The Internet of Threats: Software-Level Lua Virtual Machine Sandbox, and Hardware-Level Closed System Hardware Vulnerabilities, Exploits, and Countermeasures, Targeting Lua-Based Internet of Things and Embedded Devices

Adrian Diepeveen[1][0009-0007-6105-2284]

[1] University of Johannesburg, Auckland Park, Johannesburg, 2092, South Africa
adriandiepeveen@outlook.com

Abstract. A comprehensive review is presented on the software-level Lua Virtual Machine (VM), and hardware-level closed system hardware vulnerabilities, exploits, and corresponding countermeasures, affecting Lua-based Internet of Things (IoT) and embedded devices.

Keywords: Lua, IoT, Embedded Devices, Lua VM, Sandbox, Cybersecurity.

1 Introduction

The exponential growth of the internet has accelerated a rise in the number of IoT and embedded devices, which are predicted to reach 500 billion by the year 2030 [1]. As a result of this rapid increase in connectivity, the attack surface for cyber threats has expanded, presenting more opportunities for attackers to exploit critical vulnerabilities. Lua is one of the most popular programming languages utilised in IoT and embedded devices, particularly within the Lua VM sandbox, due to its lightweight nature [2]. As illustrated in Figure 1, in Lua-based IoT and embedded devices, the lightweight nature of Lua has amplified the number of critical software-level and hardware-level vulnerabilities exploited by attackers, in order to compromise the Lua VM sandbox and closed system hardware respectively. Consequently, in response to these vulnerabilities and exploits, countermeasures have been implemented to secure the Lua VM sandbox and closed system hardware of these devices, which is discussed in Section 2 and Section 3 respectively. This research article conducts a thorough analysis of each vulnerability, in conjunction with its associated exploits and countermeasures, which are highlighted in Table 1 below. Therefore, the objective of this research article is to comprehensively investigate the software-level Lua VM sandbox, and hardware-level closed system hardware vulnerabilities, exploits, and associated countermeasures, specifically targeting Lua-based IoT and embedded devices, as well as the resultant effects on the security of these devices, and the privacy of their end-users.

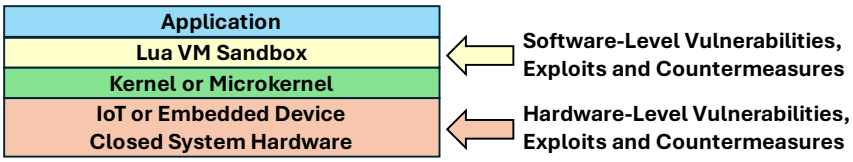

Fig. 1. System Architecture Diagram of Lua-Based IoT and Embedded Devices, Highlighting the Lua VM Sandbox and Closed System Hardware Entry Points for Attackers

Table 1. Summary of Vulnerabilities, Exploits and Associated Countermeasures

Entry Point	Vulnerabilities	Exploits	Notable Countermeasures
Lua VM Sandbox	Buffer Overflow	Denial of Service	Address Space Layout Randomisation
Lua VM Sandbox	Redis	Remote Code Execution	Malware Detection Machine Learning Model
Lua VM Sandbox	Lua VM Scripts	LuaBot Botnet	Long Short-Term Memory Model
System Hardware	ESP8266 Bus	Bus Sniffing	Encryption and Cryptographic Validation
System Hardware	Device Antenna	Radio Frequency	Frequency Hopping Spread Spectrum
System Hardware	ESP32 Chip	Fault Injection	Voltage Sensor

2 Software-Level Lua VM Vulnerabilities, Exploits, and Countermeasures, Targeting Lua-Based IoT and Embedded Devices

Lua-based IoT and embedded devices are sandboxed within the Lua VM, which possesses the purpose of protecting these devices and the end-users themselves, from compromised applications. Attackers can exploit these vulnerabilities by compromising the Lua VM in order to bypass the limitations imposed by this sandbox, thereby gaining access to other programs, sensitive data, and control over the IoT or embedded device.

2.1 Buffer Overflow within the Lua VM (CVE-2014-5461)

Vulnerabilities. The Lua VM sandbox serves as an execution environment for Lua scripts, which are utilised in IoT and embedded devices, due to Lua's lightweight nature. However, this lightweight nature of Lua, has resulted in numerous vulnerabilities. In 2014, a critical vulnerability, namely vulnerability CVE-2014-5461 in common vulnerabilities and exposures (CVE), was a buffer overflow within the Lua VM sandbox which existed in Lua versions 5.1 through 5.2.2. This critical vulnerability presented an opportunity for attackers to manipulate memory and remotely execute arbitrary code on IoT and embedded devices [3]. Therefore, due to Lua VM's improper handling of memory, this buffer overflow vulnerability, presented attackers with the opportunity to execute malicious code [4].

Exploits. Therefore, as a result of the aforementioned vulnerabilities, the Lua VM has encountered numerous exploits which involved breaking out of the sandbox. In 2014, a buffer overflow exploit allowed attackers to cause a denial of service (DoS) attack

[5], thereby allowing the execution of arbitrary code on IoT and embedded devices running Lua scripts, therefore exploiting memory allocation flaws. This exploit allowed attackers to gain unauthorised access to other programs, data and system resources on the IoT or embedded device, which infringed on the privacy and personal data of the end-users of these devices, ultimately leading to full system compromises outside of the Lua VM sandbox [6].

Countermeasures. In order to mitigate and prevent the buffer overflow vulnerabilities and associated exploits from occurring again in the future, a wide variety of countermeasures have been implemented by Lua developers. The first approach employed was enhancing the Lua VM's memory management mechanisms in order to prevent these buffer overflows. The vulnerability was fixed in the source code of Lua version 5.2.3 [7] by modifying the *vararg* functions in *ldo.c*, in order to prevent buffer overflows. Additionally, this updated version of Lua introduced additional checks to ensure that the number of arguments did not exceed expected limits [4, 8]. Furthermore, Address Space Layout Randomisation (ASLR) was adopted by the Lua VM sandbox to reduce the risk of memory corruption exploits, such as buffer overflows. ASLR is a memory-protection process which mitigates the threats of buffer overflow attacks by randomising the location where system executables are loaded into memory [9]. Ultimately, these aforementioned countermeasures have ensured that Lua VM sandbox does not allow access to other programs, sensitive data, or control over the IoT or embedded devices.

2.2 Redis Embedded Scripting Engine and Remote Code Execution (CVE-2022-0543)

Vulnerabilities. The Lua VM sandbox is a core component of IoT and embedded devices, because it is responsible for executing Lua scripts on these devices. Vulnerabilities in the Lua VM can lead to sandbox escapes, ultimately allowing attackers to gain unauthorised access to other programs, sensitive data and system resources. Redis is an open source in-memory data structure store utilised by millions of developers as a database, cache, message broker and streaming engine [10]. In order to further optimise these functions, Redis provides an embedded scripting engine which is based on the Lua programming language [11]. In 2022, a vulnerability in the Redis data structure was identified, namely CVE-2022-0543 [12], which granted attackers an opportunity to escape the Lua VM sandbox, and therefore execute arbitrary remote commands on the underlying system of the IoT or embedded device [13]. The aforementioned vulnerability was granted a critical severity score (CVSS) of 10.0, which is the highest possible severity score, emphasising the danger of this threat to the IoT and embedded devices, as well as the privacy of its end-users [12].

Exploits. Therefore, upon identification of this vulnerability, attackers exploited this threat, by executing the package *loadlib()*, in order to dynamically load modules from *liblua*. Furthermore, attackers gained access to a restricted function and loaded the *luaopen_io* function from a runtime library, enabling malicious code to be executed on

the host IoT or embedded device [10]. Ultimately, the Lua VM sandbox was bypassed successfully, granting attackers access to other programs, sensitive data, and control over the IoT or embedded devices.

Countermeasures. In order to mitigate the impact of the aforementioned vulnerabilities and exploits, Redis and Lua developers subsequently implemented a software patch. Additionally, the Lua VM sandbox design was hardened and redesigned in a more secure manner, in order to minimise the attack surface and prevent unauthorised access to system resources, through the implementation of stricter validation checks. Furthermore, developers have utilised active countermeasures to identify similar vulnerabilities and exploits, by leveraging the exponential advancements in AI research. Moreover, a malware detection machine learning (ML) model was also utilised to actively identify anomalies in the embedded Reddis scripting engine and Lua VM sandbox, by continuously monitoring system behaviour and analysing logs for anomalous activities, ensuring that the security of the Lua VM sandbox was maintained [10].

2.3 LuaBot Botnet

Vulnerabilities. Due to IoT and embedded device's limited computational resources, and networking capabilities, Lua VM sandboxes are increasingly becoming more popular targets among attackers. In 2016, a vulnerability for Lua-based IoT and embedded devices was initially discovered and analysed by researchers named MalwareMustDie and Symantec [14-15]. The aforementioned researchers identified critical vulnerabilities in web-cameras, closed-circuit television (CCTV) cameras, cable modems, and video surveillance systems. After thorough analysis of the Lua scripts executed by the Lua VM sandboxes on these IoT and embedded devices, it was evident that a vulnerability existed, whereby these devices could potentially be infected with malware, adding them to a botnet, and be utilised to perform large-scale attacks [16-17].

Exploits. Discovered in 2016, an exploit called LuaBot, was the first and most destructive IoT botnet implemented in the Lua programming language [18-19]. The malware of this botnet infected a wide variety of devices, namely D-Link routers, Hikvision CCTV Cameras, smart plugs, and smart light bulbs. Additionally, embedded systems, such as devices running on embedded Linux and industrial IoT devices, were infected by this malware, emphasising that these threats extended beyond the household environment and into industry as a whole [20]. LuaBot employed a centralised architecture, infected Advanced RISC Machine (ARM)-based IoT devices, and utilised a lightweight implementation of the Matrix Secure Sockets Layer (SSL) protocol for encrypted Command and Control (C2) communication. This malware contained a Lua version 5.3 interpreter along with approximately 24 Lua scripts, which opened a backdoor [21].

Furthermore, LuaBot possessed a unique ability to bypass Distributed Denial of Service (DDoS) security mechanisms utilising an advanced DDoS attack technique, thereby allowing it to be leveraged for executing DDoS attacks and providing proxy services [22]. Additionally, the attackers installed ransomware programs, further

compromising the personal information of the end-users of these devices [23]. The inherent dangers of LuaBot were amplified, by the ability of infected devices continuing to perform activities specified by the manufacturer, leaving end-users unaware of their device's participation in the botnet, and performance of malicious activities. Ultimately, this botnet exploited vulnerabilities within the Lua VM of these devices, which allowed it to bypass the restrictions imposed by the sandbox, allowing the malware to gain access to other programs, sensitive data, and control over the IoT or embedded device.

Countermeasures. Consequently, various countermeasures were implemented in order to prevent similar botnet vulnerabilities and exploits from occurring again in the future. Firstly, anti-virus and anti-spyware software was installed. This software was regularly updated, therefore ensuring that IoT and embedded device systems were secure against the latest known vulnerabilities, which could be exploited by LuaBot [24]. Furthermore, malware detection tools, such as Virus Total, were utilised to search for LuaBot signatures and behaviours, to detect and remove the malware from infected systems [14]. Additionally, firewalls were also configured in order to restrict unauthorised access to systems and filter out traffic associated with known LuaBot command and control servers [14, 25]. Moreover, network segmentation was utilised to isolate critical systems from less secure networks, ultimately limiting the spread of LuaBot infections [26]. Finally, the exponential advancements in AI research were leveraged through the utilisation of Long Short-Term Memory (LSTM), which is a type of Recurrent Neural Network (RNN) with an ability to remember long-time dependencies. Therefore, Haddad Pajouh also deployed LSTM for malware detection which analysed the behaviour of applications and network traffic, thereby identifying anomalies indicating LuaBot activity, further ensuring that the security of the Lua VM sandbox was successfully maintained [27-28].

3 Hardware-Level Closed System Hardware Vulnerabilities, Exploits, and Countermeasures, Targeting Lua-Based IoT and Embedded Devices

Lua based IoT and embedded devices are built using a wide variety of hardware components, such as ESP8266 and ESP32 microcontrollers [29], each with its own set of potential vulnerabilities. Attackers can exploit these vulnerabilities by physically compromising the device in order to bypass the closed system hardware, thereby gaining access to sensitive data, and the ability to control the device's system directly.

3.1 Inter-Integrated Circuit Bus Sniffing Attacks on the ESP8266 Microcontroller

Vulnerabilities. In Lua based IoT and embedded devices, the ESP8266 microcontroller communicates with peripheral components, such as sensors, Electrically Erasable Programmable Read-Only Memory (EEPROM) or other microcontrollers, over an Inter-

Integrated Circuit (I2C) bus [30]. András Tevesz identified a vulnerability where an attacker, with physical access to the IoT device, could infiltrate the I2C lines, Serial Data Line (SDA), and Serial Clock Line (SCL) in order to monitor and inject malicious data, consequently bypassing the closed system hardware's control over these communications [31]. An additional vulnerability was identified, which presented the attacker with an opportunity of sniffing the communication between I2C device sensors, controlling devices, memory chips, analogue-to-digital (ADC) converters, digital-to-analogue (DAC) converters, and the controllers or processors. This vulnerability presented the threat of compromising sensitive data from the I2C EEPROM, such as keys and logs [32].

Exploits. Therefore, as a result of the I2C being the most utilised communication protocol in IoT devices, the I2C bus is a valuable source of sensitive data for attackers. Firstly, clock glitching attacks, whereby the frequency of the I2C clock signal is modified, were performed on ESP8266 microcontrollers. Furthermore, attackers exploited this microcontroller through recon sniffing I2C communication, which extracted data from the I2C memory chip, and randomly modified the payload in order to intentionally generate errors in the system [32]. Therefore, by compromising the closed system hardware, attackers successfully bypassed the sandbox, resulting in DoS attacks [33]. The I2C bus was also targeted by attacks that aim to recover data and official firmware stored in the EEPROM, thereby allowing the attacker to install unofficial firmware [32, 34]. Moreover, an additional exploit involved a hardware trojan attack based on controlling both the SCL and SDA lines, allowing the attacker to read or write the memory of the microcontroller, by simply probing the lines through pull-up resistors.

Additionally, a Heartbleed attack was also performed by reading more information from the memory than requested by the master, through controlling the SDA and SCL lines. Finally, a buffer overflow attack was executed by writing additional data into the memory in order to replace security information, such as encryption keys and counters. The aforementioned attacks allowed the attacker to escape the closed system hardware and extract sensitive information stored in the secure EEPROM memory, such as the memory addresses, memory formats, bytes written, and pages written [35].

Countermeasures. Therefore, a wide variety of countermeasures were implemented in order to mitigate the effects of the aforementioned vulnerabilities and exploits. I2C encryption of data in memory and cryptographic validation between memory chips, prevented the leakage of sensitive data, and secured the data transmitted between the ESP8266 microcontroller and peripherals. Additionally, cryptographic co-processors were utilised, such as the AT88SC0104CA, which provide authentication and encryption features, therefore allowing secure read or write access [32]. Furthermore, tamper resistant enclosures or printed circuit boards (PCBs) with protective layers, restricted physical access to the I2C bus, making it more difficult for attackers to physically infiltrate the I2C bus.

Moreover, secure I2C modules were utilised which handled encryption and authentication internally, provided an added layer of security without the need for software modifications in the ESP8266 [36]. Finally, ML solutions which utilised anomaly detection algorithms to monitor I2C traffic in real time, were implemented. In the case of an attack, the amount of traffic exchanged on the communication buses significantly increased in comparison to normal behaviour, and therefore the anomalies were effectively identified and mitigated accordingly, ultimately ensuring that the security of the closed system hardware was maintained [37].

3.2 Radio Frequency Attacks on the Sonoff SNZB-02 Temperature and Humidity Sensor

Vulnerabilities. The Sonoff SNZB-02 is a small temperature and humidity sensor, which is housed in a small plastic casing. Therefore, this IoT device has minimal protection from Radio Frequency (RF) interference, leaving it vulnerable to attackers. A beaconing packet is sent out by this device, which alerts any receiving device of the current temperature and humidity in the area [38]. However, this small device possessed limited space for hardware, resulting in a lack of sufficient filtering and protection from outside interference. Therefore, due to this limitation, a vulnerability was identified, whereby an attacker could bypass the closed system hardware by physically obstructing the IoT device's antenna, leading to interference and control over the device's normal operations [39].

Exploits. Consequently, in order to exploit the aforementioned vulnerabilities of the Sonoff IoT device, attackers utilised a type of RF attack called Jamming. This specific attack was executed by successfully jamming beaconing packets on Channel 11, at the frequency of 2.405 gigahertz (GHz), utilising SDRangel which is an open-source Software Defined Radio (SDR) application [40]. Consequently, the device was unresponsive, leaving it unable to receive a signal for three minutes. Additionally, this Sonoff IoT device was also attacked by another type of RF attack called Replay Attacks, where a Universal Radio Hacker was utilised to capture the signal of the device and replay the packet back to the device when it was listening to Channel 11. The aforementioned vulnerabilities were compromised by these exploits as a result of this Sonoff IoT device lacking sufficient dynamic encryption, authentication mechanisms, and robust security features such as frequency hopping [39].

Countermeasures. Therefore, defending Sonoff IoT devices against these RF attacks, is of crucial importance in the exponentially evolving IoT environment. Dynamic encryption played a pivotal role in safeguarding data integrity and confidentiality within these IoT devices. Among various encryption methods, the Advanced Encryption Standard (AES), particularly AES-256, was implemented due to its proven robustness and efficiency [41]. As a symmetric key algorithm, AES-256 encrypts data in 128-bit blocks, therefore providing a high level of security for IoT applications contained in the sandbox, against current cryptographic threats [42]. Furthermore, Scrambling mechanisms were utilised, which effectively obfuscate the content of RF transmissions. The

signal is transposed to a different range and can only be decoded by a device utilising the same frequency offset and settings, thereby preventing incoming RF attacks. Automated AI tools, such as Deinvert developed by Oona Raisanen, further streamlined the process of identifying attacks, by autonomously monitoring the frequencies [43].

Finally, Frequency Hopping Spread Spectrum (FHSS) mechanisms were utilised as a highly effective method for mitigating RF interference and securing wireless transmissions from eavesdropping, as it is a resilient countermeasure with the ability to adapt to different IoT environments. FHSS involves modifying the official firmware in order to support frequency hopping protocols. Thereafter the carrier frequency is rapidly switched among a wide range of frequency channels, utilising a pseudorandom sequence known to both the transmitter and receiver. This technique significantly enhances resistance to RF interference, thereby minimising the risk of signal interception and unauthorised access, ultimately ensuring that the closed system hardware is secure [44].

3.3 Physical Fault Injection Attacks on the Lua ESP32 Microcontroller (CVE-2019-17391)

Vulnerabilities. Among the various hardware security attack techniques, Fault Injection (FI) attacks commonly target IoT and embedded devices [45]. FI attacks deliberately manipulate the normal execution of instructions and alter the data stored in registers within the processor, therefore forcing the hardware to induce an error in the software. Voltage glitching is a specific FI technique employed by attackers, whereby the hardware of the specific device is manipulated in order to rapidly manipulate the voltage of its components. Therefore, this attack aims to disrupt the device's normal operation, execute malicious code, attack the storage interface, and gain unauthorised access, thereby allowing the attacker to circumvent the authentication process, and ultimately bypass the closed system hardware [46]. In 2019, a vulnerability, namely CVE-2019-17391, was identified in the Espressif ESP32's mask Read-Only Memory (ROM) code version 0 through version 2. Lack of anti-glitch mitigations in the first stage bootloader of the ESP32 chip, allowed an attacker to read the contents of read-protected electronic fuses (eFuses), such as flash encryption and secure boot keys, by physically injecting a glitch into the power supply of the ESP32 chip shortly after reset [47].

Exploits. Later in 2019, a researcher named LimitedResults [48], reported an FI exploit utilising voltage glitching, where bits are stored in eFuses within the ESP32 microcontroller. These eFuses are utilised to store critical security configurations, including read-protection bits and cryptographic keys. This attack exploits a vulnerability during the transfer of these bits to shadow registers, allowing the corruption of read-protection bits. As a result, the cryptographic keys that should remain protected were exposed and read out by attackers. Once the read-protection restrictions of the closed system hardware were bypassed, the attacker extracted keys and sensitive data, ultimately executing malicious code outside the closed system hardware environment and gaining access to the entire device's memory. Therefore, as a result of bypassing the closed system

hardware, the Lua ESP32 microcontroller was further exploited by running unauthorised firmware, and consequently installing malware. A voltage glitching attack was also performed on the ESP32 ROM with full security settings enabled, triggering a full readout of the security keys [46, 49]. Finally, Colin O'Flyn documented an additional exploit, whereby a paper clip was utilised to perform a voltage glitch on the device's EEPROM in order to deliberately interrupt the communication between the memory module and the processor. Thereafter, the closed system hardware was bypassed, allowing unauthorised interaction with the locked bootloader shell [50].

Countermeasures. Consequently, responding to the above FI attacks, in 2020, Espressif hardened the security design of the ESP32 and released ESP32 Chip Revision version 3 [51]. Compared to the previous version, four significant changes were made. Firstly, the secure boot transitioned from symmetric-key cryptography to public-key cryptography. Secondly, the ROM code released by Espressif [52], illustrated redundancy where the eFuse bits were read out multiple times, which is a common FI countermeasure [53-55]. Thirdly, the Universal Asynchronous Receiver or Transmitter (UART) bootloader was disabled by utilising a dedicated eFuse bit. Finally, flash encryption was enabled [50]. Furthermore, a voltage sensor was installed on the ESP32 microcontroller, which monitors the clock signals and resets the board if the provided voltage falls below a specific threshold, which is 2.7 volts by default [56]. Additionally, in order to more effectively identify these attacks, power or clock lines were distributed across the PCB, forcing any attack to propagate throughout the entire network of lines [56]. Furthermore, the exponential advancements in AI research were leveraged, by utilising a voltage glitch attack detection system based on an advanced semi-supervised ML methodology, which utilised a hybrid combination of ML algorithms, in order to successfully identify voltage glitch attack anomalies, further ensuring that the security of the closed system hardware was successfully maintained [57].

4 Conclusion

Therefore, as clearly portrayed by the aforementioned analysis, this research article has reviewed the critical vulnerabilities and exploits affecting Lua-based IoT and embedded devices, specifically at the software-level Lua VM sandbox and hardware-level closed system hardware. Moreover, through this detailed analysis, it is evident that the relevant countermeasures, as emphasised in Table 1, offer effective solutions for safeguarding Lua-based IoT and embedded devices against the ever evolving cybersecurity vulnerabilities and exploits. Therefore, the objective of this research article, to comprehensively investigate the software-level Lua VM sandbox, and hardware-level closed system hardware vulnerabilities, exploits, and associated countermeasures, specifically targeting Lua-based IoT and embedded devices, has been successfully achieved.

Disclosure of Interests. Adrian Diepeveen has no competing interests to declare that are relevant to the content of this research article.

References

1. Zikria, Y.B., Ali, R., Afzal, M.K., Kim, S.W.: Next-generation internet of things (IoT): Opportunities, challenges, and solutions. Sensors 21, 1174 (2021)
2. Costin, A.: Lua code: security overview and practical approaches to static analysis. In: 2017 IEEE Security and Privacy Workshops (SPW), pp. 132-142. IEEE (2017)
3. CVE-2014-5461 Detail, https://nvd.nist.gov/vuln/detail/CVE-2014-5461, last accessed 2024/08/30
4. CVE-2014-5461, https://ubuntu.com/security/CVE-2014-5461, last accessed 2024/08/30
5. Lua, https://github.com/lua/lua, last accessed 2024/08/30
6. Woo, S., Hong, H., Choi, E., Lee, H.: A Precise Approach for Modified Vulnerable Code Clone Discovery from Modified Open-Source Software Components. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 3037-3053 (2022)
7. Buffer overflow in the vararg functions in ldo.c in Lua 5, https://github.com/advisories/GHSA-v3hh-4h88-w4mr, last accessed 2024/08/31
8. Lua 5.2 Reference Manual, https://www.lua.org/manual/5.2/manual.html#5.2.3, last accessed 2024/08/31
9. Address space layout randomization (ASLR), https://www.techtarget.com/searchsecurity/definition/address-space-layout-randomization-ASLR, last accessed 2024/09/01
10. Mitchell, B.S., Chandnani, A., Carter, J., Roumelioti, D., Mancoridis, S.: Malware Detection in Cloud Native Environments. Journal of Engineering and Technology Research 6 (2024)
11. What is Lua, https://www.lua.org/about.html, last accessed 2024/09/01
12. CVE-2022-0543 Detail, https://nvd.nist.gov/vuln/detail/CVE-2022-0543, last accessed 2024/09/01
13. Shrestha, M., Kim, Y., Oh, J., Rhee, J., Choe, Y.R., Zuo, F., Park, M., Qian, G.: Provsec: Cybersecurity system provenance analysis benchmark dataset. In: 2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), pp. 352-357. IEEE (2023)
14. MMD-0057-2016 - Linux/LuaBot - IoT Botnet as a Service, http://blog.malwaremustdie.org/2016/09/mmd-0057-2016-new-elf-botnet-linuxluabot.html, last accessed 2024/09/02
15. Linux.LuaBot, https://www.broadcom.com/support/security-center/vulnerability-management?docid=2016-090915-3236-99&tabid=2, last accessed 2024/09/03
16. McNulty, L., Vassilakis, V.G.: IoT botnets: Characteristics, exploits, attack capabilities, and targets. In: 2022 13th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), pp. 350-355. IEEE (2022)
17. Prokofiev, A.O., Smirnova, Y.S., Surov, V.A.: A method to detect Internet of Things botnets. In: 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp. 105-108. IEEE (2018)
18. Victor, P., Lashkari, A.H., Lu, R., Sasi, T., Xiong, P., Iqbal, S.: IoT malware: An attribute-based taxonomy, detection mechanisms and challenges. Peer-to-peer Networking and Applications 16, 1380-1431 (2023)

19. LuaBot is the first Linux DDoS botnet written in Lua Language, https://securityaffairs.com/51155/malware/linux-luabot.html, last accessed 2024/09/03
20. The Next Evolution of IoT Botnets, https://www.fortinet.com/blog/threat-research/reaper-the-next-evolution-of-iot-botnets, last accessed 2024/09/04
21. Embedded Device and Webapp Hacking, https://w00tsec.blogspot.com/2016/09/luabot-malware-targeting-cable-modems.html, last accessed 2024/09/05
22. De Donno, M., Dragoni, N., Giaretta, A., Spognardi, A.: DDoS-capable IoT malwares: comparative analysis and Mirai investigation. Security and Communication Networks 2018, 7178164 (2018)
23. Hachem, N., Mustapha, Y.B., Granadillo, G.G., Debar, H.: Botnets: lifecycle and taxonomy. In: 2011 Conference on Network and Information Systems Security, pp. 1-8. IEEE (2011)
24. Feily, M., Shahrestani, A., Ramadass, S.: A survey of botnet and botnet detection. In: 2009 Third International Conference on Emerging Security Information, Systems and Technologies, pp. 268-273. IEEE (2009)
25. Yaacoub, J.-P.A., Noura, H.N., Salman, O., Chehab, A.: Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations. International Journal of Information Security 21, 115-158 (2021)
26. Ferronato, G.: IoT white worms: design and application. University of Twente (2020)
27. Malhotra, P., Singh, Y., Anand, P., Bangotra, D.K., Singh, P.K., Hong, W.-C.: Internet of Things: Evolution, concerns and security challenges. Sensors 21, 1809 (2021)
28. HaddadPajouh, H., Dehghantanha, A., Khayami, R., Choo, K.-K.R.: A deep recurrent neural network based approach for internet of things malware threat hunting. Future Generation Computer Systems 85, 88-96 (2018)
29. Litayem, N., Al-Sa'di, A.: Exploring the Programming Model, Security Vulnerabilities, and Usability of ESP8266 and ESP32 Platforms for IoT Development. In: 2023 IEEE 3rd International Conference on Computer Systems (ICCS), pp. 150-157. IEEE (2023)
30. Jacob, A., Zakaria, W.N.W., MRB, M.T.: Evaluation of I2C communication protocol in development of modular controller boards. ARPN Journal of Engineering and Applied Science 11, 4991-4996 (2016)
31. Hardware Hacking 101: E01 I2C Sniffing, How to Listen to Your Arduino's I2C Bus, https://cujo.com/blog/hardware-hacking-101-e01-i2c-sniffing/, last accessed 2024/09/06
32. IoT Security – Part 16 (101 – Hardware Attack Surface: I2C), https://payatu.com/masterclass/iot-security-part-16-101-hardware-attack-surface-i2c/, last accessed 2024/09/05
33. Gomez-Bravo, F., Jiménez Naharro, R., Medina García, J., Gómez Galán, J., Raya, M.: Hardware attacks on mobile robots: I2c clock attacking. In: Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 1, pp. 147-159. Springer (2015)
34. Gupta, A.: The IoT Hacker's Handbook. Springer (2019)
35. Khelif, M.A., Lorandel, J., Romain, O.: Non-invasive I2C hardware trojan attack vector. In: 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1-6. IEEE (2021)
36. I2C bus sniffing and spoofing, https://electronics.stackexchange.com/questions/400968/i2c-bus-sniffing-and-spoofing, last accessed 2024/09/06

37. Lorandel, J., Khelif, M.A., Romain, O.: A low-cost hardware attack detection solution for IoT devices. In: 2022 IEEE 31st International Symposium on Industrial Electronics (ISIE), pp. 674-679. IEEE (2022)
38. Temperature and Humidity Sensor SNZB-02, https://fccid.io/2APN5SNZB-02, last accessed 2024/09/07
39. Anthi, E., Williams, L., Ieropoulos, V., Spyridopoulos, T.: Investigating Radio Frequency Vulnerabilities in the Internet of Things (IoT). IoT 5, 356-380 (2024)
40. SDRangel, https://www.sdrangel.org/, last accessed 2024/09/07
41. Sarker, M.Z.H., Parvez, M.S.: A cost effective symmetric key cryptographic algorithm for small amount of data. In: 2005 Pakistan Section Multitopic Conference, pp. 1-6. IEEE (2005)
42. Nannipieri, P., Di Matteo, S., Baldanzi, L., Crocetti, L., Zulberti, L., Saponara, S., Fanucci, L.: VLSI design of Advanced-Features AES CryptoProcessor in the framework of the European Processor Initiative. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 30, 177-186 (2021)
43. A Voice Inversion Descrambler and Scrambler, https://github.com/windytan/deinvert, last accessed 2024/09/08
44. Torrieri, D.: Principles of spread-spectrum communication systems. Springer (2005)
45. Kazemi, Z.: Fault Injection Attacks on Embedded Applications: Characterization and Evaluation. Université Grenoble Alpes (2022)
46. Pearson, B.: Discovering Vulnerabilities and Designing Trustworthy Defenses in IoT Systems and Devices. University of Central Florida (2023)
47. CVE-2019-17391 Detail, https://nvd.nist.gov/vuln/detail/CVE-2019-17391, last accessed 2024/09/08
48. Fatal Fury on ESP32: Time to Release Hardware Exploits, https://www.blackhat.com/eu-19/briefings/schedule/index.html#fatal-fury-on-esp-time-to-release-hardware-exploits-17336, last accessed 2024/09/08
49. Pwn the ESP32 Forever: Flash Encryption and Sec. Boot Keys Extraction, https://limitedresults.com/2019/11/pwn-the-esp32-forever-flash-encryption-and-sec-boot-keys-extraction/, last accessed 2024/09/09
50. Delvaux, J., Mune, C., Romero, M., Timmers, N.: Breaking Espressif's ESP32 V3: Program Counter Control with Computed Values using Fault Injection. In: 18th USENIX WOOT Conference on Offensive Technologies (WOOT 24), pp. 229-243 (2024)
51. ESP32 Chip Revision v3.0 User Guide, https://www.espressif.com/sites/default/files/documentation/esp32_chip_revision_v3_0_user_guide_en.pdf, last accessed 2024/09/09
52. ESP ROM Elfs, https://github.com/espressif/esp-rom-elfs/releases, last accessed 2024/09/10
53. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE 94, 370-382 (2006)
54. Moro, N., Heydemann, K., Encrenaz, E., Robisson, B.: Formal verification of a software countermeasure against instruction skip attacks. Journal of Cryptographic Engineering 4, 145-156 (2014)
55. Witteman, M., Oostdijk, M.: Secure application programming in the presence of side channel attacks. In: RSA conference (2008)
56. Cirne, A., Sousa, P.R., Resende, J.S., Antunes, L.: Hardware security for Internet of Things identity assurance. IEEE Communications Surveys & Tutorials (2024)

57. Jiang, W.: Machine Learning Methods to Detect Voltage Glitch Attacks on IoT/IIoT Infrastructures. Computational Intelligence and Neuroscience 2022, 6044071 (2022)