Article

Automatic Symbol Resolution on Embedded Platforms by the Example of Smart TV Device

Michał Majchrowicz ^{*,†} and Piotr Duch [†]

Institute of Applied Computer Science, Łódź University of Technology, ul. Stefanowskiego 18/22, 90-537 Łódź, Poland; pduch@iis.p.lodz.pl

* Correspondence: mmajchr@iis.p.lodz.pl

† These authors contributed equally to this work.

Abstract: Smart TV devices are gaining increasingly more popularity. Due to their nature, Smart TVs can access a lot of sensitive data. This is one of the reasons why the Smart TV has become a popular target of hacking recently. Manufacturers try to make such attacks more difficult, and one of the methods they use is the removal of symbols from the firmware. In principle, this would prevent or significantly hinder the preparation of malwares or homebrew that could run on different firmware versions. This article is focused on developing algorithms for automatic symbol resolution. We proposed two automatic symbol resolution methods designed for Smart TVs. Presented methods were tested on the firmwares of the most popular Smart TV manufacturers', Samsung and LG, devices. Furthermore, an original framework is presented, which automatically locates the desired function in the binaries based on characteristic strings used in or near searched function. The developed framework is commonly used by homebrew developers (e.g., SamyGO) and releases developers from hardcoding function's addresses for different firmwares.

Citation: Majchrowicz, M.; Duch, P. Automatic Symbol Resolution on Embedded Platforms by the Example of Smart TV Device. *Appl. Sci.* **2021**, *11*, 3674. <https://doi.org/10.3390/app11083674>

Academic Editor: Juan Francisco De Paz Santana

Received: 20 March 2021

Accepted: 13 April 2021

Published: 19 April 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: symbol resolution; Smart TV; security; embedded systems; malware

1. Introduction

We are living in the Internet of Things era, where every day we find new applications for IoT devices [1,2] and take advantages of remote [3] and distributed systems [4]. Therefore, it is not surprising that many of the devices we use every day are connected to the global network. Moreover, the security of technologies used as the base of IoT devices [5–10] are not well developed [11], which, in addition to the fact that those devices have access to a lot of our private data, causes real problems for users [12]. Between 5 and 10 years ago, we mostly cared about the security of our PCs, and the security of mobile devices was a new thing. Now, most people are aware of viruses on such platforms and know that their personal data, banking details, etc. are at risk. However, people are still not informed about the danger of using embedded devices, which are vulnerable to attacks like physical tampering, malware, or hackers taking control of them.

A good example of devices vulnerable to such attacks is modern Smart TVs, which are connected to the Internet and public networks so that they can be easily exposed to cyberattacks [13]. Smart TVs are becoming increasingly widespread, increasing from 45 percent of total TV shipments in 2015 to 64 percent in 2017 [14]. Shipment of Smart TV units is continuously growing, and in 2017 it reached 244.4 million units [14]. The systematic increase in the popularity of Smart TV means that hackers are more interested in them [15], as is the case with phones [16] and computers [17], where the most popular models/systems are the most vulnerable to attacks. Modern TVs are equipped with Internet access and a full operating system, and the newest one often has a camera and microphone. Nevertheless, when people find out that their devices are more than simple TVs and that somebody might take advantage of that to spy on them, the situation changes—as people are more concerned about their privacy than security itself [18]. To quote a study conducted

in 2020 [19], “70% of people believed data privacy is important, but only 1 in 3 actually worry about their technology being hacked” [19].

Hacking Smart TVs has some advantages over hacking smartphones: there is no problem with power, which means that everything that a camera can see or a microphone can hear might be recorded 24 h per day and may be sent to the hijacker [20,21]. Furthermore, the pictures taken by the Smart TV’s camera are always good quality because the TV cannot move. The additional problem is that some Smart TVs, i.e., Samsung TVs, do not use light-emitting diodes (LEDs) to notify whether the camera is turned on or not. In consequence, the security of Smart TVs is becoming an issue that is a subject of many scientific studies [14,22–25].

In Smart TVs, the firmware is an operating system that manages all hardware resources and provides standard services to the TV’s applications. The firmware provided by the manufacturers is mainly encrypted, but in some cases, methods used for encryption are insufficient. For example, in the case of Samsung TVs, there are already tools for decryption and, in some cases, even encryption of the Samsung TV’s firmwares [26]. Another way to increase the Smart TV’s security against an unauthorized modification of the firmware was to eliminate all symbols as much as possible, especially those that may be used in reverse engineering [27].

1.1. Area of Research

This paper focuses on testing whether removing symbols from the firmwares of embedded devices can increase their security level. The proposed methods (Sections 5.1 and 5.2) and developed framework (Section 5.3) for automatic symbol resolution will be presented and tested on two examples of Smart TV brands: LG and Samsung. Those two brands were chosen because of their popularity on the market. Right now they both have almost 50% of the Smart TV market share (<http://english.etnews.com/20180220200004>, accessed on 13 April 2021). According to the Wakefield study [28], in 2019 Samsung and LG had a 45% share of the Smart TV market in the USA (Figure 1), which makes them perfect targets for hackers [15–17]. We did not consider researching Vizio as it is mostly popular only in the US and its market share in the EU or other parts of the world is marginal. Moreover, even in the US, its market share reduced by almost 60% in the last 3 years [29]. If we analyze the world Smart TV market share, Sony is the only other manufacturer worth mentioning. However, it is constantly losing its market share to cheap Chinese devices produced at the lowest possible price, which often do not have many security mechanisms. For example, recently, researchers have found multiple security vulnerabilities [30] in devices produced by TCL, a leading Chinese Smart TV manufacturer [29]. Some researchers even suggested that the lack of security mechanisms in those devices is so severe that users should unplug them from the Internet immediately [31]. Furthermore, recently there have been many security vulnerabilities found in Sony Smart TVs [32,33]. These were the reasons we have decided to focus our research on the biggest players in the Smart TV market: LG and Samsung. To present the results of the proposed method, Smart TVs equipped with Netcast (LG) and Orsay (Samsung) operating systems were used.

1.2. Preliminary Analysis of Research Gap

Removing symbols from the firmware had a significant impact on homebrew development, especially when preparing code to work on different firmware versions. In such cases, developers were hardcoding addresses of functions, but that is not a good solution. There can be a slight difference between the same function’s address in two different versions of firmware. After extensive studies of existing papers covering resolution of missing symbols (Section 2), we noticed that currently no framework could be used to automatize the process. At the same time, there is much work required to allow new firmware versions to be supported by existing tools. To overcome this problem, we present novel methods for symbol resolution based on characteristic binary or string values. On this basis, we will try

to test different approaches for recovering missing symbols and adapt them as a part of automated framework.

Figure 1. U.S. Smart TV market share [28].

2. Related Works

Smart TVs have caught the attention of the hacking community [26,34], who are modifying TV systems to allow users to take full advantage of device capabilities. The research conducted in 2012 about various Smart TV security implementations revealed that all of the tested vendors had vulnerabilities [35]. In [36], the authors presented that it is possible to build a system for remote parental control by hacking a Samsung Smart TV. In [37], this idea has been improved and expanded into remotely managed Kids Mode. There is an analysis of Samsung Smart TV devices, with examples of how particular components of TV firmware, applications, and web browsers can be easily attacked [38,39].

One of Smart TV standards is HbbTV (Hybrid Broadcast Broadband TV), which enables broadcasting stations to deliver additional interactive content through the Internet to the end users. The data transmitted between the broadcasting station and the HbbTV applications allow monitoring home network traffic to interfere in which TV programs the users are watching [23]. In [40], the author investigated what kinds of data are exchanged between HbbTV and the broadcasting stations. It was also noticed that privacy-sensitive information could be sent unencrypted. In [41], the author discovered that LG, in a similar way, automatically transmitted data to their servers. Transmitted information concerned not only the channels viewed, but also the names of the files that were being watched from external USB devices. What is more important, all of these data were sent unencrypted. Not only the private information about the watched files can be sent to the manufacturer, in the case of TVs equipped with microphones, eavesdropped private conversations were transmitted online to a specialized voice recognition provider [42]. A few years later, the CIA developed an attack that turns a TV into a listening device [43,44]. It was also found that certain Smart TVs send voice data to the third party services to get the result of voice recognition [45].

Modern Smart TVs are exposed to attacks by transmitting malware through a radio signal [46]. In such cases, there is no need to have any physical access to the attacked TVs. Additionally, the attacker did not leave any traces of their identity, like IP address or Domain Name System (DNS) transactions. In [47], the authors described different

attacking scenarios that use two public area networks: the Asymmetric Digital Subscriber Line (ADSL) network and the Digital Video Broadcasting (DVB) network. In one of the experiments, the authors replaced one of the video streams with their live webcam feed. In [48], the researchers described an attack against a popular Samsung TV model that exploits the movie player feature. During the playback on the attacked Smart TV, a corrupted video file can give the remote attacker complete control over the TV.

Another way to attack Smart TVs is through their firmware. Analysis of the firmwares may help discover possible essential security flaws. Some studies presented, in detail, how to capture a firmware upgrade by emulating the Device Firmware Upgrade (DFU) devices [49]. In [47], the authors were able to get a live Secure Shell (SSH) connection with the Smart TV by exploiting some of the firmware's vulnerabilities. The online firmware upgrade achieved by impersonating Samsung's update servers was described in [50]. Those and many other examples [14,51–54] clearly illustrate the rising importance of security on Smart TV devices.

In the case of firmwares with removed symbols, the hackers of Samsung's Smart TVs were hardcoding addresses of used functions, as there was no reliable way to resolve their names. One of the reasons for that was that though using strings or static values to resolve unexported symbols was a known technique on regular computers on both Windows and Linux systems (<https://reverseengineering.stackexchange.com/questions/18676/how-can-i-access-an-internal-dll-function-or-piece-of-data-externally>, accessed on 13 April 2021) [55,56], they were never used in the Smart TVs. Moreover, even on PCs, those techniques were not commonly used, as on both Linux and Windows most of the Application Programming Interface (API) is available through dynamic libraries (which have exported symbols) or by specific kernel *syscalls*. On SmartTVs there is none of that (at least in terms of device specific APIs), as most of the TV's functionality is controlled using private APIs which often lack symbols. In [57], the author presented a method for finding the address of an unexported function by matching hardcoded assembler instructions. Debugged messages and string literals were used to find function names on Windows 8.1 in [58].

3. Contribution

This paper focuses on retrieving the information about symbols removed from firmwares by Smart TV manufacturers. During the conducted studies, the authors focused on two systems: Orsay, used in Samsung TV, and Netcast, used in LG. The choice of these systems was not accidental—in these two cases, companies have removed the names of the functions from the binary file to prevent using firmware for hacking [27]. Furthermore, Samsung and LG cover almost 50% of the Smart TV market and naturally hackers are interested in breaking the security systems of the most popular devices instead of attacking vendors who supply only a small percentage of the market [15–17]. We evaluated two methods for automatic symbol resolution:

- Symbol resolution for Samsung firmware based on search of characteristic binary values in previous firmwares (Section 5.2).
- New framework for symbol resolution on a Samsung TV, based on locating string references (Section 5.3).

Note that all of the methods for working with software with removed symbols mentioned in the last paragraph of the related works section were tailored down for specific use cases and could be applied only for single cases and cannot be generalized for usage in the whole OS. Moreover, it has to be stressed that those methods cannot be directly applied to IoT devices. The most popular OSs use hundreds of small dynamically loaded libraries that make localizing specific functions much easier. On the Samsung Orsay Smart TV however, there are big blob binaries (exeDSP or exeAPP/exeTV in newer devices) that have thousands of private functions and are between 100 and 200 megabytes in size. At the same time, the processing power of Smart TVs is much lower in comparison to modern computers and, as a result, a need for custom solutions arose.

4. Background

In this section, the background information about the firmware security mechanisms used by Smart TV software vendors and operating systems of the selected devices is provided.

4.1. Firmware Security

Software vendors try to secure their firmwares in order to make it more difficult or even impossible to use them during the attacks on Smart TVs. Firmware can be obtained in two ways: the first one is to dump it from the hacked device, and the second method is based on using publicly available tools that already have encryption keys extracted from hacked devices and can be used to unpack update files provided by the vendors on their websites. The four most popular methods for securing firmwares are presented and shortly discussed.

- Firmware signing is one of the ways of protecting Smart TV devices before installing corrupted firmwares. This feature is implemented by the software vendor, who signs the firmware image with a private key, kept in secret. A device with such a feature enabled will first validate the firmware before accepting its installation. In case of detection compromised firmware integrity, the device will reject its upgrade or installation. This can be omitted by turning off the feature that checks firmware signing or uploading a rootkit when the firmware is already run by the system [59].
- Firmware encryption—different vendors use different types for firmware encryption. In the case of Samsung's devices, firmwares can be encrypted using multiple algorithms. There are already available tools for decryption of such firmwares (https://wiki.samygo.tv/index.php?title=Playing_with_Firmware_Images, accessed on 13 April 2021).
- Firmware data mangling is an example of applying the approach “security by obscurity”. In this case, one or more base functions can be generated and blended into the existing code. Such an approach may mean that it will be more difficult or even impossible for the attackers to distinguish the base function's codes from the codes of the firmware [60].
- Removing symbols from firmware is another example of the “security by obscurity” approach. In this case, software vendors remove symbols (Figure 2) from the final version of the firmware [27]. In such case, if one wanted to hack the firmware, they would have to use hardcoded addresses instead of function symbols. This method is not a reliable one, because of possible differences with function's addresses between different firmwares [27] (Figure 3).

4.2. Symbols Removal

Stripping binary from symbols is a common way of obstructing reverse engineering. Binaries without symbols are harder to disassemble or reverse engineer. There are two sections responsible of storing information about symbols and their names: `.strtab` and `.symtab`. The first one stores strings containing symbolic names, and the second one stores symbol table. The symbol table is responsible for associating symbolic names with functions or variables in binary (by `Elf64_Sym` structures). The `Elf64_Sym` structures connect symbolic names with strings. In the case of stripped binaries, those two sections are often removed [61]. Symbols removal is often called symbol stripping and can be performed on object files before linking them or at the time of compilation. For example, to instruct gcc to remove symbols, it is necessary to pass `-s` as one of arguments [62].

4.3. Netcast Operating System (LG)

The Netcast OS runs Linux, which supports device drivers. The Netcast's architecture consists of four main layers: linux system, service engine, Netcast application framework, and applications. The service engine is based on WebKit.

On LG's Netcast operating system, most of the tasks are done by 22 MiB binary blob, which is stripped of many symbols, including private methods, global variables, etc. From conducted research, it appears that ~16,000 symbols were removed. The original binary still has ~44,000 symbols in place (Figure 2). As a result, it is possible to have ~36% more symbols at our disposal. Such a number can significantly impact exploits, patches, malware, or homebrew development, especially if it is expected to work on different firmware versions.

4.4. Orsay Operating System (Samsung)

The best example of similar behavior on Samsung devices was found in the C series TVs (2010 models) with most of the symbols removed. Noted that, similarly to LG, on Samsung's devices (B to E series) most of those functions are in 60 MiB in size binary named `exeDSP`. Samsung developers "eliminated all function names as much as possible which may be used in reverse engineering" [27]. As a result, their intentions are clear. In the end, from over 107 thousand symbols, only approximately 2 thousand symbols were left (Figure 2). When we look at the `exeDSP`, we clearly see that it is pretty hard even to try to start reverse engineering of such a big binary blob from which more than 98% of symbols were removed. Therefore, we can quickly conclude that developers were right and, at first glance, the reverser has no idea where to start. Usually, a quick fix for such an issue is hardcoding of the offsets for missing function symbols; however, this is not a complete solution for the problem. This has been illustrated in Figure 3, where it can be seen that if homebrew or malware cannot find offsets for firmware running on a device instead of following normal execution (green blocks), it has to abort execution (red block) due to missing symbols. Such a situation can occur if a hacker or homebrew developer did not have access to all firmware versions when development or a new version came out after the tool was developed. Note that in our research, we have identified 23 versions of C series firmware, ranging in versions from T-VALDEUC_0000.0 up to T-VALDEUC_3018.1. Though we did our best to identify every possible version, we still cannot be sure that this list is complete. Furthermore, note that Samsung did not remove symbols for the previous series and, which might sound surprising at first, for none of the future models.

Figure 2. Numbers of removed and available symbols and strings in LG and Samsung firmwares.

Figure 3. Simplified diagram illustrating the major flaw of using hardcoded offsets in order to access functions with missing symbols.

5. Materials and Methods

In the following sections, three different methods for automatic symbol resolution will be described. In the case of LG Smart TV firmware, the method used during automatic symbol resolution is similar to that used during standard reverse engineering procedures. We applied it for the first time to retrieve removed symbols from firmware. It should be noted that all previously developed methods use hardcoded offsets which require additional work for each new firmware version (Figure 3).

Methods used for symbol resolution on Samsung Smart TV firmwares were specially designed to work on giant binary blobs and on devices with limited computational capabilities.

5.1. Address to Symbol

The first method focuses on retrieving symbols from another file, which is used for generating bug reports. In the case of the LG Netcast operating system structure, it can be noticed that every RELEASE file, which is the main binary blob for this system, has a corresponding RELEASE.sym file. As it turns out, LG uses those files to resolve missing symbols and send function names instead of addresses as a part of automatically generated bug reports. The very same file can be used for finding addresses of missing symbols.

The RELEASE.sym file maps the signatures with addresses for the given firmware. Therefore, in this case, it is possible to get the names of all functions removed from the RELEASE file. In order to do that, the first step is to load the RELEASE into the disassembler and then using the prepared script, one can load the RELEASE.sym file into the memory

and retrieve all removed symbols. Now, the function symbols can be used instead of hardcoded addresses. As a result, instead of calling the `dlsym` function to get the address of exported symbol, the `symfile_addr_by_name` function can be called, which can be found in `symfile.c` (<https://github.com/openlgtv/epk2extract/blob/master/src/symfile.c>, accessed on 13 April 2021) and `symfile.h` from the `epk2extract` project. There is even a convenience method `sym2addr` (<https://github.com/openlgtv/OPENRELEASE/blob/master/libopenrelease/util.c>, accessed on 13 April 2021) that provides compatibility with `dlsym` (so it makes the code more general) (Listing 1). The written code will be independent of the version of TV firmware; function addresses may vary between different versions of the firmware.

Listing 1: Code of the function that returns the address where the symbol name is loaded into memory.

```
void *sym2addr(const char *name)
{
    void *addr = (void *)symfile_addr_by_name(name);

    if (addr == NULL)
        addr = dlsym(RTLD_NEXT, name);

    return addr;
}
```

As can be seen, the whole idea of removing the symbols in the first place and providing a `RELEASE.sym` file with the same symbols did not make much of a difference in terms of system security.

5.2. Distinctive Functions

The same method could not be used with Samsung TVs as the vendor does not provide any symbol file with the firmware, and as a consequence, a different approach is needed. In the case of Samsung firmwares, most of the reverse engineering was done as a part of SamyGO project [26], which started around 2009 when B series TVs were out in the market. Developers were not only able to reverse engineer a lot of security mechanisms, but even have provided custom firmwares as well as extracted security keys. Therefore, it should not be a surprise that removing symbols was considered as a good idea to stop or at least slow down reverse engineering of C series firmware [27]. The main flaw of this idea was that during the preparation of new firmwares, developers used the old code. Therefore, the new firmware with removed symbols was based on the previous series, and the previous versions of the firmware could be used to retrieve removed symbols.

This can be illustrated in the example of TVs movie player seek patch. There was already a developed version for the B series (<https://forum.samygo.tv/viewtopic.php?t=1270>, accessed on 13 April 2021), which was based on patching seek values in function `CWProVideoPlayer::ProcessDirectionKey`. Unfortunately, the same approach cannot be directly applied to C series as there is no information where this method is located because of the removed symbols. However, through the function content analysis, it is possible to find where the same function is located on the C series firmware.

In the analyzed case, the most interesting part of the function are two specific places with default seek values for *down* and *up* keys, see Listings 2 and 3, respectively.

Listing 2: Part of the function `CWProVideoPlayer::ProcessDirectionKey` which sets the value `0xCE800000` when the *down* key is pressed.

```
LDR    R2, [R6,#0xC]      ; onDown
LDR    R4, =0xCE800000    ; float -1073741824.00
MOV    R0, R2
```


Listing 3: Part of the function `CWProVideoPlayer::ProcessDirectionKey` which sets the value `0x4D4CCCCD` when the *up* key is pressed.

```
LDR    R2, [R6,#0xC]      ; onUp
LDR    R4, =0x4D4CCCCD    ; float 214748368.00
MOV    R0, R2
```

Those are simply huge positive and negative float values, which as a result cause the TV to jump to the beginning or end of the movie on key down and key up. At this point, symbols do not matter anymore as it can be safely assumed that the same or at least similar code was used on the C series. A quick binary search for `0x4D4CCCCD` gets us to the patch location (Listing 4).

Listing 4: Code in C series which sets the value `0x4D4CCCCD` when the *up* key is pressed.

```
LDR    R3, [R5,#8]        ; onUp
LDR    R4, =0x4D4CCCCD    ; float 214748368.00
MOV    R0, R3
```

As different firmware versions are often completed with the same or at least similar compiler, it is also possible to locate missing symbols by searching for specific assembler instruction sets. The approach described above has been implemented in `find_func_by_binary_data`. The function with a removed symbol is localized based on characteristic, binary values, or a set of assembler instructions unique for specific functions (signatures).

5.3. Framework Description

In most cases, much better results may be obtained by trying to locate string references. Therefore, the authors have developed an automatic symbol resolution framework consisting of the following main functions:

- `find_func_by_string` - localizes the function by specific string.
- `find_nth_func_by_string` - localizes the function which is presented in assembly below or above the function with specific string.
- `find_nth_func` - localizes the function present below or above specific function.
- `find_nth_func_from_export` - localizes function which is present in assembly below or above the function with specific exported symbol.
- `find_function_start` - localizes the function's entry point.

Using the mentioned above set of functions allows for a reliable and firmware version-independent symbol resolution, as shown on the example in Listing 5.

Listing 5: The code of the function with characteristic error message.

```
.text:01647AB4 loc_1647AB4 ; CODE XREF: sub_164790C+
    B0 j
.text:01647AB4 MOV R0, #3
.text:01647AB8 MOV R1, R0
.text:01647ABC LDR R2, =aErrorCmcaption ;
"[ERROR] CmCaptionWnd::Create() Failed !" ...
.text:01647AC0 LDR R3, =sub_146D1C4
.text:01647AC4 BLX R3 ; sub_146D1C4
.text:01647AC8 MOV R1, R4
.text:01647ACC B loc_1647A84
```

As can be seen, this debug message is pretty specific and even in this case gives the original function name, but it can be any string reference unique enough to locate a function reliably. Based on this information, we can get an idea of a generic algorithm for finding missing symbols. We start by finding the closest exported symbol, as we need to know where to look from. After that, we apply some offset shift; as exported symbols are pretty rare, we have to add or subtract some predefined constant value. From that point, we jump by 4 bytes, which is the size of a single ARM instruction, and look for LDR. Once a string reference is found, it can be used in simple string comparison in order to determine whether it is the symbol we were looking for or not. This generic idea is realized by the

following API (<https://forum.samygo.tv/viewtopic.php?f=12&t=7350>, accessed on 13 April 2021) shown in Listing 6.

Listing 6: An example of a function call that searches for the error message in C series firmware.

```
caption_addr=find_func_by_string(pid, symtab, "CRYPTHW_SetIV",
"[ERROR] CmCaptionWnd::Create() Failed !!!", F_SEEK_DOWN, -0x30000);
```

The function takes a process id (pid), in this case it will be a exeDSP process' pid (the one that we want to get functions from so we need to inject the library into it), symtab—a structure holding information for the exported symbols, the name of the exported function which we want to start the search from (in this case we are starting with the CRYPTHW_SetIV function), the characteristic string we are trying to locate ([ERROR] CmCaptionWnd::Create() Failed !!!) which allows us to find the sought function, the direction of the search (either F_SEEK_DOWN or F_SEEK_UP), and the offset from the exported function from which we start the search; it can be positive or negative.

In some cases, we will not have any string references as part of our interest's function, though there is a pretty simple solution to such a situation. It is enough to find the nearest function with such reference and count the number of functions up or down. It is assumed that there is little difference in the function's location between different firmwares (<https://forum.samygo.tv/viewtopic.php?f=75&t=9038>, accessed on 13 April 2021) (Listing 7).

Listing 7: An example of a function call that searches for the previous function (−1) to export.

```
addr=find_nth_func_from_export(h, "_ZN14CCStateMachinel1t_FindStateEi", -1);
```

In this case, we pass the function handler to the process (h), the name of the exported function, and the number of functions that should be skipped to find the desired function.

Moreover, this approach is especially successful, as in exeDSP binary, we are dealing with over 80,000 unique strings to which there are almost 280,000 references (Figure 2). The developed framework does not need to have a string reference in the exact function we are trying to resolve. Therefore, we can use this vast amount of string references as a unique identifier for specific functions. As a result, it is a highly reliable approach for automatic symbol resolution.

This process can even be automatized as there is a firmware version for the C series, called 0000 (<https://forum.samygo.tv/viewtopic.php?t=3318>, accessed on 13 April 2021). To be precise, such a thing exist probably for most if not all series, though it is kept secret by Samsung's developers, but in a few cases secrecy was not enough and there was a leak of such software. This kind of tool is developed for internal purposes so it is no surprise that the 0000 firmware for the C series did not have any symbols stripped. As a result, it is possible to develop a patch or malware for this version using provided symbols and add support for others by automatically going through binary, finding the nearest symbol available in the production firmwares, after that the nearest string reference and counting functions (<https://forum.samygo.tv/viewtopic.php?f=12&t=7548>, accessed on 13 April 2021). In most cases, symbol resolution on Smart TV platforms can be highly automatized by applying the developed framework and, as a result, removing symbols from firmwares is not a good solution for protecting against hackers or malware.

5.4. Experimental Setup

The developed framework was tested on a Samsung LE32C650 with *libSchedulePVR* library. The device itself had *SamyGO sysroot* installed. Various firmware versions were used for the tests; however, the development process itself was conducted on *T-VALDEUC_1016.0* firmware version. This device is running a variant of Linux OS called by Samsung Orsay. The firmware was decompiled, using a *gcc* compiler, in the first step. Next, using static analysis tools, it was possible to locate characteristic strings. We needed the offset between

the function with the characteristic string and the function which we wanted to use. Using this information, we prepared a new version of the *libSchedulePVR* library and compiled it with our framework. Finally, the *libSchedulePVR* library was run on the tested TV. To check the prepared library's compatibility with different versions of firmwares, we downloaded and installed on TV 23 versions of C series firmware (from T-VALDEUC_0000.0 up to T-VALDEUC_3018.1). The prepared library was working with all of those firmwares.

6. Results

As a result of our research, we have developed and extensively tested two methods for resolving removed symbols. One is based on finding characteristic binary values (Section 5.2) and the other is based on using dynamic localization of missing symbols based on string references (Section 5.3). As the second method proved to be much more reliable, we chose it as the basis for the new framework for symbol resolution on a Samsung Smart TV.

To create malware or homebrew, one has to know which functions are available and which functions can be used. Therefore, the first step is to check binary for interesting functions. Then, knowing the function symbol, it is easier to proceed. However, in some firmwares (e.g., Samsung firmwares for series C) where symbols were removed, it is impossible to use the function name. Instead, one has to hardcode the function address, although it is only a temporary solution. In different versions of firmware functions, the address may differ. Using the framework presented in the article, a developer has to find a unique string reference, which is in the function they needed, or near it (because there is a slight difference in functions order between different firmwares), and localizes it in the used firmware. This approach solves a problem with different versions of firmware and also makes developed homebrew more universal.

The prepared framework is a novelty in IoT security. It makes it much easier to resolve symbols in binaries of 200 megabytes or more in size that have thousands of unexported functions. By calling simple API (`find_func_by_string`), an attacker can deal with such a huge amount of binary data on a Smart TV and resolve all necessary functions without a negative effect on the device performance, which makes writing reliable exploits much easier (Figure 4). We have prepared a summary of different methods (Table 1) of resolving missing symbols in comparison to the developed framework. To further illustrate the differences between different methods, we have used bold font to highlight those aspects that are advantageous (take less time, allow for broader support, etc.). We used the same method to highlight methods that pass all criteria, which at the moment is only provided by the developed framework (Table 1).

One of the possible attack scenarios using the presented framework may look as follows: The hacker may download the newest version of the firmware from the software vendor's website. Although many firmwares are encrypted, there are already available tools for firmware decryption. The problem of signed software can be overcome by turning off the options for checking whether the firmware is corrupted or not. When signing cannot be disabled, it is still possible to load shellcode into an already running application (like a web browser) and execute it using some kind of vulnerability that completely bypasses all kinds of security mechanisms.

Moreover, thanks to the application of the developed framework, there is no need to modify the application in any way to support different firmware versions (Figure 4). Everything is handled internally, and the hacker or homebrew developer does not have to worry about missing symbols anymore. That is why stripping symbols is not a security but an obscurity, which cannot stop hackers and only slows down the homebrew community from extending the device capabilities. For example, before an automatic symbol resolution framework has been developed, projects available at the SamyGO forum were utilizing hardcoded addresses of functions which often made those tools unreliable (Figure 3).

Figure 4. Simplified activity diagram illustrating the execution of an application utilizing developed framework and its advantages over using hardcoded offsets in order to access functions with missing symbols.

Table 1. Comparison of other methods and developed framework. Bold font illustrates preferable answer.

Method of Resolving Missings Symbols	Does Method Work across Different Firmware Versions (Including Future Ones)	Does Method Require Changes in Code and Recompilation in Order to Support New Firmwares	Is Method Capable of Recovering All of Missing Symbols
Hardcoding functions off-sets	No	Yes	Yes
Using relative offset to known symbols	No	Yes	No
Finding functions with specific binary values	Yes	No	No
Finding functions with specific string references	Yes	No	No
Providing symbol files for known fw versions	No	Yes	Yes
Developed framework	Yes	No	Yes

Listing 8: Only modification necessary in `util.h` file to support C series.

```

STATIC int samyGO_whacky_t_init(void *h, void *paramCTX, uint32_t cnt)
{
    samyGO_CTX_t *ctx;
    ctx=paramCTX;
    for(int i = 0; i < cnt ; i++)
    {
        if(!ctx->procs[i])
            continue;

        void *fn = dlsym(h, ctx->procs[i]);
        <         if(!fn)
        ----
        >         if(!fn && !(fn=C_find(h,ctx->procs[i])))
        log("dlsym '%s' failed.\n", ctx->procs[i]);
        else
        log("%s [%p].\n", ctx->procs[i], fn);
        ctx->procs[i] = fn;
    }
    return 0;
}

```

Another example of an application of the developed framework, that is not harmful to the end users, is homebrew programs. An example of such a project is SamyGO libSchedulePVR (<http://forum.samygo.tv/viewtopic.php?f=75&t=9038>, accessed on 13 April 2021). This project can be used to remotely schedule and control the Personal Video Recorder (PVR) on the TV. In the beginning, the project was prepared to work on almost all Samsung TV versions except for C, as it lacked symbols for many necessary functions. Using the prepared framework, it was possible to make the project available for C series TVs users. It was accomplished by adding to the project three framework files (`C_find.c`, `C_support.c`, and `C_support.h`) and modifying one line (adding a call to `C_find` function) in file `util.h` (Listing 8).

Tests of Developed Framework

We have tried coding the same library using different methods for resolving missing symbols and comparing how long it will take to support different numbers of firmware versions (Figure 5). There are only two methods worth comparing to: hardcoding function offsets and providing symbols files. Any other method does not work for all functions and any firmware version (Table 1). However, symbol files are very rarely used, as the amount of work required for implementing them is almost the same as in case of hardcoding function offsets, but at the same time the developer must also generate them, which gives unnecessary overhead. This is the reason why coders very rarely use them. In our test, when trying to support a single firmware version, implementing libSchedulePVR on C series by using hardcoded offsets (on average) took approximately 7 min, whereas when using yjr developed framework it took approximately 10 min (Figure 5). Moreover, the benefits of the developed framework become more visible when we compare the results for 5, 10, or 15 firmware versions (Figure 5) we want to support. On average, we needed approximately 4 min to support a new firmware version using the hardcoded offset. The exact number is not so important, as for more complex projects it will higher even by one or two orders of magnitude. At the same time binary compiled with developed framework did not require any additional work to support new firmware versions (Figure 5).

Figure 5. Numbers of minutes required to implement support for new firmware version(s).

7. Legality and Ethics of Conducted Research

An important part of our research was testing the security mechanisms themselves and finding a way to break or bypass them. As a result, the topic of legality and ethics of our research might be a concern. First, note that both under US [63] and EU [64] law modification of devices owned by the users is completely legal. Moreover, breaking or bypassing security mechanisms is the only way to test them [65] and check how secure they really are. It is important to remember that “assessing the security state is a continuous and necessary task to understand the risks there exist. This assessment is usually performed through security tests” [66]. It always comes to a simple rule: the chain is as strong as its weakest link [67]. Of course, after finding flaws it is important to follow the procedure commonly known as “responsible disclosure” [68–70]. In short terms, the procedure states that no information regarding a security flaw should be published before respective vendors have been informed and fix or patch released. Usually, it is common to wait around 90 days after such release. The flaws we describe in this paper relate to TV series released almost 10 years ago, though we were able to identify similar issues in models released in recent years. Therefore, we describe them only to illustrate the problem and highlight the importance of applying solutions we suggest, as those can be applied to fix issues often found in modern devices. As a result, those specific flaws serve only demonstration purposes and do not pose a severe risk to the user. At the same time, as it is pretty common for people to use TVs for 10 or more years (sometimes not updating their firmwares at all), we will not present details about our recent findings as it might endanger users. Therefore, through this publication we hope to raise awareness about security issues in modern Smart TVs and hopefully increase the percentage of people who regularly update their firmware.

8. Discussion

In this article, the automatic symbol resolution methods on embedded platforms by the example of Smart TV devices were presented. Those methods were tested on two operating systems installed on the TVs of two major Smart TVs manufacturers: LG and Samsung (Figure 1). Both manufacturers in recent years have changed their operating systems. In the case of LG, it was a switch from Netcast to WebOS, whereas in the case of Samsung, it was a switch from Orsay to Tizen operating system. In order to follow the principle of responsible disclosure and not endanger users’ security in a meaningful

manner (Section 7), we described issues found in old operating systems used by both brands (Netcast and Orsay). However, note that in both cases, the problems we illustrated as well as techniques used for automation of symbol resolution can also be adapted for usage with the newest operating systems provided by both vendors.

We could obtain the correct address for almost every function we tried to localize during our research using manual analysis. Moreover, even if we could not find anything characteristic for the specific function, we could still use the closest one that meets our criteria and use it as an “anchor point” to resolve the function name that we were interested in.

However, as checking 107,000 functions manually is not a proper approach, we have generated distinct signatures to function from one of the earlier firmware versions and labeled them using the func_NUM schema. After that, we applied those signatures to one of the newest firmware versions, and though it was very rough and only a proof of concept, we still were able to resolve almost 31 thousand functions names. Quick inspection for random functions that we could not localize automatically proved that a lot of them can be resolved by manually adjusting the search parameters passed to the framework. Therefore, we believe that by application of our automatic framework resolution of most symbols is possible. Moreover, this framework has been successfully applied in many SamyGO projects without any issues.

Significance of Further Research in this Field and the Current State of the Industry

Legal and ethical issues (Section 7) were an essential factor that we had to consider while working on this paper. Especially because, as we explained in (Section 7), it is widespread for people to use their Smart TVs for more than 10 years, often without updating them regularly. Specific issues we described are no longer present in recent versions of devices produced by both manufacturers. We believe that the multiple reports we provided and our close cooperation with Smart TV manufacturers positively impacted Smart TV devices, as their security level increased significantly over the years. Nevertheless, the problem of removing symbols is still applicable to recent models and proposed techniques can be used to solve it. However, the presented framework has to be adapted and cannot be used directly on those devices. Of course, we have also reported those new problems; however, due to bug report limitations (and ethical reasons we already explained), we cannot currently reveal more information about those specific issues. We can, however, explain that we are constantly trying to direct our research in the direction of improving the security of Smart TVs and other types of IoT devices. Note that security is never a closed topic and, even on PCs, where we have decades of experience with improving their security, there still can be a simple bug that can affect most of them [71]. IoT devices are still (at least in terms of security) an industry at its early stages and there is still a lot that has to be done to improve their security. Nevertheless, as we hope this paper illustrates, even on Smart TVs, there are still issues that need to be addressed. We also hope that by this research, we will be able to convince users to perform more regular updates of their devices and at the same time make the details of our work accessible to other Smart TV manufacturers that we were not able to check personally.

Author Contributions: Conceptualization, M.M.; methodology, M.M.; software, M.M.; data curation, P.D.; writing—original draft preparation, P.D.; writing—review and editing, P.D. All authors have read and agreed to the published version of the manuscript.

Funding: Part of the presented work describing Samsung Smart TV issues is the result of scientific research conducted as part of cooperation with Samsung Electronics Co., Ltd., whose aim was to increase the level of security of the products developed by the company. This work was financed by the Lodz University of Technology, Faculty of Electrical, Electronic, Computer and Control Engineering as a part of statutory activity (project no.501/12-24-1-39).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Duch, P.; Jaworski, T. Enriching Computer Science Programming Classes with Arduino Game Development. In Proceedings of the 11th International Conference on Human System Interaction (HSI), Gdansk, Poland, 4–6 July 2018; pp. 148–154. [CrossRef]
2. Perenc, I.; Jaworski, T.; Duch, P. Teaching programming using dedicated Arduino educational board. *Comput. Appl. Eng. Educ.* **2019**, *27*, 943–954. [CrossRef]
3. Duch, P.; Jaworski, T. Dante—Automated Assessments Tool for Students’ Programming Assignments. In Proceedings of the 11th International Conference on Human System Interaction (HSI), Gdansk, Poland, 4–6 July 2018; pp. 162–168. [CrossRef]
4. Majchrowicz, M.; Kapusta, P.; Jackowska-Strumiłło, L.; Banasiak, R.; Sankowski, D. Multi-GPU, multi-node algorithms for acceleration of image reconstruction in 3D Electrical Capacitance Tomography in heterogeneous distributed system. *Sensors* **2020**, *20*, 391. [CrossRef] [PubMed]
5. Papoutsakis, M.; Fysarakis, K.; Spanoudakis, G.; Ioannidis, S.; Koloutsou, K. Towards a Collection of Security and Privacy Patterns. *Appl. Sci.* **2021**, *11*, 1396. [CrossRef]
6. Tawalbeh, L.; Muheidat, F.; Tawalbeh, M.; Quwaider, M. IoT Privacy and security: Challenges and solutions. *Appl. Sci.* **2020**, *10*, 4102. [CrossRef]
7. Lee, J.; Choi, S.; Kim, D.; Choi, Y.; Sun, W. A Novel Hardware Security Architecture for IoT Device: PD-CRP (PUF Database and Challenge–Response Pair) Bloom Filter on Memristor-Based PUF. *Appl. Sci.* **2020**, *10*, 6692. [CrossRef]
8. Pieroni, A.; Scarpato, N.; Felli, L. Blockchain and IoT Convergence—A Systematic Survey on Technologies, Protocols and Security. *Appl. Sci.* **2020**, *10*, 6749. [CrossRef]
9. Horalek, J.; Sobeslav, V. Cybersecurity Analysis of IoT Networks. In *International Conference on Computational Collective Intelligence*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 488–499.
10. Shelupanov, A.; Evsyutin, O.; Konev, A.; Kostyuchenko, E.; Kruchinin, D.; Nikiforov, D. Information Security Methods—Modern Research Directions. *Symmetry* **2019**, *11*, 150. [CrossRef]
11. Fujdiak, R.; Mlynek, P.; Mrnustik, P.; Barabas, M.; Blazek, P.; Borcik, F.; Misurec, J. Managing the secure software development. In Proceedings of the 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Canary Islands, Spain, 24–26 June 2019; pp. 1–4.
12. Adat, V.; Gupta, B. Security in Internet of Things: issues, challenges, taxonomy, and architecture. *Telecommun. Syst.* **2018**, *67*, 423–441. [CrossRef]
13. Kang, S.; Kim, S. How to obtain common criteria certification of smart TV for home IoT security and reliability. *Symmetry* **2017**, *9*, 233. [CrossRef]
14. Alam, I.; Khusro, S.; Naeem, M. A review of smart TV: Past, present, and future. In Proceedings of the International Conference on Open Source Systems Technologies (ICOSST), Lahore, Pakistan, 18–20 December 2017; pp. 35–41. [CrossRef]
15. Goud, N. FBI issues Cyber Attack Warning against Smart TVs. 2020. Available online: <https://www.infosecurity-magazine.com/news/fbi-issues-smart-tv-cybersecurity/> (accessed on 13 April 2021).
16. Goud, N. Which Phone Protects Your Security the Best? We Asked the Experts. 2017. Available online: <https://time.com/4905774/which-phone-is-most-secure/> (accessed on 13 April 2021).
17. Grimes, R.A. Macs’ Low Popularity Keeps Them Safer from Hacking and Malware. 2009. Available online: <https://www.csoonline.com/article/2629811/mac--low-popularity-keeps-them-safer-from-hacking-and-malware.html> (accessed on 13 April 2021).
18. Ghiglieri, M.; Volkamer, M.; Renaud, K. Exploring consumers’ attitudes of smart TV related privacy risks. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 656–674.
19. ECU Concerns over Hacking and Data Privacy. 2020. Available online: <https://studyonline.ecu.edu.au/blog/concerns-over-hacking-and-data-privacy> (accessed on 13 April 2021).
20. Feamster, N. Outsourcing Home Network Security. In *ACM SIGCOMM Workshop on Home Networks*; HomeNets ’10; ACM: New York, NY, USA, 2010; pp. 37–42. [CrossRef]
21. Lee, S.; Kim, S. Hacking, Surveilling and Deceiving Victims on Smart TV. Blackhat USA. 2013. Available online: <https://docs.huihoo.com/blackhat/usa-2013/US-13-Lee-Hacking-Surveilling-and-Deceiving-Victims-on-Smart-TV-Slides.pdf> (accessed on 10 January 2020).
22. Arabo, A.; El-Mousa, F. Security framework for smart devices. In Proceedings of the International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), Kuala Lumpur, Malaysia, 26–28 June 2012; pp. 82–87. [CrossRef]
23. Ghiglieri, M.; Tews, E. A privacy protection system for HbbTV in Smart TVs. In Proceedings of the IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2014; pp. 357–362. [CrossRef]
24. Landau, S. What Was Samsung Thinking? *IEEE Secur. Priv.* **2015**, *13*, 3–4. [CrossRef]
25. Xin, W.; Suiyng, Y. The Security File System in Smart TV Operating System. *Inf. Secur. Comput. Fraud.* **2014**, *2*, 48–51. [CrossRef]
26. Altinyurt, E.U. SamyGO. Available online: <http://www.samygo.tv:samygo> (accessed on 30 September 2018).

27. Wikileaks. Samsung's Countermeasure Fixing Security Holes. Available online: https://wikileaks.org/sony/docs/07/Documents/@Deals/An%20Overview%20of%20Security%20Model%20applied%20on%20Samsung%202010%20TV_BDPs%20-%2020Oct%202010.pdf (accessed on 30 September 2018).
28. Kolnowski, T. Smart TVs Continue to Resonate with Global Consumers. 2019. Available online: <https://digitized.house/smart-tvs-continue-to-resonate-with-global-consumers/> (accessed on 13 April 2021).
29. Department, S.R. Share of Smart TVs by Manufacturer in U.S. Wi-Fi Households from 2017 to 2020. 2021. Available online: <https://www.statista.com/statistics/782217/smart-tv-share-by-oem-in-the-us/> (accessed on 13 April 2021).
30. Paul Roberts, S.L. Security Holes Opened Back Door to TCL Android Smart TVs. 2021. Available online: <https://securityledger.com/2020/11/security-holes-opened-back-door-to-tcl-android-smart-tvs/> (accessed on 13 April 2021).
31. Richi Jennings, S.B. Disconnect Your TCL Smart TV from the Internet—NOW. 2021. Available online: <https://securityboulevard.com/2020/11/disconnect-your-tcl-smart-tv-from-the-internet-now/> (accessed on 13 April 2021).
32. Loi, T.; Fortinet, I. The Sony Smart TV Exploit: An Inside View of Hijacking Your Living Room. 2021. Available online: <https://www.fortinet.com/blog/threat-research/sony-smart-tv-exploit-inside-view-hijacking-your-living-room> (accessed on 13 April 2021).
33. Tara Seals, T. Android-Based Sony Smart-TVs Open to Image Pilfering. 2021. Available online: <https://threatpost.com/android-smart-tvs/144133/> (accessed on 13 April 2021).
34. Openlgtv. Available online: http://openlgtv.org.ru/wiki/index.php/Wiki_index (accessed on 30 September 2018).
35. Kuipers, R.; Starck, E.; Heikkinen, H. Smart TV Hacking: Crash Testing Your Home Entertainment. 2012. Available online: <https://www.yumpu.com/en/document/view/33718312/smart-tv-hacking-crash-testing-your-home-codonomicon> (accessed on 13 April 2021).
36. Majchrowicz, M.; Kapusta, P.; Fastryjak, D.; Jackowska-Strumillo, L. System for remote parental control and management of rooted smart TVs. In Proceedings of the International Interdisciplinary PhD Workshop (IIPhDW), Swinoujscie, Poland, 9–12 May 2018; pp. 357–360. [CrossRef]
37. Majchrowicz, M.; Kapusta, P.; Jackowska-Strumillo, L. Application of Different Kinds of Interfaces in Modern Devices for Taking Care of People. In Proceedings of the 11th International Conference on Human System Interaction (HSI), Gdansk, Poland, 4–6 July 2018; pp. 207–213. [CrossRef]
38. Grattafiori, A.; Yavor, J. The outer limits: Hacking the Samsung Smart TV. Blackhat USA. 2013. Available online: <https://infocondb.org/con/black-hat/black-hat-usa-2013/the-outer-limits-hacking-the-samsung-smart-tv> (accessed on 13 April 2021).
39. Majchrowicz, M.; Kapusta, P. Rooting Smart TVs by exploiting design flaws in application framework. In Proceedings of the International Interdisciplinary PhD Workshop 2017, 2017; pp. 8–14.
40. Smit, L. What Does Your Television Know about You. 2015. Available online: <https://www.semanticscholar.org/paper/What-Does-Your-Television-Know-About-You-Smit/cce770c96595ff53ce15a483d0ff9b471f575d8b> (accessed on 13 April 2021).
41. DoctorBeet. LG Smart TVs Logging USB Filenames and Viewing info to LG Servers. 2013. Available online: <http://doctorbeet.blogspot.com/2013/11/lg-smart-tvs-logging-usb-filenames-and.html> (accessed on 13 April 2021).
42. Crossley, D. Samsung's Listening TV Is Proof That Tech Has Outpaced Our Rights. 2015. Available online: <https://www.theguardian.com/media-network/2015/feb/13/samsungs-listening-tv-tech-rights> (accessed on 13 April 2021).
43. Gross, G. WikiLeaks' CIA document dump shows agency can compromise Android TVs. *PC World* **2017**, *24*, 08131384.
44. Welt, N. Weeping Angel: The Latest Surveillance Tool, That Can Turn Your Smart TV into a Bug TV. 2017. Available online: <http://mastersofmedia.hum.uva.nl/blog/2017/09/25/weeping-angel-cia-bug-smart-tv/> (accessed on 13 April 2021).
45. Munro, K. Is Your Smart TV Listening to You?—Update. 2015. Available online: <https://www.pentestpartners.com/security-blog/is-your-samsung-tv-listening-to-you-update/> (accessed on 13 April 2021).
46. Oren, Y.; Keromytis, A.D. From the aether to the ethernet—Attacking the internet using broadcast digital television. In Proceedings of the 23rd {USENIX} Security Symposium ({USENIX} Security 14), San Diego, CA, USA, 20–22 August 2014; pp. 353–368.
47. Bachy, Y.; Basse, F.; Nicomette, V.; Alata, E.; Kaâniche, M.; Courrége, J.; Lukjanenko, P. Smart-TV Security Analysis: Practical Experiments. In Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, Brazil, 22–25 June 2015; pp. 497–504. [CrossRef]
48. Michéle, B.; Karpow, A. Watch and be watched: Compromising all Smart TV generations. In Proceedings of the IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2014; pp. 351–356. [CrossRef]
49. Travis, G. Emulating Usb Dfu to Capture Firmware. 2012. Available online: <http://travisgoodspeed.blogspot.com/2012/10/emulating-usb-dfu-to-capture-firmware.html> (accessed on 13 April 2021).
50. Sidiropoulos, N.; Stefopoulos, P. Smart tv hacking. *Res. Proj.* **2013**, *1*, 2012–2013.
51. Bachy, Y.; Nicomette, V.; Kaâniche, M.; Alata, E. Smart-TV security: risk analysis and experiments on Smart-TV communication channels. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 61–76. [CrossRef]
52. Chernyshev, M.; Hannay, P. Security assessment of IoT devices: The case of two smart TVs. In Proceedings of the 13th Australian Digital Forensics Conference, Perth, Western Australia, 30 November–2 December 2015; pp. 85–94.

53. Niemietz, M.; Somorovsky, J.; Mainka, C.; Schwenk, J. Not so Smart: On Smart TV Apps. In Proceedings of the 2015 International Workshop on Secure Internet of Things (SIoT), Vienna, Austria, 21–25 September 2015; pp. 72–81. [CrossRef]
54. Wang, X. Research Of Security Strengthen Tactics Based On Smart Tv Os. *J. Investig. Med.* **2015**, *63*, S32.
55. j00ru. Unexported SSDT Functions Finding Method. 2009. Available online: <https://j00ru.vexillum.org/2009/11/unexported-ssdt-functions-finding-method/> (accessed on 13 April 2021).
56. Suiche, M. Windows Vista 64bits and Unexported Kernel Symbols. 2007. Available online: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.174.2486&rep=rep1&type=pdf> (accessed on 13 April 2021).
57. Leeqwind. From CVE-2017-0263 to Windows Menu Management Component. 2018. Available online: <https://xiaodaozhi.com/exploit/117.html> (accessed on 13 April 2021).
58. Jurczyk, M. One Font Vulnerability to Rule Them All. 2015. Available online: <https://j00ru.vexillum.org/talks/recon-one-font-vulnerability-to-rule-them-all/> (accessed on 13 April 2021).
59. Leclercq, M. Firmware Authentication and Deciphering for Secure TV Receiver. U.S. Patent 9,177,152, 11 March 2015.
60. Johnson, H.; Gu, Y.X.; Wiener, M. Securing Accessible Systems Using Dynamic Data Mangling. U.S. Patent 9,698,973, 4 July 2017.
61. Andriesse, D. *Practical Binary Analysis: Build Your Own Linux Tools for Binary Instrumentation, Analysis, and Disassembly*; no starch press; 2018; pp. 31–55.
62. Free Software Foundation. *GCC Online Documentation*; Free Software Foundation: Boston, MA, USA, 2020.
63. Wired. U.S. Declares iPhone Jailbreaking Legal, Over Apple’s Objections. 2018. Available online: <https://www.wired.com/2010/07/feds-ok-iphone-jailbreaking/> (accessed on 13 April 2021).
64. ComputerWeekly. *Apple iPhone Jailbreaking Is ‘Okay under EU Law’*; ComputerWeekly, Available online: <https://www.computerweekly.com/news/1280093396/Apple-iPhone-jailbreaking-is-okay-under-EU-law> (accessed on 13 April 2021).
65. Bishop, M. About Penetration Testing. *IEEE Secur. Priv.* **2007**, *5*, 84–87. doi:10.1109/MSP.2007.159. [CrossRef]
66. Bertoglio, D.; Zorzo, A. Overview and open issues on penetration test. *J. Braz. Comput. Soc.* **2017**, *23*. [CrossRef]
67. Zhao, J.J.; Zhao, S.Y.; Zhao, S.Y. Opportunities and threats: A security assessment of state e-government websites. *Gov. Inf. Q.* **2010**, *27*, 49–56. [CrossRef]
68. Cavusoglu, H.; Cavusoglu, H.; Raghunathan, S. Efficiency of Vulnerability Disclosure Mechanisms to Disseminate Vulnerability Knowledge. *IEEE Trans. Softw. Eng.* **2007**, *33*, 171–185. [CrossRef]
69. Ding, A.Y.; De Jesus, G.L.; Janssen, M. *Ethical Hacking for Boosting IoT Vulnerability Management: A First Look into Bug Bounty Programs and Responsible Disclosure*; ICTRS ’19; Association for Computing Machinery: New York, NY, USA, 2019; pp. 49–55. [CrossRef]
70. Kinis, U. From Responsible Disclosure Policy (RDP) towards State Regulated Responsible Vulnerability Disclosure Procedure (hereinafter—RVDP): The Latvian approach. *Comput. Law Secur. Rev.* **2018**, *34*, 508–522. [CrossRef]
71. Qualsys. *Heap-Based Buffer Overflow in Sudo*; Available online: <https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-heap-based-overflow-sudo.txt> (accessed on 16 April 2021).