Turbulence: Ransomware Proof of Concept for Resource-Constrained IoT Devices

Calvin Brierley[1][0000−0001−8766−822X], Yuxiang Huang[2][0009−0001−5122−7710], Yichao Wang[1][0000−0002−4633−3690], James Pope[3][0000−0003−2656−363X], George Oikonomou[2][0000−0002−1684−6989], and Budi Arief[1][0000−0002−1830−1587]

[1] School of Computing, University of Kent, Canterbury, UK
[2] School of Electrical, Electronic & Mechanical Engineering, University of Bristol, UK
[3] School of Engineering Mathematics & Technology, University of Bristol, UK

Abstract. The "Internet of Things" (IoT) is a term used to describe smart devices that are capable of connecting to a network. IoT devices can take many forms, such as cameras, televisions, or home assistants, and are often designed to perform specific tasks. While they only require limited processing power to achieve their intended purpose, their connected nature means they are still vulnerable to attack. Most IoT-based malware is designed to infect devices using General Purpose Operating Systems, such as Linux. Malware targeting "constrained" IoT devices, which have lower hardware specifications and implement bare-metal firmware or a Real Time Operating System, are significantly less common, as they present a number of challenges that can hinder malware development. In this work, we identify these challenges and assess the viability of implementing functional ransomware that targets constrained IoT devices. We then test our findings by developing a ransomware Proof of Concept capable of locking a target system and spreading throughout a network. Finally, we analyse the ransomware's performance against an intentionally vulnerable testbed to identify the requirements and limitations of – as well as potential countermeasures against – ransomware targeting constrained IoT devices.

Keywords: Ransomware · Internet of Things · Constrained Device · Proof of Concept · Malware

1 Introduction

The Internet of Things (IoT) has become popular in our daily lives, spanning a wide range of devices such as smart homes, wearable devices, and even industrial sensors. These devices employ various communication technologies to improve convenience across diverse contexts. For instance, smart thermostats in home settings can automatically adjust indoor temperatures based on user preferences and environmental conditions, thereby improving energy efficiency and comfort. Despite the widespread deployment of commercial IoT devices, the associated privacy and security issues have been widely discussed in academia [3,24,34],

and remain an urgent and pressing concern. Several sectors implementing industrial IoT devices have also been subjected to ransomware attacks, such as in healthcare [9], manufacturing [19], and smart grids [48]. These instances have highlighted the severe impact such attacks can inflict, including potential endangerment of human life [26].

IoT networks have the potential to become very profitable for malicious actors [7,23,44], especially given the recent surge in ransomware attacks [25]. Previous instances of IoT-based malware primarily targeted vulnerable Linux-based IoT devices, while constrained IoT devices, particularly those running Real Time Operating Systems (RTOSes), remain comparatively under-explored. Such devices are often deployed in critical applications such as industrial control systems, healthcare, and smart infrastructure. As the adoption of RTOS-based IoT devices continues to increase, it becomes important to understand their potential vulnerabilities, threat models, and address the unique security challenges presented by their limited computing resources.

In this paper, we explore the viability of functional ransomware for constrained IoT devices. In the context of this work, we define constrained IoT devices as low-spec hardware running RTOS-based or bare-metal firmware. We identify methods that could be used to successfully infect and ransom such devices, then attempt to implement them to create a Proof of Concept (PoC). We then assess the effectiveness of the PoC by performing a simulated attack against a constrained-IoT device in a semi-realistic environment.

For the purposes of this PoC, the device ran firmware we had developed to be "vulnerable-by-design". While this may seem like an idealised scenario, similar exploits have often been found in the wild [50,36,35], and are likely to continue being discovered. One particularly relevant example discovered during this work was a buffer overflow vulnerability present in Zephyr OS's Bluetooth stack, which when exploited, could lead to remote code execution [28]. Additionally, while progressively more IoT devices are implementing security features to impede attackers, such as secure boot and Data Execution Prevention (DEP), previous works indicate that many are still being left vulnerable [1,54,5].

Contributions. The contributions in this paper are as follows:

- Provides insight as to how ransomware can be developed and deployed on constrained IoT devices.
- Created a Proof of Concept ransomware that was successfully implemented on a constrained IoT device in a simulated attack.
- Identifies pitfalls attackers may encounter during the ransomware development process for IoT devices.
- Suggests potential countermeasures that can be implemented by developers and users to reduce the effectiveness of such attacks.

2 Related Work

IoT's popularity continues to grow, with devices being widely adopted in both household and industrial settings. The potential consequences of attacks against

IoT devices are far reaching, and can impact victims in a variety of ways. Research has shown that attackers can brick devices to render them unusable, or infect devices such as smart cameras to invade users' privacy [46]. Ronen et al. [45] described a novel attack in which IoT "smart lamps" are infected with a worm capable of wirelessly spreading to neighbours via the Zigbee protocol, which can then be used to perform Denial of Service (DoS) attacks, jam nearby wireless signals, or brick infected devices.

Existing malware targeting IoT devices running relatively robust operating systems (OS), such as embedded Linux, have also been subject to analysis by researchers [13,56]. One type of malware, named Mirai, was found to be exploiting security vulnerabilities present in various Linux-based IoT firmware, creating botnets containing hundreds of thousands of IoT devices [15,22]. Methods to gain persistence have also been identified, which would allow IoT malware to maintain long-term infections of affected IoT devices [11].

Ransomware, as a class of malware, has become a serious worldwide cyber threat in recent years. Traditional crypto-ransomware often aims to encrypt valuable files, then extort the device's owner in exchange for decryption or restoration [21,42]. In addition to the financial losses of paying the ransom, victims may face other negative effects, such as disruption of infrastructure and services [2], or the theft of data [52]. While ransomware is often used to target IT infrastructure or personal computers, ransomware targeting IoT devices using general purpose operating systems (GPOSs) have also been a subject of study. Brierley et al. [10] developed a PoC ransomware for Linux-based IoT devices, which was capable of disabling targeted devices and encrypting their bootloaders, rendering infected devices bricked should they be rebooted. Pen Test Partners [43] demonstrated how they were able to infect a smart thermostat running vulnerable firmware using Linux with ransomware. Once installed, the malware would use the device's screensaver feature to display a ransom note, then perform disruptive actions, such as turning the heating or air conditioning on or off. Similar attacks were also performed against other household devices, such as smart coffee machines [32] and smart televisions [16].

For constrained devices, however, hardware limitations may make the use of a GPOS infeasible. As an alternative, an RTOS can be used, as they are often designed with low memory usage and a compact code size in mind. If minimal software overhead is required, developers may instead opt for bare-metal firmware, which runs without the support of an operating system. Currently, there is limited research examining the feasibility and real-world impact of ransomware targeting constrained IoT devices [2]. As ransomware is restricted by the resources provided by the host, typical strategies used by ransomware targeting GPOS-based devices, such as encrypting valuable files or exfiltrating sensitive information, are unlikely to be as effective. Traditional security solutions often used in more powerful systems to defend against such attacks may also be inapplicable [12,53].

The work presented in this paper aims to overcome these limitations, and provide a concrete example of ransomware adapted for use on constrained IoT

platforms, emphasising the urgent need to re-evaluate traditional security assumptions in large-scale, low-footprint IoT ecosystems.

3 Methodology

In this section, we discuss the challenges constrained IoT devices pose to ransomware development, the methods we proposed to overcome these challenges, and finally, the design of our testbed, which is used to evaluate the viability of each method.

3.1 Challenges

Previous research has highlighted a number of challenges that can be encountered when developing ransomware for GPOS-based IoT devices, such as heavy limitations of storage space, Random Access Memory (RAM), and processing power [10]. For RTOS-based devices, these issues are likely exacerbated, as they are likely to utilise more restrictive low-spec hardware.

Additionally, adapting existing software to new devices can present other difficulties. For GPOS-based applications, this process can often be as simple as re-compiling the application with a cross-compiler that matches the target architecture. Malware of this type typically runs within the context of the target's OS, relying on the abstraction it provides to maintain the portability and compatibility of the ransomware when attempting to infect various devices. RTOS-based firmware, however, is likely to be designed for a specific purpose, and is likely to only include features necessary to complete the task intended by the firmware's developers. The availability of common features typically provided by GPOSs cannot be guaranteed, such as a filesystem, abstraction for all available hardware, or support for executable files.

These challenges can greatly increase the complexity of running or installing external software, such as ransomware, within the context of an existing RTOS-based firmware.

3.2 Ransomware Implementation

To address these challenges, we identified three possible approaches that could be used to develop ransomware suitable for use in a constrained environment. The first approach, "Living Off the Land" would attempt to limit memory usage by utilising existing code provided by the original firmware for complex operations. The second approach would involve creating a platform-independent "ransom library" simplifying the implementation and maintenance of complex functionality when adapting the ransomware to target new platforms. Finally, the last method would aim to *replace* the original firmware with a malicious ransom-firmware of the attackers own design. After evaluating the viability of each option, we found that the "firmware replacement" method would be the most likely to succeed. We cover this method in more detail here, while a detailed assessment of the

rejected approaches, and our attempts to implement them, is provided later in Section 5.3.

As mentioned previously in Section 3.1, IoT malware targeting GPOS-based systems typically takes the form of a malicious application or code block that runs alongside or "within" the context of the original firmware of the device. With this method, however, the attacker instead develops malicious "ransom-firmware", designed to *replace* the original firmware. As ransom-firmware can be developed using similar workflows to traditional firmware, it can utilise the same benefits, such as building upon a pre-existing RTOS. This would allow attackers to utilise any features and board support the chosen RTOS provides, greatly improving the ease of development and hardware compatibility of the ransomware. In addition, this method allows the attacker to configure which features are set to be included during the compilation stage, thereby drastically reducing reliance on the original firmware's functionality.

3.3 Testbed

In order to test the viability of the suggested approaches, a testbed was created to simulate a possible target. Here, we discuss the hardware and software components of the testbed.

Target Environment For the purpose of evaluating a PoC, the target network was expected to be installed within an enclosed environment, such as in an industrial setting. The attacker is assumed to have access to a device on the network, which could feasibly be achieved by an insider threat, an attacker with unauthorised physical access, or the use of remote connectivity devices such as a drone [45]. Previous attacks on IoT devices were able to run malware by exploiting vulnerable software that was exposed to the internet [33], or via "local" communication protocols such as Zigbee [45]. We assume that attackers would be able to exploit the target device in a similar fashion, such that they can run arbitrary code.

Hardware The nRF52840-DK [37] – an ARM based device created by Nordic Semiconductor – was chosen as the target hardware since it is a popular platform that supports Zephyr [63], the RTOS we wanted to exploit in this research. Further details of this device can be found in Appendix A.

Target Software For the purposes of testing a realistic exploitation scenario, a vulnerable application was developed for use on the target devices. The application itself is built upon the Zephyr RTOS, an open source project that boasts a number of useful features that may appeal to developers, such as: (i) a modular structure, allowing features to be enabled or disabled to preserve space [58]; (ii) support for hundreds of boards [57], allowing for easy adaption to new hardware; and (iii) support for various communication protocols, such as Bluetooth, Bluetooth Low Energy (BLE), and Wi-Fi [60,61].

By using an RTOS such as Zephyr, developers can abstract many lower level complexities that they may otherwise have to manage for themselves. Instead, common tasks such as writing to flash [62] are provided as a standard Application

Enabled	RESV	Temp Low	Temp High	Action ID	Arg 1	Arg 2	Arg 3
01	00	00 15	01 40	00 03	41 4c 52 4d	00 00 00 00	00 00 00 00
TRUE	Reserved	-21C	320C	ID: 3	ALRM	[Null]	[Null]

Fig. 1. Example of an Alert Structure

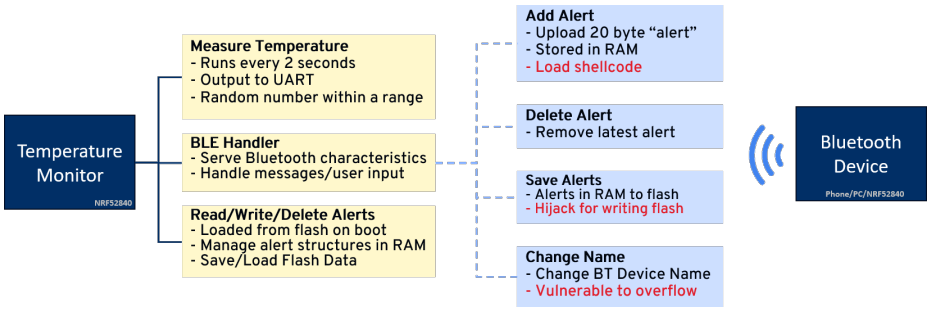

Fig. 2. Temperature Monitor Features

Programming Interface (API) to the developer, which will remain consistent regardless of the underlying hardware.

Software Features The firmware installed on the test devices acts as a temperature monitoring system, which measures the temperature of a connected device, and outputs it to the Universal Asynchronous Receiver/Transmitter (UART) port. Temperature alerts can be set, which the application would reference when a new reading is taken. Alerts can be stored in RAM or saved in flash, with each device able to support 20 active alerts at a time. Alerts are represented as a 20 byte structure, which is shown in further detail in Figure 1. It should be noted that for the purposes of this PoC, some features are simulated or unimplemented[1], as they would not impact the development or implementation of ransomware.

The temperature monitors are designed to be managed via BLE using a "central device", such as a phone or laptop. Users can access this interface to perform various actions, such as changing the device ID, adding temperature alerts, deleting temperature alerts, and saving current alerts to flash memory.

Software Vulnerabilities For the purposes of testing the viability of a ransomware attack, the device and its firmware are "vulnerable-by-design" with no additional hardening, such that an attacker is able to remotely exploit the device. As shown in Figure 2, the user is able to change the "device ID" via Bluetooth. The device ID is expected to be numeric, and only 2 bytes are allocated for storage, however, as the length of the user input is not checked, the user can provide up to 20 bytes as part of the function call. This can lead to a stack overflow, a common vulnerability that can allow an attacker to overwrite the return address, gain control of the program counter, and run arbitrary code [49,27].

[1] E.g. "Temperature readings" are simulated by generating a random number within a set range, alerts will not trigger during measurements or perform any actions.

It should be noted that while this work uses Bluetooth as the exploit vector, the methods described in this work could theoretically be achieved via other communication technologies, as long as the attacker can run arbitrary code.

4 PoC: Turbulence

As part of this work, we developed a PoC ransomware designed to target constrained embedded devices, which we named "Turbulence". In this section, we outline Turbulence's development process, followed by an explanation as to how it can be used to infect a vulnerable device.

4.1 Features

During the initial development, three features were chosen as the benchmark for ransomware to be considered "functional":

- **DoS:** Once the ransomware is activated, infected devices should be rendered incapable of performing their intended tasks. Once a sufficient payment is made to the attacker, the user should be able to recover the functionality of the device.
- **Persistent:** The ransomware should remain on the device in the event of device reboots or power loss, preventing easy removal.
- **Wormable:** Once a device is infected, the ransomware should attempt to spread to other nearby devices of a similar type. Ideally, this should be performed without any further interaction being required from the attacker.

4.2 Exploitation

In order to infect a device with ransomware, the attacker must be able to execute arbitrary code on the target. This could be achieved in a number of ways, such as using physical access to reprogram a target, obtaining credentials that allow them to force a malicious update, or finding a vulnerability that can be exploited to run custom code. In this case, we have simulated the latter scenario by developing a firmware with an intentional stack overflow vulnerability.

By sending a large ID to the "Change ID" Bluetooth service, an attacker can overwrite the return address of the function, and gain control of the program counter. In a typical stack overflow vulnerability, program execution would be redirected to attacker-controlled data within RAM, such as a variable that contains user input [8]. By compiling assembly code into "shellcode", and providing it as input to the application, the attacker can store machine instructions within these variables, allowing them to execute arbitrary code and perform malicious actions.

During early testing, we attempted exactly this approach. First, we created a shellcode that runs an infinite loop, then uploaded it by embedding it within an "alert" structure. The structure is then uploaded to the device via the BLE service (discussed in Section 3.3), which is then stored in the device's RAM.

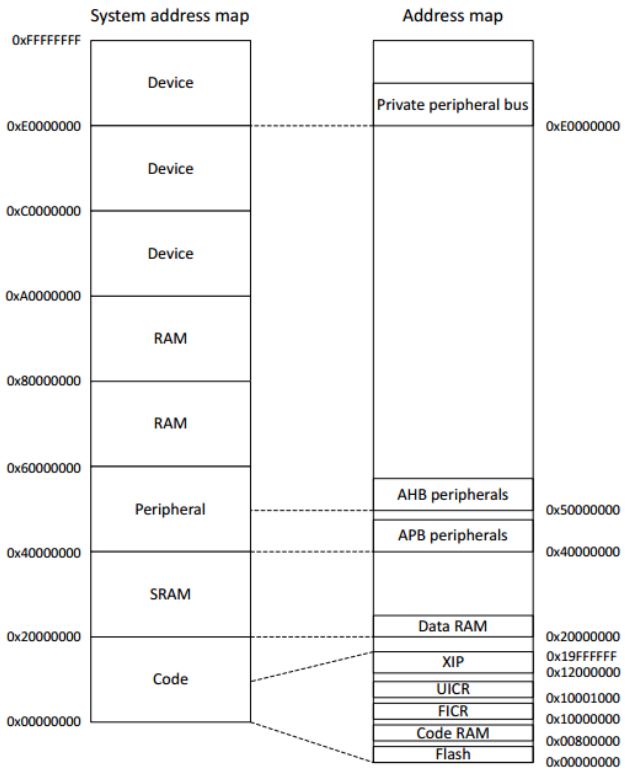

Fig. 3. Memory Map of the nRF52840

We then attempted to use the stack overflow exploit to redirect the program counter to the address of the newly uploaded shellcode. However, Zephyr implements DEP, which causes the system to crash when the program counter is set to point to data in RAM. Upon discovering this, we examined the documentation for the nRF52840, with the aim of bypassing DEP. We found that in the memory map of the device (shown as Figure 3), the RAM is mapped into memory *twice*, once as "Data RAM" (starting at `0x20000000`), and once as "Code RAM" (starting at `0x00800000`) [38]. For this device, both mappings reference the *same physical RAM*, and by referencing areas in these regions with the same offset (i.e. `0x20001234` and `0x00801234`), the same data was returned, indicating that changes the data RAM are reflected in the code RAM area. Therefore, we would be able to make changes to memory via data RAM, then execute the changes as code by accessing them via code RAM.

4.3 Implementation

Once we were able to execute arbitrary code, the next step was to develop ransomware capable of infecting the target device, while addressing the challenges

discussed in Section 3.1. To achieve this, each of the methods presented in Section 3.2 were tested as part of the PoC development stage.

The "firmware replacement" method was found to be the most suitable, as it allowed the attacker to gain complete control of the environment in which the ransomware is executed, and removed any dependence on the original firmware providing any required features. A summary of our attempts to implement the rejected methods can be found in Section 5.3.

Just like the original firmware, Turbulence utilised Zephyr OS, as it offered support for the target board and Bluetooth communication. The development process followed many of the steps described in the Zephyr documentation, similar to producing "normal" firmware.

For testing, the ransom-firmware could be installed directly to a nRF52840 development board, without requiring an attack to be performed against a real target. Once complete, Turbulence was compiled into a binary file, ready to be loaded as part of the exploitation stage.

4.4 Loading

In order to infect a targeted system with ransomware, we developed a loader capable of remotely installing Turbulence via Bluetooth. The installation was separated into three stages: (i) *Loading*, in which the ransomware is uploaded to temporary memory, then written to permanent memory; (ii) *Redirection*, where the existing firmware is modified to redirect execution towards the ransomware upon reboot; and (iii) *Reset*, which forces the target to reboot, thereby activating the ransomware.

Stage 1: Loading Malware The first stage aims to load the ransomware into the device's RAM. As mentioned previously in Section 4.2, the target reserves an area of memory for "Alerts" that can be manipulated by an attacker. As the application supports up to 20 alerts, and each alert is 20 bytes in size, 400 bytes worth of space is available for use.

To write the uploaded data to permanent storage, a shellcode sequence was used to call a function that was originally used to save configuration settings: "`flash_write`". While this was previously highlighted that the 400 byte area could be used to store shellcode, it can *also* be used to store data intended to be written to flash. By appending the data we wish to write after the shellcode we wish to run, and manipulating the arguments of the function call to target the appended data, it could be written to areas of flash memory with sufficient "free" space. In this case, Turbulence was written to the address `0x35000`.

As 400 bytes was not sufficient to store all 23kb of the ransomware, Turbulence must instead be uploaded in segments, requiring around 59 "rounds" of exploitation in total. For each round, the alerts list is wiped, new data is uploaded, then it is written to persistent memory with `flash_write`.

Stage 2: Redirection Once Turbulence is written to flash, the existing firmware then has to be modified, such that when the device is rebooted, Turbulence is executed rather than the original OS. While this may seem simple in theory, a number of issues arose that complicated matters significantly.

First, modifying code as it is running can result in undefined behaviour. If the original application attempts to run code that we have modified, it is likely that the application will crash, and may result in the device being left in a "bricked" state (i.e. it cannot be used any more, or it is in an irrecoverable state).

Second, due to the nature of flash memory, it is not possible to arbitrarily "write" bits of 1 to flash; instead, only bits of 1 can be set to 0 [39]. Bits that are set to 0 cannot be changed back to 1 without erasing the "block" that the bit is contained in, which sets *all* bits within the block to 1. In the case of the nRF52840, a block is 512 bytes in size.

To facilitate eventual recovery, triggering wipes should be avoided to preserve as much of the original firmware as possible. To avoid crashes, modifications to existing firmware code should also be limited.

Redirect Calculator To redirect execution to the ransomware, an attacker should aim to take control of the boot process as early as possible. For the nRF52840, the boot state is managed by the addresses `0x0` and `0x4`, which load the initial stack pointer and program counter respectively. Therefore, the value in `0x4` could be modified to redirect execution, thus allowing the attacker to take control of the device on boot. However, as highlighted previously, a series of one bits cannot be written to positions that have been set to zero. Therefore, the value of the boot address can only be *reduced*, not increased.

In the testbed, the original boot target was set to `0x5431`. Therefore, this value could not be changed to `0x35000` without requiring a page wipe, which would result in significant data loss. Instead, a tool was created to calculate every address that could be generated via zero-bit writes to the original boot target. The values stored at those addresses would then be scanned to identify whether any useful instructions could be created by further zero-bit writes to those locations.

As an example, the `B 0x35000` instruction could be used to move execution directly to the ransomware. However, this instruction would require a reachable address to contain 21 correctly positioned one bits. To increase the chance of finding potential matches, the `B 0x35000` instruction can instead be split into separate, smaller instructions.

First, the address must be written to an area in memory nearby the current instruction address. A `LDR Rx, <value>` instruction can load that value into a register, followed by a `MOV PC, Rx` instruction to move it into the program counter, redirecting execution.

Normally, these instructions would need to be contiguous, as otherwise any code between the new instructions belonging to the original firmware would run, which may result in unexpected behaviour. However, it was found that the byte sequence `0x0000` is interpreted as a pseudo-`NOP` instruction, as it moves the contents of `r0` into `r0`. As this does not require any 1 bits to be written, it will always be possible to write this instruction to memory. Therefore, split instructions can be used reliably, drastically increasing the search space.

As this process would be tedious to perform manually, a Python script was developed to automatically identify the changes required to redirect execution.

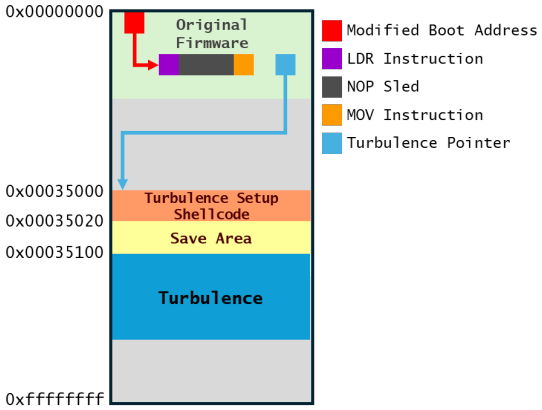

Fig. 4. Overview of Exploit's Edits to Flash Memory

When given a target binary, original boot address, and ransomware install address, the program produces a list of candidates of potential changes. Before performing the edit, the original values are saved to an area of memory reserved by the ransomware for later restoration, then the changes are written to flash.

Stage 3: Resetting A final exploit is used to run shellcode that forces the device to reset. When the device reboots, the new boot address will cause the redirect code to execute, and finally, run Turbulence. A simplified overview of the edits made to flash during the exploitation stage is shown in Figure 4.

4.5 Execution

At this point, the device will boot into Turbulence, fully under the attacker's control. The ransomware attack itself is split into three main stages outlined below.

Backup As mentioned previously in Section 4.4 (Stage 2), any areas of memory that would be damaged during the initial exploit stage can be backed up to an area `0x80` bytes before the Turbulence firmware install address, such that they could be restored to their original locations should the ransom be paid.

Ransom By replacing the original firmware, Turbulence "locks" the target in a state that prevents it from being used for its original purpose. While it is possible to encrypt the data stored on the device, it is likely unnecessary, as at this stage, the victim is no longer expected to have access to the application or the storage medium. This lack of access is the main barrier preventing victims from regaining control of the device, rather than the loss of data.

Spread To spread the infection, any devices infected by Turbulence must be capable of recreating the exploit against other nearby devices. In this case, each device acts as a "peripheral", waiting for a "central" Bluetooth device to connect and send commands.

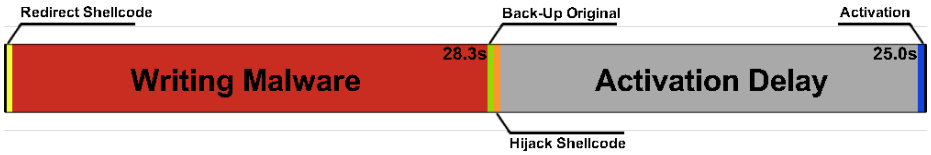

Fig. 5. Timing Breakdown of Exploit Stages

As the device is expected to assume the peripheral role, the original firmware is not configured to support the central role. However, the replacement firmware that is installed during the loading stage can be configured to include any features supported by the target hardware. Turbulence is configured to act as a central device, allowing it to communicate with other target peripheral devices.

Once executed, Turbulence immediately scans for nearby Bluetooth devices, and upon discovering them, will attempt to connect and perform the exploit as defined in Section 4.4. As long as there is a device in range that is vulnerable to this exploit, Turbulence is able to propagate. If the victim is able to manually recover a device without paying, it is still liable to be re-infected by other nearby infected devices, unless the vulnerability is patched.

During testing, infecting devices from an attacking machine took around 54 seconds to complete after the initial connection. A breakdown of the time taken for each stage can be seen in Figure 5. The infected device then attempted to automatically spread the infection to a nearby device without attacker intervention, taking approximately 64 seconds.

4.6 Post-Infection

When Turbulence has infected an acceptable amount of the network, the attacker can then attempt to procure payment from the victim.

Advertise After a device has been successfully encrypted or locked, ransomware will often attempt to deliver a ransom note to the victim. For ransomware targeting personal computers or phones, this is a relatively simple process, typically achieved by spawning an application window or message within the infected operating system [29].

For constrained devices, however, the attacker cannot rely on "standard" methods of communication being used by the victim. Previous research has also encountered this issue when attempting to develop ransomware for Linux-based IoT devices. It was found that by hijacking communication methods provided by targeted devices, such as Hypertext Transfer Protocol servers or connected screens, they could instead be used to deliver ransom notes to the victim [10]. This method could be applied to constrained IoT devices, but the success of this method would heavily rely on the support and abstraction provided by the RTOS for the chosen method of communication.

Alternatively, if the attacker knows the identity of the victim, such as in a targeted attack, communication hijacking is likely unnecessary, as the attacker can message the victim directly via "out-of-band" communication channels, such as email.

Recovery If payment is provided by the victim, the attacker is expected to provide guidance to unlock any infected devices. For traditional "crypto-based" ransomware, this would typically involve the attacker providing an encryption key, which can be used by the victim to regain access to encrypted files. While Turbulence does not implement encryption, a similar process can be used.

During compilation, an "unlock key" could be generated, and its hash hard-coded into the ransomware binary. Once the victim has paid, the victim would be provided the key, which must then be input to the infected device via a communication channel supported by the device, such as Bluetooth or a UART port. If the hashed input matches the hash stored on the device, Turbulence could reverse the changes made during the exploitation stage using the backup stage described in Section 4.5 (Backup), then trigger a reboot. At this point, the device would boot into the original firmware, and return to normal operation.

To prevent re-infection from other infected devices, a "magic value" could be written at a chosen offset, which would be read during the exploitation stage defined in Section 4.2. If the offset was found to contain the magic value, the infection stage would end prematurely, and the ID of the target added to a blacklist on the attacking device.

Victims could attempt to recover the device themselves without paying the ransom. However, there are a number of requirements for manual recovery to be considered viable. This would include the technical knowledge as to how to attempt a recovery, adequate time to access and reprogram each infected device, a method to regain access to the device (such as via an exposed and enabled debug port), a backup of the original firmware and configuration, the ability to wipe and reprogram storage, and the ability to prevent re-infection.

5 Discussion

Once Turbulence was proven capable of exploiting, locking, and propagating via the target device, a post-infection analysis was performed. Here, we discuss the key implications, countermeasures, and limitations of the attack.

5.1 Key Implications

Targeting constrained devices on bare metal or RTOS-based firmware presents several new challenges that may discourage attackers, such as limited firmware cross-compatibility, limited features due to "opt-in" configuration, and missing functionality typically provided by GPOSs (filesystem, standard interface, etc.).

Turbulence shows that despite these issues, ransomware for constrained devices can be developed efficiently, and can be used to successfully infect, lock, and ransom a target. While constrained devices are currently unappealing to

attackers, they could be targeted in the future. By abstracting complex functionality, providing support for many hardware targets, and supplying easily configurable features, RTOSes simplify the process of writing firmware for IoT developers. Unfortunately, attackers can *also* leverage these benefits, and ransomware designed with these tools can be more easily adapted to additional targets.

As Turbulence is designed to replace the firmware of the target device, the attacker does not need to rely on the target including the functionality required for the ransomware to function. However, during this work, a minimal list of requirements was identified for this technique to be applicable:

- **Board support** - The board must be supported by the chosen RTOS[2].
- **Remotely Exploitable** - The attacker must be able to exploit the target remotely[3], such that they can force the target to run arbitrary code. This will allow the attacker to gain their initial foothold and subsequently allow the ransomware to spread through the network.
- **Writeable Flash** - The attacker must be able to write arbitrary values to flash, such that changes are maintained after a reboot.
- **OS Redirection** - The attacker must be able to redirect execution to the new operating system. In this case, this is achieved by modifying the boot process and forcing a soft-reboot.

5.2 Countermeasures

Here, we discuss possible countermeasures that can be used to reduce the effectiveness of this approach. These countermeasures have been split into two parts: detection and prevention.

Detection By detecting ransomware early, victims can isolate or decommission affected devices to limit the spread of an infection. Previous work has shown that by monitoring certain properties of an IoT device, such as metadata in wireless traffic [20] or power consumption [6], behavioural analysis can be employed to identify malicious behaviour. However, constrained devices may present unique challenges for IoT developers, as the use of such countermeasures may conflict with logistical requirements, such as minimising power consumption and computational overhead.

Various tools have been produced in an attempt to address these issues, either by adapting to the constrained environment, or protecting devices at the network level. Some examples include: a lightweight neural-network that can be used to identify malicious binary code [4], an anomaly monitor designed to be run within the trusted execution environment [30], and a traffic classification system intended for use on edge devices [17]. As tools of this type are unlikely

[2] While Zephyr is used in this work, ransom-firmware can be implemented by any RTOS of the attacker's choice.

[3] It should be noted that while BLE was used for this PoC, other communication mediums such as WiFi, Zigbee, or LoRaWAN could be used, as long as it is accessible to the attacker.

to be designed with ransomware in mind, their effectiveness against the specific techniques described in this work would likely have to be tested, but they could still potentially be used to detect similar malicious behaviour commonly found in other malware, such as during the exploitation stage.

Prevention Developers can also make proactive steps to defend against ransomware attacks. For example, scanning their firmware's code for common vulnerabilities, such as stack overflows, could reduce the number of vulnerabilities for the attacker to exploit. Memory protection features, such as DEP, stack canaries, or Address Space Layout Randomisation (ASLR) [47] can also be used to drastically increase the complexity of an attack, and should be used when supported by the underlying hardware.

While these methods are often recommended for hardening devices during development, past research indicates that a significant portion of devices may not implement these exploit mitigation techniques. Analysis of publicly available firmware for Linux-based devices showed that only 34.48% had enabled DEP, and only 5.65% used position independent executables, which would be able to effectively utilise ASLR [1].

By utilising a bootloader with support for firmware verification, developers can prevent any unsigned firmware from being booted[31]. In a survey of 33 manufacturers of IoT devices performed by the UK government, 67% had implemented "software integrity" checks for all their products. However, 15% had only introduced it to "some" products, and the remaining 18% stated that they had not yet implemented it at all [54]. Another survey with responses from 655 embedded developers showed that only 30% had utilised secure boot in their projects [5].

It is also worth noting that secure boot should not be considered a "silver bullet" for protecting IoT devices against the methods described in this work. For instance, previous Unified Extensible Firmware Interface (UEFI) bootkits such as BlackLotus [51] – or other fault-based exploits [14,55] – have rendered this preventative approach ineffective. Also, while this method would prevent ransom-firmware from being used, the attacker would still be able to impact the device's functionality by damaging the original firmware, essentially performing a DoS attack.

Previous research suggests that the adoption of security features is heavily influenced by hardware architecture [1]. Therefore, adoption rates could also be influenced by the severe hardware limitations imposed by constrained IoT devices. For example, previous work has suggested that the low adoption rate of ASLR on IoT devices could be due to the limited address space, which can decrease the available entropy, reducing its effectiveness [1]. If a constrained device is found to be unable to implement certain protections due to a lack of relevant hardware (such as a Memory Management Unit for hardware-enforced DEP), developers may instead have to seek alternative methods designed with these constraints in mind [27].

5.3 Rejected Methods

During this work, we assessed three methods that could be used to develop functional ransomware for constrained IoT devices. One of these methods, "firmware replacement", was found to be effective, and was utilised in development of the Turbulence PoC. However, the other methods were deemed unfit for purpose. In this section, we discuss these methods' weaknesses, and our attempts at implementing them.

Live Off the Land For this approach, we considered developing ransomware that would run alongside the original firmware by loading a very small malicious program into RAM, which is then executed via an exploit. Complex features already implemented by the original firmware that would facilitate an attack (such as saving data to flash) could then be utilised by calling functions directly via their address in memory.

However, this method is highly dependent on the target device's implementation. If a feature required by the ransomware is unavailable in the original firmware, the attacker would need to develop and implement it themselves, significantly increasing the overall storage requirement and development time. As the ransomware would reside entirely in RAM, the space available to the attacker is likely to be very limited.

This method would also be difficult to maintain, as in order to develop functional shellcode that can run within the context of another application in RAM, the ransomware would have to be written in assembly rather than C, significantly slowing down the development process. Minor modifications made by device updates could also require the attacker to make drastic redesigns of the ransomware to adapt to changing function addresses, behaviour, and availability.

Finally, other threads and processes implemented by the original firmware may run in parallel, which could result in interference, undefined behaviour, or even crashes. In some cases, functionality provided by the original firmware, such as remote installation of firmware updates, may allow victims to attempt manual recovery of the device.

When attempting to implement this method, it was found that while it was possible to inject shellcode that could call existing functions present in the original firmware, the target device was configured as a "peripheral" Bluetooth device. As mentioned in Section 4.5, to use the "central" Bluetooth role necessary to communicate with other devices, the device would have to be fully reconfigured, which was not simple to achieve with this method. As this would prevent the ransomware from spreading autonomously, this method was found to be inadequate.

Ransom Library Pre-existing projects, such as Zephyr [63], have already performed the arduous task of programming a robust and adaptable API that can be utilised on various platforms. With this method, an attacker could extract useful features and code from such projects, then compile them into a library that can be uploaded for use during an attack.

This approach would drastically reduce the reliance on the original firmware providing the necessary features, and reduce the work required by the attacker to produce such features themselves. However, it would also significantly increase the size of the uploaded program. Additionally, similar to the previous approach, processes and threads created by the original firmware could still cause interference.

To implement this method as part of the PoC, attempts were made to extract the required Bluetooth functionality from pre-existing code in the Zephyr codebase [59]. However, identifying and extracting the relevant code and all its dependencies took significantly more time and effort than expected. It was determined that even if this method eventually proved successful, the process would have to be repeated for any new feature required by the attacker, increasing the development time for future targets.

5.4 Limitations

While this work has made significant strides in exploring the development process for constrained ransomware, there are a few areas that could stand to see improvements.

Single Device PoC While this work focused on producing a singular PoC, significant strides were made in maintaining the "adaptability" of the methods and the development process. When targeting a new device, if the hardware is supported by the RTOS utilised by the ransomware, only minimal changes to the code are likely to be required, as hardware differences should be handled by the abstraction provided by the RTOS's build system. However, the exploitation and loading stages would likely need to be fully rewritten, unless similar firmware, hardware, or vulnerabilities are used. The method used to hijack the boot-process in the execution stage is likely to be hardware dependant, but could be somewhat standardised between chips of the same series or producer. Finally, any security features provided by the target (if implemented), such as those described in Section 5.2, could require additional changes in the exploitation and execution stages. As an example, the method used in this PoC to bypass DEP in the nRF52840 is unlikely to work for a new target with different hardware, unless a chip of a similar design is used.

However, for this PoC, Turbulence was only tested against one target board. To more thoroughly assess the viability and generalisation of the methods defined in this work, they should also be tested against other devices. Ideally, these devices should implement different board types (e.g. nRF540, STM32, CC2650), operating systems (e.g. Zephyr, Contiki-NG, FreeRTOS), and communication mediums (e.g. Thread, Zigbee, Wi-fi).

Full Wipe Feature on the nRF52840 While Turbulence is capable of disabling, re-programming, and even programmatically locking a target device, the nRF52840 is designed such that the user can always use the "NVMC ERASEALL" command, which triggers a full wipe of internal memory [39]. While this does not constitute a full "recovery", if the victim is aware of this feature, and the pre-

requisites defined in Section 4.6 (Recovery) are fulfilled, the victim may be able to subsequently recover the device by restoring the device's original firmware.

While there is no "official" way to disable this full-wipe mechanic, errata documents show that certain edge cases in the device state could cause the erase functions to fail [40]. While we were not able to recreate the effects defined in the document, these issues could potentially allow attackers to retain control and prevent a reset from occurring. It should be noted that similar devices such as the nRF5340 have implemented protection features that allow developers to prevent the erase command from being used [41].

Bluetooth Realism The temperature monitoring testbed firmware was designed to be relatively simplistic, and followed examples given by the Zephyr codebase for best practices. By default, the Bluetooth features pair with other devices without requiring authentication, physical interaction, or "PIN-based" pairing. If authentication is required by a targeted device or network, additional steps may be required to spread throughout the network.

6 Conclusions and Future Work

In this paper, we demonstrated the viability of developing and implementing ransomware targeting constrained IoT devices. First, we identified the challenges that constrained IoT devices present to ransomware development, followed by three approaches that attackers may use to address them. We then attempted to create a compatible PoC ransomware using each of the proposed development approaches. It was found that while the first two techniques defined in Section 3.2 (Live Off the Land and Ransom Library) were not fit for purpose, the Firmware Replacement technique would allow attackers to develop ransom-firmware, which can be more easily adapted for use on other targets by using a RTOS.

In order to test the PoC ransomware, we developed intentionally vulnerable firmware based on the Zephyr operating system, which was then run on a testbed of constrained IoT devices. We simulated a ransomware attack against the testbed via Bluetooth, which was able to infect the target device, and spread to other nearby devices. We then analysed the attack, identifying the ransomware's requirements and limitations. Finally, we suggested a list of potential countermeasures that developers could implement to reduce the effectiveness of such attacks.

In this work, we prioritised the use of development methods that would allow malware to be adapted to new targets. Future work could evaluate the adaptability of this technique by simulating attacks on other constrained devices with known vulnerabilities. We also suggest a number of countermeasures that developers could implement to defend against the discussed techniques; future work could attempt to assess these countermeasures to determine their effectiveness at detecting ransomware, address any logistical issues the constrained nature of the target devices may present when employing them, and investigate whether such protections could be evaded.

Acknowledgments. This work was supported by the funding received from the UK EPSRC project grants EP/X036707/1 and EP/X036871/1 on Countering HArms caused by Ransomware in the Internet Of Things (CHARIOT). The authors would also like to thank the anonymous reviewers for their constructive feedback.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

A nRF52840-DK Board

The Nordic Semiconductor nRF52840-DK board (see Figure 6) features 1MB of flash memory, 256kb of RAM, and is capable of communicating via BLE and Zigbee [37].

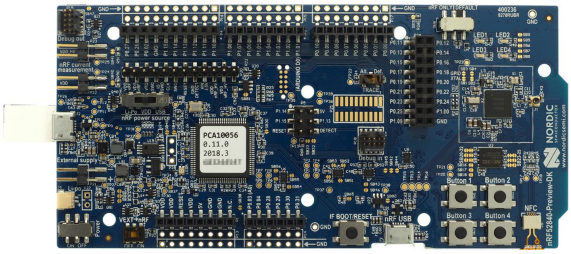

Fig. 6. An image of the nRF52840-DK [18]

References

1. Akiyama, M., Shiraishi, S., Fukumoto, A., Yoshimoto, R., Shioji, E., Yamauchi, T.: Seeing is not always believing: Insights on iot manufacturing from firmware composition analysis and vendor survey. Computers & Security **133**, 103389 (2023)
2. Al-Rimy, B.A.S., Maarof, M.A., Shaid, S.Z.M.: Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Computers & Security **74**, 144–166 (2018)
3. Alrawi, O., Lever, C., Antonakakis, M., Monrose, F.: Sok: Security evaluation of home-based iot deployments. In: 2019 IEEE symposium on security and privacy (sp). pp. 1362–1380. IEEE (2019)
4. Anand, S., Mitra, B., Dey, S., Rao, A., Dhar, R., Vaidya, J.: Malite: Lightweight malware detection and classification for constrained devices. IEEE Transactions on Emerging Topics in Computing (2025)
5. Aspencore: Embedded survey - the current state of embedded development (2023), https://www.embedded.com/wp-content/uploads/2023/05/Embedded-Market-Study-For-Webinar-Recording-April-2023.pdf [Accessed: February 2026]

6. Azmoodeh, A., Dehghantanha, A., Conti, M., Choo, K.K.R.: Detecting crypto-ransomware in iot networks based on energy consumption footprint. Journal of Ambient Intelligence and Humanized Computing **9**(4), 1141–1152 (2018)
7. Benmalek, M.: Ransomware on cyber-physical systems: Taxonomies, case studies, security gaps, and open challenges. Internet of Things and Cyber-Physical Systems **4**, 186–202 (2024)
8. Bierbaumer, B., Kirsch, J., Kittel, T., Francillon, A., Zarras, A.: Smashing the stack protector for fun and profit. In: ICT Systems Security and Privacy Protection: 33rd IFIP TC 11 International Conference, SEC 2018, Held at the 24th IFIP World Computer Congress, WCC 2018, Poznan, Poland, September 18-20, 2018, Proceedings 33. pp. 293–306. Springer (2018)
9. Brewster, T.: Medical devices hit by ransomware for the first time in US hospitals (2017), https://www.forbes.com/sites/thomasbrewster/2017/05/17/wannacry-ransomware-hit-real-medical-devices/ [Accessed: February 2026]
10. Brierley, C., Pont, J., Arief, B., Barnes, D.J., Hernandez-Castro, J.: Paperw8: an iot bricking ransomware proof of concept. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–10 (2020)
11. Brierley, C., Pont, J., Arief, B., Barnes, D.J., Hernandez-Castro, J.: Persistence in linux-based iot malware. In: Secure IT Systems: 25th Nordic Conference, NordSec 2020, Virtual Event, November 23–24, 2020, Proceedings 25. pp. 3–19. Springer (2021)
12. Celdrán, A.H., Sánchez, P.M.S., Scheid, E.J., Besken, T., Bovet, G., Pérez, G.M., Stiller, B.: Policy-based and behavioral framework to detect ransomware affecting resource-constrained sensors. In: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium. pp. 1–7. IEEE (2022)
13. Costin, A., Zaddach, J.: Iot malware: Comprehensive survey, analysis framework and case studies. BlackHat USA **1**(1), 1–9 (2018)
14. Cui, A., Housley, R.: BADFET: Defeating modern secure boot using Second-Order pulsed electromagnetic fault injection. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, Vancouver, BC (Aug 2017), https://www.usenix.org/conference/woot17/workshop-program/presentation/cui
15. De Donno, M., Dragoni, N., Giaretta, A., Spognardi, A.: Ddos-capable iot malwares: comparative analysis and mirai investigation. Security and Communication Networks **2018**(1), 7178164 (2018)
16. Echo Duan, Veo Zhang, K.Y.: FLocker mobile ransomware crosses to smart tv (2016), https://www.trendmicro.com/ru_ru/research/16/f/flocker-ransomware-crosses-smart-tv.html [Accessed: February 2026]
17. Eichler, C., Röckl, J., Jung, B., Schlenk, R., Müller, T., Hönig, T.: Profiling with trust: system monitoring from trusted execution environments. Design Automation for Embedded Systems **28**(1), 23–44 (2024)
18. Farnell: Farnell - nRF52840-DK (2025), https://uk.farnell.com/nordic-semiconductor/nrf52840-dk/dev-kit-bluetooth-low-energy-soc/dp/2842321 [Accessed: February 2026]
19. Flores, R.: The impact of modern ransomware on manufacturing networks (2020), https://www.trendmicro.com/en_us/research/20/l/the-impact-of-modern-ransomware-on-manufacturing-networks.html [Accessed: February 2026]
20. Gao, M., Wu, L., Li, Q., Chen, W.: Anomaly traffic detection in iot security using graph neural networks. Journal of Information Security and Applications **76**, 103532 (2023)
21. Gazet, A.: Comparative analysis of various ransomware virii. Journal in computer virology **6**, 77–90 (2010)

22. Griffioen, H., Doerr, C.: Examining mirai's battle over the internet of things. In: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security. pp. 743–756 (2020)
23. Habibzadeh, H., Nussbaum, B.H., Anjomshoa, F., Kantarci, B., Soyata, T.: A survey on cybersecurity, data privacy, and policy issues in cyber-physical system deployments in smart cities. Sustainable Cities and Society **50**, 101660 (2019)
24. He, W., Zhao, V., Morkved, O., Siddiqui, S., Fernandes, E., Hester, J., Ur, B.: Sok: Context sensing for access control in the adversarial home iot. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 37–53. IEEE (2021)
25. Honeywell: Ransomware attacks targeting industrial operators surge 46% in one quarter, honeywell report finds (2025), https://www.honeywell.com/us/en/press/2025/06/ransomware-attacks-targeting-industrial-operators-surge-46-percent-in-one-quarter-honeywell-report-finds [Accessed: February 2026]
26. Islam, S.R., Kwak, D., Kabir, M.H., Hossain, M., Kwak, K.S.: The internet of things for health care: a comprehensive survey. IEEE access **3**, 678–708 (2015)
27. Jung, J., Kim, B., Son, H., Jang, D., Lee, B., Cho, J.: A segmented stack randomization for bare-metal iot devices. Computers & Security **151**, 104342 (2025)
28. Kao, W.C.: bt: hci: Dos and possible rce (2023), https://github.com/zephyrproject-rtos/zephyr/security/advisories/GHSA-j4qm-xgpf-qjw3 [Accessed: February 2026]
29. Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., Kirda, E.: Unveil: A large-scale, automated approach to detecting ransomware. In: 25th USENIX security symposium (USENIX Security 16). pp. 757–772 (2016)
30. Kumar, A., Lim, T.J.: Edima: Early detection of iot malware network activity using machine learning techniques. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). pp. 289–294. IEEE (2019)
31. Löhr, H., Sadeghi, A.R., Winandy, M.: Patterns for secure boot and secure storage in computer systems. In: 2010 International Conference on Availability, Reliability and Security. pp. 569–573. IEEE (2010)
32. Margaritelli, S.: Reversing the smarter coffee iot machine protocol to make coffee using the terminal (2016), https://www.evilsocket.net/2016/10/09/IoCOFFEE-Reversing-the-Smarter-Coffee-IoT-machine-protocol-to-make-coffee-using-terminal/index.html [Accessed: February 2026]
33. 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)
34. Naeini, P.E., Bhagavatula, S., Habib, H., Degeling, M., Bauer, L., Cranor, L.F., Sadeh, N.: Privacy expectations and preferences in an iot world. In: Thirteenth symposium on usable privacy and security (SOUPS 2017). pp. 399–412 (2017)
35. NIST, Amazon: Cve-2025-5688 (2025), https://nvd.nist.gov/vuln/detail/CVE-2025-5688 [Accessed: February 2026]
36. NIST, Apache Software Foundation: Cve-2025-35003 (2025), https://nvd.nist.gov/vuln/detail/CVE-2025-35003 [Accessed: February 2026]
37. Nordic Semiconductor: nRF52840 product specification v1.11. pp. 22-23. (2024), https://docs-be.nordicsemi.com/bundle/ps_nrf52840/attach/nRF52840_PS_v1.11.pdf [Accessed: February 2026]
38. Nordic Semiconductor: nRF52840 product specification v1.11. pp. 23. (2024), https://docs-be.nordicsemi.com/bundle/ps_nrf52840/attach/nRF52840_PS_v1.11.pdf [Accessed: February 2026]

39. Nordic Semiconductor: nRF52840 product specification v1.11. pp. 25-26. (2024), https://docs-be.nordicsemi.com/bundle/ps_nrf52840/attach/nRF52840_PS_v1.11.pdf [Accessed: February 2026]
40. Nordic Semiconductor: nRF52840 revision 3 errata v1.3. pp. 20-21. (2024), https://docs-be.nordicsemi.com/bundle/errata_nRF52840_Rev3/attach/nRF52840_Rev_3_Errata_v1.3.pdf [Accessed: February 2026]
41. Nordic Semiconductor: nRF5340 product specification v1.6. pp. 402. (2025), https://docs-be.nordicsemi.com/bundle/ps_nrf5340/page/nRF5340_PS_v1.6.pdf [Accessed: February 2026]
42. O'Kane, P., Sezer, S., Carlin, D.: Evolution of ransomware. Iet Networks **7**(5), 321–327 (2018)
43. Pen Test Partners: Thermostat ransomware: a lesson in iot security (2016), https://www.pentestpartners.com/security-blog/thermostat-ransomware-a-lesson-in-iot-security [Accessed: February 2026]
44. Rani, S., Kataria, A., Chauhan, M., Rattan, P., Kumar, R., Sivaraman, A.K.: Security and privacy challenges in the deployment of cyber-physical systems in smart city applications: State-of-art work. Materials Today: Proceedings **62**, 4671–4676 (2022)
45. Ronen, E., Shamir, A., Weingarten, A.O., O'Flynn, C.: Iot goes nuclear: Creating a zigbee chain reaction. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 195–212. IEEE (2017)
46. Rostami, A., Vigren, M., Raza, S., Brown, B.: Being Hacked: Understanding Victims' Experiences of IoT Hacking. In: Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022). pp. 613–631 (2022)
47. Saito, T., Watanabe, R., Kondo, S., Sugawara, S., Yokoyama, M.: A survey of prevention/mitigation against memory corruption attacks. In: 2016 19th International Conference on Network-Based Information Systems (NBiS). pp. 500–505. IEEE (2016)
48. Schneider Electric SE: Cve-2022-46680 detail (2023), https://nvd.nist.gov/vuln/detail/CVE-2022-46680 [Accessed: February 2026]
49. Selvaraj, M., Uddin, G.: A large-scale study of iot security weaknesses and vulnerabilities in the wild. ACM Transactions on Software Engineering and Methodology **34**(2), 1–40 (2025)
50. Seri, B., Vishnepolsky, G., Zusman, D.: Critical vulnerabilities to remotely compromise vxworks, the most popular rtos. White paper, ARMIS., URGENT/11 (2019)
51. Smolár, M.: Blacklotus uefi bootkit: Myth confirmed. Online. Bratislava, SK: ESET, spol. s ro (2023)
52. Symantec Threat Hunter Team: Ransomware 2025: A resilient and persistent threat (2025), https://www.security.com/sites/default/files/2025-02/2025_02_Ransomware_2025.pdf [Accessed: February 2026]
53. Tan, X., Ma, Z., Pinto, S., Guan, L., Zhang, N., Xu, J., Lin, Z., Hu, H., Zhao, Z.: Sok:where's the "up"?! a comprehensive (bottom-up) study on the security of arm cortex-m systems. In: 18th USENIX WOOT conference on offensive technologies (WOOT 24). pp. 149–169 (2024)
54. UK Government - Department for Science, Innovation & Technology: Cyber security of consumer iot - manufacturer survey (2024), https://www.gov.uk/government/publications/cyber-security-of-consumer-iot-manufacturer-survey/cyber-security-of-consumer-iot-manufacturer-survey#analysis-of-psti-awareness-compliance-and-impacts [Accessed: February 2026]

55. Vasselle, A., Thiebeauld, H., Maouhoub, Q., Morisset, A., Ermeneux, S.: Laser-induced fault injection on smartphone bypassing the secure boot. In: 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 41–48. IEEE (2017)
56. Wang, H., Zhang, W., He, H., Liu, P., Luo, D.X., Liu, Y., Jiang, J., Li, Y., Zhang, X., Liu, W., et al.: An evolutionary study of iot malware. IEEE Internet of Things Journal **8**(20), 15422–15440 (2021)
57. Zephyr Project: Supported boards and shields - zephyr project (2025), https://docs.zephyrproject.org/latest/boards/index.html [Accessed: February 2026]
58. Zephyr project: Zephyr - application development (2025), https://docs.zephyrproject.org/latest/develop/application/index.html#overview [Accessed: February 2026]
59. Zephyr project: Zephyr - Github (2025), https://github.com/zephyrproject-rtos/zephyr [Accessed: February 2026]
60. Zephyr project: Zephyr - supported features (2025), https://docs.zephyrproject.org/latest/connectivity/bluetooth/features.html [Accessed: February 2026]
61. Zephyr project: Zephyr - wi-fi management (2025), https://docs.zephyrproject.org/latest/connectivity/networking/api/wifi.html [Accessed: February 2026]
62. Zephyr Project: Zephyr api documentation: Flash interface (2025), https://docs.zephyrproject.org/latest/doxygen/html/group__flash__interface.html [Accessed: February 2026]
63. Zephyr Project: Zephyr project (2025), https://www.zephyrproject.org/ [Accessed: February 2026]