Blinding Snort: Breaking the Modbus OT Preprocessor

claroty.com/2022/04/14/blog-research-blinding-snort-breaking-the-modbus-ot-preprocessor/

April 14, 2022

By Uri Katz | April 14, 2022

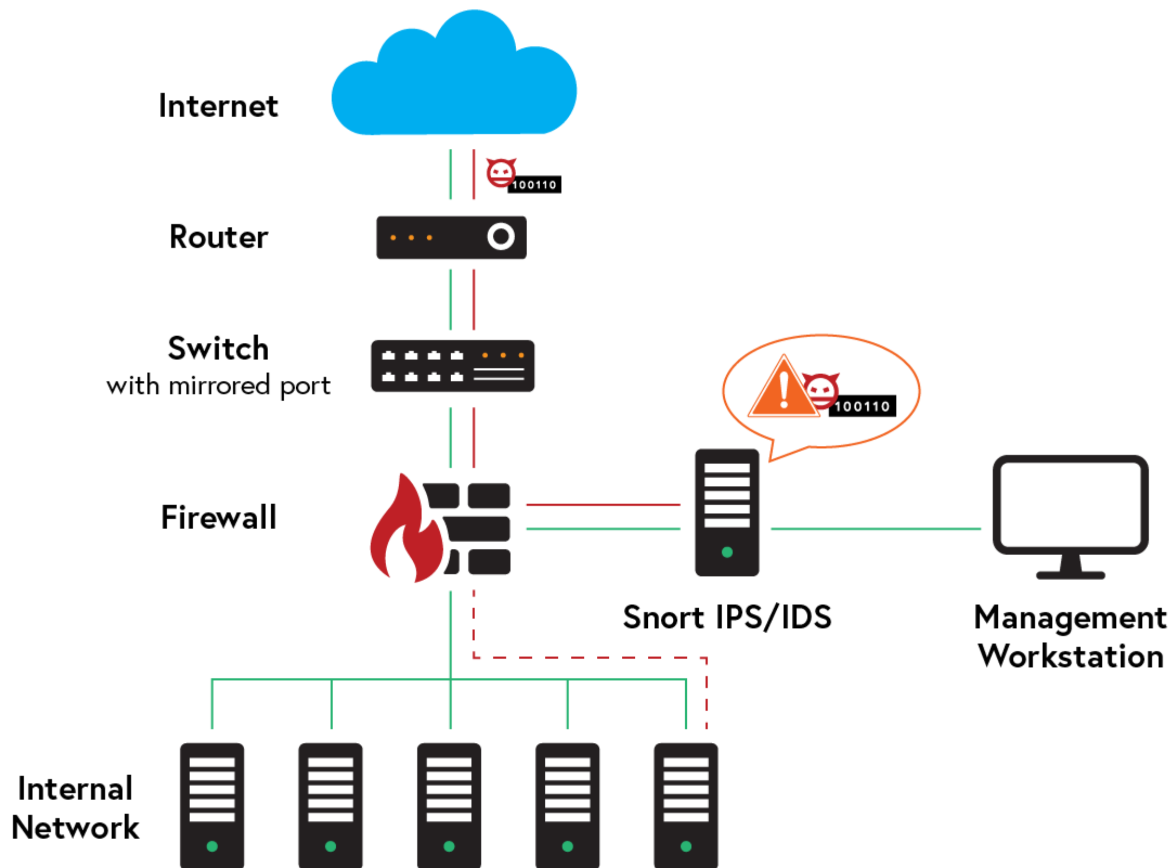
Executive Summary

- Team82 discovered a means by which it could blind the popular Snort intrusion detection and prevention system to malicious packets.
- The vulnerability, CVE-2022-20685, is an integer-overflow issue that can cause the Snort Modbus OT preprocessor to enter an infinite while-loop.
- A successful exploit keeps Snort from processing new packets and generating alerts.
- The vulnerability, which can be attacked remotely, has been patched by Cisco and the Snort team.
- All open source Snort project releases earlier than 2.9.19 and release 3.1.11.0 are vulnerable.
- Read Cisco's advisory here for commercial product patching and mitigation information.

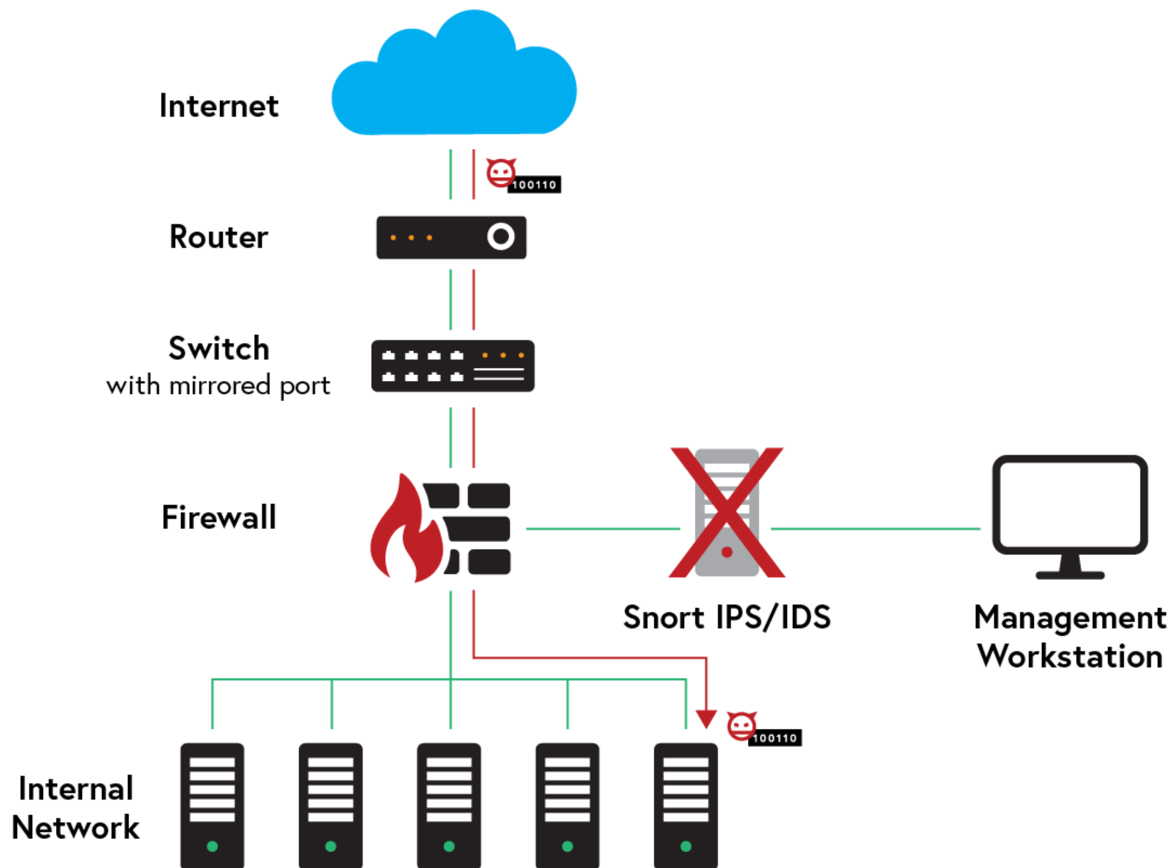
Introduction

Network analysis tools are integral to keeping networks secure by providing real-time logging and analysis of events and traffic. Snort is atop this list of analysis tools as the most popular network intrusion detection and prevention system. The open-source version of Snort still has an active community of contributors and developers, while Cisco has developed commercial versions of Snort since acquiring parent company Sourcefire in 2013.

Snort is largely used passively on the network, but it can also take action on malicious packets, making it a powerful detection tool for defenders. An attacker who could blind this tool to malicious traffic, however, could gain an important advantage over network defenders.



In this report, Team82 will demonstrate how it was able to do just that through a vulnerability, (CVE-2022-20685) we uncovered in Snort's Modbus OT preprocessor. Exploiting this vulnerability allowed us to blind Snort's ability to detect further attacks and run malicious packets on the network.



Snort Rules and Alerts

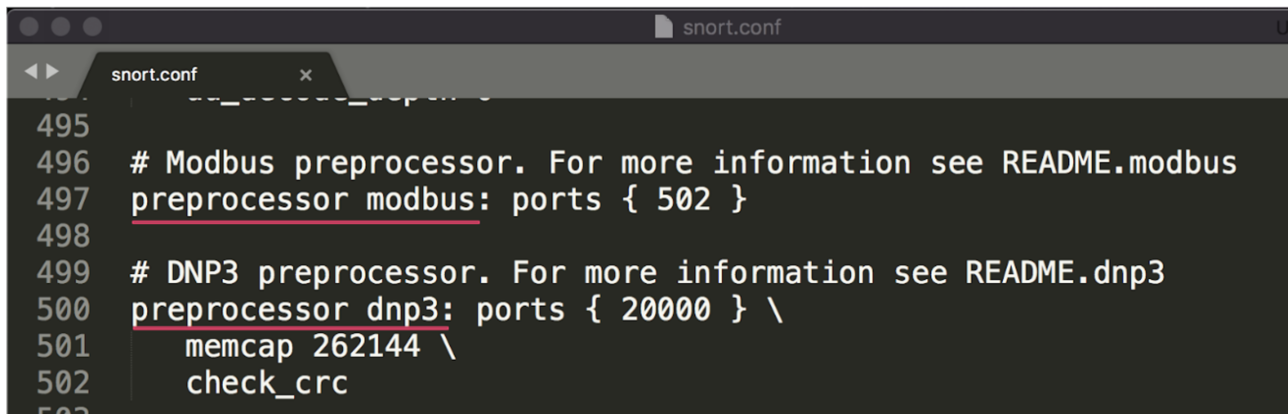
Snort's open-source network-based intrusion detection/prevention system (IDS/IPS) has the ability to perform real-time traffic analysis and packet logging on internet protocol (IP) networks. Snort performs protocol analysis, content searching, and matching based on a predefined rule set. Snort rules can be based on raw data or on Snort's built-in protocol parsers.

Rules may trigger one of three actions:

- Alert rules: Generate an alert
- Log rules: Alert and log the alert
- Pass rules: Ignore the packet

To make the rule-writing process simpler and improve the detection capabilities, Snort comes with a set of preprocessors that are on by default and analyze and structure network traffic into objects that can be referenced later in Snort rules. Some of the preprocessors

included in Snort are: ARP, DNS, SSH and some OT (operational technology) protocols, such as MODBUS / DNP3.



Snort's default configuration (snort.conf).

When writing Snort rules, one can easily use these objects—for example to check the Modbus function ID the modbus.func_id attribute can be used, instead of checking offset of the sixth byte in the packet. Here is an example for a snort rule that use modbus preprocessor attributes:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 502 (msg:"PROTOCOL-SCADA Modbus write single coil -
invalid state"; flow:to_server,established; modbus_func:write_single_coil; content:"|00
00|"; depth:2; offset:2; content:"|00|"; depth:1; offset:11; content:!"|FF|"; depth:1;
offset:10; content:!"|00|"; depth:1; offset:10; metadata:policy max-detect-ips drop;
reference:url,www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf;
classtype:protocol-command-decode; sid:29200; rev:3;)
```

Modbus

Modbus is an industrial protocol developed in 1979, first intended to transfer data over a serial line. Later it was expanded to include TCP/UDP support. The main Modbus function codes are:

Function name	Function code
Read Discrete Inputs	2
Read Coils	1
Write Single Coil	5
Write Multiple Coils	15
Read Input Registers	4
Read Multiple Holding Registers	3
Write Single Holding Register	6
Write Multiple Holding Registers	16

CVE-2022-20685: Technical Details

While researching Snort OT preprocessors, we decided to focus on Modbus because it was one of the more complex OT preprocessors Snort supports. To understand what we found, we first need to examine the structure of the Modbus Write File Record function code.

Write File Record (command 0x15)

The Write File Record Modbus command writes multiple groups of file registers to the Modbus server. A file is an organization of records. Each file may contain up to 10,000 records, addressed 0000 to 9999 decimal or 0x0000 to 0x270F.

The Write File Record Modbus command allows writing multiple groups of references. Each group is defined in a separate sub-request field that contains 7 bytes plus the data:

- **Reference type**: 1 byte (must be specified as 6)
- **File number**: 2 bytes
- **Starting record number within file**: 2 bytes
- **Length of record to be written**: 2 bytes
- **Data to be written**: 2 bytes of data per register

The number of registers to be written, combined with all other fields in the request, must not exceed the allowable length of the Modbus protocol data unit (PDU), which is 253 bytes.

Here is a summary of the Write File Record Modbus command request:

	DESCRIPTION	LENGTH	VALUE
Header	Funtion Code	1 Byte	0x15
	Request Data Length	1 Byte	0x09 to 0xFB
Group 1	Sub-request: Reference Type	1 Byte	06
	Sub-request: File Number	2 Bytes	0x0001 to 0xFFFF
	Sub-request: Record Number	2 Bytes	0x0000 to 0x270F
	Sub-request: Record Length	2 Bytes	N1
	Sub-request: Record Data	N1 x 2 Bytes	Data
Group 2	Sub-request: Reference Type	1 Byte	06
	Sub-request: File Number	2 Bytes	0x0001 to 0xFFFF
	Sub-request: Record Number	2 Bytes	0x0000 to 0x270F
	Sub-request: Record Length	2 Bytes	N1
	Sub-request: Record Data	N1 x 2 Bytes	Data

Vulnerability and Exploitability

The Modbus preprocessor handles multiple Modbus function codes. Snort uses the ModbusCheckRequestLengths function to calculate the expected size for each packet.

If we look at the function ModbusCheckRequestLengths in the file modbus_decode.c, we see a while-loop that goes over all of the groups in the packet, in order to calculate the total record lengths.

```
tmp_count = *(packet->payload + MODBUS_MIN_LEN);
if (tmp_count == modbus_payload_len - MODBUS_BYTE_COUNT_SIZE)
{
    uint16_t bytes_processed = 0;

    while (bytes_processed < (uint16_t)tmp_count)
```

Function *ModbusCheckRequestLengths* ***in file*** *modbus_decode.c*

Can you already spot a potential problem? Let's go over the steps for exploitation:

Step 1

The tmp_count parameter is initialized using a value from the packet->payload and represents the number of bytes remaining in the payload, according to the payload_length parameter. After tmp_count is set, we enter a while-loop with the exit condition of bytes_processed < tmp_count. Therefore, so far tmp_count = 10, which is constant and won't change during the loop. As long as bytes_processed remains less than 10, the while-loop will continue looping.

To do so, let's look at the content of the while-loop. We see that bytes_processed is affected by the record_length parameter, which consists of two bytes from the Modbus payload (Group → Record Length).

```
{
    uint16_t record_length = 0;

    /* Check space for sub-request header info */
    if ((modbus_payload_len - bytes_processed) <
            MODBUS_FILE_RECORD_SUB_REQUEST_SIZE)
        break;

    /* Extract record length.
    MODBUS_MIN_LEN = 8
    MODBUS_BYTE_COUNT_SIZE = 1
    MODBUS_FILE_RECORD_SUB_REQUEST_LEN_OFFSET = 5

    */
    record_length = *(packet->payload + MODBUS_MIN_LEN +
        MODBUS_BYTE_COUNT_SIZE + bytes_process  ed +
        MODBUS_FILE_RECORD_SUB_REQUEST_LEN_OFFSET);

    record_length = record_length << 8;

    record_length |= *(packet->payload + MODBUS_MIN_LEN +
        MODBUS_BYTE_COUNT_SIZE + bytes_processed +
        MODBUS_FILE_RECORD_SUB_REQUEST_LEN_OFFSET + 1);

    /* Jump over record data.
    MODBUS_FILE_RECORD_SUB_REQUEST_SIZE = 7
     */
    bytes_processed += MODBUS_FILE_RECORD_SUB_REQUEST_SIZE +
                        2*record_length;
```

***Function** ModbusCheckRequestLengths **in file** modbus_decode.c*

Step 2

The record_length parameter is of type uint16_t, with a value from the user-controlled Modbus payload. bytes_processed is also uint16_t, and is calculated by multiplying record_length by 2 + the sub-request header size, which is 7.

However, the result of the multiplication can be more than the maximum uint16_t size, thus overflowing the value.

For example:

- **record_length** = 0xfffe
- **MODBUS_FILE_RECORD_SUB_REQUEST_SIZE** = 7

- **bytes_processed** = 7 + (2 * 0xfffe)

In this example, the bytes_processed will be 0x20003, which is:

0000000000000010|0000000000000011

High **Low**

In binary, when the result is cast to uint_16t, the lower 16 bits are kept, meaning the bytes_processed will be 0000000000000011, which equals 3. If the bytes_processed is 3, we do not exit the while-loop, because 3 (bytes_processed) < 10 (tmp_count), and we enter another iteration.

Step 3

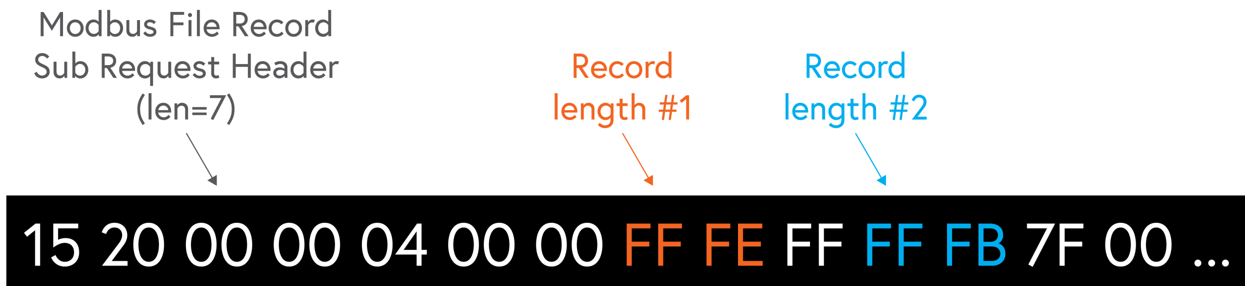
Now, the new record_length will be taken from the user-controlled payload, from a specific offset that is partially affected by the bytes_processed value. Since we fully control the value of the bytes_processed using the integer-overflow bug, we can craft the payload in such a way that the newly calculated record_length will be any number we choose.

Therefore, if the next value that is read into the record_length (allegedly the next group's record length) is 0xfffb, then the bytes_processed will be calculated as follows:

bytes_processed = bytes_processed + MODBUS_FILE_RECORD_SUB_REQUEST_SIZE + 2*record_length

bytes_processed = 3 + 7 + 2*(0xfffb) = 0

So, the bytes_processed is now 0.The next time we enter the while-loop, the bytes_processed is 0 again, so we will go through steps 2 and 3 over and over, until the process is terminated by the user. This essentially keeps the process stuck in the while-loop (steps 2 & 3) "blinding" it forever. In this state, Snort will not process new packets and will not alert.



Conclusion

Successful exploits of vulnerabilities in network analysis tools such as Snort can have devastating impacts on enterprise and OT networks.

CVE-2022-20685, uncovered by Team82, targeted just one facet of this popular network intrusion detection and prevention system. It can be exploited remotely to create a denial-of-service condition in Snort, keeping it from processing new packets, and generating alerts.

Team82 believes network analysis tools are an under-researched area that deserves more analysis and attention, especially as OT networks are increasingly being centrally managed by IT network analysis familiar with Snort and other similar tools.

CVE-2022-20685

CWE-190: Integer Overflow or Wraparound

CVSSv3 score: 7.5

Description: An integer overflow vulnerability in the Snort Modbus OT preprocessor enables an attacker to remotely send a crafted packet to a vulnerable system, triggering an infinite while-loop and creating a denial-of-service condition.