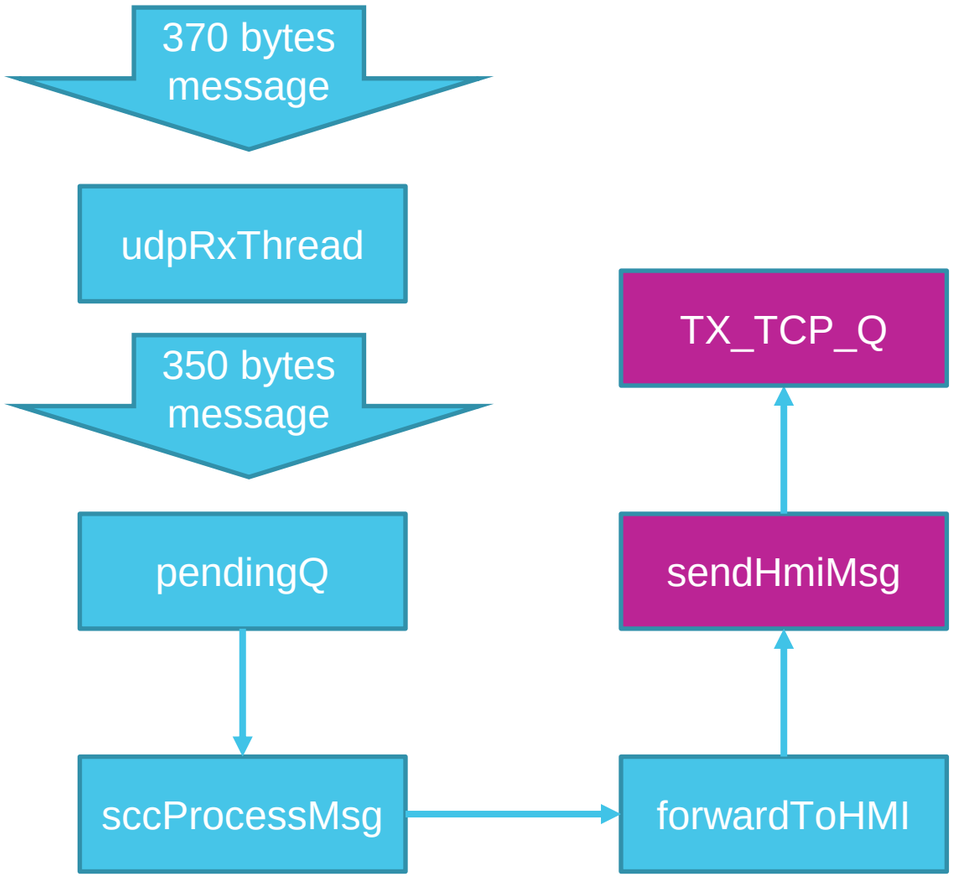
```
sccProcessMsg(q_buf);
```

#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
if ( first_data_byte == 0x37 )
{
   forwardToHMI(q_buf);
```

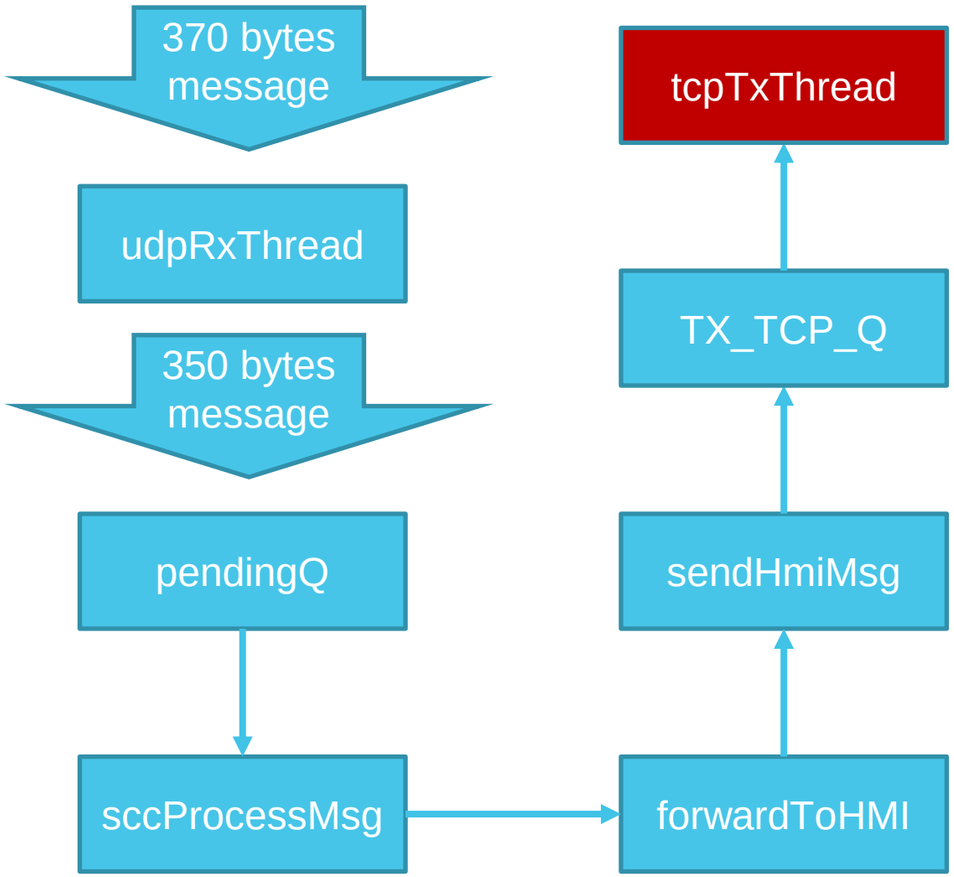
#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
1 int __fastcall forwardToHMI(q_buffer *q_buf)
2 {
3   q_buffer *q_buf_copy; // [sp+Ch] [bp-8h]
4
5   do
6     q_buf_copy = Q_remove_block((q_buffer *)&freeQ);
7   while ( !q_buf_copy );
8   setHmiBuffer(q_buf_copy, q_buf);
9   return sendHmiMsg(q_buf_copy);
10 }
```

```
1 int __fastcall sendHmiMsg(q_buffer *a1)
2 {
3   q_buffer *v2; // [sp+4h] [bp-8h]
4
5   v2 = a1;
6   a1->should_process_using_hmi = 1;
7   a1->op_buf_8 = 1;
8   pthread_mutex_lock(&TX_TCP_flag_mutex);
9   Q_add_block(v2, (q_buffer *)&TX_TCP_Q);
10  pthread_cond_broadcast(&TX_TCP_flag_cv);
11  return pthread_mutex_unlock(&TX_TCP_flag_mutex);
12 }
```

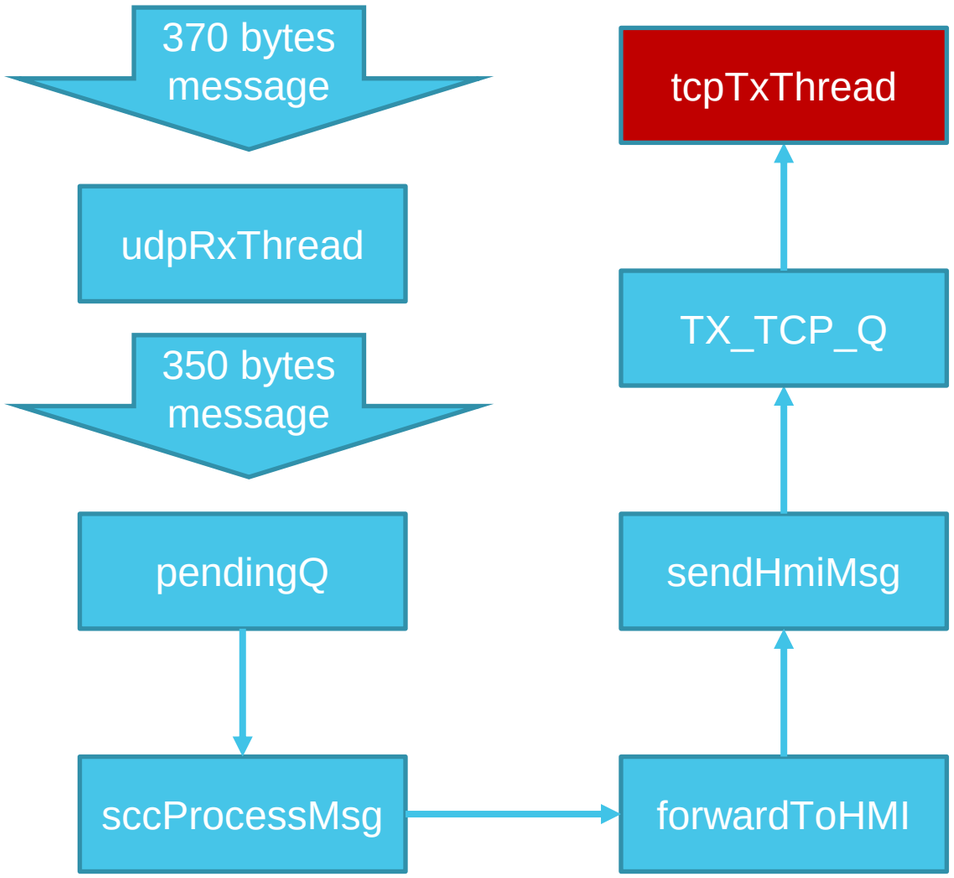
#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
void tcpTxThread()
{
  char buf[352]; // [sp+18h] [bp-17Ch]
  q_buffer *buffer_to_send; // [sp+178h] [bp-1Ch]
  ...
  while ( 1 )
  {
    while ( 1 )
    {
      ...
      buffer_to_send = Q_remove_block((q_buffer *)&TX_TCP_Q);
      ...
    }
    buf[0] = 2;
    if ( buffer_to_send->op == 2 )
      ...
    else
      ...
      buf[2] = buffer_to_send->data_len;
      memcpy(&buf[3], buffer_to_send->data,
             (unsigned __int16)buffer_to_send->data_len);
      addCRC((int)&buf[1], buffer_to_send->data_len + 2);
      ...
  }
}
```

#8 Off-by-three stack overflow in tcpTxThread (RCE)

#8 Off-by-three stack overflow in tcpTxThread (RCE)




```
void tcpTxThread()
{
  char buf[352]; // [sp+18h] [bp-17Ch]
  q_buffer *buffer_to_send; // [sp+178h] [bp-1Ch]
  ...
  while ( 1 )
  {
    while ( 1 )
    {
      ...
      buffer_to_send = Q_remove_block((q_buffer *)&TX_TCP_Q);
      ...
    }
    buf[0] = 2;
    if ( buffer_to_send->op == 2 )
      ...
    else
      ...
      buf[2] = buffer_to_send->data_len;
      memcpy(&buf[3], buffer_to_send->data,
             (unsigned __int16)buffer_to_send->data_len);
      addCRC((int)&buf[1], buffer_to_send->data_len + 2);
      ...
  }
}
```

#9 No firmware update verification (RCE)

The system is updated using an unauthenticated UDP command.

```
1 int startNewApp()
2 {
3   int result; // r0
4   char dest[84]; // [sp+0h] [bp-54h]
5
6   strcpy(dest, "cp /root/HMI3 /root/HMI3-back");
7   system(dest);
8   strcpy(dest, "mv /tmp/app_download /root/HMI3-new");
9   system(dest);
10  strcpy(dest, "sync");
11  result = system(dest);
12  exitType = 1;
13  LOBYTE(appSTOPrequest) = 1;
14  return result;
15 }
```

Exploitation

Exploitation – the easy way

1. Upload a new malicious FW
2. Connect using the default user and use the PE

```
burek ~/Dev/research/SwisslogPTS/Demo (master) $ telnet 192.168.100.40
Trying 192.168.100.40...
Connected to 192.168.100.40.
Escape character is '^]'.


BN5


freescale login: user
Password:
user@freescale ~$ ls -l /home/user/hmi/run
-rwxr-xr-x    1 user     user           480 Jan  1  1970 /home/user/hmi/run
user@freescale ~$
```

Heap Overflow Exploitation Plan

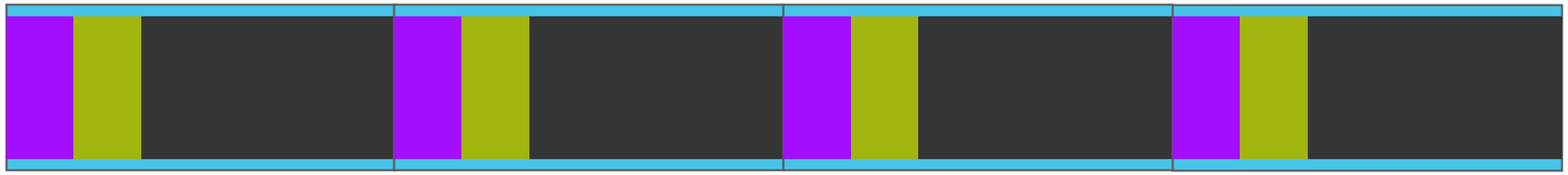
Off-By-Three Stack Overflow

- Corrupt buffer_to_send via the stack overflow
- Move buffer_to_send to the .got section where all the fun(c) pointers can be overwritten
- Send another UDP packet that will trigger the use of the overwritten buffer
- Overwrite the *memcpy* function pointer in the .got section with a call to a shellcode in the heap

```
void tcpTxThread()
{
  char buf[352]; // [sp+18h] [bp-17Ch]
  q_buffer *buffer_to_send; // [sp+178h] [bp-1Ch]
  ...
  while ( 1 )
  {
    while ( 1 )
    {
      ...
      buffer_to_send = Q_remove_block((q_buffer *)&TX_TCP_Q);
      ...
    }
    buf[0] = 2;
    if ( buffer_to_send->op == 2 )
      ...
    else
      ...
      buf[2] = buffer_to_send->data_len;
      memcpy(&buf[3], buffer_to_send->data,
             (unsigned __int16)buffer_to_send->data_len);
      addCRC((int)&buf[1], buffer_to_send->data_len + 2);
      ...
  }
}
```

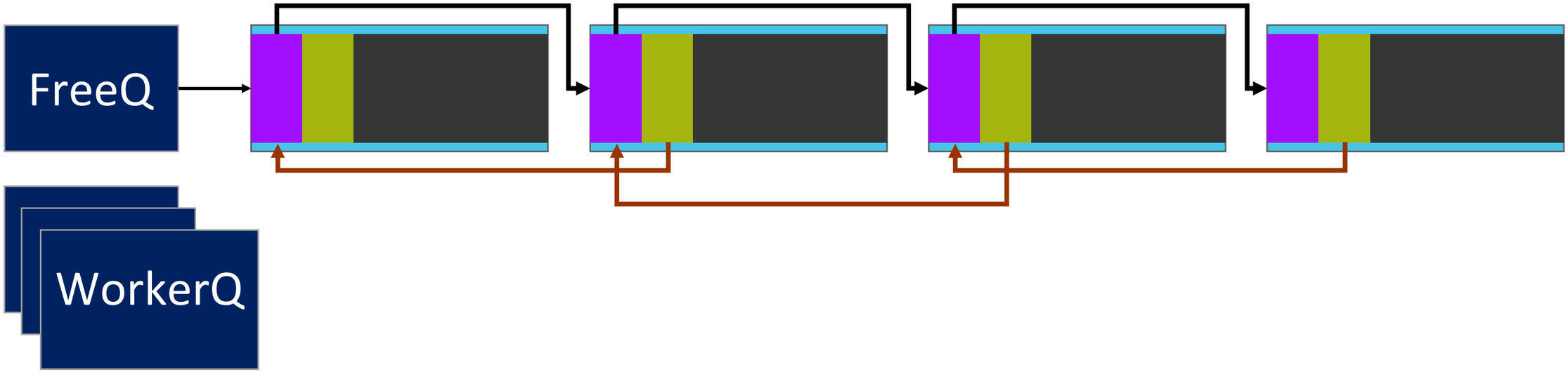
Heap examination

- 59 pre-allocated “heap” blocks in the bss section
- “heap” blocks are moved between queues
- Each block is of size 0x180 bytes



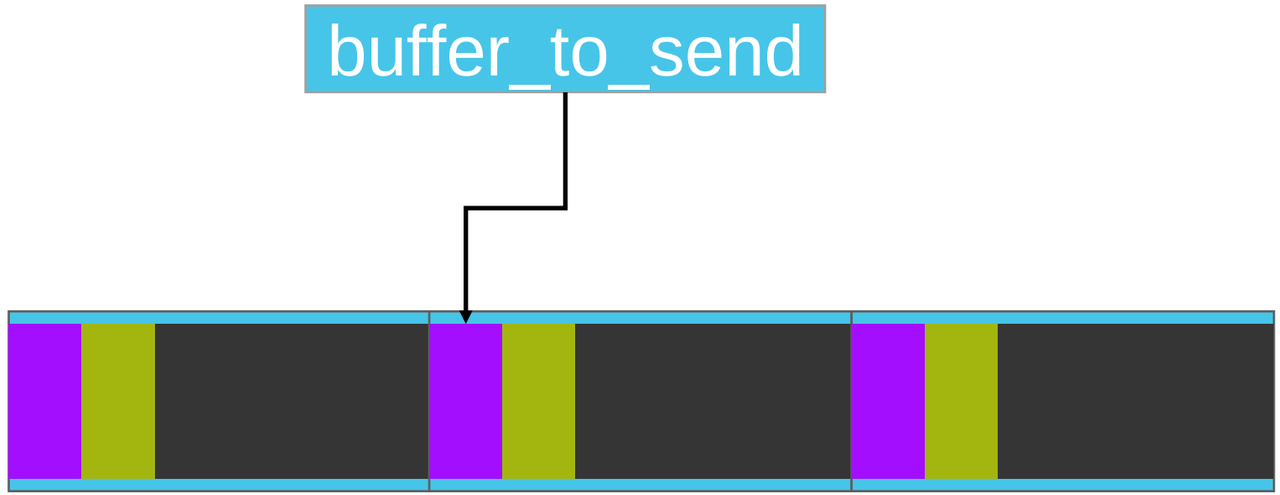
Heap examination

- 59 pre-allocated “heap” blocks in the bss section
- “heap” blocks are moved between queues
- Each block is of size 0x180 bytes



Exploitation

```
void tcpTxThread()
{
  char buf[352]; // [sp+18h] [bp-17Ch]
  q_buffer *buffer_to_send; // [sp+178h] [bp-1Ch]
  ...
  while ( 1 )
  {
    while ( 1 )
    {
      ...
      buffer_to_send = Q_remove_block((q_buffer *)&TX_TCP_Q);
      ...
    }
    buf[0] = 2;
    if ( buffer_to_send->op == 2 )
      ...
    else
      ...
      buf[2] = buffer_to_send->data_len;
      memcpy(&buf[3], buffer_to_send->data,
             (unsigned __int16)buffer_to_send->data_len);
      addCRC((int)&buf[1], buffer_to_send->data_len + 2);
      ...
  }
}
```



Exploitation

```
void tcpTxThread()
{
  char buf[352]; // [sp+18h] [bp-17Ch]
  q_buffer *buffer_to_send; // [sp+178h] [bp-1Ch]
  ...
  while ( 1 )
  {
    while ( 1 )
    {
      ...
      buffer_to_send = Q_remove_block((q_buffer *)&TX_TCP_Q);
      ...
    }
    buf[0] = 2;
    if ( buffer_to_send->op == 2 )
      ...
    else
      ...
      buf[2] = buffer_to_send->data_len;
      memcpy(&buf[3], buffer_to_send->data,
             (unsigned __int16)buffer_to_send->data_len);
      addCRC((int)&buf[1], buffer_to_send->data_len + 2);
      ...
  }
}
```

3 bytes overflow of buf

buffer_to_send is attacker controlled

buffer_to_send

Memory layout

LOAD	00008000	00008034	R	.	X	.	L	byte	03	public	CODE	32	00	12
PHDR	00008034	00008174	R	.	X	.	L	dword	02	public	CODE	32	00	12
LOAD	00008174	00008FF8	R	.	X	.	L	byte	03	public	CODE	32	00	12
.init	00008FF8	00009008	R	.	X	.	L	dword	07	public	CODE	32	00	12
.plt	00009008	000093DC	R	.	X	.	L	dword	08	public	CODE	32	00	12
LOAD	000093DC	000093E0	R	.	X	.	L	byte	03	public	CODE	32	00	12
.text	000093E0	0001A9E4	R	.	X	.	L	qword	09	public	CODE	32	00	12
.fini	0001A9E4	0001A9F0	R	.	X	.	L	dword	0A	public	CODE	32	00	12
.rodata	0001A9F0	0001BE4C	R	.	.	.	L	dword	0B	public	CONST	32	00	12
.ARM.exidx	0001BE4C	0001BE74	R	.	.	.	L	dword	0C	public	CONST	32	00	12
.eh_frame	0001BE74	0001BE78	R	.	.	.	L	dword	0D	public	CONST	32	00	12
.init_array	00023EF4	00023EF8	R	W	.	.	L	dword	0E	public	DATA	32	00	12
.fini_array	00023EF8	00023EFC	R	W	.	.	L	dword	0F	public	DATA	32	00	12
.jcr	00023EFC	00023F00	R	W	.	.	L	dword	10	public	DATA	32	00	12
LOAD	00023F00	00024000	R	W	.	.	L	byte	04	public	DATA	32	00	12
.got	00024000	00024150	R	W	.	.	L	dword	11	public	DATA	32	00	12
.data	00024150	00024174	R	W	.	.	L	dword	12	public	DATA	32	00	12
LOAD	00024174	00024178	R	W	.	.	L	byte	04	public	DATA	32	00	12
.bss	00024178	0002A11C	R	W	.	.								
.prgend	0002A11C	0002A11D	?	?	?	.								
extern	0002A120	0002A268	?	?	?	.								
abs	0002A268	0002A28C	?	?	?	.								

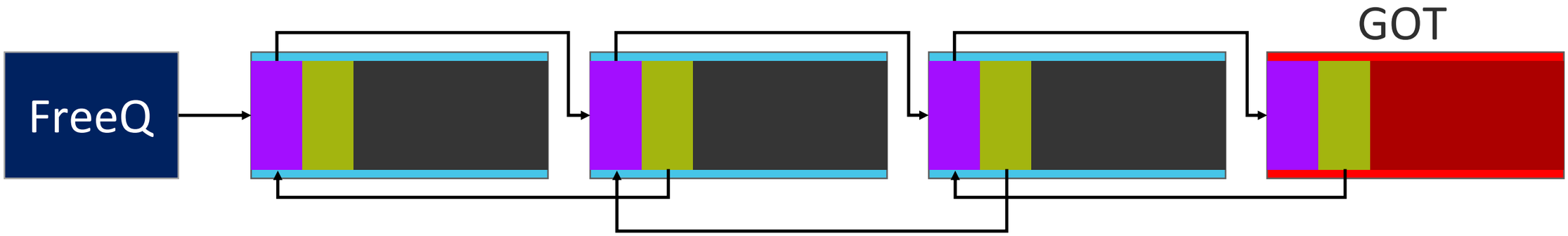
Heap Blocks

```
root@freescale ~/igal$ cat /proc/2183/maps
00008000-0001c000 r-xp 00000000 b3:01 72004      /root/HMI3
00023000-00024000 r-xp 00013000 b3:01 72004      /root/HMI3
00024000-00025000 rwxp 00014000 b3:01 72004      /root/HMI3
```

.GOT BLOCK - Controlling a function pointer

The first two dwords are unused, perfect for the new .got block start!

```
.got:000240CC E0 A1 02 00 strtoul_ptr      DCD __imp_strtoul          ; DATA XREF: strtoul+8↑r
.got:000240D0 E4 A1 02 00 pthread_create_ptr DCD __imp_pthread_create
.got:000240D0                                                         ; DATA XREF: pthread_cre
.got:000240D4 E8 A1 02 00 memcpy_ptr       DCD __imp_memcpy           ; DATA XREF: memcpy+8↑r
```



From GOT Block to RCE

When the removed block is the one in the GOT, *seq_num* will overwrite the *memcpy* address with a call to our shellcode (in the heap)

memcpy
Overwrite

```
void __noreturn udpRxThread()
{
        ...
        rec_len = recvfrom(udp_socket, buf, 370u, 0, &addr, &addr_len);
        ...
        q_buf_1 = Q_remove_block(freeQ);
        q_buf_1->seq_num = ntohl(buf[4]);
        ...
        q_buf_1->data_len = rec_len - 20;
        q_buf_1->should_process_using_hmi = 0;
        memcpy(q_buf_1->data, &buf[20], rec_len - 20);
```

From GOT Block to RCE

Memcpy(shellcode) is used right after it is set

memcpy Overwrite

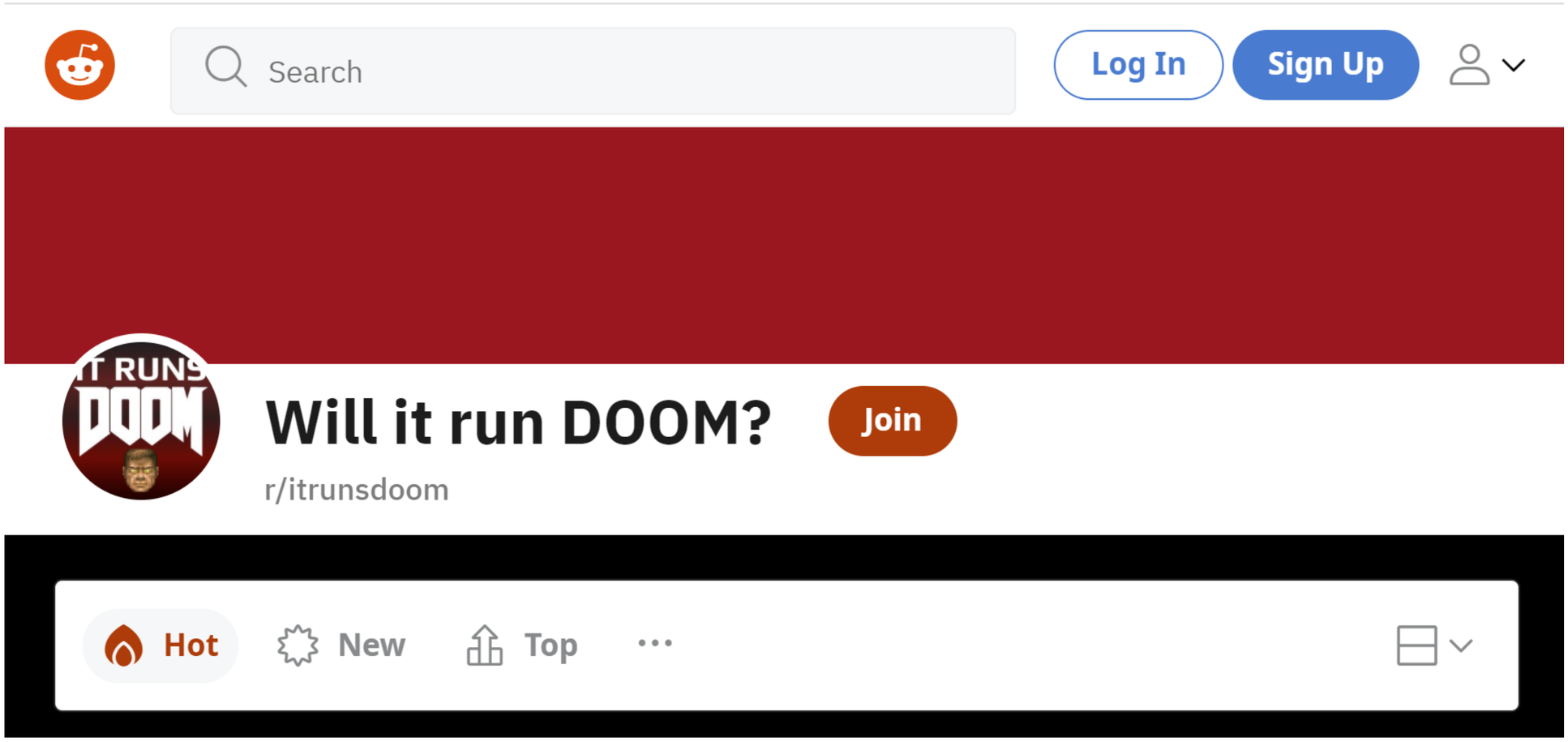
Memcpy(shellcode) Usage

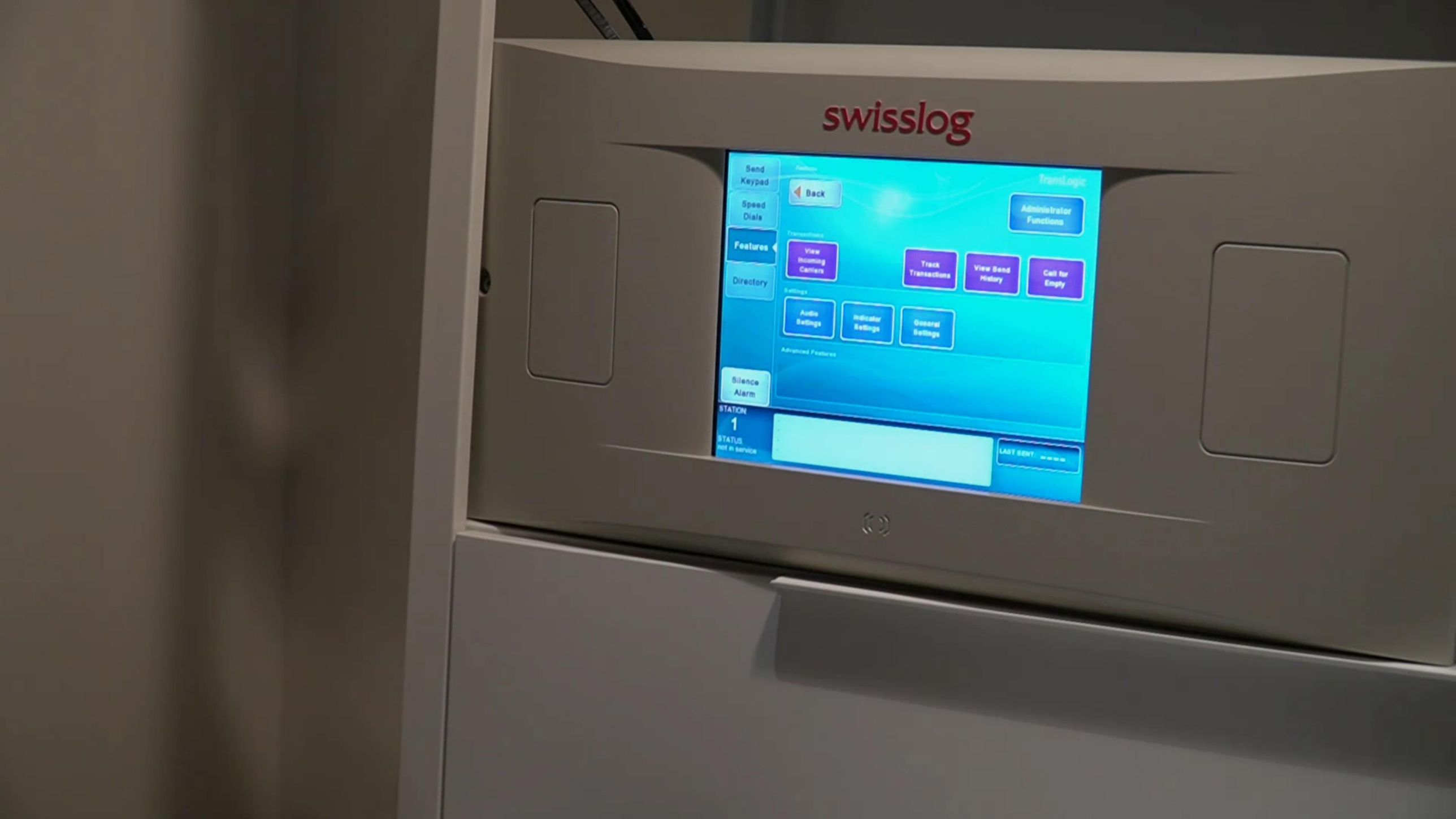
```
void __noreturn udpRxThread()
{
        ...
        rec_len = recvfrom(udp_socket, buf, 370u, 0, &addr, &addr_len);
        ...
        q_buf_1 = Q_remove_block(freeQ);
        q_buf_1->seq_num = ntohl(buf[4]);
        ...
        q_buf_1->data_len = rec_len - 20;
        q_buf_1->should_process_using_hmi = 0;
        memcpy(q_buf_1->data, &buf[20], rec_len - 20);
```

Heap Overflow Exploitation - Summary

1. Spray the heap with shellcode buffers
2. Trigger the off-by-three vulnerability to move one heap block to the .got section
3. Spray the heap (again) with the shellcode and the shellcode address as the sequence number.
4. Once the .got block is used, the *memcpy* pointer will be point to the shellcode, and then the shellcode will be triggered
5. Demo time!

Demo prep


Search
Log In
Sign Up
Will it run DOOM?
Join
r/itrunsdoom
Hot
New
Top


swisslog
Send Keypad
Speed Dials
Features
Directory
Silence Alarm
Back
TransLogic
Administrator Functions
View Incoming Carriers
Track Transactions
View Send History
Call for Empty
Audio Settings
Indicator Settings
General Settings
STATION
1
STATUS
not in service
LAST SENT

Final Thoughts

- Pneumatic Tube Systems require more research
- They are critical infrastructure – like electricity or elevators
- The Swisslog case is a classic case of embedded devices gone wrong
- Developing robust security mitigations to safeguard these systems is essential
- Adding DOOM to pneumatic systems would make any hospital visit much more entertaining ;)

Questions?

- More info at: https://www.armis.com/pwnedPiper