Possible Attacks on SOHO and Security Analysis

B.J.Annie Neeraja[1], Prof.R.J.Rama Sree[2]

[1]Computer Center, SPMVV, Tirupati.

[2]Department of Computer Science, National Sanskrit University, Tirupati.

Abstract:

Due to many unusual constraints imposed on their design, embedded systems carry a large set of challenges related to feature implementation and security. Many systems found today prove the high difficulty of achieving the latter, namely designing a secure software architecture. Nevertheless, IoT systems are crucial in many environments ranging from large-scale, tightly secured commercial deployments to relatively unprotected home networks. The danger that embedded devices may face in such versatile environments coupled with the constantly rising popularity and feature scope of IoT systems is one of the main factors which motivated this paper. In this work, we focus on SOHO (Small Office/Home Office) embedded systems equipped with Linux OS. Our goal is to identify and address some of their security problems. We also propose a security testing framework-based on our enhanced methodology. Our methodology focuses on increasing the simplicity and intuitiveness of the tasks as well as making them more straight forward. The methodology covers device data collection, network scanning, traffic gathering, software and hardware testing, then preparing the tools, the attack scenario, and its execution. We also advocate for using free, open-source testing tools. This work also provides security recommendations for manufacturers, solution architects and security researchers in the field of internet of things.

Keywords: Embedded system, IoT, smart home, security, threats, cyber security, counter measures, vulnerabilities.

1 INTRODUCTION

The modern world is changing at an incredible pace. To keep up with this momentum, many individuals and companies often abandon important manufacturing engineering and safety tasks. Often such decisions are not preceded by an analysis of how the change or omission will affect the end product. Numerous oversights, as well as limitations of IoT devices, result in an unbelievably large number of security flaws and vulnerabilities that are often exploited by attackers. Nowadays, these devices are usually not the main target of hackers, but just another tool that can be used for larger scale attacks. Devices and systems of this type are often used to distribute malware, mine cryptocurrencies, and are also used as part of botnets used for DDOS attacks.

The times we live in are the times of information, so DDOS attacks are the most impactful as we lose connection with the world. In the first quarter of 2022 strategic targets for national defense. With this we refer to the large attacks in Russian and Ukrainian cyberspace and neighboring countries[2]. The number of vulnerabilities in the industry is becoming more severe and flooding the market(IBMX-Force, 2022). In the Figure 1 we can see the number of vulnerabilities discovered in the last 10 years.

It has been a long time since researchers highlighted what the problem with such solutions might be and yet the market has not listened much. Back in 2016, Ericka Chickowski wrote on *Dark Reading* that Internet of Things devices would become *Botnet All The Things* (Chickowski, 2016).In recent years, the OWASP organization has released a list of the 10 most popular vulnerabilities for IoT[3], most of them are trivial to solve. Unfortunately, we can still find them in the popular IoT solutions.

The zero-day vulnerability in the Apache Log4jlibrary caused quite a sensation in the security world in late 2021(CVE-2021-44228[4]). This vulnerability is we have seen increased activity of hackers attacking

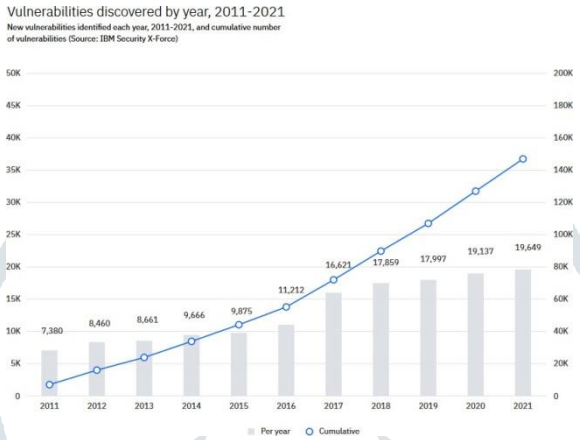

Figure 1: IBM Security X-Force Report on vulnerabilities discovery in year span of 2011-2021(IBMX-Force, 2022)

also known as Log4shell, which allowed remote code execution on a server(Starks, 2021). The vulnerability is also present in IoT devices that use the vulnerable library(Herold, 2021). Another example of major vulnerability discovered in a defibrillator was a remote execution exploit. This was one of a few vulnerabilities that could realistically and directly threaten human life (CISA, 2021).

Motivation, Structure and Contribution

Recently, the Internet of Things sector and especially smart home solutions are starting to reign among regular users. Household penetration of Smart Home solutions will be 14.2% in 2022 and is expected to hit 25.0% by 2026 worldwide. In the more developed markets, this rate is expected to reach a value of about 50%[5]. Smart home solutions strive to make our homes more comfortable and automated. They also help remote management, e.g. by a mobile application or through a website.

Unfortunately, recently we see more and more security vulnerabilities that are being exploited, often for massive DDoS attacks, as well as compromising the security of users of IoT devices. A well-known example of such exploitation is the Mirai botnet which infects IoT devices and uses them for massive net- work attacks[6].

These are our motivations for writing the paper that deals with security testing of IoT devices as well as providing recommendations on how to create and manage such devices to minimize common mistakes. We analyzed different classes of SOHO[7] devices, such routers (several models of e.g., TP-Link, D-Link or Net Gear), IP-cameras(e.g. D-Link), and smart- home solutions (e.g. Fibaro). For over 90% of them, we identified that their software ran on top of some Linux-based Operating System, and thus, we decided to focus on this subclass of IoT devices.

The structure of the paper is as follows. An introduction to the subject is presented in Section 2. This part is accompanied by an initial exploration and categorization of embedded systems. Additionally, in scope of this section the marked trends in embedded environments are investigated in order to find popular software stacks used within them. Afterwards, potential areas and weaknesses which could pose a threat to the overall security of said systems are explored in Section3. Section**??** illustrates a practical methodology developed and utilized by us when approaching several embedded devices with the goal of evaluating their security. Besides the methodology itself, general security threats encountered numerously during our research are also presented as part of sections 5 and6. Those findings are accompanied by a description of their theoretical background, contribution to the overall system security as well as recommendations for possible fixes or improvements.

2 Modern IoT Security

Embedded systems can be generally described as devices composed of physical components controlled by a processing unit. This general definition applies to a large range of devices found today-from ASICs and FPGAs which employ very specific computing resources to systems such as IoT devices which make use of more general-purpose environments, not unlike the ones also found in desktop and server class computers.

There exists a variety of different embedded systems. For the purposes of this paper it is useful to divide them into two following classes:

- Systems permanently connected to a standard computer, which receives all their output and provides entirety of their input.

  Example: ASIC[8] chips
- Systems which, under normal conditions, operate autonomously and provide a variety of services to external systems.

  Example: routers, switches, IoT equipment...

While the first class is not exempt from vulnerabilities, they can be generally considered to be “protected” from possible attacks by the system which they are connected to. As long as the connected computer remains secure, the vulnerable areas remain inaccessible to potential attackers.

The second class in the above list poses a much bigger problem, since those devices are usually operating in computer networks shared with both trusted as well as untrusted devices, without any “filter” for malicious operations. Additionally, the second class usually has a much larger attack surface, as a result of a much broader set of functionalities. Not only does a SOHO router have to implement network packet routing, it should also provide a configuration interface accessed via telnet or HTTP, coupled with diagnostic functionality and in many cases integration with on- line services, such as Dynamic DNS.

With the rapidly growing popularity of IoT systems, the security weaknesses found in embedded devices are quickly becoming critical. This is mainly a consequence of the fact that such systems are usually deployed in home networks which are rarely monitored and secured in the same manner as enterprise ones. Despite that, those devices are steadily becoming responsible for monitoring as well as configuring home appliances, climate control systems and even security devices such as cameras.

Many aspects specific to embedded devices make securing them much more difficult. Limitations on the processing power, energy consumption as well as available storage are often the main reasons behind abandoning existing, well tested, secure software, in favor of custom purpose-specific implementations.

In specific cases the available hardware may simply not support commonly used security enhancements, such as memory protection. This might not cause any major problems when the device works as expected, however it makes successful exploitation of any potential buffer overflows much easier.

General Purpose Embedded Operating Systems

Embedded devices may run on a variety of platforms depending on their particular use-cases. In cases where only basic functionality is necessary the main software may run without any underlying Operating System at all, or with a very minimalistic OS designed to run in constrained environments.

Considering the ever-increasing hardware capabilities when compared to price, and growing scope of functionalities expected from embedded devices it is, however, not uncommon to see the use of a general purpose Operating System with a software stack similar to those found in desktop-class computers, albeit usually tuned to decrease storage requirements.

Indeed, one of the most popular OS kernels in the embedded systems area is Linux-a kernel also widely utilized on desktop and server systems. As indicated by the *Embedded Market Study* performed in 2019 by Aspen Core(Aspen Core, 2019), Embedded Linux was the most popular OS used by 21% of all surveyed companies (followed by in-house developed systems and Free RTOS). In the survey of operating systems considered to be used in future development, Embedded Linux was selected by 31% of participants.

The large number of functionalities that Linux- based systems provide, as well as free access to most of their source code and patch histories, makes them one of the more prominent attack vectors. This, coupled with their increasing popularity are the main reasons behind choosing embedded, Linux-based systems as the primary scope of subsequent sections in this document.

3 Security problems of embedded systems

Before attempting a security analysis of a given embedded device it is important to consider its, of- ten unavoidable, weaknesses. This chapter serves as an overview of significant areas which are worth exploring during such analysis. This overview divides common security problems into two significant parts - those that require physical access to the device and those without that requirement.

Weaknesses which require physical access to the device

While the real-world impact of physical vulnerabilities may not be as large as that of software-based attacks, hardware-based problems are still important to explore since in many cases they may be used as an easy entry point to any further security assessment.

One of the leading examples of such weaknesses is the presence of onboard debug ports usually included to allow internal testing or maintenance. Such ports often use well-known protocols such as JTAG or UART and can be easily accessed by anyone.

Upon gaining access, one may be able to inspect the contents of flash memory, obtain a standard Linux shell, or interrupt and override the normal boot process. This poses a high threat to the device, since usually, boot loaders allow applying custom firmware patches to the internal flash storage without any prior integrity

or signature checks.

Weaknesses which do not require physical access to the device

The second category of weaknesses encompasses software-related areas of embedded systems which can be exploited without direct physical access to the device.

Some of the more prevalent problems in this category are related to administrative interfaces exposed by many embedded devices.

In the past, one of the leading examples was us- age of short and fixed preconfigured passwords which could be easily brute-forced. This issue has been largely addressed, since in most devices available today such passwords are usually more complex and generated based on some device-specific information like the serial number.

Another issue related to the login process comes from the use of protocols like telnet or more commonly HTTP to expose configuration panels. Because those protocols offer no encryption on their own, even secure passwords may be uncovered by the adversary through a man in the middle attack or eaves dropping. Many security issues stem from diagnostic end- points with poorly implemented input sanitization. Such end points, when provided with specially crafted malformed input, can allow an adversary to gain control over a given system, often without prior authentication.

Another source of weaknesses in embedded environments can come from the operating system itself. If the software stack is not updated on a frequent basis, vulnerabilities discovered in components like the Linux kernel itself, or the *Busy box* tool may stay exploitable for a long time.

This holds especially true for the Linux kernel if the vendor relies on custom, proprietary drivers. The LinuxABI[9] is not considered stable and can often require embedded vendors to manually adapt their code during kernel upgrades. Because of this it is not uncommon to still find Linux 2.6.x kernels present on otherwise up to date devices.

Finally, a major security issue exhibited by many embedded devices is the usage of sensitive information like passwords or security keys hardcoded into firmware binaries. While this practice might prevent casual discovery of encrypted information, a determined attacker will usually be able to inspect the firmware, find and extract such keys and use them to decrypt sensitive data (e.g. configuration backups containing passwords).

4 ENHANCED TESTING METHODOLOGY

Methodology plays an important role in the security testing process of IoT devices as it helps in adhering to the desired security standards. It also facilitates portability of tests between different types of devices and reliable comparison of results, as well as reproduction of the process by other researchers. A well- designed framework prevents confusion about the sequence, skipping steps, or executing incorrect tests.

Chosen methodology

We adapted and extended testing methodology from the paper *Vulnerabilities in IoT Devices for Smart Home Environment*(daCostaetal., 2019). Authors focus on identifying threats and vulnerabilities in tested IoT devices. They also focus on recommending open- source tools for those performing such testing.

The usage of the scheme they proposed was very helpful in analyzing the selected devices. However, during testing, we found out there is not much potential for parallel work. Noticing this problem, we significantly flattened the structure of the test frame- work. This way, individual tasks can be distributed within the testing team among people who specialize in a particular topic.

Our proposed scheme also involves more activity during data collection and analysis. Later, we planned that all those involved in the previous phases would come together and combine their information into an action plan. Based on the plans, testers will prepare tools, then attack the device and make a detailed re- port. Our scheme will be more intuitive and easier to implement for testing teams. We tried to make it as close as possible to the modern security testing standards used by professionals.

Method structure

The testing framework divides the entire process into different stages and tasks. A flow diagram for our methodology is available in Figure 2. Different steps and procedures can be executed in parallel, but only for the same level in the diagram. The following steps can only be performed if all tasks and steps have been completed for the given level. This rule does not apply to performing subsequent tasks in a branch at the same level. We point out that not all steps can be applied to every device, those tasks that for some reason cannot be performed (e.g. the device does not have a web interface) should be skipped. Each level can be considered completed if all tasks reach the black block in the diagram.

Whole structure can be presented by this high level steps:

1. Setting up a test environment
2. Information gathering
3. Information analysis
4. Attempts to exploit discovered vulnerabilities

The entire process will be described in detail in the next few subsections.

Set up testing environment

Ideally, testing should be performed in a physically or logically isolated environment. It is good to already have test environments in place that are well configured and known by the testers. Any operating system can be used, although we recommend using Kali Linux or Parrot OS. These systems come with sets of predefined tools used for security testing of variety of systems. Such approach also facilitates reproducibility of tests.

Information Gathering

This phase can be divided into two parts: the left- hand branch focuses on passively acquiring information about the system under investigation. It consists of the following steps:

Identify the Device and its Components - identification information about the analyzed device as well as its components should be collected and described. Examples include model and serial numbers, name- plates or part identifiers.

Collect Documentation and Information- we need to review provided documentation about device, check the manufacturer website. Sometimes it may include for instance developer documentation for open API. Community forums very often provide known work a rounds and unofficial solutions for many problems. In this phase, OSINT (Open-Source Intelligence[10]) techniques and tools will be most useful.

Collect Information about the Mobile Application

-we try to identify what functionality is provided for mobile apps connected to IoT solutions. Such information can be obtained from public source repositories, or, since in most cases the source code is not available publicly, decompiled application packages.

Looking at the right-hand branch, we can see the active gathering information about the system. There are following stages:

Device identification aims to check which devices of the test solution we are able to identify in the network.

Network Traffic Capture - in this step, we perform normal system operation while capturing traffic. Let us prepare a test plan in which we include the cases to test and scenarios to support the work. It is important that the capture is done in monitor mode, because only then will it show all traffic in the network, and not only the traffic addressed to the network interface behind which *Wireshark* is located.

Port and Service Identification - we want to query all the services running on a given network and identify their type and version. In this step, we can add verification whether a particular version has security vulnerabilities.

Vulnerability Scanning should be performed in two variants: with or without user/account credentials. This allows us to conduct broaden reconnaissance and make it harder to miss some non-obvious vulnerabilities.

Once we have completed all the tasks in this category, we can move on in testing.

Analysis stage

The next step in our scheme is the analysis phase in which we analyze the results gathered in previous part. The advantage of our methodology is that we can work concurrently on few issues. Another add-on is that the individual tasks can be distributed among people who specialize in particular activities.

Vulnerability Assessment is the first step in this phase, which allows us to analyze the results of the vulnerability scans and check their validity in a running system. It is worth checking the vulnerabilities databases, because quite often it is possible to find prepared exploits for a given vulnerability.

Port and Protocol analysis focuses on analyzing ports, communication protocols, and services that expose their functionality to the outside world. At this point, we try to use bugs or information about used versions of services to conduct an exploration in order to check the usefulness of found gaps.

Traffic Analysis is the next activity in the analysis phase. It will focus on analyzing the traffic captured in the previous phase. We will try to identify the communication endpoints, see if there is any unsecured communication channel and if it is possible to intercept the data or files being transmitted.

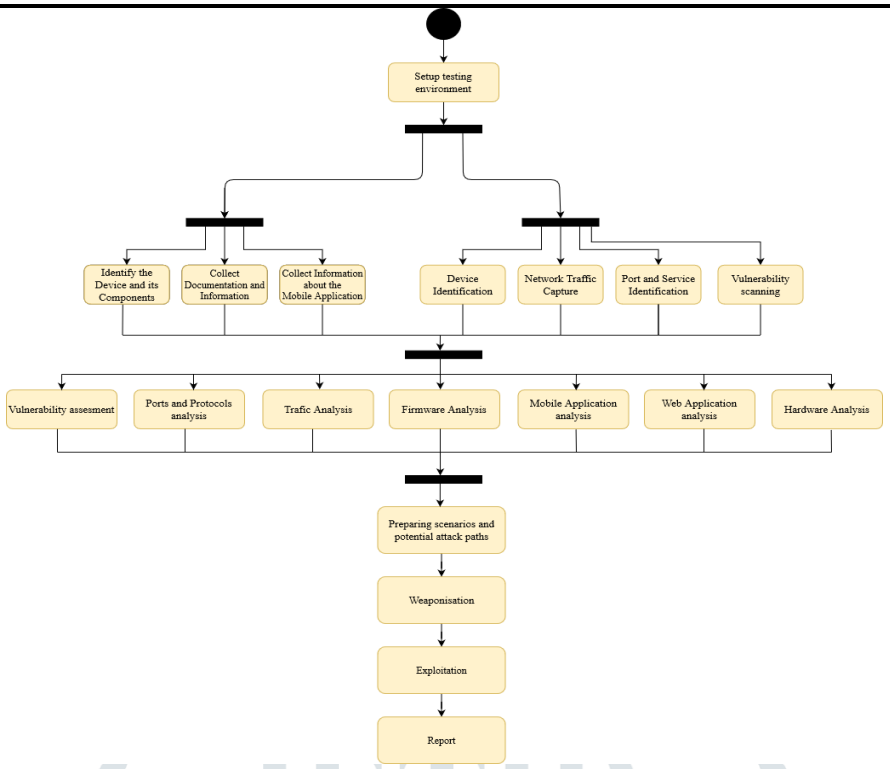

Figure 2: Enhanced testing methodology for IoT devices-flow diagram

Firmware Analysis is one of the most important tasks in the analysis stage because the researcher has direct access to the code implementation. Inspecting it allows us to identify the weak points in the system, and the knowledge gained during the analysis will help to verify the vulnerabilities in the next stage. It also aids in adjusting the appropriate parameters during exploitation. Code analyzers can be helpful in this phase as they can immediately qualify pieces of code as vulnerable.

Mobile App Analysis consists of decompiling the found packages in the mobile app analysis phase and performing static code analysis. In the analysis, we look for errors in implementation of security mechanisms, cloud computing endpoints, hard-coded API keys, and other similar issues that can decrease IoT solutions' security level.

Web App Analysis refers to some of the activities performed as part of regular penetration testing of web applications. The analysis of the application should consist of determining the existing vulnerabilities as well as checking the possibility of their exploitation.

Hardware Analysis is one of the additional tasks that is worth doing if we have expertise in analyzing electronic components. Here we can check the tamper protection. Then we can locate interfaces used in production phase as well as in debug phase during product development. If the connection to these interfaces is possible, we can try to get directly to the device's memory, extract firmware and try to retrieve the hardcoded keys, if there are any.

Preparing an attack

During the attack preparation phase we already have the analysis results and, based on these results, can discuss and propose potential attack scenarios and paths to deal with each scenario. We can create non-trivial attack paths against IoT devices with a cross- section of people with different specializations. Individuals who specialize in this type of activity can prepare each attack step.

Weaponization

In this phase, we already know the scenarios and attack paths. Based on these, we create and customize payloads, scripts and programs for the attack. Thanks to the information gathered and their analysis, we are able to choose the best type of solution for each phase of the attack. We can also create scripts to detect vulnerabilities and automate the attack.

Exploitation

The exploitation phase consists of combining all the prepared scripts and payloads and using them in the field. This way, we will be able to verify the correctness of the attack scenarios. If an attack fails, we can change the script configurations or the executed step to try alternative solutions to the problem. Each step and the result of each action must be recorded carefully, because it will be described in the report, which is the next step.

Report

We recommend using fixed structure for the report being prepared. A well-designed template and awareness of it by the researchers will allow for aggregating important information more intuitively and understandably. The report should include all the key information about the tested devices, even those, that have not been exploited, but are potentially dangerous and can be part of the attack chain. A negative result or failure is not a bad result because it tells other researchers which path leads nowhere. Knowing about the failures is crucial as it helps a lot in putting yourself on a better track in further studies.

5 ADDRESSING DETECTED SECURITY ISSUES

As previously stated, during our research we have utilized the described methodology in order to identify vulnerabilities and weaknesses in a wide range of SOHO IoT systems. The following section describes some of the most commonly discovered vulnerabilities and security issues along with proposals of possible improvements in order to mitigate them completely or at least minimize their impact on the overall system.

Hard coded credentials

One of the first issues encountered during the analysis was the use of hard-coded, weak credentials for accessing device's administrative features. A similar problem was later encountered once more, where a fixed DES key was used for encrypting the exported system configuration file.

Two relatively simple solutions could be used to mitigate such problems. The first one is not to set the administrative password at all and instead requires the user to set it when the system is first setup. The second solution, which could be considered more user friendly, and which has already gained some adoption in newer systems, is to derive the default credentials from some information available only when one has direct physical access to the device, like its serial number. It can still lead to problems when such a device is publicly available and attached so that the label can be read, but it does at least prevent easy access by simply attempting to use some well-known default password.

A similar technique could also be used in the case of the private key, where instead of a fixed value, a random sequence would be assigned during the manufacturing, essentially preventing a well-known secret attack.

Obfuscation over validation

Besides the issue of hard coding a secret value in the firmware binary, the particular usage of this key raises another question - why was encryption used at all when exporting the configuration backup?

Two possible answers are that it was implemented as tampering prevention or to obfuscate the username and password visible within the file's contents. If so, then it is essential to mention that both problems could be addressed in much less error-prone ways. Like any other user input, the configuration import should follow the usual access controls and validation applied to direct user input. As for the visible credentials and particularly the password, a better approach would be not to store it anywhere at all and instead hash and salt it right after setup.

Lack of digital signing

Another encountered problem has to do with the firmware binary itself. It appears that in many cases the firmware format used by the embedded system does not include any digital signature which could be verified during the update process. This makes it very simple to create malicious copies of the official firmware as long as the size constraints would be met. For instance, in case of TP-Link WR740 this issue is addressed in the newer router models, since the latest version of firmware image format includes RSA signature of its data. The signature is verified during the standard update process, however it still is possible to boot modified firmware binaries with help of the serial U-Boot console. This decision was most likely intentional to enable third-party modifications such as installing alternative operating systems and considering the necessity of a physical access can be treated as relatively low risk. While unrelated to network equipment like the tested router, it might be worth to add that if the embedded device stores some confidential information, a third party modification like booting unsigned software should require removal of such data before hand.

Inflexible up date mechanism

On the topic of a software update, both solutions used by the analyzed router as well as many implementations encountered elsewhere, appear to be severely lacking when compared to common solutions used in mobile, desktop, or server systems. The majority of currently available embedded devices use a relatively bare-bones solution where the entire system is completely shutdown, completely reinstalled and started up once again. While this approach may not be surprising considering the hardware limitations of typical embedded devices, it may also be considered to be one of the major problems which prevent proper security maintenance of such devices. To make matters worse, this procedure usually requires the user to either download and upload the firmware themselves, or at least manually trigger an on-device version update check and install.

A much more user-friendly solution can be observed in systems like Android, where instead of reflashing the entire system, updates are generally applied only to specific components. The major benefit of this solution is that only a relatively small portion of software updates, in particular only those involving kernel or boot loader update actually require a full system restart. Most user level applications can instead be updated during the systems normal operation and restarted often without any noticeable downtime. If the update does not modify any functionality in major way but only fix edge cases of existing behavior, such as with most security patches, it may additionally, much like in case of Android, be performed completely automatically without major risk or interruptions.

6 DEFENSE AGAINST ATTACKS

Given the expected longevity of many embedded systems as well as the previously described major increase of functionality due to increasing popularity of IoT solutions it is worth to research not only how to address existing security threats but also to mention the ways of protecting or minimizing the effect of inevitable future vulnerabilities.

This defensive approach was analyzed in scope of the article titled *A systems and control perspective of CPS security* (Dibaji et al., 2019) where methods of defense were classified into three categories: prevention, resilience as well as detection and isolation.

The goal of prevention is to guard against disclosure attacks usually by means of randomization or cryptography. The importance of prevention is, of course, dependent by the amount of confidential in-formation that the device handles. One example of a prevention mechanism was already mentioned in the previous section, where a scenario of an attacker obtaining an encrypted configuration file and using it to learn the administrative credentials could have been prevented by either randomizing the key per device or simply hashing the password using a secure one-way hash algorithm. This category can of course gain much bigger importance in certain IoT solutions. For example Internet-accessible IP cameras should most definitely utilize strong encryption both during transmission of their capture as well as its remote storage to mitigate the possibility of massive breaches of their owners' privacy.

The next category - resilience - encompasses all mechanisms used to minimize the impact of an attack and continue operating as close to normal as possible. This particular category is worth exploring in depth because it is applicable to most of the embedded systems and yet during the previously carried out experiments has been found mostly unaddressed.

As a reminder, the majority of functionality of the inspected router like the Web configuration interface, multiple diagnostic tools, software update and many others was contained within a single executable running completely unconstrained under the super user account. This type of design leads to several problems, since it effectively bypasses most security features offered by unix- like operating systems such as Linux. The single privileged executable meant that any directory traversal attack can, for example, enable the adversary to inspect the contents of both/etc/passwd as well as the normally restricted/etc/ shadow data bases.

More importantly this approach enables any HTTP server related vulnerability to open a reverse shell, or trigger the firmware flashing update function. In a scenario like this the device's firmware could be permanently altered using a malicious attacker- provided binary. This would be possible even in cases when the update process verifies software signature, since the signature checking is only performed by the same, vulnerable, process.

To protect against those types of vulnerabilities, modularization of the system's software is strongly encouraged. This approach takes advantage of the fact that unlike desktop or server systems, the feature scope of a given embedded system is usually well known in advance and as a result, the entire execution environment might be designed around it.

This way, built-in features, like the address-space separation, may be used to isolate potentially high- risk

processes like the HTTP server and limit their capabilities, in this case only to serving Web content and occasionally triggering other applications using some IPC[11] mechanism.

In addition to the mentioned security benefits, separating monolithic embedded software into smaller components can also greatly help to address the previously mentioned software update issue. Purpose specific components which communicate only using well defined interfaces should be much easier to individually patch, test and update while also decreasing the need to fork specific functionality between all device-specific firmware releases.

An example of functionality which can easily enable this type of sandboxing are the *pledge* and *unveil* system calls found in Open BSD which allow to respectively limit the system calls available to a process as well as hide unwanted portions of the file system. If the process gets hijacked and attempts to access a blocked resource it would either receive an error or simply get killed by the operating system and most likely restarted by some supervisor. A similar functionality to that of *unveil* is currently being introduced to the Linux kernel inform of the *Landlock project*[12]. While akin to many other unix-like systems Linux introduces a vast scope of IPC mechanisms, it is important to mention that many of them may not be applicable to the constraints of embedded hardware. In general, it appears that common higher-level IPC systems like *D-Bus*, used extensively by the system d suite, can simply be too resource intensive on such systems. A possible solution may be offered in form of the Binder IPC system which is included with the Linux kernel used within Android.

If the range of functionality provided by the operating system is not large, it may also be worth to evaluate fitness of smaller and thus more efficient systems or kernels. In case the custom software is expected to cover most of system's features then using a real time kernel like *Free RTOS* might be a viable option.

As another possibility, a microkernel like *seL4* or the unix-like *QNX* may be used instead. Due to their architecture those systems have the benefit of a readily available highly performant message passing system and generally closely follow the suggested practice of composing the final software from a large number of isolated purpose-specific modules. This means that not only following such approach may be simpler when relying on micro kernels, but that the kernel itself should be easier to secure due to its severely limited attack surface.

The last of the three sections includes methods of detection as well as isolation. As the name suggests, the purpose of methods falling under this category is to identify attacks, isolate the threat and restore normal mode of operation. Out of all the mentioned categories, means of detection and isolation have most likely been explored the least in embedded space. This is unsurprising given the relatively high cost of anomaly detection and other continuous methods of threat analysis.

One possible solution is to conduct threat analysis of embedded devices by means of existing anti-malware software found on many desktop systems. Using this approach, the comparatively powerful desktop hardware would not only detect and isolate immediate threats but also gather and analyze data from other devices present on the network. The problem with this approach is that even though most embedded systems do collect some logs of user interaction and administrative access, it could be difficult to access this information outside of the device.

One example of such functionality was found during the exploration of the TP-Link router's firmware. In this case the device allows to send captured log data to a specified mail account by using an external SMTP server. Additionally, a configuration option exists, which enables the user to configure the log sending to occur periodically. This approach is certainly a step in the right direction and in cases of enterprise deployments in closed networks may be in fact sufficient. It does sadly possess some critical ease of usage problems for home users, considering that most Internet mail providers have disabled plain-text SMTP access as a security measure.

7 CONCLUSIONS

Our analysis performed on the several dozen of SOHO IoT devices with Linux OS revealed a number of vulnerabilities, weak factory defaults as well as usage of obsolete cryptographic algorithms and software. These practices are unfortunately not uncommon in commonly found embedded devices.

These problems often arise from the limitations imposed by the hardware on which those types of systems run. For example, when developing a system targeting a weak CPU and low on board memory, the decision to incorporate most of the functions exposed by the router into a single monolithic binary may at first glance seem quite beneficial, since this type of design allows for easy code reuse and entirely bypasses any IPC overhead.

Similarly, the need to control proprietary hard- ware components usually requires the vendor to implement their own device drivers. Because Linux kernel ABI is neither stable nor well suited for maintenance of external, closed kernel modules, many vendors simply deploy old kernel releases in new firmware binaries.

In practice, decisions like those make it rather difficult to maintain proper level of security in the soft- ware used by such devices. This is highly problematic due to the fact that the expected functionality of embedded devices usually remains unchanged for extensive periods of time, often long after their end-of-maintenance date. Going back to the example of routers, it is not uncommon to find systems remaining in use for ten or even more years, especially at homes. After all, as long as the hardware is functional without any issue and the software still works like expected there is very little incentive to replace them. To make matters worse, even if security patches for those devices are still released and available, they still may not be applied for long periods of time, since for the most part firmware updates have to be either performed or at least triggered manually.

The aforementioned model of device utilization indicates that, as opposed to many examples we analyzed, in particular embedded systems should be held to rather high standards of security. In particular, systems like SOHO IoT should be designed with longevity in mind to better match their usual life spans.

REFERENCES

AspenCore (2019).Embedded markets study - integrating iot and advanced technology de- signs, application development & processing environments. https://www.embedded.com/wp-content/uploads/2019/11/EETimes_ Embedded_2019_ Embedded_Markets_Study. pdf.

Chickowski, E. (2016).Hacker 2016 To-Do List: Botnet All The Things! https://www. dark reading.com/iot/hacker-2016-to- do-list-botnet-all-the-things-.

CISA (2021) .ICS Medical Advisory (ICSMA-21- 161-01). https://www.cisa.gov/uscert/ics/advisories/icsma-21-161-01.

daCosta, L.T., Barros, J.P., and Tavares, M.(2019). Vulnerabilities in iot devices for smarthome environment. In *ICISSP*.

Dibaji, S. M., Pirani, M., Flamholz, D. B., An- naswamy, A. M., Johansson, K. H., and Chakrabortty, A.(2019). A systems and control perspective of cps security. *Annual Reviews in Control*, 47:394–411.

Herold, R. (2021).Suggestions for consumers using iot products containing log4j.https:// privacy security brainiacs.com/privacy- professor-blog/Log4j/.

IBM X-Force (2022). IBM - X-Force Threat Intelligence Index 2022.https://www.ibm.com/downloads/cas/ADLMYLAZ.

Starks, T. (2021).Cisa warns 'most serious' log4j vulnerability likely to affect hundreds of millions ofdevices.https://www.cyberscoop.com/log4j-cisa-easterly-most-serious/.

Footnotes

[1]This is the preprint of the paper published in the Journal of Computer Fraud & Security Volume 2023, Issue 11 under DOI 10.12968/S1361-3723(23)70053-5

[2]https://blog.cloudflare.com/ddos-attack-trends-for-2022-q1/

[3]https://owasp.org/www-project-internet-of-things/

[4]https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[5]https://www.statista.com/outlook/dmo/smart-home/worldwide

[6]https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/

[7]SmallOffice/HomeOffice

[8]Application-SpecificIntegratedCircuit

[9]ApplicationBinaryInterface

[10]https://www.osinttechniques.com

[11]Inter-ProcessCommunication

[12]https://landlock.io/