Attackers Deploy New ICS Attack Framework “TRITON” and Cause Operational Disruption to Critical Infrastructure

www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html

Introduction

Mandiant recently responded to an incident at a critical infrastructure organization where an attacker deployed malware designed to manipulate industrial safety systems. The targeted systems provided emergency shutdown capability for industrial processes. We assess with moderate confidence that the attacker was developing the capability to cause physical damage and inadvertently shutdown operations. This malware, which we call TRITON, is an attack framework built to interact with Triconex Safety Instrumented System (SIS) controllers. We have not attributed the incident to a threat actor, though we believe the activity is consistent with a nation state preparing for an attack.

TRITON is one of a limited number of publicly identified malicious software families targeted at industrial control systems (ICS). It follows Stuxnet which was used against Iran in 2010 and Industroyer which we believe was deployed by Sandworm Team against Ukraine in 2016. TRITON is consistent with these attacks, in that it could prevent safety mechanisms from executing their intended function, resulting in a physical consequence.

Malware Family	Main Modules	Description
TRITON	trilog.exe	Main executable leveraging libraries.zip
	library.zip	Custom communication library for interaction with Triconex controllers.

Table 1: Description of TRITON Malware

Incident Summary

The attacker gained remote access to an SIS engineering workstation and deployed the TRITON attack framework to reprogram the SIS controllers. During the incident, some SIS controllers entered a failed safe state, which automatically shutdown the industrial process and prompted the asset owner to initiate an investigation. The investigation found that the SIS controllers initiated a safe shutdown when application code between redundant processing units failed a validation check -- resulting in an MP diagnostic failure message.

We assess with moderate confidence that the attacker inadvertently shutdown operations while developing the ability to cause physical damage for the following reasons:

- Modifying the SIS could prevent it from functioning correctly, increasing the likelihood of a failure that would result in physical consequences.
- TRITON was used to modify application memory on SIS controllers in the environment, which could have led to a failed validation check.
- The failure occurred during the time period when TRITON was used.
- It is not likely that existing or external conditions, in isolation, caused a fault during the time of the incident.

Attribution

FireEye has not connected this activity to any actor we currently track; however, we assess with moderate confidence that the actor is sponsored by a nation state. The targeting of critical infrastructure as well as the attacker's persistence, lack of any clear monetary goal and the technical resources necessary to create the attack framework suggest a well-resourced nation state actor. Specifically, the following facts support this assessment:

The attacker targeted the SIS suggesting an interest in causing a high-impact attack with physical consequences. This is an attack objective not typically seen from cyber-crime groups.

The attacker deployed TRITON shortly after gaining access to the SIS system, indicating that they had pre-built and tested the tool which would require access to hardware and software that is not widely available. TRITON is also designed to communicate using the proprietary TriStation protocol which is not publicly documented suggesting the adversary independently reverse engineered this protocol.

The targeting of critical infrastructure to disrupt, degrade, or destroy systems is consistent with numerous attack and reconnaissance activities carried out globally by Russian, Iranian, North Korean, U.S., and Israeli nation state actors. Intrusions of this nature do not necessarily indicate an immediate intent to disrupt targeted systems, and may be preparation for a contingency.

Background on Process Control and Safety Instrumented Systems

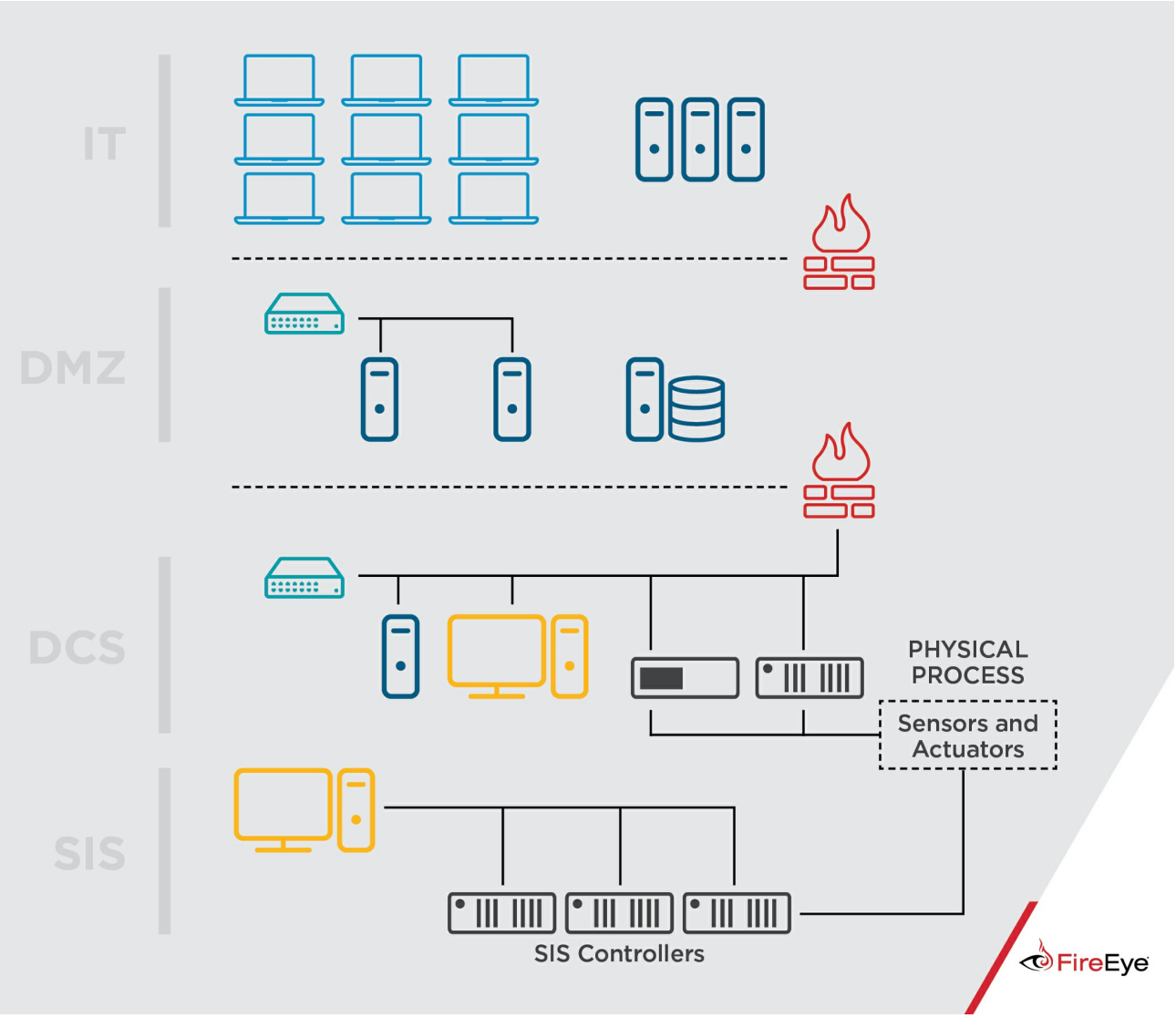

Figure 1: ICS Reference Architecture

Modern industrial process control and automation systems rely on a variety of sophisticated control systems and safety functions. These systems and functions are often referred to as Industrial Control Systems (ICS) or Operational Technology (OT).

A Distributed Control System (DCS) provides human operators with the ability to remotely monitor and control an industrial process. It is a computerized control system consisting of computers, software applications and controllers. An Engineering Workstation is a computer used for configuration, maintenance and diagnostics of the control system applications and other control system equipment.

A SIS is an autonomous control system that independently monitors the status of the process under control. If the process exceeds the parameters that define a hazardous state, the SIS attempts to bring the process back into a safe state or automatically performs a safe shutdown of the process. If the SIS and DCS controls fail, the final line of defense is the design of the industrial facility, which includes mechanical protections on equipment (e.g. rupture discs), physical alarms, emergency response procedures and other mechanisms to mitigate

dangerous situations.

Asset owners employ varied approaches to interface their plant's DCS with the SIS. The traditional approach relies on the principles of segregation for both communication infrastructures and control strategies. For at least the past decade, there has been a trend towards integrating DCS and SIS designs for various reasons including lower cost, ease of use, and benefits achieved from exchanging information between the DCS and SIS. We believe TRITON acutely demonstrates the risk associated with integrated designs that allow bi-directional communication between DCS and SIS network hosts.

Safety Instrumented Systems Threat Model and Attack Scenarios

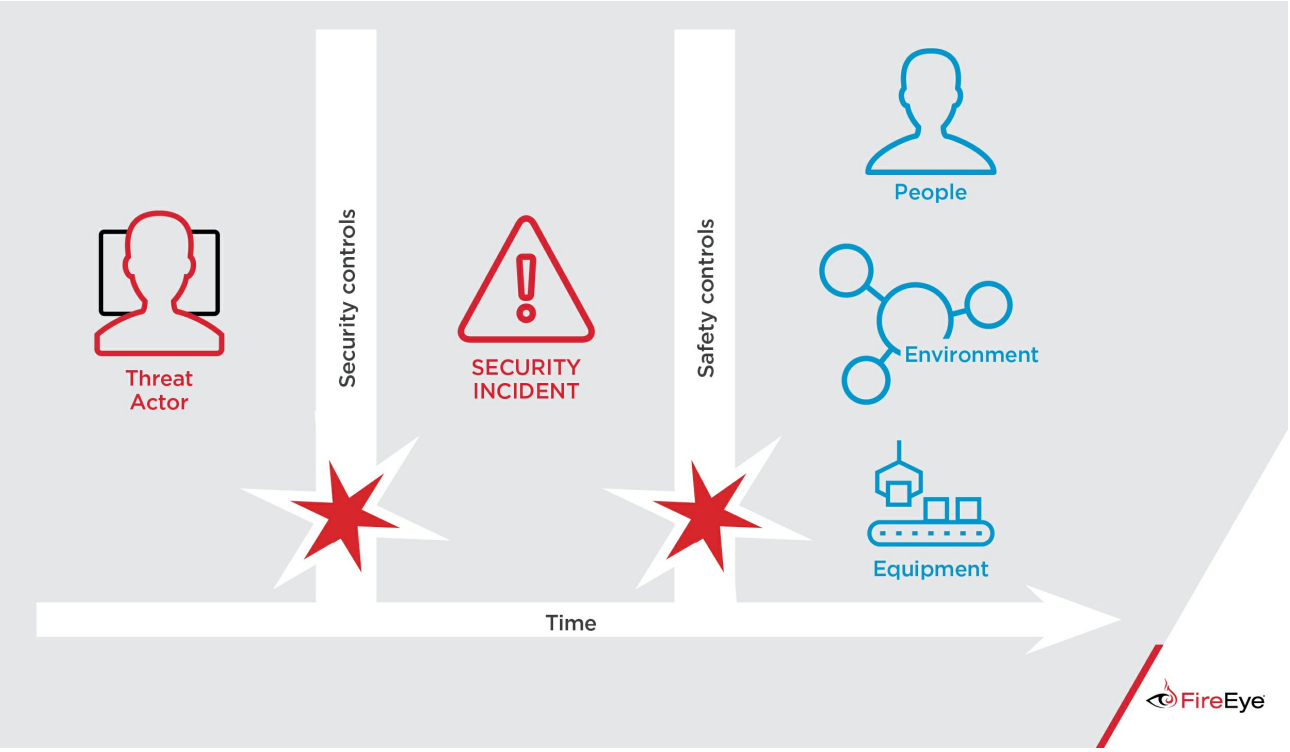

Figure 2: Temporal Relationship Between Cyber Security and Safety

The attack lifecycle for disruptive attacks against ICS is similar to other types of cyber attacks, with a few key distinctions. First, the attacker's mission is to disrupt an operational process rather than steal data. Second, the attacker must have performed OT reconnaissance and have sufficient specialized engineering knowledge to understand the industrial process being controlled and successfully manipulate it.

Figure 2 represents the relationship between cyber security and safety controls in a process control environment. Even if cyber security measures fail, safety controls are designed to prevent physical damage. To maximize physical impact, a cyber attacker would also need to bypass safety controls.

The SIS threat model below highlights some of the options available to an attacker who has successfully compromised an SIS.

Attack Option 1: Use the SIS to shutdown the process

- The attacker can reprogram the SIS logic to cause it to trip and shutdown a process that is, in actuality, in a safe state. In other words, trigger a false positive.
- Implication: Financial losses due to process downtime and complex plant start up procedure after the shutdown.

Attack Option 2: Reprogram the SIS to allow an unsafe state

- The attacker can reprogram the SIS logic to allow unsafe conditions to persist.
- Implication: Increased risk that a hazardous situation will cause physical consequences (e.g. impact to equipment, product, environment and human safety) due to a loss of SIS functionality.

Attack Option 3: Reprogram the SIS to allow an unsafe state – while using the DCS to create an unsafe state or hazard

- The attacker can manipulate the process into an unsafe state from the DCS while preventing the SIS from functioning appropriately.
- Implication: Impact to human safety, the environment, or damage to equipment, the extent of which depends on the physical constraints of the process and the plant design.

Analysis of Attacker Intent

We assess with moderate confidence that the attacker's long-term objective was to develop the capability to cause a physical consequence. We base this on the fact that the attacker initially obtained a reliable foothold on the DCS and could have developed the capability to manipulate the process or shutdown the plant, but instead proceeded to compromise the SIS system. Compromising both the DCS and SIS system would enable the attacker to develop and carry out an attack that causes the maximum amount of damage allowed by the physical and mechanical safeguards in place.

Once on the SIS network, the attacker used their pre-built TRITON attack framework to interact with the SIS controllers using the TriStation protocol. The attacker could have caused a process shutdown by issuing a halt command or intentionally uploading flawed code to the SIS controller to cause it to fail. Instead, the attacker made several attempts over a period of time to develop and deliver functioning control logic for the SIS controllers in this target environment. While these attempts appear to have failed due one of the attack scripts' conditional checks, the attacker persisted with their efforts. This suggests the attacker was intent on causing a specific outcome beyond a process shutdown.

Of note, on several occasions, we have observed evidence of long term intrusions into ICS which were not ultimately used to disrupt or disable operations. For instance, Russian operators, such as Sandworm Team, have compromised Western ICS over a multi-year period without causing a disruption.

Summary of Malware Capabilities

The TRITON attack tool was built with a number of features, including the ability to read and write programs, read and write individual functions and query the state of the SIS controller. However, only some of these capabilities were leveraged in the trilog.exe sample (e.g. the attacker did not leverage all of TRITON's extensive reconnaissance capabilities).

The TRITON malware contained the capability to communicate with Triconex SIS controllers (e.g. send specific commands such as *halt* or read its memory content) and remotely reprogram them with an attacker-defined payload. The TRITON sample Mandiant analyzed added an attacker-provided program to the execution table of the Triconex controller. This sample left legitimate programs in place, expecting the controller to continue operating without a fault or exception. If the controller failed, TRITON would attempt to return it to a running state. If the controller did not recover within a defined time window, this sample would overwrite the malicious program with invalid data to cover its tracks.

Recommendations

Asset owners who wish to defend against the capabilities demonstrated in the incident, should consider the following controls:

- Where technically feasible, segregate safety system networks from process control and information system networks. Engineering workstations capable of programming SIS controllers should not be dual-homed to any other DCS process control or information system network.
- Leverage hardware features that provide for physical control of the ability to program safety controllers. These usually take the form of switches controlled by a physical key. On Triconex controllers, keys should not be left in the PROGRAM mode other than during scheduled programming events.
- Implement change management procedures for changes to key position. Audit current key state regularly.
- Use a unidirectional gateway rather than bidirectional network connections for any applications that depend on the data provided by the SIS.
- Implement strict access control and application whitelisting on any server or workstation endpoints that can reach the SIS system over TCP/IP.
- Monitor ICS network traffic for unexpected communication flows and other anomalous activity.

Figure 3: Triconex Key Switch (source)

Appendix: Technical Analysis

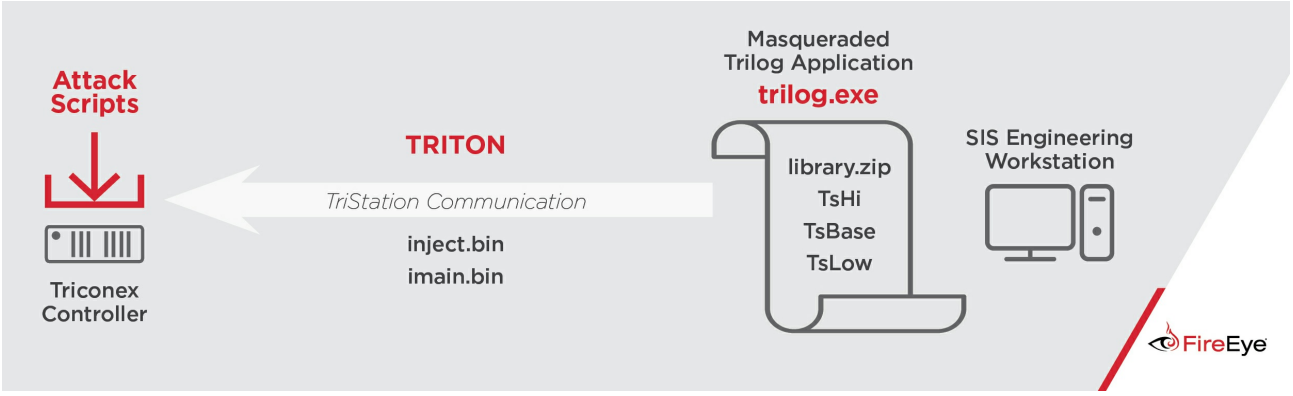

Figure 4: TRITON Architecture and Attack Scenario

TRITON was deployed on an SIS engineering workstation running the Microsoft Windows operating system. The malware was named to masquerade as the legitimate Triconex Trilog application. This application is used for reviewing logs and is a part of the TriStation application suite. The malware was delivered as a Py2EXE compiled python script dependent on a zip file containing standard Python libraries, open source libraries, as well as the attacker-

developed Triconex attack framework for interacting with the Triconex controllers. Along with the executable, two binary files, *inject.bin (malicious function code)* and *imain.bin (malicious control logic)*, were deployed as the controller's payload. These file names were hard coded in the Py2EXE compiled python script.

Trilog.exe took one option from the command line, which was a single IP address of the target Triconex device. It did not leverage the underlying TRITON library's capability for Triconex device discovery, instead an instance of trilog.exe had to be invoked separately for each target controller in the environment. Once invoked, trilog.exe checked the status of the controller, then read the configuration information exposed by the TriStation protocol. If the controller was in a running state, trilog.exe encoded the two payload files *inject.bin* and *imain.bin* and passed them to the communication libraries to be appended to the controller's program memory and execution table.

After payload files were inserted into memory on the Triconex controller, the script initiated a countdown, periodically checking the status of the controller. If an error was detected, the communication library's method *SafeAppendProgramMod* attempted to reset the controller to the previous state using a TriStation protocol command. If this failed, trilog.exe attempted to write a small 'dummy' program to memory. We assess that this was an anti-forensics technique to hide the presence of the attacker code on the Triconex controller.

Working with the asset owner, Mandiant ran trilog.exe in a lab environment with a valid Triconex controller and discovered a conditional check in the malware that prevented the payload binary from persisting in the environment. Mandiant confirmed that, after correcting patching the attack script to remove this check, the payload binary would persist in controller memory, and the controller would continue to run.

TRITON implements the TriStation protocol, which is the protocol used by the legitimate TriStation application, to configure controllers.

TsHi is the high-level interface created by the malware's authors that allows the threat actor's operators to implement attack scripts using the TRITON framework. It exposes functions for both reconnaissance and attack. The functions generally accept binary data from the user, and handle the code 'signing' and check sums prior to passing the data to lower level libraries for serialization on to the network.

TsBase, another attacker-written module, contains the functions called by *TsHi*, which translate the attacker's intended action to the appropriate TriStation protocol function code. For certain functions, it also packs and pads the data in to the appropriate format.

TsLow is an additional attacker module that implements the TriStation UDP wire protocol. The *TsBase* library primarily depends on the *ts_exec* method. This method takes the function code and expected response code, and serializes the commands payload over UDP. It checks the response from the controller against the expected value and returns a result data structure indicating success or a *False* object representing failure.

TsLow also exposes the connect method used to check connectivity to the target controller. If invoked with no targets, it runs the device discovery function detect_ip. This leverages a "ping" message over the TriStation protocol using IP broadcast to find controllers that are reachable via a router from where the script is invoked.

Indicators

Filename	**Hash**
trilog.exe	MD5: 6c39c3f4a08d3d78f2eb973a94bd7718 SHA-256: e8542c07b2af63ee7e72ce5d97d91036c5da56e2b091aa2afe737b224305d230
imain.bin	MD5: 437f135ba179959a580412e564d3107f SHA-256: 08c34c6ac9186b61d9f29a77ef5e618067e0bc9fe85cab1ad25dc6049c376949
inject.bin	MD5: 0544d425c7555dc4e9d76b571f31f500 SHA-256: 5fc4b0076eac7aa7815302b0c3158076e3569086c4c6aa2f71cd258238440d14
library.zip	MD5: 0face841f7b2953e7c29c064d6886523 SHA-256: bef59b9a3e00a14956e0cd4a1f3e7524448cbe5d3cc1295d95a15b83a3579c59
TS_cnames.pyc	MD5: e98f4f3505f05bf90e17554fbc97bba9 SHA-256: 2c1d3d0a9c6f76726994b88589219cb8d9c39dd9924bc8d2d02bf41d955fe326
TsBase.pyc	MD5: 288166952f934146be172f6353e9a1f5 SHA-256: 1a2ab4df156ccd685f795baee7df49f8e701f271d3e5676b507112e30ce03c42
TsHi.pyc	MD5: 27c69aa39024d21ea109cc9c9d944a04 SHA-256: 758598370c3b84c6fbb452e3d7119f700f970ed566171e879d3cb41102154272
TsLow.pyc	MD5: f6b3a73c8c87506acda430671360ce15 SHA-256: 5c776a33568f4c16fee7140c249c0d2b1e0798a96c7a01bfd2d5684e58c9bb32
sh.pyc	MD5: 8b675db417cc8b23f4c43f3de5c83438 SHA-256: c96ed56bf7ee85a4398cc43a98b4db86d3da311c619f17c8540ae424ca6546e1

Detection

```
rule TRITON_ICS_FRAMEWORK
{
    meta:
        author = "nicholas.carr @itsreallynick"
        md5 = "0face841f7b2953e7c29c064d6886523"
        description = "TRITON framework recovered during Mandiant ICS incident response"
    strings:
        $python_compiled = ".pyc" nocase ascii wide
        $python_module_01 = "__module__" nocase ascii wide
        $python_module_02 = "<module>" nocase ascii wide
        $python_script_01 = "import Ts" nocase ascii wide
        $python_script_02 = "def ts_" nocase ascii wide

        $py_cnames_01 = "TS_cnames.py" nocase ascii wide
        $py_cnames_02 = "TRICON" nocase ascii wide
        $py_cnames_03 = "TriStation " nocase ascii wide
        $py_cnames_04 = " chassis " nocase ascii wide

        $py_tslibs_01 = "GetCpStatus" nocase ascii wide
        $py_tslibs_02 = "ts_" ascii wide
        $py_tslibs_03 = " sequence" nocase ascii wide
        $py_tslibs_04 = /import Ts(Hi|Low|Base)[^:alpha:]/ nocase ascii wide
        $py_tslibs_05 = /module\s?version/ nocase ascii wide
        $py_tslibs_06 = "bad " nocase ascii wide
        $py_tslibs_07 = "prog_cnt" nocase ascii wide

        $py_tsbase_01 = "TsBase.py" nocase ascii wide
        $py_tsbase_02 = ".TsBase(" nocase ascii wide

        $py_tshi_01 = "TsHi.py" nocase ascii wide
        $py_tshi_02 = "keystate" nocase ascii wide
        $py_tshi_03 = "GetProjectInfo" nocase ascii wide
        $py_tshi_04 = "GetProgramTable" nocase ascii wide
        $py_tshi_05 = "SafeAppendProgramMod" nocase ascii wide
        $py_tshi_06 = ".TsHi(" ascii nocase wide

        $py_tslow_01 = "TsLow.py" nocase ascii wide
        $py_tslow_02 = "print_last_error" ascii nocase wide
        $py_tslow_03 = ".TsLow(" ascii nocase wide
        $py_tslow_04 = "tcm_" ascii wide
        $py_tslow_05 = " TCM found" nocase ascii wide

        $py_crc_01 = "crc.pyc" nocase ascii wide
        $py_crc_02 = "CRC16_MODBUS" ascii wide
        $py_crc_03 = "Kotov Alaxander" nocase ascii wide
        $py_crc_04 = "CRC_CCITT_XMODEM" ascii wide
        $py_crc_05 = "crc16ret" ascii wide
        $py_crc_06 = "CRC16_CCITT_x1D0F" ascii wide
        $py_crc_07 = /CRC16_CCITT[^_]/ ascii wide

        $py_sh_01 = "sh.pyc" nocase ascii wide

        $py_keyword_01 = " FAILURE" ascii wide
        $py_keyword_02 = "symbol table" nocase ascii wide

        $py_TRIDENT_01 = "inject.bin" ascii nocase wide
        $py_TRIDENT_02 = "imain.bin" ascii nocase wide

    condition:
        2 of ($python_*) and 7 of ($py_*) and filesize < 3MB
}
```