

TECHNICAL REPORT

IIT TR-06/2023

Understanding the MIRAI botnet: scanning process, infection method and key features

F. Lauria

Understanding the MIRAI botnet: scanning process, infection method and key features

Filippo Maria LAURIA
filippo.lauria@iit.cnr.it

Technology Unit Computer and Communication Networks
Institute of Informatics and Telematics — Italian National Research Council
via G. Moruzzi, 1 — 56124 Pisa, Italy

Abstract

This document provides a comprehensive analysis of the MIRAI botnet, a sophisticated malware that specifically targets vulnerable Internet of Things (IoT) devices. The analysis focuses on the bot's infection process, key features, PRNG implementation, information storage, execution flows and loader's functionalities. The MIRAI botnet demonstrates a high level of automation and adaptability, employing scanning techniques and various attack vectors to compromise IoT devices. The PRNG implementation utilizes the Xorshift128 algorithm, optimized for resource-constrained IoT devices. The storage of crucial information within the bot is examined, highlighting the use of obfuscation techniques. The execution flows involve processes for network scanning, attack coordination and attempts to gain unauthorized access using default credentials. The loader component operates with a multi-threaded architecture, efficiently managing the infection process. Additionally, the document explores the loader's features, such as selecting appropriate executables based on hardware architectures and utilizing different file upload methods. These insights shed light on the complexity and versatility of the MIRAI botnet, emphasizing the need for robust security measures. Manufacturers and users are encouraged to prioritize strong passwords, regular firmware updates and network segmentation to mitigate the risks posed by this malicious botnet.

Introduction

The MIRAI botnet, discovered in 2016, exploited vulnerabilities in IoT devices to carry out large-scale distributed denial-of-service (DDoS) attacks. By targeting devices with weak passwords, MIRAI compromised and enlisted them into its botnet network. [1] Despite being a botnet of the past, MIRAI had a significant and lasting impact on the threat landscape, inspiring the emergence of similar botnets with comparable behavior patterns. [2] This section provides an introduction to key concepts related to botnets, including worms, DDoS attacks, the Internet of Things (IoT) and master-slave botnets. Understanding these concepts is crucial for comprehending the inner workings of the MIRAI botnet.

Worm

A computer worm is a self-replicating program designed to spread autonomously, often utilizing the Internet as a means of propagation. While worms themselves may not possess malicious intent, they are frequently accompanied by a harmful code segment known as a *payload*, which can cause damage to the victim's device. [3]

Distributed Denial-of-service (DDoS)

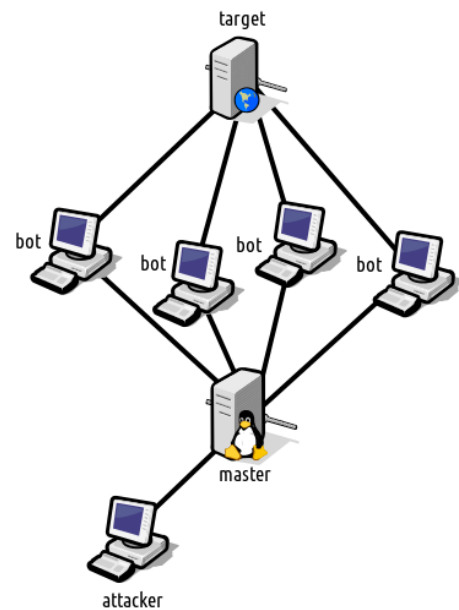
A Denial-of-service (DoS) attack occurs when an attacker targets a network service, node or resource with the intention of preventing authorized users from accessing it. When such attacks are orchestrated from multiple network nodes simultaneously, they are referred to as Distributed Denial-of-service (DDoS) attacks. [4]

Internet of Things (IoT)

The term Internet of Things (IoT) refers to a vast network of interconnected intelligent devices capable of exchanging information with other nodes on the network, leveraging the pervasive nature of the Internet. [5]

Master-slave botnet

In a master-slave botnet, as illustrated in the figure to the right, numerous nodes, referred to as *bots*¹ are connected to a central *master* node². The master node serves as a command center available to the *attacker* who can orchestrate and coordinate the actions to be performed by the connected bots against the *target*.



Overview of MIRAI

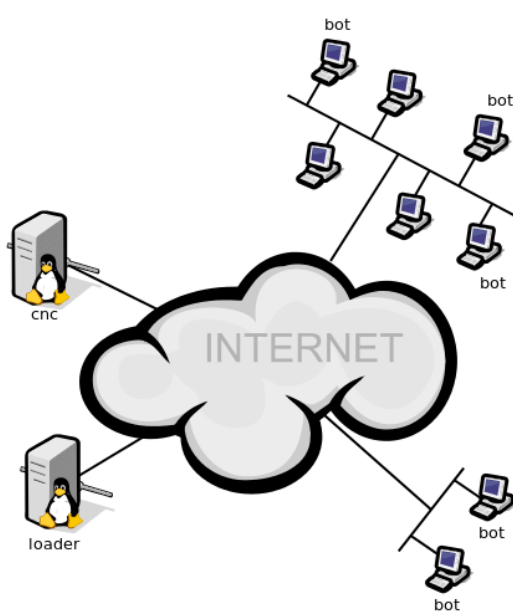
MIRAI is a prominent botnet that specifically targets IoT devices, taking advantage of their susceptibility to exploitation due to weak default credentials assigned by manufacturers. The combination of vulnerable devices and a lack of cybersecurity awareness among device owners creates an ideal environment for the proliferation of malware targeting embedded devices.

¹ or also slaves or zombies

² also known as CNC or C&C or C2

Initially discovered in September 2016, MIRAI gained notoriety for launching the first large-scale DDoS attack on the Internet. This attack generated an unprecedented volume of traffic, exceeding 1 Tbps, directed at a single victim. [2] In October 2016, the source code of MIRAI was released by its author(s), marking a significant turning point. [7] We conducted an analysis of the source code and developed monitoring tools based on sandboxes and honeypots to gain insights into this phenomenon. Despite the release of the MIRAI source code, the botnet and its variants continued to propagate and persist within the IoT ecosystem.

MIRAI operates on a master-slave architecture, with a Command and Control (CNC) server serving as the central coordinator for the bots. Although the source code of the CNC server, written in the Go programming language, was not included in our analysis, it plays a crucial role in the botnet's operation. MIRAI's CNC server employs the single fast-flux technique³, making it more challenging to detect and shut down.



The MIRAI botnet primarily comprises compromised IoT devices that establish connections with the Command and Control (CNC) server, awaiting further instructions. In addition, these infected devices actively conduct targeted scans within a subset of the entire IPv4 address space to identify and exploit new vulnerable devices.

Unlike traditional master-slave architectures, MIRAI incorporates a significant component known as the *loader*, which plays a pivotal role in efficiently infecting new devices. The loader receives information about potential targets either from the bots themselves or directly from the botnet's operator, [14] who provides contact information for specific devices. This loader component facilitates the seamless integration of new devices into the botnet's network, expanding its reach and potential impact.

MIRAI-like Botnets

Since the emergence of MIRAI, numerous botnets with MIRAI-like characteristics have appeared, employing similar attack strategies. The release of the MIRAI source code has provided aspiring threat actors with a blueprint to create and deploy their own botnets targeting vulnerable IoT devices. These devices, often lacking adequate security measures, become prime targets for various malicious activities.

³ the fast-flux technique essentially involves registering a single domain name that simultaneously points to multiple IP addresses. When connecting to the Command and Control (CNC) server, the bots query the DNS, which responds with a list of addresses. This list is dynamically ordered based on load balancing algorithms typically used by name resolvers (e.g., round-robin), ensuring that the order of addresses varies. The CNC nodes are programmed to periodically register/unregister their IP address associations with the domain, continuously updating the list of addresses held by the DNS server. [6]

In the following sections, we will explore the scanning processes employed by MIRAI, shedding light on the intricacies of its operations.

Scanning process

The scanning process is a crucial aspect of the MIRAI botnet's operation. Each bot in the botnet persistently conducts scans across a subset of the entire IPv4 address space, specifically targeting TCP ports such as port 23 (telnet) or ports 22 and 2222 (ssh). When a vulnerable device is detected, the bot transmits the contact information of the compromised device to the loader for further action.

In the next section, we will delve into the infection processes employed by MIRAI, providing a deeper understanding of how the botnet compromises and controls vulnerable IoT devices.

Infection process

The infection process involves the loader targeting a specific device for infection. Let's examine the steps of the infection process in detail:

1. the loader, armed with the contact information, initiates a connection attempt to the target device;
2. upon successfully establishing a connection, the loader verifies if the device is vulnerable to infection by searching for directories with write/execute permissions and other indicative factors;
3. once the device's vulnerability is confirmed, the loader proceeds to download and execute the MIRAI executable, effectively transforming the device into a bot;
4. the newly transformed bot takes its first action by establishing a connection to the Command and Control (CNC) server within the network. Simultaneously, it begins scanning the internet in search of potential victims, as described in the scanning process section.

In the next section, we will discuss the key features of the bot that have been deduced by analyzing its source code.

Bot's key features

Let's delve into some key features of the MIRAI bot code that contribute to its functionality and evasiveness. Upon execution, the bot immediately takes action by deleting itself from the file system. However, the process continues to run, ensuring that the infection persists until the device is rebooted. To prevent automatic device restart, the MIRAI bot disables the embedded device's watchdog timer⁴. These initial actions highlight the bot's proactive measures on IoT devices.

⁴ the watchdog is a timer that periodically triggers an integrity check routine of the system. If the routine detects a fault, it restarts the system to restore the initial conditions and bypass the anomaly. [8]

To protect vital operational information, such as the addresses of the Command and Control (CNC) server and the loader, the MIRAI bot employs cryptographic techniques for obfuscation. This ensures that the stored information remains unintelligible to software analysts during static analysis. By encrypting critical data within the bot itself, MIRAI enhances its resistance to detection and analysis, making it more challenging for security professionals to uncover its network infrastructure and control mechanisms.

Understanding these key features provides valuable insights into the design and capabilities of the MIRAI bot. Its ability to delete itself, disable device restart and obfuscate crucial information demonstrates the bot's sophistication and intent to remain hidden and operational within compromised IoT devices.

In the following subsections, we will explore other notable features of the MIRAI bot binary, shedding light on its PRNG implementation, storing information within the binary and execution flows. These features further contribute to the bot's effectiveness and resilience in carrying out malicious activities.

PRNG implementation

The author(s) of MIRAI implemented a pseudo-random number generator (PRNG) within the bot, using the *Xorshift128 algorithm*. [9] This algorithm falls into the category of linear feedback shift register (LFSR) algorithms. As it can be noticed in the snippet below, the PRNG implementation in MIRAI consists of two main functions: `rand_init()` for initializing the seed values and `rand_next()` for generating the next random number.

```
uint32 x, y, z, w;

// seeds initialization
void rand_init() {
    x = now();
    y = getpid() ^ getppid();
    z = clock();
    w = z ^ y;
}

// PRNG
uint32 rand_next() {
    uint32 t = x;
    t = t ^ (t << 11);
    t = t ^ (t >> 8);
    x = y; y = z; z = w;
    w = w ^ (w >> 19);
    w = w ^ t;
    return w;
}
```

In the `rand_init()` function, the seed values `x`, `y`, `z` and `w` are initialized. The `x` value is set to the current time, `y` is calculated by performing a bitwise XOR operation between the process ID and the parent process ID, `z` is set to the current clock time and `w` is obtained by performing a bitwise XOR operation between `z` and `y`.

The `rand_next()` function generates the next random number based on the current state of `x`, `y`, `z` and `w`. It follows a series of XOR and shift operations to update the values of `x`, `y`, `z` and `w` and finally returns the updated value of `w` as the random number.

The Xorshift128 algorithm is known for its simplicity and efficiency, making it suitable for devices with limited computational power and resources. It has a period of 2^{128-1} , meaning it can generate a large number of unique random values before repeating and its main purpose, within the MIRAI bot, is the generation of random IPv4 addresses to be used as scanning targets.

Storing information within the binary

Let's explore how MIRAI stores information within the binary and the obfuscation techniques it employs. To store the information, MIRAI utilizes a structure called `table`, which is essentially a C array. Each element in this array holds a variable-length string of bytes. Now, let's shift our focus to how the stored information is accessed and obfuscated using the xor operator.

MIRAI's obfuscation function utilizes the xor operator, which has symmetric properties allowing it to be used for both obfuscation and deobfuscation. However, it appears that the author(s) made an error in implementing this function. The provided code snippet showcases the obfuscation/deobfuscation process, where a 32-bit key is applied to each byte of the information in the table. The key is divided into four components, `k0` through `k3` and each byte is xor-ed with these components in a repetitive manner.

```
...
// hard coded key
uint32 key = 0xdeadbeef;

// obfuscation/deobfuscation
// function
void toggle_obfuscation(i) {
    k0 = key[0] ... k3 = key[3]
    for (j=0;j<len(tab[i]);j++) {
        uint8& b = table[i][j];
        b = b ^ k0;
        b = b ^ k1;
        b = b ^ k2;
        b = b ^ k3;
    }
}
...
```

However, due to the associative property of the xor operator, it is clear that the effective key size reduces to only 8 bits.

This means that the entire obfuscation/deobfuscation process can be simplified to a single expression: `b ^ 0x225`.

Consequently, in this case, each byte in the information is xor-ed with the constant value `0x22`, weakening the strength of the obfuscation/deobfuscation.

Despite the reduced key size, this runtime obfuscation/deobfuscation technique still poses challenges for dynamic analysis using a debugger. It makes the analysis of the information and understanding of the bot's behavior more challenging. Additionally, MIRAI incorporates other well-known anti-debugging techniques, such as capturing SIGTRAP to terminate the executable during debugging. These measures are implemented to impede the efforts of analysts in comprehending the bot's inner workings.

In conclusion, by employing obfuscation and anti-debugging techniques, MIRAI aims to increase its resilience against analysis, making it more difficult for security professionals to dissect its code and identify its malicious activities.

⁵ $\{[(b \oplus k_0) \oplus k_1] \oplus k_2\} \oplus k_3 = b \oplus (k_0 \oplus k_1 \oplus k_2 \oplus k_3) = b \oplus (0xDE \oplus 0xAD \oplus 0xBE \oplus 0xEF) = b \oplus 0x22$

Execution flows

Let's delve into the execution flows within the MIRAI bot. The bot utilizes the PRNG to obfuscate its process name running on the infected device. The parent process generates two child execution flows:

- *scanning handler*: this flow is responsible for scanning the network and identifying new potential victims for infection. It utilizes the PRNG to generate IP addresses for potential connections. The generated IPv4 addresses undergo a validation process that considers private or reserved addresses, as well as broadcast and multicast addresses, as invalid. Additionally, specific addresses associated with certain companies are also deemed invalid. Once a valid IPv4 address is identified, the bot attempts to establish a connection to the target device.
- *processes killer*: this flow terminates concurrent or suspicious processes that may interfere with the bot's operations. This includes terminating other botnets or loggers running on the infected device.

The parsing of attack requests takes place within the parent process. Therefore, the parent process is responsible for managing and coordinating attacks. Attack requests have specific structures with mandatory fields, including the attack vector, targets and duration. MIRAI implements various attack vectors, such as TCP/UDP/HTTP floods and introduces a new type of DNS attack known as DNS Water Torture⁶.

During the scanning process, the MIRAI bot employs a dictionary-based attack strategy. It utilizes a predefined dictionary that includes 60 pairs of factory default credentials commonly used by manufacturers, which are known to be vulnerable. The bot systematically scans potential targets and attempts to gain unauthorized access by leveraging these default credentials. This approach is designed to exploit weaknesses in the security measures of IoT devices, allowing the bot to infect and propagate further.

An analysis of the execution flows reveals that MIRAI operates with a well-structured and organized approach. It dynamically scans networks to identify vulnerable devices, coordinates attacks through its parent process and utilizes a diverse range of attack vectors and credential-based exploits. This systematic methodology enables MIRAI to maximize its effectiveness in compromising and controlling IoT devices, highlighting the botnet's sophisticated strategies and techniques.

Loader's key features

The loader component of MIRAI demonstrates a robust and efficient approach through its multi-threaded architecture, enabling smooth handling of the infection process. The main thread acts as a listener, receiving contact information from victim devices transmitted by the bots. Upon receiving details about a new victim, the loader creates a dedicated worker thread responsible for managing the infection process specific to that device.

⁶ DNS Water Torture is an attack that forces recursion of DNS resolvers by creating requests for definitely nonexistent subdomains. If such requests come from many bots, they can overwhelm the DNS servers of the targeted domain. [10]

During the infection process, the loader utilizes a connection descriptor that holds essential information about the remote device. This information may include the presence of a download path or the hardware architecture of the device. Understanding the hardware architecture is crucial for the loader to select the appropriate executable file to be downloaded and executed on the victim device. MIRAI's loader is equipped with a diverse range of cross-compiled executables⁷ tailored to prevalent hardware architectures in the IoT domain, which is more varied compared to traditional personal computers.

The loader faces the decision of selecting the method for uploading the binary file onto the victim device. It provides three options: WGET, TFTP or ECHO:

- the WGET method utilizes the HTTP protocol to retrieve the designated binary file from a remote repository; [12]
- similarly, the TFTP method employs the TFTP protocol for file retrieval; [13]
- on the other hand, the ECHO method takes a distinct approach where the loader serializes the content of the binary file and transmits it to the device through the established communication channel.

Once the binary file is successfully uploaded, it is executed on the victim device, establishing the presence of the bot. Subsequently, the connection between the loader and the infected device is terminated, allowing the bot to operate autonomously within the IoT network.

The loader's features showcase its ability to handle the infection process efficiently, adapt to diverse hardware architectures and employ multiple file retrieval methods. Understanding the loader's functionalities provides insights into the intricacies of MIRAI's deployment and underscores the sophistication of its infection mechanism in compromising IoT devices.

⁷ cross-compilation is the process by which it is possible to compile an executable for a specific hardware architecture from a different host machine with a different architecture. [11]

Conclusion

In conclusion, the analysis of the MIRAI bot code has provided valuable insights into its operation and key features. This botnet exhibits a range of malicious behaviors and techniques that enable it to compromise and control vulnerable IoT devices. From the scanning process that targets specific ports to the infection process that transforms devices into bots, MIRAI demonstrates a high level of automation and adaptability.

The bot's key features, such as the implementation of a PRNG for random number generation, the storage of critical information within the binary using obfuscation techniques and the execution flows that coordinate various processes, highlight the complexity and evasiveness of the bot. These features contribute to its ability to remain hidden, resist detection and operate autonomously within compromised IoT networks.

Moreover, the loader component plays a crucial role in the infection process, employing multi-threading to efficiently handle victim devices and selecting appropriate executables based on hardware architectures. The choice of file upload methods, including WGET, TFTP and ECHO, showcases the versatility of the loader in adapting to different network environments.

Overall, the analysis of the MIRAI bot code emphasizes the importance of robust security measures for IoT devices. It underscores the need for manufacturers and users to prioritize strong passwords, regular firmware updates and network segmentation to mitigate the risk of botnet infections. Additionally, it highlights the ongoing challenge faced by security professionals in detecting and combating evolving botnet threats.

References

- [1] Mirai (malware) - [https://en.wikipedia.org/wiki/Mirai_\(malware\)](https://en.wikipedia.org/wiki/Mirai_(malware))
- [2] Who Makes the IoT Things Under Attack? - <https://krebsonsecurity.com/2016/10/who-makes-the-iot-things-under-attack/>
- [3] Computer worm - https://en.wikipedia.org/wiki/Computer_worm
- [4] Denial-of-service attack - https://en.wikipedia.org/wiki/Denial-of-service_attack
- [5] Internet of things - https://en.wikipedia.org/wiki/Internet_of_things
- [6] Fast flux - https://en.wikipedia.org/wiki/Fast_flux
- [7] Mirai Source Code - <https://github.com/jgamblin/Mirai-Source-Code>
- [8] Watchdog timer - https://en.wikipedia.org/wiki/Watchdog_timer
- [9] Xorshift - <https://en.wikipedia.org/wiki/Xorshift>
- [10] Water Torture: A Slow Drip DNS DDoS Attack - <https://secure64.com/2014/02/25/water-torture-slow-drip-dns-ddos-attack/>
- [11] Cross compiler - https://en.wikipedia.org/wiki/Cross_compiler
- [12] Wget - GNU Project - Free Software Foundation - <https://www.gnu.org/software/wget/>
- [13] The TFTP protocol (rev. 2) - <https://datatracker.ietf.org/doc/html/rfc1350>
- [14] Botnet - <https://en.wikipedia.org/wiki/Botnet>