Regular Paper

IoTProtect: Highly Deployable Whitelist-based Protection for Low-cost Internet-of-Things Devices

Chun-Jung Wu[1,a)] Ying Tie[1,b)] Satoshi Hara[2,c)] Kazuki Tamiya[1,d)]
Akira Fujita[3,4,e)] Katsunari Yoshioka[3,4,f)] Tsutomu Matsumoto[3,4,g)]

Received: November 28, 2017, Accepted: June 8, 2018

Abstract: In recent years, many Internet-of-Things (IoT) devices, such as home routers and Internet Protocol (IP) cameras, have been compromised through infection by malware as a consequence of weak authentication and other vulnerabilities. Malware infection can lead to functional disorders and/or misuse of these devices in cyberattacks of various kinds. However, unlike personal computers (PCs), low-cost IoT devices lack rich computational resources, with the result that conventional protection mechanisms, such as signature-based anti-virus software, cannot be used. In this study, we present IoTProtect, a light-weight, whitelist-based protection mechanism that can be deployed easily on existing commercial products with very little modification of their firmware. IoTProtect uses a whitelist to check processes running on IoT devices and terminate unknown processes periodically. Our experiments using four low-cost IoT devices and 4,981 in-the-wild malware binaries show that IoTProtect successfully terminated 99.92% of the processes created by the binaries within 44 seconds after their infection with central processing unit (CPU) overhead of 24% and disk space overhead of 288 KB.

Keywords: IoT, malware process, Whitelisting

1. Introduction

The Internet of Things (IoT) is a network of physical devices, such as vehicles, furniture, and buildings, embedded with electronics, sensors, and network connectivity. Connectivity enables these objects to collect and exchange data for further applications and business use. However, a threat from IoT malware has materialized. In Oct. 2016, an IoT Malware called Mirai, reported to have infected over 100,000 compromised IoT devices such as Internet Protocol (IP) cameras, conducted the largest ever distributed denial of service (DDOS) attack against Dyn DNS [1]. We have been using IoTPOT [2], a honeypot that monitors attacks on IoT devices, to observe cyberattacks against such devices and analyze the threats from IoT malware. As shown in **Fig. 1**, the number of attacking hosts, many of which are indeed IoT devices compromised and misused by attackers, has increased rapidly since Aug. 2016.

Our observations show that most of the compromised devices are home routers [3] and IP cameras [4]. Although many security vendors have developed commercial anti-virus software for embedded systems, such as those listed in **Table 1**, these are not suitable for protecting the above-mentioned low-cost devices as a result of resource constraints and unsupported platforms. Moreover, all of the commercial products require substantial modification of the firmware that would incur high engineering costs, especially if the manufacturer wants to deploy the security product on existing products.

In addition to this commercial security software, there are many studies that deal with the protection of low-cost IoT devices [11]. These have deployment costs similar to the commercial options resulting from required firmware modifications.

The remainder of this paper is structured as follows. The second section contains a literature review of related work and existing techniques. The third section presents preliminaries. The fourth section presents the details of IoTProtect. After explaining our method, we introduce the evaluation experiments. Finally, our results are discussed, and conclusions are drawn.

2. Related work

Pareek et al. consider that blacklisting-based solutions for detecting malware suffer from problems of false positives and negatives. They share the idea of application whitelisting that has been applied by security vendors and various other solutions. They also provide details regarding the design and implementation approaches and discuss challenges to developing an effective whitelisting solution [12].

Obermeier et al. apply whitelisting to applications for protecting industrial automation and control systems. They find application whitelisting to be an effective means of preventing the instal-

1 Yokohama National University, Yokohama, Kanagawa 240–8501, Japan
2 FUJI SOFT INCORPORATED, Yokohama, Kanagawa 231–8008, Japan
3 Graduate School of Environment and Information Sciences, Yokohama National University, Yokohama, Kanagawa 240–8501, Japan
4 Institute of Advanced Sciences, Yokohama National University, Yokohama, Kanagawa 240–8501, Japan
a) wu-chun-jung-zt@ynu.jp
b) tie-ying-fc@ynu.jp
c) tamiya-kazuki-gj@ynu.jp
d) harasato@fsi.co.jp
e) fujita@ynu.ac.jp
f) yoshioka@ynu.ac.jp
g) tsutomu@ynu.ac.jp

© 2018 Information Processing Society of Japan

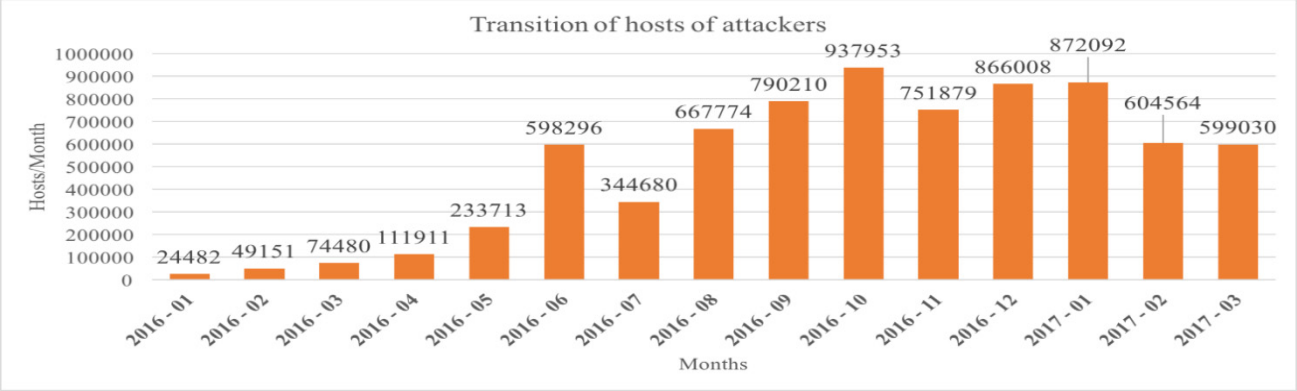

Fig. 1 Statistics regarding attacking hosts observed by IoTPOT from January 2016 to March 2017.

Table 1 Commercial secure software against embedded systems.

Product name	Supported platform	Min. Resource Constraint	Other Constraint
McAfee Embedded Control 6.x [5]	RHEL 4/5/6/7 CentOS 5/6/7 SuSE 10/11 Open SuSE 10/11 Solaris 9/10 WEPOS, POSReady 2009 Windows Embedded Systems (WES) 2009 Windows XP/vista/7/8 Windows NT/2000/2003/2008	512 MB RAM 80 MB free disk space	Rebuild kernel module [9]
Kaspersky Embedded Systems Security 2.0 [6]	Windows xp/7/8/10 WEPOS 2009/7 WES xp/7/8 Windows 10 Redstone Windows 10 IoT Ent	256 MB RAM 50 MB free disk space	N/A
Trend Micro OfficeScan 10.6 [7]	WEPOS 2009/7 WES XP/vista/7	256MB RAM 350 MB free disk space	N/A
Symantec Critical System Protection 5.2 (Agent) [8]	RHEL 5/6 CentOS 5/6 SUSE 8/9/10/11 Solaris 10/11 Oracle 5/6 QNX IBM AIX 5L HP Unix 11 WEPOS, 2009/7 WES xp/7 Windows XP/vista/7/8 Windows 2003/2008/2012	256 MB RAM 100 MB free disk space	Additional management server [10]

lation of malware [13].

Bhardwaj et al. developed a process monitoring system based on blacklisting and whitelisting of process names [14]. Further, they developed an application called Debsums to calculate the MD5 sums of an installed package and compare them with those from existing processes [15]. However, an adversary can easily alter process names and thus evade detection.

Paleari et al. present an architecture for the automatic generation of procedures for recovery from malicious programs. This method extracts the behavior of applications and monitors system calls using QEMU, an emulator and monitor of virtual machines. In addition, they propose clustering the behavior of malware to construct recovery procedures [16].

In 2011, Shahzad et al. proposed a classification-based method that analyzes a minimal feature set of 11 features for distinguishing benign and malicious processes. This method provides 93% detection accuracy with a 0% false alarm rate within 100 milliseconds [17].

In 2017, Tamiya et al. proposed a method for disinfecting IoT devices by merely rebooting or resetting the infected devices [18]. Their experiments show that 45 existing IoT malware could be erased by the simple operation of rebooting, but they did not present a detection method for these malware binaries.

Koike et al. developed a whitelisting-based execution control technique called WhiteEgret for the Linux operating system (OS). This module uses the bprm_check_security hook and the mmap_file hook to monitor the absolute path of executable files. WhiteEgret permits execution if the absolute path is contained in the whitelist and the hash value of the executable file is also contained in the hash value whitelist [11].

There are some secure Linux OS with implementations of mandatory access control (MAC), such as TOMOYO Linux [19] and SELinux [20]. The secure OS hooks system calls in the kernel space and checks the validity with predefined policies files. This solution incurs efforts of labeling and policies files. Moreover, for existing IoT devices, applying secure OS result in substantial modification of firmware.

As shown in the above literature review, there is no existing research on IoT cybersecurity conducted on low-cost devices together with a process-level defense mechanism other than [11]. Moreover, all of the existing technologies require substantial modification of firmware and incur a significant engineering cost if deployed on existing products. We propose a protection method that is very light-weight and easy to deploy on existing low-cost IoT devices.

© 2018 Information Processing Society of Japan

```
address           perms offset  dev   inode       pathname
00400000-00452000 r-xp 00000000 08:02 173521      /usr/bin/dbus-daemon
00651000-00652000 r--p 00051000 08:02 173521      /usr/bin/dbus-daemon
00652000-00655000 rw-p 00052000 08:02 173521      /usr/bin/dbus-daemon
00e03000-00e24000 rw-p 00000000 00:00 0           [heap]
00e24000-011f7000 rw-p 00000000 00:00 0           [heap]
...
35b1800000-35b1820000 r-xp 00000000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a1f000-35b1a20000 r--p 0001f000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a20000-35b1a21000 rw-p 00020000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a21000-35b1a22000 rw-p 00000000 00:00 0
35b1c00000-35b1dac000 r-xp 00000000 08:02 135870  /usr/lib64/libc-2.15.so
35b1dac000-35b1fac000 ---p 001ac000 08:02 135870  /usr/lib64/libc-2.15.so
35b1fac000-35b1fb0000 r--p 001ac000 08:02 135870  /usr/lib64/libc-2.15.so
35b1fb0000-35b1fb2000 rw-p 001b0000 08:02 135870  /usr/lib64/libc-2.15.so
```

Fig. 2 Format of the maps [25].

```
rootfs / rootfs rw 0 0
/dev/root / squashfs ro,relatime 0 0
devtmpfs /dev devtmpfs rw,relatime,size=127672k,
proc /proc proc rw,relatime 0 0
tmpfs /tmp tmpfs rw,relatime 0 0
sysfs /sys sysfs rw,relatime 0 0
devpts /dev/pts devpts rw,relatime,mode=600 0 0
/dev/mtdblock4 /jffs jffs2 rw,noatime 0 0
usbfs /proc/bus/usb usbfs rw,relatime 0 0
```

Fig. 3 Filesystems of ASUS Wi-Fi router RT-AC3200.

3. Preliminaries

3.1 Linux processes information

Linux is a free OS developed by many companies and groups. The GNU/Linux system is the core component, which is branched off into many different Linux distributions [21]. Among these distributions, such as Fedora, Ubuntu, Debian, and Mandriva Linux, there is a common design called the proc filesystem for providing system information to users or applications. This filesystem is not associated with any hardware device such as disk drives. Instead, proc is an agent into the running Linux kernel. Files in the proc filesystem are objects that behave similarly to normal files but provide access to parameters, data structures, and statistics in the kernel. The contents of these files are not always fixed, but are generated on the fly by the Linux kernel. For embedded Linux systems, users can use open source tools such as the Yocto Project to produce their distribution [22]. The Yocto tool retains the feature that supports the proc and sys filesystems [23]. Therefore, users and applications can read process information using proc on an embedded Linux system as long as the device developers are willing.

The proc filesystem contains a directory entry for each process running on a Linux system. The name of each directory is the process identifier (PID) of the corresponding process. These directories appear and disappear dynamically as processes start and terminate on the system. Each directory contains several entry files providing information regarding the running process [24]. There are three entry files that contain pathname or filename information regarding the binary of the corresponding process:

- The exe file is a symbolic link that points to the executable image running in the process.
- The maps file displays the range of addresses in the process's address space into which the file is mapped, the permissions on these addresses, the name of the file, and other information.
- The cmdline file records the complete command line for the process unless the process is a zombie or kernel. In the zombie process, there is nothing in this file. The general cmdline file contains Linux commands or pathname of executable files, together with arguments [25].

As shown in **Fig. 2**, users and applications can find the pathname of the running process. Moreover, if there is a whitelist of benign binaries' pathnames, we can distinguish between normal and abnormal processes.

3.2 Files in IoT devices

In this research, we focus on Linux-based IoT devices because many open-source OS's for IoT devices are based on Linux distributions, such as Brillo, OpenWrt, and Ostro Linux [26]. Linux has a single hierarchical directory structure that starts from the root directory, represented by / and then expands into subdirectories. The Filesystem Hierarchy Standard (FHS) defines the directory structure and contents in most Linux distributions [27]. However, IoT devices can apply various storages such as flash storage. This kind of IoT device can contain multiple filesystems in one device. For example, the ASUS Wi-Fi router RT-AC3200 mounts nine filesystems, according to the `/proc/mounts` file shown in **Fig. 3**. The format and meaning of each line are as follows [25], [28]:

1. The first field specifies the device that is mounted.
2. The second field specifies the mount point.
3. The third field specifies the filesystem type.
4. The fourth field describes whether the filesystem is mounted read-only (ro) or read-write (rw).
5. The fifth field is used by the `dump` command to determine which filesystems are to be dumped.
6. The sixth field is used by the `fsck` command to determine the order in which filesystem checks are performed at boot time.

The rootfs filesystem is a simple filesystem that exports Linux's disk caching mechanisms as a dynamically resizable random access memory (RAM)-based filesystem [29]. Squashfs is a compressed read-only filesystem for Linux and is intended for general read-only filesystem use, for archival [30]. Devtmpfs permits the kernel to create an instance called devtmpfs very early at kernel initialization. Every device will provide a device node in devtmpfs before any driver-core device is registered. Devtmpfs can be changed and altered by users at any time [31]. The proc filesystem contains system information, and the files in `/proc` are generated by the kernel on the fly [24]. The tmpfs filesystem is a temporary file storage facility on many Unix-like operating systems. It does not use traditional non-volatile media to store file data; instead, tmpfs files exist solely in a virtual memory maintained by the UNIX kernel [32]. Sysfs is a pseudo filesystem provided by the Linux kernel that exports information regarding various kernel subsystems, hardware devices, and associated device drivers [33]. Devpts is a virtual filesystem available in the Linux kernel since version 2.1.93 (April 1998). It is usually mounted at `/dev/pts` and contains solely device files representing slaves to the multiplexing master located at `/dev/ptmx` [34]. JFFS2 is a log-structured filesystem designed for use on flash devices in embedded systems. It is based on the work begun in the original JFFS by Axis Communications, AB [35]. The usbfs filesystem is

© 2018 Information Processing Society of Japan

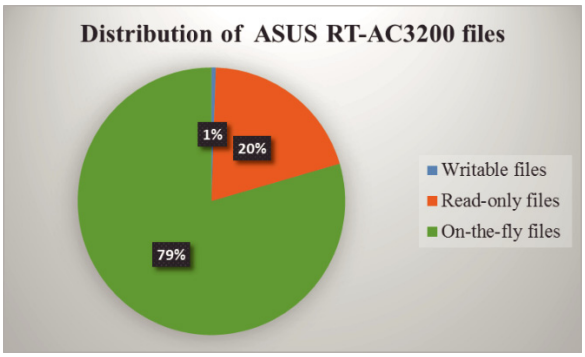

Fig. 4 Distribution of ASUS RT-AC3200 files.

a dynamically generated one, similar to the proc filesystem. Usbfs complements the normal device node system and can be used to write user space device drivers [36].

Based on the privileges and features of these filesystems, we categorize three kinds of files in Linux-based IoT devices:

- Writable files
- Read-only files
- On-the-fly files

Writable files are those that come from user-writable filesystems. Most of them are the input/output (I/O) of systems or configuration files. A read-only filesystem comes from some mounted image or read-only filesystems. The files are libraries and applications for creating the functions and services of IoT devices. On-the-fly files are the files that are in the proc or usbfs filesystems, are in the kernel, or are generated dynamically by users. The whitelist criteria are simple. First, ignore on-the-fly files because they are system information entries or mounted by USBs outside the device. Secondly, create the whitelist of pathnames by read-only files. There are many libraries and executable files in a read-only filesystem. Finally, create the whitelist of hashing values by writable files. For example, there are 14,514 files in the ASUS RT-AC3200 Wi-Fi router. The distribution of the files is shown in **Fig. 4**. Of these files, 79% are on-the-fly files generated by the kernel. Therefore, the number of files to be whitelisted is only 3,048.

3.3 Major premises of IoTProtect

We assume that IoTProtect is implemented by the device developers and uses the whitelists they created. There are four conditions. First, the checker and whitelists must be merged into the kernel or executed in the initial process to prevent attackers or malware from killing the checker process. Second, we assume that the developers do not use the mmap function to produce anonymous mappings. There is a case when the pathname fields of maps in `/proc` are blank. This is an anonymous mapping as obtained via the mmap function of the system call. There are no easy means of coordinating this back to a process's source, as there is no field giving the pathname [25]. Therefore, this is a constraint under which the developers must develop their devices in order to implement IoTProtect. More precisely, when loading files into memory, they must not set the MAP_ANONYMOUS argument for creating the memory mappings. Third, the exe files in `/proc` can sometimes lose the links to the pointed files. Under Linux 2.2 and later, the exe files in `/proc` are a symbolic link containing the actual pathnames of the executed commands. Attempting to open an exe file will indeed open the original executable. However, this symbolic link can typically be dereferenced. If the pathname has been unlinked, the symbolic link will contain the string '(deleted)' appended to the original pathname. In a multithreaded process, the contents of this symbolic link are not available if the main thread has already been terminated [25]. Developers must not dereference the exe link to create hash values of the executable binaries. Moreover, the hash algorithm must be available on the IoT device. Note that we use MD5 for the actual implementation of IoTProtect tested in the following evaluation. Fourth, if the developer would like to apply whitelists of cmdline content, the libraries and application files must be allocated in read-only partitions. Furthermore, the full or unique pathname of the corresponding binaries must be included in the command line to prevent file replacement or false positives.

4. IoTProtect

IoTProtect is a whitelisting method for protecting low-cost IoT devices. IoTProtect consists of three whitelists and a checker program. The pathname whitelist is a list of pathnames of all legitimate executables. The hash value whitelist records MD5 hash values of binaries on IoT devices. The cmdline whitelist is a list of string which we extract from cmdline files of the legitimate processes. The comparison and whitelist of cmdline content are optional and performed only if there are processes that cannot display their pathname and exe links in the proc filesystem.

We first explain the creation of whitelists. Here we assume that the device to be protected has already been developed and that the device developer is to install IoTProtect on top of the existing system. We skip the files coming from on-the-fly filesystems, such as sysfs, proc, usbfs, and I/O files. If developers know precisely which executable files to include on the whitelist, they can create their own whitelist manually. However, recent IoT device products are often not developed by a single manufacturer, and each developer does not know all of the legitimate executables exactly. In such a case developers can still create whitelists that include all executables existing in the system by using the Linux command `find` with the `-exec` expression and `md5sum`. Moreover, the cmdline whitelist can be created by `find` with the `-exec` expression and `cat` Linux commands.

IoTProtect conducts process checks through the following steps. The input data come from entry files of the proc filesystem and whitelists. The output is the removal of malicious processes. The notations used in the pseudocode are shown in **Table 2**.

We explain the details of the IoTProtect procedures with the following pseudocode:

1. while true
2. find and grep Pn_i from M, i = 1 to n
3. PN ← $\{Pn_1, Pn_2, \ldots, Pn_n\}$
4. Comp (PN, WLP)
5. SP ← $\{Pid_1, Pid_2, \ldots, Pid_j\}$ $\forall Pid_i \in A$: $Pn_i \notin$ WLP, I = 1 to j
6. $H_j \leftarrow MD5(E_j)$ $\forall E_j \in B$: $Pid_i \in$ SP, j = 1 to |SP|

© 2018 Information Processing Society of Japan

Table 2 Table of symbols.

Notations	Definitions and Descriptions
T	Integer variable, the period of process checker
N	Integer variable, number of processes to be checked
M	Set of /proc/[pid]/maps files
Cmd	Set of /proc/[pid]/cmdline files
CL	Set of cmdline
PN	Set of pathnames
WLP, H, C	Whitelists of the pathname, hash value, and cmdline
Pn_1, Pn_2	Entity of pathname
Pid_1, Pid_2	Entity of process id
E_1, E_2	Entity of exe links
H_1, H_2	Entity of hash value
SP	Set of suspicious process id
Cl_1, Cl_2	Entity of cmdline content
MD5 (E_i)	Calculate the MD5 hash value of the linked binary
Comp (S, WL)	Comparison of set S and whitelists
Kill (S)	Kill the process of set S by its Pid
Sleep (t)	Pause process checker of IoTProtect for t seconds
←	Assignment
-	Remove entities from the set
A, B, C, D	Sets
|D|	Size of set 'D'

7. H ← {H_1, H_2, . . . , H_j}, j=1 to |SP|
8. Comp (H, WLH)
9. SP − {Pid_1, Pid_2, . . . , Pid_k} ∀Pid_k ∈ C: H_k ∈WLH
   /*optional step of cmdline whitelisting */
10. find and grep Cl_i from Cmd, i=1 to n
11. CL ← {Cl_1, Cl_2, . . . , Cl_i}, i = 1 to n
12. Comp (CL, WLC)
13. SP − {Pid_1, Pid_2, . . . , Pidr} ∀Pidr ∈ D: CL_r ∈ WLC
    /*optional step of cmdline whitelisting */
14. Kill (SP)
15. Sleep (t)
16. Endwhile

IoTProtect first filters processes that are not included in the pathname whitelist, and then filters the remaining processes according to the hash value whitelist. It then filters the remainder with the cmdline whitelists if there are any processes with no pathname and exe links. Finally, it removes all remaining processes.

5. Evaluation

We developed a prototype of IoTProtect using shell scripts with Linux commands and AWK scripts, such as grep, find, and head. We conducted experiments with four actual IoT devices and 4,981 malware binaries captured by our IoT honeypot for evaluation. We show three different experiments to evaluate the effectiveness and overhead of IoTProtect.

5.1 Data collection and experimental devices

We chose four IoT devices for conducting experiments. These devices were known to be vulnerable and compromised by IoT malware [37], [38], [39]. The brands and specifications of the devices are listed in **Table 3**. According to their disk information, previous commercial products cannot be installed on the four devices. The Dahua IPC-HFW3300 does not support MD5 hash libraries. Therefore, IoTProtect checks only the pathnames and cmdline of corresponding processes in the IP Camera.

Table 3 IoT devices used in conducting the experiments.

IoT Device	Model	CPU Arch.	CPU spec.	RAM (MB)	Disk (MB)	Appr. Price
Dahua IPC-HFW3300	IP Camera	ARM	300 MHZ	92	8	325 USD
ASUS RT-AC3200	Wi-Fi Router	ARM	1GHz 2cores	256	30.8	199.99 USD
ASUS RT-N66U	Wi-Fi Router	MIPSEL	600 MHZ	256	22.3	84.99 USD
PRINCETON ShAirDisk	Wi-Fi storage	MIPSEL	360 MHZ	28	4.6	26.7 USD

Table 4 IoT malware used for conducting the experiments.

CPU	Bashlite	Tsunami	Mirai	sum.
ARM	3123	51	74	3248
MIPSEL	1514	27	192	1733
All	4637	78	266	4981

IoTPOT collected 4,981 different IoT malware binaries for ARM and MIPSEL from January 2016 to March 2017. The malware labels, such as Bashlite, Tsunami, and Mirai, come from local scans by Kaspersky, an anti-virus engine that we consider, from our previous experience of submitting 12,000 samples to the Virustotal web service application programming interface (API), to be one of the most reliable anti-virus products for IoT malware [40]. VirusTotal is a website that aggregates many antivirus products and online scan engines [41]. The distribution of our malware samples is shown in **Table 4**.

5.2 Removal experiment

We conducted experiments involving the malware removal process on the four IoT devices as follows:

1. Login to the device as root via telnet.
2. Download malware using the `wget` or `tftp` command.
3. Assign the `755` privilege to the malware binary.
4. Execute the downloaded malware.
5. Conduct a process check using IoTProtect
6. Check whether IoTProtect can kill the malware process.

5.2.1 Experimental Results

The results are shown in **Table 5**. We see that there are many segmentation faults (7% to 14%) and bus errors (0% to 0.8%) when we execute the malware binaries on these devices. There are two ARM malware binaries and one MIPSEL binary that finished execution before we started process checks of IoTProtect. These three malware binaries are similar and contain the same functions in their binaries. All they attempted was to install python on target devices using `apt-get` and `yum`. When the installation failed as a result of the installation utilities not being available on these devices, the malware simply terminated. The

© 2018 Information Processing Society of Japan

Table 5 Results of the removal experiments.

IoT Device	model	CPU Arch.	Kill	segmentation fault	bus error	fail
IPC-HFW3300	IP Camera	ARM	2774	446	26	2
RT-AC3200	Wi-Fi Router	ARM	2732	483	31	2
RT-N66U	Wi-Fi Router	MIPSEL	1608	123	1	1
ShAirDisk	Wi-Fi storage	MIPSEL	1622	108	2	1

Table 6 IoTProtect overheads.

IoT Device	Model	Disk overhead	CPU overhead	Time cost
IPC-HFW3300	IP Camera	124.5K	24%	44 sec
RT-AC3200	Wi-Fi Router	288.4K	7%	2 sec
RT-N66U	Wi-Fi Router	176.5K	17.6%	2 sec
ShAirDisk	Wi-Fi storage	155.5K	19%	4 sec

complete execution of the malware takes less than one second, and the process disappeared after termination. The purpose of the malware is to infect an IoT OS that can install python, such as the IBM Watson IoT Platform [42]. IoTProtect successfully removed the processes of all but three of the malware binaries. The success rate of removal by IoTProtect against triggered malware was 99.92% if the above three cases are considered as failed protection, but was 100% if they are considered as successful protection because the malware could not function properly.

The overheads of IoTProtect on the four devices are shown in **Table 6**. The disk overheads include the sizes of whitelists. The size of the IoTProtect checker program is only 1.6 KB. Our pathname whitelist includes all of the pathnames from the read-only filesystem. The manufacturer of a device might use a more efficient whitelist. The central processing unit (CPU) overheads are the maximum values during execution time. The three devices other than the IPC-HFW3300 can finish a process check of IoTProtect in four seconds. Despite the fact that the IP Camera spent 44 seconds executing the checker program, the original monitor and display systems of the camera functioned normally without delays.

5.3 Mitigating outgoing attacks

In reality, IoTProtect would continuously check existing processes in some designated scan intervals. Therefore, it is important to ask whether IoTProtect is sufficiently fast to kill a malware process before it conducts outgoing attacks against other devices. To measure the worst case, we chose a Mirai variant, one of the fastest spreading IoT worms, which conducts a telnet scan on port 2323/tcp right after its execution before even connecting its command-and-control server. The MD5 hash value of

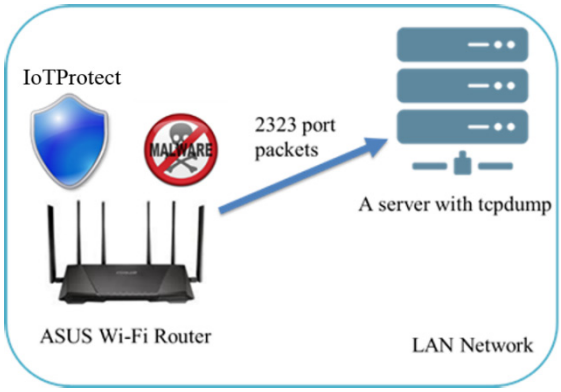

Fig. 5 Experimental environment for measuring outgoing attack mitigation by IoTProtect.

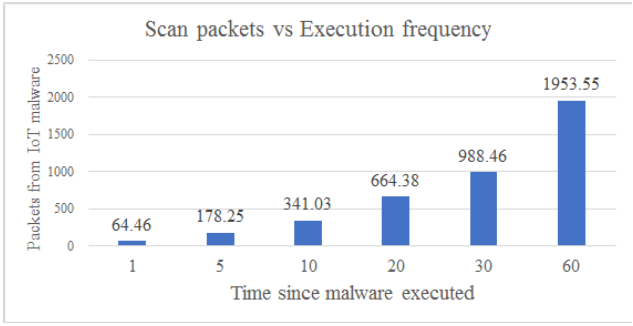

Fig. 6 Results ofexperiments onmitigating outgoing attacks.

the sample is "d6e99a59d44b83e8360745145fa5d2b3."

5.3.1 Design of experiment

As shown in **Fig. 5**, we conducted this experiment on the ASUS RT-AC3200 Wi-Fi Router. All traffic is contained in a LAN network. At the beginning of the experiment, we executed malware. After a fixed scan interval, we executed IoTProtect to conduct a process check. To simulate different detection timings, we started the process check of IoTProtect at one, five, ten, 20, 30, and 60 seconds after malware execution, respectively. We then measured how many packets went out from the devices before the IoTProtect checker killed the malware process. We conducted this trial 100 times for each setting.

5.3.2 Experimental results

The results of the experiment are illustrated in **Fig. 6**. Those results confirm that IoTProtect cannot block every scan by Mirai but does reduce the number of scan packets significantly. Measurement shows that this Mirai variant generates nearly 2,000 scan packets for one minute after it begins its execution and would continue to scan at the same rate if it were not killed by IoTPOT.

5.4 Trade-off between security and device performance

We measured the impact of IoTProtect on the performance of the devices. We chose a low-cost device, ShAirDisk, and analyzed the trade-off between the security and overhead of IoTProtect.

5.4.1 Experimental design

As illustrated in **Fig. 7**, we conducted this experiment in a location at which there were no other Wi-Fi access points. Then, we uploaded a 200 MB file five times into Wi-Fi storage to measure the device performance for uploading files. We conducted

© 2018 Information Processing Society of Japan

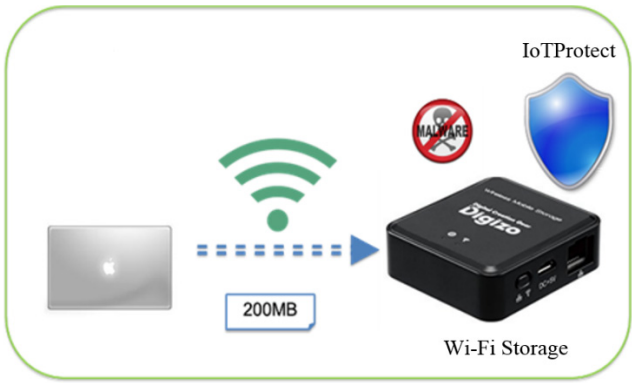

Fig. 7 Experimentalenvironment formeasuringthetrade-off between performance and security.

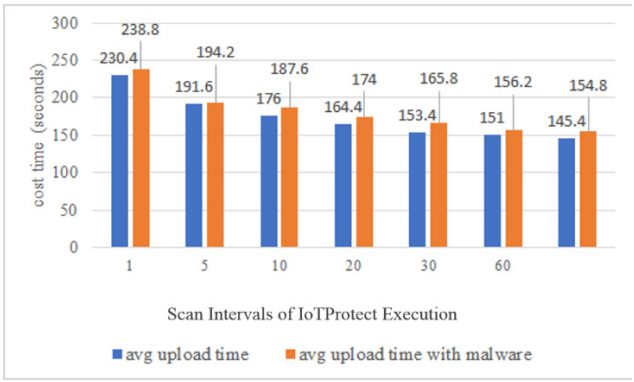

Fig. 8 Results of experimentsmeasuring trade-off.

this experiment under two conditions, one with only IoTProtect running and the other with IoTProtect and malware running. The same procedure as in the previous experiment was followed for malware execution. The MD5 hash of the Mirai variant used for this experiment is "018cb18e9cb415af453ee020fa33aa28."

5.4.2 Experimental results

Figure 8 presents the different upload time costs under different scan intervals of IoTProtect. In the figure, the values of the blue bars are the average upload times in a situation with only IoTProtect. The values of the orange bars are the average upload times in the situation in which both malware and IoTProtect are executed. We can see that the differences between the orange and blue bars in the same scan interval are not significant, measuring less than 12.4 seconds. This means the malware infection caused a limited delay of fewer than 12.4 seconds of file upload time. However, if we shorten the scan interval of IoTProtect process checks to 1.0 second to increase security, the overhead increases significantly, measuring a 55% increase in upload time compared to the case without IoTProtect. On the other hand, we can also see that scan intervals of more than 30 seconds do not harm performance significantly.

5.5 Evaluation of easy deployment

The deployment of IoTProtect involves two steps, the creation of whitelists and the installation of the IoTProtect checker. We create worst-case whitelists as we are not the developers of these devices. These whitelists can filter out only sysfs, proc, usbfs, and I/O files. The time costs are shown in **Table 7**. General users can quickly create worse-case whitelists in only a few minutes.

Table 7 Cost of creating whitelists.

IoT Device	model	whitelists size	creating whitelists time cost
IPC-HFW3300	IP Camera	123.0K	*29 sec
RT-AC3200	Wi-Fi Router	285.9K	218 sec
RT-N66U	Wi-Fi Router	175.0K	157 sec
ShAirDisk	Wi-Fi storage	154.0K	188 sec

*This IP camera lacks hash libraries and contains some processes without pathname and valid exe links. We created the cmdline whitelist instead of hash value whitelists.

The installation procedure for the IoTProtect checker is very light and quick. The checker program is written using Bash scripts, leading to portability between different Linux distributions. Moreover, the fact that the size of the checker program is only 1.5 KB resulted in easy deployment on low-cost IoT devices. Finally, the installation procedures of this program include only a copy of a file and assignment of the execute privilege. The checker script was executed independently of most Linux kernel modules. Users can easily invoke it in the Linux startup process and have it run in the background or as a daemon.

6. Discussion

From the removal experiment, we see that our method applies to different CPU architectures and models of IoT devices. Furthermore, IoTProtect successfully removed several thousand different malware processes with nearly 100% success. According to the mitigation of outgoing attacks, IoTProtect reduced the scan attack traffic caused by a rapidly spreading Mirai variant, even if the process check is not very frequent. The results of the performance experiment show that IoTProtect can be installed in some low-cost devices without a significant drop in performance if the process checking interval is configured appropriately.

We found that IoTProtect was significantly slow when implemented in one of the tested devices, the Dahua IP Camera, as shown in Table 4. We could improve the performance of IoTProtect by implementing it in the C language and by reducing the size of the whitelists. According to a comparative study of programming languages in 2015, C is the best language for computing-intensive tasks [43]. Moreover, the whitelists we created for the experiments in the worst case contain thousands of pathnames and MD5 hashing values. Manufacturers can build much better whitelists for their products.

For mitigating outgoing attacks, we find that IoTProtect cannot block all outgoing scan packets. It can remove the malware process, but the malware has already conducted thousands of scan packets before it is killed. We consider this shortcoming to be a limitation of IoTProtect. If we shorten the scan interval of IoTProtect's process checks from 60 seconds to 20 seconds, 66% of scan packets can be reduced. Moreover, a scan interval of one second could stop 96.72% of the scan packets that could have been sent out in one minute. Note that we used a Mirai variant,

© 2018 Information Processing Society of Japan

one of the fastest known IoT worms that begins scanning right after execution as the worst-case scenario. However, most Mirai malware would connect the command-and-control server first and then start the scan and DoS attack after receiving commands. Hence, in most real cases, IoTProtect could have blocked most of the outgoing attacks within an acceptable scan interval.

Our performance experiment on Wi-Fi storage shows that the file upload speed of the device is significantly affected by the scan interval of IoTProtect's process check. On the tested Wi-Fi storage devices, the best scan interval can be 20 to 30 seconds, which will introduce a 7.1% to 12.4% increase in file upload time while protecting from and mitigating most of the attacks by the malware infection, as discussed above.

IoTProtect is easy to deploy, but the creation of the proper whitelists can take some effort. Supposing that the developers use some third-party libraries and an open source OS for their IoT products, they might know only the processes caused by their own applications and have limited information for all of the other benign processes. In such a case, the developers must pick up all execution files installed in the device, such as the files in `/bin` and `/usr/bin`, as we did in the experiment. When they conduct a software testing process, they must record all of the created processes to avoid false positive detection by IoTProtect.

6.1 Comparison with previous studies

- The method by Paleari et al. must apply QEMU and behavior clustering [16], which are too expensive to implement on low-cost devices.
- In Ref. [17], Shahzad et al. analyzed 11 features from the kernel and achieved 93% detection accuracy. However, the system requests many features, executes a decision tree algorithm, and is difficult to install on low-cost devices. IoTProtect, in contrast, was able to remove 99.92% of malware processes from four thousand malware binaries. We assume here that our method does not cause false positives as long as the whitelist is created appropriately by the device developers. However, as discussed in the previous section, the creation of whitelists can involve difficulties during the manufacturing process. Our future work will include investigating proper whitelisting.
- Tamiya et al. investigated a simple solution for malware removal by rebooting the device, which can be applied to low-cost IoT devices [18]. However, they do not offer the detection methodology of the malware infection, and they also mention that the connected vulnerable devices would again be infected after removal unless the vulnerability is fixed. Therefore, their solution would not be able to defend the device.
- There are platform and resource constraint issues for McAfee Embedded Control 6.x. These solutions cannot be installed on low-cost IoT devices. Moreover, McAfee Embedded Control 6.x must rebuild the kernel when installed on a Linux distribution, introducing significant engineering cost, especially if deployed on existing commercial products.
- Koike et al. developed a whitelisting-type execution control module WhiteEgret on Linux [11]. Similarly to McAfee Embedded Control, WhiteEgret also builds the Linux kernel upon installation, also introducing substantial engineering cost.

6.2 Limitations

IoTProtect does have some limitations. Many of the limitations come from the design of Linux process information and our whitelisting idea. First, IoTProtect depends on exe and maps entries in the proc filesystem. Kernel-level malware and toolkits that disable or alter these functionalities would evade detection by IoTProtect. Moreover, checks and removal by IoTProtect are performed on filesystems, with the result that code injection on a legitimate process in memory cannot be detected.

Second, the defense offered by IoTProtect is not prevention but mitigation of malware infection. It would help substantially in defending against long-lasting malicious activities such as DDoS, spamming, bitcoin mining, click fraud, and stepping stones for other attacks. On the other hand, attacks that can be performed in a very short time, such as credential and privacy data exfiltration, might not be mitigated well. Applying a whitelist before malware execution would require process creation hooking. We did not choose this approach for two reasons. First, the hooking of process creation would involve modification of the Linux kernel [44] and hence increase the deployment cost for device developers. We believe that IoTProtect is easier to implement and use than the hooking method. Second, hooking every process creation and checking all created processes before they are executed would slow down the principal functionality of the devices, especially at the time of device boot-up when many processes are created and checked.

When designing and developing new products of IoT devices, developers may be able to select and use secure OS distribution. However, changing an OS for existing already-developed products involves a lot of engineering efforts. If the manufacturers change the OS of an existing product to a secure OS solution, such as Tomoyo Linux [19] or SELinux [20], they have to modify a considerably large part of the firmware including rebuilding the kernel and installing necessary libraries, each of which requires a careful test to see the product still fulfill the specification of the product after the modification. They may be able to use an already prepared system image for some secure OS but since those images are not built to run on particular products, the developers still need to install additional libraries and programs to the image required by the products. Such customization also involves considerable engineering cost including the product tests.

Compared with these secure OS solutions, IoTProtect is user space solution without the need of kernel rebuilding and requires a simple script with few dependent programs such as hash function. As a result, the modification necessary to adopt IoTProtect is much smaller compared to the adoption of the secure OS. For a large scale of compromised IoT devices, such as Mirai, the observed population initially fluctuated between 200,000 and 300,000 devices with a brief peak of 600,000 devices from September 2016 to February 2017. The targeted devices consist of more than 84 kinds of devices [45]. Moreover, building

© 2018 Information Processing Society of Japan

a secure OS for each device will consume substantial engineering efforts and time. To cyberattacks like Mirai, IoTProtect is the ultimate solution to protect these devices.

There are four major conditions that a developer must follow in order to deploy IoTProtect as described in Section 3.3. These can be the constraints for device developers. In addition, if the conditions are not satisfied by existing devices, this might require additional effort to modify the firmware, thereby limiting the advantage of easy deployment. We can at least say that these conditions are satisfied for the four existing devices we tested.

7. Conclusions

We have shown that IoTProtect is a valid solution that can remove IoT malware processes with reasonable implementation and resource costs. Moreover, we implemented a shell script prototype and showed that it could be executed successfully on low-cost IoT devices, such as Wi-Fi routers and storage, with marginal cost. We tested more than four thousand different IoT malware binaries, and IoTProtect removed 99.92% of these malicious processes successfully.

Acknowledgments A part of this work was conducted under the auspices of the MEXT Program for Promoting the Reform of National Universities. A part of this work was funded by the WarpDrive: Web-based Attack Response with Practical and Deployable Research Initiative project, supported by the National Institute of Information and Communications Technology (NICT).

References

[1] Loshin, P.: Details emerging on Dyn DNS DDoS attack, Mirai IoT botnet (2016), available from ⟨http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet⟩ (accessed 2016-11-20).

[2] Pa, Y.M.P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T. and Rossow, C.: IoTPOT: A Novel Honeypot for Revealing Current IoT Threats, *Journal of Information Processing*, Vol.24, No.3, pp.522–533 (2016).

[3] Auchard, E.: Deutsche Telekom attack part of global campaign on routers (2016), available from ⟨https://www.reuters.com/article/us-deutsche-telekom-outages/deutsche-telekom-attack-part-of-global-campaign-on-routers-idUSKBN13O0X4⟩ (accessed 2017-11-26).

[4] Franceschi-Bicchierai, L.: How 1.5 Million Connected Cameras Were Hijacked to Make an Unprecedented Botnet (2016), available from ⟨https://motherboard.vice.com/en_us/article/8q8dab/15-million-connected-cameras-ddos-botnet-brian-krebs⟩ (accessed 2017-11-26).

[5] McAfee Embedded Control, available from ⟨http://support.intelsecurity.com/us/products/embedded-control.aspx⟩ (accessed 2017-11-20).

[6] Kaspersky Embedded Systems Security 2.0, available from ⟨https://support.kaspersky.com/kess2#requirements⟩ (accessed 2017-11-19).

[7] Supported embedded operating systems in OfficeScan 10.6, available from ⟨https://success.trendmicro.com/solution/1060451-supported-embedded-operating-systems-in-officescan-10-6⟩ (accessed 2017-11-19).

[8] Symantec™ Critical System Protection Version 5.2 RU9 MP6 Platform and Feature Matrix, available from ⟨https://symwisedownload.symantec.com/resources/sites/SYMWISE/content/live/DOCUMENTATION/8000/DOC8022/en_US/SCSP_Platform_Feature_Matrix.pdf?_gda_=1511283679_42c5dda9b7a1075c7b46cc29d7137977⟩ (accessed 2017-11-20).

[9] User Guide McAfee Embedded Control 6.5.1, available from ⟨https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/25000/PD25615/en_US/mec_651_ug_en_us.pdf⟩ (accessed 2017-11-20).

[10] Symantec Critical System Protection 5.2.9 Installation Guide, available from ⟨https://origin-symwisedownload.symantec.com/resources/sites/SYMWISE/content/live/DOCUMENTATION/5000/DOC5944/en_US/SCSP_Installation_Guide.pdf⟩ (accessed 2017-11-20).

[11] Koike, M., Ogura, N., Takumi, S., Hanatani, Y. and Haruki, H.: Development of WhiteEgret™: A Whitelisting-type Execution Control on Linux, *Computer Security Symposium 2017*, Session 3D3-4 (2017).

[12] Pareek, H., Romana, S. and Eswari, P.R.L.: Application whitelisting: Approaches and challenges, *International Journal of Computer Science, Engineering and Information Technology* (*IJCSEIT*), Vol.2, No.5 (2012).

[13] Obermeier, S., Schierholz, R. and Hristova, A.: Securing industrial automation and control systems using application whitelisting, *2014 IEEE Emerging Technology and Factory Automation* (*ETFA*), pp.1–4, IEEE (2014).

[14] Bhardwaj, R., Daftari, M., John, D., Shinde, N. and Deshpande, V.: Whitelisting and Blacklisting for Private Execution of Processes in Linux (2015).

[15] Debsums: check the MD5 sums of installed Debian packages, available from ⟨http://manpages.ubuntu.com/manpages/zesty/en/man1/debsums.1.html⟩ (accessed 2017-06-25).

[16] Paleari, R., Martignoni, L., Passerini, E., Davidson, D., Fredrikson, M., Giffin, J.T. and Jha, S.: Automatic Generation of Remediation Procedures for Malware Infections, *USENIX Security Symposium*, pp.419–434 (2010).

[17] Shahzad, F., Bhatti, S., Shahzad, M. and Farooq, M.: In-execution malware detection using task structures of linux processes, *2011 IEEE International Conference on Communications* (*ICC*), pp.1–6, IEEE (2011).

[18] Tamiya, K., Nakayama, S., Ezawa, Y., Tie, Y., Wu, C., Yang, D., Yoshioka, K. and Matsumoto, T.: Experiment on removal and prevention of IoT malware using real devices, *Symposium on Cryptography and Information Security 2017*, Session 3E1-5 (2017).

[19] Harada, T., Horie, T. and Tanaka, K.: Task oriented management obviates your onus on Linux, *Linux Conference*, Vol.3, p.23 (2004).

[20] Peter Loscocco, N.S.A.: Integrating flexible support for security policies into the Linux operating system, *Proc. FREENIX track: USENIX Annual Technical Conference* (2001).

[21] What is GNU/Linux?, available from ⟨http://www.getgnulinux.org/en/linux/⟩ (accessed 2017-06-21).

[22] Yocto Project, available from ⟨https://www.yoctoproject.org/⟩ (accessed 2017-11-27).

[23] Yocto Project Linux Kernel Development Manual, available from ⟨http://www.yoctoproject.org/docs/2.0.2/kernel-dev/kernel-dev.html⟩ (accessed 2017-11-27).

[24] Mitchell, M., Oldham, J. and Samuel, A.: Advanced linux programming, pp.147–156 (2001).

[25] Proc - process information pseudo-filesystem, available from ⟨http://man7.org/linux/man-pages/man5/proc.5.html⟩ (accessed 2017-06-21).

[26] Brown, E.: Open Source Operating Systems for IoT (2016), available from ⟨https://www.linux.com/news/open-source-operating-systems-iot⟩ (accessed 2017-07-30).

[27] Nguyen, B.: Linux Filesystem Hierarchy, Binh Nguyen (2003).

[28] fstab: static information about the filesystems, available from ⟨http://man7.org/linux/man-pages/man5/fstab.5.html⟩ (accessed 2017-06-21).

[29] Landley, R.: Ramfs, rootfs and initramfs (2005), available from ⟨https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt⟩ (accessed 2017-07-10).

[30] Lougher, P. and Lougher, R.: SquashFS (2008).

[31] G.K.H.: Driver Core: devtmpfs - kernel-maintained tmpfs-based /dev (2009), available from ⟨https://lwn.net/Articles/345480/⟩ (accessed 2017-07-10).

[32] Snyder, P.: Tmpfs: A virtual memory filesystem, *Proc. Autumn 1990 EUUG Conference*, pp.241–248 (1990).

[33] Mochel, P.: The sysfs filesystem, *Linux Symposium*, p.313 (2005).

[34] Brown, N.: Containers, pseudo TTYs, and backward compatibility (2016), available from ⟨https://lwn.net/Articles/688809/⟩ (accessed 2017-07-10).

[35] Woodhouse, D.: JFFS: The journalling flash filesystem, *Ottawa Linux Symposium*, Vol.2001 (2001).

[36] Hards, B.: The Linux USB sub-system, Sigma Bravo Pty Ltd., available from ⟨http://www.linux-usb.org/USB-guide/book1.html⟩.

[37] Vulnerability Details: CVE-2017-7253. (n.d.), available from ⟨http://www.cvedetails.com/cve/CVE-2017-7253/⟩ (accessed 2017-07-30).

[38] Cimpanu, C.: 40 Asus RT Router Models Are Vulnerable to Simple Hacks (2017), available from ⟨https://www.bleepingcomputer.com/news/security/40-asus-rt-router-models-are-vulnerable-to-simple-hacks/⟩ (accessed 2017-07-30).

[39] Sudo, T.: 無線 LAN 機器，出荷停止 サイバー攻撃に悪用の恐れ (2016)，available from ⟨http://www.asahi.com/articles/

© 2018 Information Processing Society of Japan

ASJDN5GJ5JDNUUPI00C.html⟩ (accessed 2017-07-30).
[40] VirusTotal Public API v2.0, available from ⟨https://www.virustotal.com/en/documentation/public-api/⟩ (accessed 2017-06-23).
[41] Lardinois, F.: Google Acquires Online Virus, Malware and URL Scanner VirusTotal, TechCrunch (2012), available from ⟨https://techcrunch.com/2012/09/07/google-acquires-online-virus-malware-and-url-scanner-virustotal/⟩ (accessed 2017-06-22).
[42] IBM Watson IoT, available from ⟨https://github.com/ibm-watson-iot⟩ (accessed 2017-06-23).
[43] Nanz, S. and Furia, C.A.: A comparative study of programming languages in Rosetta Code, *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (*ICSE*), Vol.1, pp.778–788, IEEE (2015).
[44] Morris, J., Smalley, S. and Kroah-Hartman, G.: Linux security modules: General security support for the linux kernel, *USENIX Security Symposium* (2002).
[45] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J. and Kumar, D.: Understanding the mirai botnet, *USENIX Security Symposium* (2017).

Chun-Jung Wu received his B.S. in mathematics from National Taiwan University, Taiwan in 2003 and M.S. in computer Science from National Taiwan University of Science and Technology, Taiwan in 2007. From 2008 to 2016, he was an engineer at Institute for Information Industry, Taiwan. Currently, he is a doctoral student at the Graduate School of Environment and Information Sciences, Yokohama National University. His research interest is IoT Security.

Ying Tie received her M.E. and Ph.D. in information sciences from Yokohama National University in 2012 and 2018 respectively. Her research interests include network security and malware analysis.

Satoshi Hara received his B.E. and M.E. in mechanical engineering from Meiji University in 2002 and 2005. He is currently a doctor course student at the Graduate School of Environment and Information Sciences, Yokohama National University. He also works as an engineer in the FUJISOFT INCORPORATED, Japan from 2005. His research interest is embedded device security.

Kazuki Tamiya is currently second year Mater student of Information Media and Environmental Science Course of Graduate School of Environmental Sciences, Yokohama National University. He is going to finish M.E. in Computer Engineering in March 2019. His research interest is IoT Security.

Akira Fujita received his B.A., M.Sc. and Ph.D. in information science from Yokohama National University in 2008, 2009 and 2012 respectively. After working as a researcher at Yokohama National University and National Institute of Informatics, Dr. Fujita is a project assistant professor at Yokohama National University. His research interests include network security, natural language processing and cognitive science.

Katsunari Yoshioka received his B.E., M.E. and Ph.D. degrees in Computer Engineering from Yokohama National University in 2000, 2002, and 2005, respectively. From 2005 to 2007, he was a Researcher at the National Institute of Information and Communications Technology, Japan. Currently, he is an Associate Professor at the Graduate School of Environment and Information Sciences, Yokohama National University. His research interest covers wide range of information security, including malware analysis, network monitoring, intrusion detection, etc. He was awarded 2007 Prizes for Science and Technology by The Commendation for Science and Technology by the Minister of Education, Culture, Science and Technology.

© 2018 Information Processing Society of Japan

Tsutomu Matsumoto is a professor of Faculty of Environment and Information Sciences, Yokohama National University and directing the Research Unit for Information and Physical Security at the Institute of Advanced Sciences. He received Doctor of Engineering from the University of Tokyo in 1986. Starting from Cryptography in the early 80's, he has opened up the field of security measuring for logical and physical security mechanisms. Currently he is interested in research and education of Embedded Security Systems such as IoT Devices, Network Appliances, Mobile Terminals, In-vehicle Networks, Biometrics, Artifact-metrics, and Instrumentation Security. He is serving as the chair of the IEICE Technical Committee on Hardware Security, the Japanese National Body for ISO/TC68 (Financial Services), and the Cryptography Research and Evaluation Committees (CRYPTREC) and as an associate member of the Science Council of Japan (SCJ). He was a director of the International Association for Cryptologic Research (IACR) and the chair of the IEICE Technical Committee on Information Security. He received the IEICE Achievement Award, the DoCoMo Mobile Science Award, the Culture of Information Security Award, the MEXT Prize for Science and Technology, and the Fuji Sankei Business Eye Award.

© 2018 Information Processing Society of Japan