Scrutinizing WPA2 Password Generating Algorithms in Wireless Routers

Eduardo Novella Lorente
The Kerckhoffs Institute
Radboud University, The Netherlands.
ednolo@alumni.upv.es

Carlo Meijer
The Kerckhoffs Institute
Radboud University, The Netherlands.
carlo@youcontent.nl

Roel Verdult
Institute for Computing and Information Sciences
Radboud University, The Netherlands.
rverdult@cs.ru.nl

Abstract

A wireless router is a networking device that enables a user to set up a wireless connection to the Internet. A router can offer a secure channel by cryptographic means which provides authenticity and confidentiality. Nowadays, almost all routers use a secure channel by default that is based on Wi-Fi Protected Access II (WPA2). This is a security protocol which is believed not to be susceptible to practical key recovery attacks. However, the passwords should have sufficient entropy to avert brute-force attacks.

In this paper, we compose a strategy on how to reverse-engineer embedded routers. Furthermore, we describe a procedure that can instantly gather a complete wireless authentication trace which enables an offline password recovery attack. Finally, we present a number of use cases where we identify extremely weak password generating algorithms in various routers which are massively deployed in The Netherlands.

The algorithms are used to generate the default WPA2 password. Such a password is loaded during device initialization and hardware reset. Users that did not explicitly change their wireless password are most likely vulnerable to practical attacks which can recover their password within minutes. A stolen password allows an adversary to abuse someone else's internet connection, for instance compromising the firewall, making a fraudulent transaction or performing other criminal activities.

Together with the Dutch National Cyber Security Centre we have initiated a responsible disclosure procedure. However, since these routers are also used by many other companies in various countries, our findings seem to relate an international industry wide security issue.

1 Introduction

Most people use various devices at home to connect to the Internet. Examples of such devices include computers, phones, tablets, e-readers and smart-TV's. Nowadays, the majority of these devices use a wireless network interface and connect to a wireless base station (*router*) that gives access to the Internet. Such a router often serves as a firewall and is the first line of defence against malicious intruders that are active on the Internet. The user's devices operate in a internal network environment, the Local Area Network (LAN), which is separated by the router to protect against outside traffic, the Wide Area Network (WAN).

To gain access to a protected wireless LAN interface, the user needs to provide a WPA2 password (the wireless key). Such a password is often printed on a sticker which is attached on the bottom of a router. An example of such sticker is shown in Figure 1.

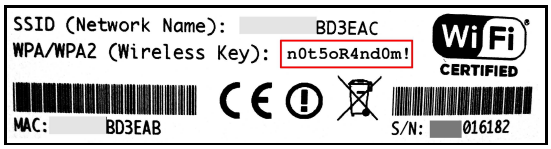

Figure 1: Sticker on the bottom of a wireless router.

The WPA2 password is used to perform mutual authentication between the user's device and the router. To sufficiently protect a wireless network, the router needs to be configured with a strong (randomly) chosen password that consists of a large number of characters to provide sufficient entropy [1]. Without a strong password the router is susceptible to brute-force attacks. Although it is often possible to change the password in the web-interface of the router, it is a bit of a hassle, especially when the router is regularly (remotely) reset where its settings return to the factory defaults. Subsequently, since the password that is printed on the sticker often looks very complicated (a lot of seemingly random characters), the user may be under the impression that is secure to leave the default password in place. We discovered that for Dutch routers this is not the case.

This paper contains a security analysis of WPA2 password generating algorithms that are used in many Dutch wireless routers. The major Telecom Companies (Tel-

cos) and Internet Service Providers (ISPs) in The Netherlands directly supply their customers with an (ASDL or Cable) Internet modem. Nowadays, almost all modems have the wireless router functionality embedded into the device. Therefore, these routers are currently massively deployed and used in The Netherlands.

We discovered that the tested routers generate wireless passwords by applying insecure proprietary obfuscation algorithms. The algorithms utilized in Dutch routers generate easy to predict network names and weak wireless passwords. The output is derived from public or predictable information such as broadcast messages and (incremental) serial numbers. Moreover, we verified with practical experiments that the WPA2 password of routers that utilize such password generating functions can be recovered within minutes.

Impact We have carried out invasive attacks to reverse engineer several wireless routers and concluded that the default wireless keys are trivial to recover, taken into account that an adversary has access to the algorithm. In the research, we have successfully recovered the proprietary algorithms from several major Dutch Telcos and ISPs. An adversary can mount practical attacks against those wireless networks to recover the password within minutes and use the compromised internet connection for fraudulent activity. Possible abuses include: stealing sensitive information, manipulating online electronic bank activity, infect client's computers with malware or simply commit digital crimes through the Internet connection of the customer such as downloading child pornography.

Although we have limited our research to analyzing the security of Dutch wireless routers, we have strong indications that many more routers are affected worldwide. Especially, since the same routers are being used by a number of other Telcos and ISPs in various countries.

Contribution The contribution of this paper is manifold. First, we show how a malicious adversary can instantly force a client to (re-)authenticate with the router. This allows the interception of a complete successful authentication trace. Such a trace can be used to offline verify a router password candidate and quickly eliminates false positives. Then, we present a general methodology how we recovered custom and proprietary hash algorithms from several Dutch routers. We expect that our method enables fellow researches and computer security experts to perform a similar risk analysis of the wireless router infrastructure in their country. Finally, we present use cases which practically demonstrate the insecurity of a number routers which are currently deployed by millions of users.

Responsible Disclosure We have strictly followed the responsible disclosure guidelines of the Dutch government [2]. These guidelines propose that the corresponding vendors be informed six months prior to full disclosure, giving them ample time to resolve the issues, inform their customers and hence preventing widespread abuse. We informed the Dutch government as well as all major Telcos and ISPs in the Netherlands in an early stage about the finding in our research. Consequently we are currently coordinating a national notification to the general public together with the Dutch National Cyber Security Centre (NCSC), formerly known as GovCERT, which is part of the Dutch Ministry of Security and Justice.

Overview The remainder of this paper is organized as follows: The related work is outlined in Section 2. Section 3 presents the technical background which introduces the techniques used later in this paper. Next, we present a general router security analysis methodology in Section 4. Five concrete use cases of router security analysis are presented in Section 5. Then, we evaluate several mitigating measures and possible solutions in Section 6. Finally, the conclusion of our study is given in Section 7.

2 Related work

This section contains the related work to our research. We have not limited ourselves to refer only to the academic literature. The reason for this is purely practical. Most of the related research is published in blog posts which are scattered over the Internet. This section starts with a general overview of wireless security issues. Then, it addresses related research about password generating algorithms. Finally, it gives a quick overview of related reverse-engineering projects which also analyzed the security of wireless routers.

2.1 Wireless security

There are several protection mechanisms introduced in the last few decades, including the well-known and widely deployed techniques Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA) and Wi-Fi Protected Access II (WPA2). The first two techniques are known to be vulnerable to several attacks [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. Recently, there are also some issues identified regarding WPA2 [16, 17, 18]. However, as far as the authors know, there is currently no practical password recovery attack proposed in the literature that can be mounted against the WPA2 protocol.

Some wireless routers support the Wi-Fi Protected Setup (WPS) authentication protocol. It enables a computer to connect with the wireless network by entering a single 8-digits PIN code instead of a long wireless password. The WPS protocol itself is vulnerable to a online practical brute-force attack. Suck attack can retrieve the PIN code from a WPS enabled router within a few hours [19] or in a few seconds when weak Pseudo Ran-

dom Number Generators (PRNG) are used to initialize the credentials [20]. However, modern routers have effective countermeasures against such attacks. Examples include a physical button that enables WPS for only one minute and a limited number of sequential failed authentication attempts.

2.2 Password generating algorithms

There were a number of incidents in the last decade that concerned insecure WPA2 password generating algorithms in routers. However, there is no general study published in the literature that addresses this issue specifically. Most of the incidents were made public in Internet blog posts or in Common Vulnerabilities and Exposures (CVE) reports.

The publication of the Thomson routers [21] had a serious impact for major ISP that is active in The Netherlands. In 2008, the ISP had massively deployed the vulnerable Thomson Speedtouch 780 router. After proactively informing their customers the ISP has now replaced most of these vulnerable routers.

Similar issues exist with routers from ADB / Pirelli. Several recent studies [22, 23, 24, 25, 26, 27] show that it is trivial to recover the default WPA2 password. Furthermore, issues were found in Comtrend routers [28] that are used by a large Spanish ISP. The researchers claim to have notified the manufacturer and ISP about these issues more than five years ago. However, it seems that these vulnerable routers are still actively being used in Spain. Then, issues exist within Arcadyan routers. A forum post from 2011 [29] points out that the password generating function is actually published in the form of a patent [30]. After the discovery, several variants of this algorithm were identified in other Arcadyan routers [31, 32, 33]. Finally, a number of consumer routers exist containing weak password generating algorithms [34, 35, 36, 37, 38, 39, 40]. Most of these consumer routers are currently still being sold in common consumer electronics stores.

2.3 Reverse-engineering routers

We identified several publicly available blog posts that specifically focus on reverse-engineering wireless routers [41, 42, 37, 32]. We have generalized their techniques and approaches in our methodology and use them to structure our analysis phase.

In this study we use non-invasive to invasive methods [43, 44] to recover the firmwares of the routers. The methods we used are described in detail in Section 4.1. Furthermore, publicly available tutorials demonstrate how to interface embedded hardware without requiring expensive lab equipment [45, 46, 47].

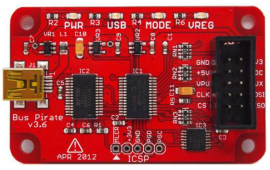

Figure 2: Bus Pirate.

Such methods include the JTAG debugger [48] and serial communication peripherals [49]. Most of these firmware recovery techniques can be carried out by using the Bus Pirate [50], shown in Figure 2, which is an off-the-shelf open hardware device that costs only USD $30. It supports a variety of communication buses and hardware protocols such as I2C, SPI, 1WIRE, UART and JTAG.

3 Introduction to WPA2

The WPA2 protocol can be set up in *Enterprise* or *Personal* mode. Enterprise mode uses a 802.1x RADIUS server for the authentication process whereas WPA2 personal uses a pre-shared key (PSK). Domestic networks normally use WPA2 personal. Unlike RADIUS server's online authentication, WPA2 personal does not rely on a Diffie-Hellman key exchange, however the shared secret must be previously established between the two parties using a separate channel. In this section we first introduce the key derivation of the WPA2-PSK protocol. Then, we explain how the mutual authentication is performed. Finally, we describe the deauthentication request that is included in the WPA2 protocol.

3.1 WPA2 key derivation

WPA2-PSK uses the key derivation function called `PBKDF2` (Password-Based Key Derivation Function 2) [51] to compute the shared secret key *PMK*. The PBKDF2 function requires the following input:

```
Derived Key = PBKDF2 (
    pseudo random function,
    password,
    salt,
    iterations,
    derived key length
)
```

`PBKDF2` combines the password *pw* and the wireless network identifier *ssid* as cryptographic salt to iterate a certain amount of times until obtain a derived key called *Pairwise Master Key* (PMK). WPA2 applies the function 4096 iterations to generate a 256 bits key by computing a HMAC-SHA1 of the passphrase and ssid.

$$PMK = PBKDF2(HMAC\_SHA1, pw, ssid, 4096, 256)$$

3.2 WPA2 authentication

Once this PMK is generated with the shared secret in both sides of the communication, a 4-way handshake which performs mutual authentication that proofs both sides have access to the shared secret *PMK*, see [52] for more details. A simplified overview of the authentication procedure is shown in Figure 3.

Once authenticated, the WPA2 protocol uses the Advanced Encryption Standard (AES) [53] in CCM encryp-

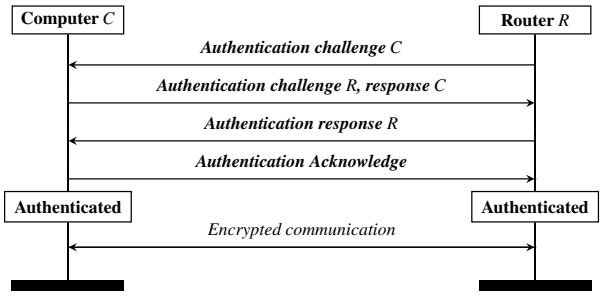

Figure 3: Simplified WPA2 authentication [52].

tion mode [54, 55] as specified in [52] to protect the confidentiality and authenticity of the messages that are transmitted between the computer and the router.

3.3 WPA2 deauthentication

The WPA2 protocol suffers, just like many other 802.11-based networks, from a serious security weakness. These protocols support a deauthentication (and deassociation) request which allows an entity to gracefully disconnect from the wireless network. Moreover, to let computers to disconnect which do not have the correct cryptographic credentials (or became out-of-sync), the deauthentication packet is not cryptographically protected in any kind. Such a feature can be convenient to use from an engineering perspective. However, it also introduces a serious security issue since it allows an adversary to mount a deauthentication attack to instantly gather all the information that is required to recover the wireless password. The problem was first discussed in [56] and later further analyzed in [57]. Figure 4 shows a simplified procedure that an adversary would perform to mount a deauthentication attack.

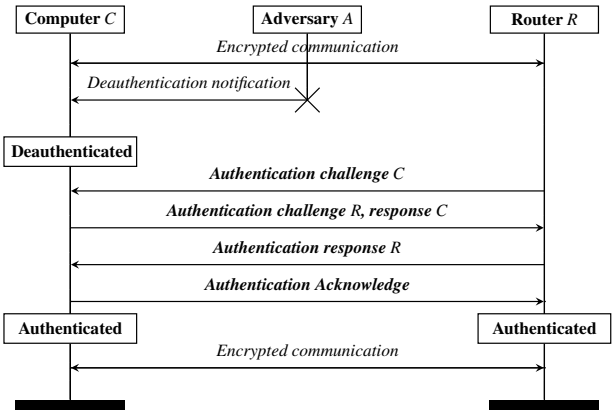

Figure 4: Deauthentication of WPA2 connection.

During an deautentication attack the adversary impersonates the router and transmits a deauthentication notification to the client. A plaintext packet is injected at an arbitrary time by the adversary without any knowledge of the shared secret PMK. The only requirement is that the adversary needs to spoof the network MAC address of the router, which is a trivial exercise. Additionally, the MAC address of the client has to be known since it is used as the destination address of the deauthentication packet. However, some clients will even accept the packet in case it is sent to the broadcast address.

After receiving the packet, a client will immediately terminate the connection to the router. The client will then automatically re-connect and authenticate itself again. The adversary now simply records the 4-way handshake. This handshake can be used later to perform an offline key recovery attack.

There are currently a few open-source attack tools available that can forge and inject a deauthentication packet into an active wireless connection between a computer and a router [58, 59, 60]. Furthermore, there are several publicly available tutorials show how such an attack can be executed in practice with the use of only ordinary consumer hardware [61, 62, 63, 64, 65].

4 Methodology

In this section we will go over the steps of obtaining the WPA2 default key generating algorithm from a router.

4.1 Obtaining the firmware

There are a number of ways to recover the embedded firmware of a router. They mostly vary in invasiveness and difficulty.

4.1.1 Downloading from the manufacturer's website

Obtaining the firmware can be as simple as visiting the manufacturer's website, selecting the router model, and downloading the image. Though this is typically not the case for routers deployed by ISPs. For such routers there seems to be a general reluctance against offering firmware images for download. Though this may hamper an adversary in obtaining the firmware and eventually the WPA2 default key generating algorithm, it is not to be considered a proper defense strategy, as will be demonstrated in the next sections.

4.1.2 Interfacing the router's serial console

Most routers offer a serial interface that allows to debug the device to some extent. The capabilities offered through the serial interface vary greatly. Hence, it is not always a useful strategy to obtain the firmware.

Serial interfaces can usually be identified by 4 lined-up pins, from which one of the outer pins connects to the ground. Most of the routers we encountered, the serial interface was not populated, requiring us to solder a 4-pin header onto it. In the case the serial interface is not found by inspecting the PCB, an online search may be performed. The OpenWRT wiki [66] is usually a good place to look for information such as where the serial in-

terface resides. Alternatively, a search for a datasheet of the System-on-Chip (SoC) may be performed, identifying which pins are the serial interface and subsequently inspecting the board in an attempt to discover where they are connected to. Once the serial interface is identified, a TTL-to-USB converter can be used to communicate with it.

Once communication is established between the PC and the router via the serial interface, the next step is to identify the capabilities the device offers over the serial interface. Typically, once the router is booted, it only outputs diagnostic logs, and does not offer the capability to send commands. However, when the router is booting, it is very typical that the device's boot loader offers a way to interrupt the normal boot sequence over the serial console, e.g. by pressing a key before a timer expires. The options offered to the serial console user differ from boot loader to boot loader, although they usually include downloading a new firmware image over the TFTP protocol and either flashing it or booting it directly, without performing a flash operation. Some boot loaders even allow to make a backup of the flash chip, meaning our quest to obtaining the firmware ends here.

If this is not the case, an attempt may be made to craft an image that the boot loader will accept and boot. Once the boot loader accepts the image, code execution is obtained on the router, hence full control over the router is obtained, including the ability to dump its flash. However, this is only possible in case the image format has been documented, either by the manufacturer or through reverse engineering, and does not require the manufacturer to include a cryptographic signature based on asymmetric cryptography (such as RSA or elliptic curves). It is likely that this is the case when the router is supported by OpenWRT, DDWRT, or a similar open source after market firmware. Although it is generally feasible to craft such an image, it is very likely easier to obtain code execution by debugging the router with a JTAG interface.

4.1.3 Debugging the router with JTAG

Most SoCs offer debugging capabilities through JTAG. However, in our experiments we typically do not encounter the JTAG pins being connected on the PCB. Similar to finding the serial interface, the JTAG interface may be found by visually inspecting the PCB. The existence of 10, 14 or 20 pins placed together is usually a strong indicator of a JTAG header. As with the serial interface, a description of where the JTAG pins can be found on the PCB may be found online or by using a datasheet. In the case the JTAG pins cannot be identified, automated techniques exist that identify the JTAG pins by taking a "brute-force" approach. Such techniques can be implemented in Arduino devices [67]. Alternatively, a device designed for this specific purpose exists [68]. The technique identifies pins and enumerates undocumented opcodes from test points and/or component pads.

Once the JTAG interface is identified, a hardware debugger can be used to communicate with the microcontroller. A JTAG debugger comes as cheap as $6 USD.

Once the router is connected through the JTAG interface, control over its execution may be obtained. For example, register values can be inspected/manipulated, breakpoints can be set, code and/or data in RAM can be inspected/manipulated, etc. In case the debugging software used properly supports the flash chip, it is possible to read it out directly. If this is not the case, it may be possible to extend the debugging software to support this particular flash chip. Alternatively, the code within the firmware that is used to read from the flash chip may be recycled for our goal of reading it out entirely. As a final option, we may simply dump the contents of the RAM while the router is up and running. It is very typical for firmware of embedded devices to be loaded in its entirety in RAM upon boot, hence it is very likely to hold the entire firmware, hence also holding WPA2 default key generating algorithm.

4.1.4 Exploiting a known vulnerability

Sometimes routers run a firmware for which a known vulnerability exists that allows us to gain control over its execution. If this is the case, we may be able to exploit it and use the firmware's internal capabilities to create a backup of its software and transfer it over the wire. The easiest way may be an OS command injection, buffer overflows in the web server or other services exposed over the network such as telnet, FTP, TFTP, etc.

4.1.5 Desoldering the flash chip

In the case none of the methods posed above work, a rather invasive method way of obtaining the firmware may be used: by desoldering the flash chip and extracting its contents with an EEPROM reader. It requires a chip programmer that costs a few hundred dollars and the router will most likely be destroyed in the process. However, it is a very reliable way of obtaining the firmware in the sense that it almost always works: unless a router manufacturer resorts to very drastic measures such as hardware encryption with a per-device key, the firmware can be recovered by desoldering the flash chip.

4.2 Decompressing and de-obfuscating

Once the firmware image is obtained, a typical first step would be to decompress it. In order to do so, a strategy that worked very well during our experiments is to attempt to pinpoint signatures of a number of well-known compression formats, such as GZIP or LZMA, and attempt to decompress starting from that offset. In case this does not work, we may be able to find documentation on this topic online. Usually with Linux-based devices,

the image contains a SquashFS filesystem, which is used as the root filesystem. Since version 4.0, SquashFS supports LZMA compression. Router manufacturers typically also use LZMA in SquashFS versions prior to 4.0. They do so by means of adding proprietary extensions to the SquashFS code. Therefore, in order to successfully extract the root filesystem, we need to these extensions into account. Fortunately, the General Public License, under which the SquashFS code is licensed, requires the manufacturer to release the source code of these extensions. Hence, all the tools required to extract the root filesystem should be offered by the manufacturer for download.

It happens that router manufacturers also add an obfuscation layer. However, by design, the router's CPU needs to be able to decompress/de-obfuscate the code prior to being able to execute it. Typically, a software routine is present in the boot loader that does this, which may or may not be present in a firmware update image. In order to remove the obfuscation layer, the routine implementing the de-obfuscation either has to either be run in an emulator or be reverse engineered. Although it adds difficulty to the process of eventually obtaining the WPA2 default key generating algorithm, it will certainly not stop a dedicated attacker. On top of that, many the obfuscation algorithms have been published online, hence completely defeating the purpose of the obfuscation [41, 69]. Alternatively, the router manufacturer could add hardware that performs the decompression and de-obfuscation, although this adds additional manufacturing costs and the gain in additional security is questionable. We have not encountered such hardware protection in any of the routers we experimented with.

4.3 Identifying the algorithm

Once the actual code is obtained, it can be analyzed by loading it into a disassembler tool such as IDA Pro. Finding the WPA2 default key generating algorithm is a task of which the difficulty varies greatly between different routers. For example, depending on the file format, (e.g. the raw image, or an executable file found in the root filesystem), a symbol table may be available, hinting to what the underlying code is doing. In our experiments, if it is available, it is often the easiest route to identify the algorithm. However, if it is not the case, we must resort to other means of finding it.

Suppose that one is in possession of a number of sample keys for a certain type of router. From this it can be reasonably deduced what the character set used in the WPA2 default keys is. Typically, the algorithm computes indices that are subsequently used in an array containing all possible characters. Hence, in order to find the WPA2 default key generating algorithm, it is often a good strategy to look for an array containing the character set, and look up where this array is referenced.

Furthermore, ESSIDs are typically also diversified over all routers. Suppose that the ESSID is `<ISP name> + <5 digits>`. Then the function generating the ESSID may refer to a string such as `<ISP name>%05d`. In the case we find such a string, we look up where the string is referenced, leading us to the ESSID generating function. It is very likely that the ESSID generating function is invoked in code that performs a factory reset or similar. Hence, once the function generating the ESSID is identified, it is likely the WPA2 default key generating algorithm can be pinpointed by analyzing the code that invokes this function.

Finally, another strategy is to look for the code that performs the factory reset. Since this code is often very verbose, it should be easy to pinpoint strings used within this code (e.g. strings used to print the status of the factory reset to the serial interface). Once the factory reset code is identified, the functions invoked can be analyzed in order to identify which generates the WPA2 default key.

4.3.1 Verifying the existence

Suppose the target function is still not found despite considerable effort. Although, in our experiments we have not encountered it, it may be the case that the algorithm is actually not present in the firmware. A reliable way to verify this proposition is to change the password such that it is different from the default one. Subsequently, the flash chip is re-read using any of the methods described in sections 4.1.2, 4.1.3, 4.1.4 or 4.1.5. Next, it can be checked whether the image obtained still contains the default password. If this is the case, then this is a strong indicator that the default password is stored on the flash chip and is re-instated when the user performs a factory reset.

4.4 Analyzing the algorithm

Suppose that the target algorithm is located with great certainty. The next step is to determine what input is fed to this algorithm. This information may be deduced from hints such as the number of characters used from the input.

Additionally, an attempt may be made to run the algorithm in an emulator, such as QEMU. Although not a necessity, it greatly simplifies the process of reverse engineering the algorithm, since it enables one to perform dynamic analysis, diagnose intermediate results, etc. In order to do so, a tiny piece of code is written that invokes `mmap` (present in any modern C library) to map the firmware image to the base address used by the router. Then, the code performs a call to the address of the function and pass the expected input values to it. The code is then compiled with a compiler for the router's architecture and run within the emulator. This should output the

correct WPA2 default key, in the case the inputs are as the algorithm expects. Note that the algorithm itself may depend on data in RAM being properly initialized, which is obviously not the case when it is invoked in this fashion. This happens for example when the algorithm calls `sprintf`. In this example, a straightforward workaround is to replace the call with the `sprintf` from libc, which is initialized upon execution of the binary.

4.5 Reverse engineering the algorithm

Once all steps described above are completed, the process of reverse engineering can start. Reverse engineering typically is a slow process, where one takes a number of instructions, tries to make sense of them, and rewrites them in a higher-level programming language. This method is comparable to other security analysis of embedded devices described in the literature [70, 71, 72, 73, 74, 75, 76].

Optionally, the correctness of intermediate of the code may be verified by means of emulating the code. This process is described in the last section. Once the algorithm is successfully reverse engineered, it can be used it to recover the default password from another router of the same type or product family.

4.6 Recovering the inputs

With no exceptions, all WPA2 default key generating algorithms that were recovered during our experiments use either the router's MAC address or serial number, or both, as input. Possible inputs such as the serial number, are assumed unknown to us and hence the strategy becomes to try every possible serial number. However, routers exist that also generate the ESSID and channel number based on the serial number. If these functions are also reverse engineered, in addition to the WPA2 key generating algorithm, their results may be used to rule out the vast majority of candidate keys. In order to further narrow down the set of candidate keys, one may attempt to correlate MAC address and serial number. Since both are typically assigned sequentially, it is very likely that the vast majority of candidate keys may be ruled out this way. Though this possibility has not been explored during our experiments.

Finally, another variable that was encountered during our experiments as an input to the key generating algorithm is the MAC address of the ethernet (LAN) adapter. Although this may seem as a variable that is unknown to an adversary, its contents is typically quite predictable. For example, it may be tightly correlated to the Wi-Fi MAC address of the router, which is public. During our experiments, routers were encountered where these MAC addresses differ in a single digit. Besides such an obvious correlation, the first three bytes of the MAC address are tied to the router manufacturer, which leaves only a search space with an entropy of 24 bits to find the ethernet MAC address.

Encrypted wireless network packets include information in plain-text such as the BSSID, the source and destination MAC address. This information is sent unencrypted because this allows a wireless network device to decide whether a packet should be ignored, for example in case the device is not the recipient, prior to performing decryption. In one specific router, the ethernet and wireless interfaces are bridged together in a single interface. The default behaviour in Linux-based devices is to assign the MAC address of the first interface added to the bridge. In this case, the bridge interface is assigned the ethernet MAC address. Hence, every packet sent by a client that is destined for the router itself has the router's ethernet MAC set as its destination. Since the router is the default gateway for traffic destined outside the local subnet, i.e. the Internet, all packets sent to any destination on the Internet will have the router's ethernet MAC address assigned as its destination. Hence, all that is needed in order to recover the ethernet MAC address is to capture a single (encrypted) packet from the air and inspect its destination address. A packet dissected by Wireshark demonstrating this phenomenon is depicted in Figure 5.

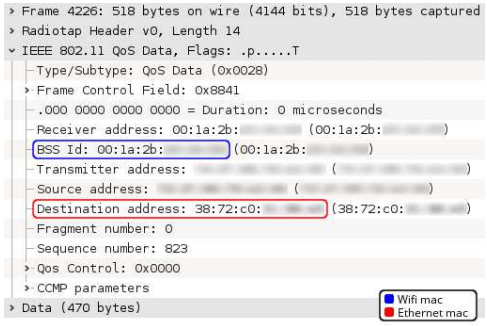

Figure 5: Packet dissected by Wireshark revealing the router's ethernet MAC.

4.7 Building an attack

The number of candidate keys directly depends on the input fed to the algorithm. For example, the router's MAC address is public. Hence, in case the algorithm uses only this value as input, its default WPA2 key can be immediately computed.

The size of the resulting set of candidate keys is sometimes so small that attempting to authenticate to the device using all the candidate keys is a feasible way of recovering the WPA2 key. However, in situations where the set is larger, we can deploy a different strategy. Suppose that one is in possession of a captured and stored authentication handshake between a client and an access point. It can be used to verify whether a candidate key is indeed the correct key, see Section 3.3. This method of

verifying the key is significantly faster than attempting to authenticate, allowing to verify several thousand candidates within seconds. On top of that, once the handshake is captured, an attacker need not be in proximity of the router anymore. Furthermore, in order to capture such a handshake, it is not necessary to wait for a client to initiate an authentication. Instead, a de-authentication packet can be sent to a client who is already connected. Since this packet is not cryptographically authenticated, the client will simply disconnect. Typical behaviour of wireless clients is to automatically re-connect, allowing an authentication handshake to be captured. The de-authentication is described in detail in Section 3.3. Hence, all that is needed is a single client being connected to the router in order to significantly speed up the attack, typically recovering the key within seconds.

5 Use cases

In this section we present the results found in several routers during our experiments.

5.1 Router 1

The first router security analysis include the recovery of hardcoded credentials, identification of a default WPA2 key which is based solely on public data, an OS command injection in the Telnet service and a stack buffer overflow in the HTTP server. This router is used by a major telecommunications operator, which has many million customers in several European countries.

5.1.1 Obtaining the firmware

The router embeds an active serial port interface, which can be used by soldering a header onto the board. When powering on the device, the boot loader waits for input from the serial port for 2 seconds before continuing the boot process and ultimately loading the firmware. During this time-frame it is possible to interrupt the boot process and supply an alternate firmware which is loaded into RAM and executed directly, hence without the necessity to perform a flash. Hence, we used the method described in Section 4.1.2 to recover firmware.

5.1.2 Locating the algorithm

The router only offers a very limited web interface to its user. We assumed that an additional, more privileged, "system" account had to exist, which is used by e.g. technical support. The HTML and Javascript code served by the web server also hints in this direction. E.g. terms such as *usrPassword*, *sysPassword* and *sptPassword* are self-explanatory. By simply searching for these strings in the firmware binaries, the "system" username and password can be recovered. The "system" account allowed us to enable the *telnet* service.

From the telnet console we found another command: *md5wpakey* which, interestingly, outputs the default WPA2 key which is set when the router ships, even in the case when the user had set the WPA2 key to a different one. Hence, it seemed a good starting point for finding the function that generates the default WPA2 key.

We searched through the file system for `md5wpakey` and found only a single file that contained that string: `/bin/cfm`. When loading the binary into IDA Pro, we noticed that the symbols in the binary were not stripped, which greatly simplifies the analysis.

The key is obtained by taking the lower-case hex representation of the first six bytes of the following hash:

```
password = MD5(
    constant seed,
    lowercase WAN mac address,
    uppercase LAN mac address
)
```

Section 4.6 shows how to observe and recover both MAC addresses using only wireless interception of network packets.

5.1.3 Telnet command execution

We noticed that there was some input validation active for the *telnet* service. For instance, execution of `sysinfo && sh` fails with the following error message:

```
Warning: operator & is not supported!
```

From this we blindly assumed that the input is properly sanitized. Though, having the ability to execute commands allows for a more thorough analysis of the software. Therefore, we decided to patch the command interpreter such that is does not sanitize the inputs. During this process we noticed that the command is not executed when a `&` (ampersand) or `;` (semicolon) is found. However, there was no validation that looked for the `|` (pipe) symbol.

5.1.4 Stack buffer overflow

During our study we also identified a (indirect) remote executable exploit. A malicious website can redirect the client's web browser to send specifically crafted HTTP requests to the router. We found a stack-based buffer overflow vulnerability by looking into code that references `strcpy` in `/bin/cfm`, which holds the web server, with IDA Pro and subsequently checking the source and destination pointers. The vulnerable function is `cgiOpt60Add`, it can be triggered by requesting `/dhcpOption60.cmd?action=add&VendorID=input1&IpStart=input2&ipEnd=input3`.

All the input variables are copied onto the stack without performing bounds checking. The stack is flagged non-executable. However, ASLR is used neither by the binary, nor by any loaded shared library. Additionally no stack smashing protection is used. Hence, the

buffer overflow can be exploited with the use of Return Oriented Programming (ROP) [77].

5.2 Router 2

The second router is similar to the first router, but a slightly different variant, it is deployed by the same ISP. In this section, we will briefly state the difference between the two routers regarding vulnerabilities.

5.2.1 Obtaining the firmware

The firmware images for this router are also not available online. We extracted the firmware using the serial interface as we did with Router 1, however, we also found a more convenient alternative way. We can exploit the same command injection vulnerability as available in router 1, with a slight difference that the `sysinfo` command was removed, though the `ping` command could be used instead. In contrast to the first router, this one came with the `nc` command installed, which allows one to send/receive data. Hence, we obtained a shell on the router using the command injection vulnerability and subsequently used `nc` to read the router's firmware and send it over the network.

Also in this router, Telnet could be activated in the web interface of the "system" account. However, we noticed that this router uses different global "system" credentials for the web interface. Interestingly, the option to enable the telnet is for a "normal" user only not visible in the web interface, but given the correct url, also the "normal" user can activate it.

5.2.2 Locating the algorithm

The default WPA2 key generating function is identical to the one applied in router 1. However, the configuration of router is different: we noticed that the Ethernet mac address equals the WiFi mac address, with the last digit decremented by one. This means that the default WPA2 password can be computed when only the BSSID is known.

5.3 Router 3

This router is deployed by a major multi-national telecommunication operator who offer services in a number of countries under different brands.

From the router we recovered algorithms used to generates the default WPA2 key for a number of brands, taking the router's serial number as input. The range of possible serial numbers is not wide enough in order to provide sufficient entropy for the security of the wireless network.

Additionally, the default ESSID and wifi channel number are also generated using the serial number, allowing us to narrow down the set of possible serial numbers to several thousands, which can be checked against a captured authentication handshake within seconds.

5.3.1 Obtaining the firmware

Neither the ISP nor the router manufacturer offers firmware images for downloading off their websites. Hence, we had to extract the firmware image from the device itself. We obtained a firmware image by desoldering the flash storage chip and using an EEPROM reader to extract its contents. The firmware image is stored compressed using LZMA, and gets decompressed on start-up by the boot loader. However, the LZMA image is in a slightly different format: the uncompressed size is missing, which was found before on routers that use the same boot loader [42]. By simply inserting a too large uncompressed size, the file can be decompressed with standard LZMA tools. The file obtained is a binary blob, which is mapped in memory on a static address and subsequently executed. Hence, no executable headers and thus symbols, sections, etc. are available.

5.3.2 Locating the algorithm

As a starting point for finding the WPA2 key generating function, we speculated that finding the function that generates the default ESSID should point us in the right direction. We found it by searching for the string "`<ISP name> %07d`" (all the routers of this model we encountered have a default ESSID in the form of the ISP name + a 7-digit number). Oddly, the function that generates the ESSID, returns a static value in the case certain input is fed into it. A quick search yielded that in the past, the ISP had been experimenting with a second hidden wireless network named as such. Users have discovered the constant key that can be used to access this network by using the *backup settings* functionality. Apparently, this functionality had not been completely removed in later firmware revisions.

We successfully reverse engineered both the ESSID and WPA2 key generating functions. Both functions take the router's serial number as input and perform some obfuscation. We assume the router's serial number is unknown to an attacker. To the best of our knowledge, it cannot be obtained from wireless communication with the router without already knowing its WPA2 key. However, given that the ESSID of the router is public, the attacker can build a candidate list of possible serial numbers that yield the correct ESSID and compute their corresponding WPA2 key. Doing so typically leaves only several thousands of candidate keys. The candidates can be verified using a deauthentication attack as described in Section 3.3. The complete attack procedure can be performed in a few minutes. This technique can be applied for the rest of algorithms that were revealed in this firmware image. We have counted up to 8 different password generating algorithms all based on the serial number and a cryptographic hashing function. All these routers are wide spread over the world.

This vulnerability seems also to be present in another router from the same manufacturer, also deployed by this ISP in another country. Although the ESSID generating function seems to differ. Finally, speculating on the results found, this vulnerability is likely also present in different router models from the same brand deployed by this ISP in other countries. However, as of yet, we were unable to practically verify whether this is the case.

5.4 Router 4

This section describes our reverse engineering research on a couple of routers deployed by a large Telco in the Netherlands. We recovered the WPA2 password generating algorithm without reverse-engineering the firmware.

5.4.1 Recovering the algorithm

Surprisingly, we were surprised when we applied a previously published algorithm [29] for similar routers and noticed that it partially worked. By executing their algorithm, we detected that 9 out of 12 digits matched. Therefore, we suspected that the vendor was reusing its previous method described in this patent [30]. With some elementary linear algebra the "new" constant seed can be recovered which is used together with an exclusive-or (XOR) operation in the algorithm.

To recover the key an adversary only need to mount a brute-force attack with at most 10^5 candidates, from which the false candidates can be eliminated in a matter of seconds.

5.5 Router 5 (family)

This section describes vulnerabilities discovered in a large family of routers made by the same manufacturer.

5.5.1 Obtaining the firmware

We downloaded a large set of firmwares and used previously published techniques [36, 38, 37] to de-obfuscated them. Old versions were not obfuscated though.

5.5.2 Locating the algorithm

We have seen a distinction of default WPA2 keys generation algorithm in many firmwares. It seems that there are two main algorithms that cover the majority of these routers and they use either the algorithm currently known [38] or a slightly modified version.

In our reverse engineering research, we have observed that certain routers are using the new password generation function. The difference between this function and the old one is simply the characters set alteration. Basically, the algorithm uses the last 3 bytes from the mac address to mangle it with some 'secret numbers'. After that, these numbers are substituted by the modular position in the characters set.

Another model of this family was reverse engineered. This model contained an stripped binary called '`AutoWPA`' responsible for generating the default WPA key. Such binary was emulated and found out that only mac address was necessary as input. After a dynamic analysis, we managed to recover the algorithm which can be attacked in matter of seconds. The algorithm was simply using many times bitwise operations (and, nor, xor and or) with the mac address and constants values. Eventually, we realized that MD5 was used to generate a hash and subsequently converted to different character set.

6 Mitigation and countermeasures

We divide the mitigation strategy in two categories: the short-term notification phase and the implementation of long-term countermeasures.

Short-term The severity of weak default WPA2 passwords in many routers demands for an immediate response. There are most likely millions of house-holds that use the default WPA2 password that is printed on the sticker of the Internet router. We are currently strongly encouraging Telcos, ISPs and manufacturers to embrace their responsibility and start informing their customers about the insecurity of weak default passwords in routers. A quick security improvement could be achieved by stimulating users to change their default key by choosing a strong wireless password [1]. However, it is well-known that user defined password are often not so difficult to guess [78, 79]. Therefore, this can be seen as an effective, but palliating countermeasure, that only marginally increases the protection of wireless networks.

Long-term There are several countermeasures that can be applied to improve the security of wireless routers. The most important change should be the removal of the password generating algorithm. We noticed during our experiments that every router is uniquely personalized during manufacturing. Specifically, a unique serial number and network MAC addresses are programmed into EEPROM. Furthermore, the sticker on the router contains the same serial number, MAC addresses and the wireless password. Therefore, we see no reason why a strong and randomly chosen password can not be programmed into EEPROM as well in stead of being derived from the other two values. Moreover, there are well-known royalty free statistical test suites that can help implementing the best practices for generating random strings [80].

7 Conclusion

We are surprised to notice little improvement in the default password protection of Dutch routers. In 2008, the practically exploitable security issue in SpeedTouch routers [21] generated some serious media attention in The Netherlands. We had expected that successor and improved routers would be much more secure. However, our study reveals that various modern and massively deployed routers still use weak methods to generate default

passwords. Users that did not explicitly changed their wireless password are vulnerable to practical password recovery attacks which enables an adversary to remotely intrude their network within minutes.

We have strictly followed the principles of responsible disclosure [81]. The guidelines, defined by the Dutch government [2], propose a time-frame of six months advance notice for embedded security issues prior to full disclosure. We informed the Dutch government as well as all major Telco and ISPs in the Netherlands in an early stage about the findings of our research. Moreover, we are currently coordinating a nation wide notification to the general public together with the Dutch National Cyber Security Centre (NCSC). With this course of action we hope to motivate vulnerable users to change their weak default WPA2 key.

Although, we focused our research solely on the analysis of Dutch wireless routers, we have reason to believe that this issue is an industry-wide problem and applies to many routers deployed in several countries. We noticed that some vendors reuse their WPA2 password generating algorithms with small modifications in other countries. This suggests that many more routers are vulnerable to practical password recovery attacks.

References

[1] Sheila L. Brand and Jeffrey D. Makey. Password management guideline. Technical Report CSC-STD-002-85, Department of Defense Computer Security Center (DoD-CSC), Fort George G. Meade, Maryland 20755, April 1985. Library No. S-226,994.

[2] National Cyber Security Centre (NCSC). Policy for arriving at a practice for responsible disclosure. *https://www.ncsc.nl/english/current-topics/news/responsible-disclosure-guideline.html*, 2013.

[3] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *8th International Workshop on Selected Areas in Cryptography (SAC 2001)*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2001.

[4] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *9th Network and Distributed System Security Symposium (NDSS 2002)*. The Internet Society, 2002.

[5] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *7th International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pages 180–189. ACM, 2001.

[6] Nancy Cam-Winget, Russ Housley, David Wagner, and Jesse Walker. Security flaws in 802.11 data link protocols. *Communications of the ACM*, 46(5):35–39, 2003.

[7] Russ Housley and William Arbaugh. Security problems in 802.11-based networks. *Communications of the ACM*, 46(5):31–34, 2003.

[8] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Transactions on Information and System Security*, 7(2):319–332, 2004.

[9] Itsik Mantin. A practical attack on the fixed RC4 in the WEP mode. In *11th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2005)*, volume 3788 of *Lecture Notes in Computer Science*, pages 395–411. Springer-Verlag, 2005.

[10] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in WEP's coffin. In *27th IEEE Symposium on Security and Privacy (S&P 2006)*, pages 386–400. IEEE Computer Society, 2006.

[11] Rafik Chaabouni et al. Break WEP faster with statistical analysis. Technical report, technical report, EPFL, LASEC, 2006.

[12] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In *8th International Workshop on Information Security Applications (WISA 2007)*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer-Verlag, 2007.

[13] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In *2nd ACM Conference on Wireless Network Security (WISEC 2009)*, pages 79–86. ACM, 2009.

[14] Pouyan Sepehrdad, Petr Susil, Serge Vaudenay, and Martin Vuagnoux. Smashing WEP in a passive attack. In *20th International Workshop on Fast Software Encryption (FSE 2013)*, volume 8424 of *Lecture Notes in Computer Science*, pages 155–178. Springer-Verlag, 2013.

[15] Mathy Vanhoef and Frank Piessens. All your biases belong to us: Breaking rc4 in wpa-tkip and tls. In *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C., 2015. USENIX Association.

[16] Achilleas Tsitroulis, Dimitris Lampoudis, and Emmanuel Tsekleves. Exposing wpa2 security protocol vulnerabilities. *International Journal of Information and Computer Security*, 6(1):93–107, 2014.

[17] Pieter Robyns, Bram Bonné, Peter Quax, and Wim Lamotte. Short paper: exploiting wpa2-enterprise vendor implementation weaknesses through challenge response oracles. In *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*, pages 189–194. ACM, 2014.

[18] Mayank Agarwal, Santosh Biswas, and Sukumar Nandi. Advanced stealth man-in-the-middle attack in wpa2 encrypted wi-fi networks. *Communications Letters, IEEE*, 19(4):581–584, 2015.

[19] Stefan Viehböck. Brute forcing wifi protected setup. *https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf*, December, 2011.

[20] Dominique Bongard. Offline bruteforce attack on wifi protected setup. *Presentation at Hacklu*, 2014.

[21] Kevin Devine. Default wep/wpa key algorithm for thomson routers. *http://www.hakim.ws/st585/KevinDevine*, 2008.

[22] Muris Kurgas. Pirelli discus drg a225 wifi router. default wpa2-psk algorithm vulnerability. *http://www.remote-exploit.org/content/Pirelli_Discus_DRG_A225_WiFi_router.pdf*, 2009.

[23] WiFi researchers. Alice agpf: The algorithm. *http://wifiresearchers.wordpress.com/2010/06/02/alice-agpf-lalgoritmo*, June, 2010.

[24] WiFi researchers. Telsey fastweb: Full disclosure. *https://wifiresearchers.wordpress.com/2010/09/09/telsey-fastweb-full-disclosure*, September, 2010.

[25] Stefan Viehböck. A1/telekom austria prg eav4202n default wpa key algorithm weakness. *http://sviehb.wordpress.com/2011/12/04/prg-eav4202n-default-wpa-key-algorithm*, 4 December, 2011.

[26] Eduardo Novella. Cve-2015-0558: Reverse-engineering the default wpa key generation algorithm for pirelli routers in argentina. *http://ednolo.alumnos.upv.es/?p=1883*, 2015.

[27] Eduardo Novella. Hacking again pirelli routers: Adb pirelli p.dg a4000n deployed by meo portugal. *http://ednolo.alumnos.upv.es/?p=2008*, 2015.

[28] Eduardo Novella and Mambostar. Uncovering the default wpa key generation for telefonica routers in spain. *http://foro.seguridadwireless.net/desarrollo-112/fallo-de-seguridad-en-routers-comtrend-full-disclosure*, 24 November, 2010.

[29] Seguridad Wireless team. Wlan4xx: Algorithm for arcardyan routers. yacom. *http://foro.seguridadwireless.net/desarrollo-112/wlan4xx-algoritmo-routers-yacom*, 3 March, 2011.

[30] TW Arcadyan Technology Corp., Hsinchu. Arcadyan encryption scheme patent. *http://www.patent-de.com/20081120/DE102007047320A1.html*, 2007.

[31] Stefan Viehböck. Vodafone easybox default wps pin algorithm weakness. *https://www.sec-consult.com/fxdata/seccons/prod/temedia/advisories_txt/20130805-0_Vodafone_EasyBox_Default_WPS_PIN_Vulnerability_v10.txt*, 12 December, 2012.

[32] Warker Ranger. Reverse engineering of the wpa default algorithm of alice (o2) modem iad 1421 and 4421. *http://warkerranger.tumblr.com/post/67646092068/re-des-wpa-default-algorithmus-der-alice-o2-iad*, November 21, 2013.

[33] Seguridad Wireless team. Arcadyan routers used by vodafone in spain are also vulnerables. *http://ednolo.alumnos.upv.es/?p=1760*, 4 February, 2014.

[34] Jörg Schneider Jakob Lell. Cve-2012-4366: Insecure default wpa2 passphrase in multiple belkin wireless routers. *http://www.jakoblell.com/blog/?p=15*, 19 November, 2012.

[35] Alex Altea. Wpa2 cracking dictionary for tp-link routers. seeds are not so random. *http://www.backtrack-linux.org/forums/showthread.php?t=62673*, 19 November, 2013.

[36] Roberto Paleari and Alessandro Di Pinto. Multiple vulnerabilities on sitecom devices. sitecom n300/n600 devices. *http://blog.emaze.net/2013/08/multiple-vulnerabilities-on-sitecom.html*, August 19, 2013.

[37] Warker Ranger. Reverse engineering blog. *http://warkerranger.tumblr.com*, 2014.

[38] Roberto Paleari and Alessandro Di Pinto. Sitecom firmware encryption and wireless keys. *http://blog.emaze.net/2014/04/sitecom-firmware-and-wifi.html*, 22 April, 2014.

[39] Craig Heffner. Reversing belkin's wps pin algorithm. *http://www.devttys0.com/2015/04/reversing-belkins-wps-pin-algorithm*, 10 April, 2015.

[40] Craig Heffner. Reversing d-link's wps pin algorithm. *http://www.devttys0.com/2014/10/reversing-d-links-wps-pin-algorithm*, 31 October, 2014.

[41] Stefan Viehböck. Reverse engineering an obfuscated firmware image E01, unpacking. *https://sviehb.wordpress.com/tag/arcadyan-2*, 9 September, 2011.

[42] Bernardo Rodrigues. Unpacking firmware images from cable modems. *http://w00tsec.blogspot.nl/2013/11/unpacking-firmware-images-from-cable.html*, November, 2013.

[43] Ross Anderson. Protecting embedded systems the next ten years. In *3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*, volume 2162 of *Lecture Notes in Computer Science*, pages 1–2. Springer Berlin Heidelberg, 2001.

[44] Sergei P. Skorobogatov. Semi-invasive attacks – A new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005.

[45] Huawei HG612 hacking. Jtag'ing the broadcom bcm6368 (hg612). *https://huaweihg612hacking.wordpress.com/2012/11/07/jtaging-the-broadcom-bcm6368-hg612/*, 2012.

[46] Embedded Device Hacking. Re-enabling jtag and debugging the wrt120n. *http://www.devttys0.com/2014/02/re-enabling-jtag-and-debugging-the-wrt120n*, 5 February, 2014.

[47] Dpeddi and The-Lizard. Jtag support for lantiq vgv7519 devices. *https://github.com/openwrt-vgv7519/lantiq-vgv7519-original-firmware*, 2014.

[48] LAN/MAN Committee et al. IEEE standard for test access port and boundary-scan architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pages 1–444, May 2013.

[49] Embedded Device Hacking. Reverse engineering serial ports. *http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/*, November, 2012.

[50] Hendrik Hanff. How to use the bus pirate as a logic analyzer. In *Proceedings of the RIC Project Day Workgroups - Electronic Design and Mechatronic Design. RIC Project Day, July 24, Bremen, Germany*, DFKI Documents, D-14-05. DFKI Robotics Innovation Center Bremen, July 2014.

[51] B. Kaliski. Rfc2898: Password-based cryptography specification version 2.0. Technical report, RSA Laboratories, September, 2000.

[52] IEEE 802.11 Working Group et al. Ieee standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments. *IEEE Std*, 802:11p, 2010.

[53] PUB FIPS. Advanced encryption standard (AES). *National Institute for Standards and Technology (NIST)*, 197(1), 2001.

[54] Doug Whiting, Russell Housley, and Niels Ferguson. Counter with CBC-MAC (CCM). *http://tools.ietf.org/html/rfc3610*, 2003. RFC 3610.

[55] Morris Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. *NIST Special publication (800-38C)*, 38C:1–27, 2004.

[56] Daniel Lowry Lough. *A taxonomy of computer attacks with applications to wireless networks*. PhD thesis, Virginia Tech, 2001.

[57] John Bellardo and Stefan Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *12th USENIX Security Symposium (USENIX Security 2003)*, page 1527, 2003.

[58] sophron. Automated phishing attacks against wifi networks in order to obtain secret passphrases and other credential. *https://github.com/sophron/wifiphisher*, 2015.

[59] RFKiller. Mass-deauth script by rfkiller. *https://github.com/RFKiller/mass-deauth*, 2014.

[60] Dan McInerney. Continuously jam all wifi clients and access points within range. *https://github.com/DanMcInerney/wifijammer*, 2015.

[61] ASPj. Tool for denial of service on wireless networks. *http://aspj.aircrack-ng.org/#mdk3*, 2008.

[62] superkojiman. Capturing the wpa handshake using mass deauthentication. *http://blog.techorganic.com/2010/12/20/capturing-the-wpa-handshake-using-mass-deauthentication/*, 2010.

[63] darkaudax, mister_x, and sleek. Deauthentication. *http://www.aircrack-ng.org/doku.php?id=deauthentication*, 2006.

[64] Jordan. Wireless deauth attack using aireplay-ng, python, and scapy. *http://raidersec.blogspot.nl/2013/01/wireless-deauth-attack-using-aireplay.html*, 2013.

[65] Cyber Security Labs. Deauthentication/disassociation attack, 2014.

[66] OpenWRT Wireless Freedom wiki. Gnu/linux distribution for embedded devices. *http://wiki.openwrt.org*, January, 2004.

[67] Nathan Fain. Jtagenum. *27c3: JTAG/Serial/FLASH/PCB Embedded Reverse Engineering Tools and Techniques*, 2010.

[68] Joe Grand. Jtagulator. *Grand Idea Studio*, 2013.

[69] Craig Heffner. Reversing the wrt120ns firmware obfuscation. *http://www.devttys0.com/2014/02/reversing-the-wrt120n-firmware-obfuscation*, February, 2014.

[70] Benedikt Driessen, Ralf Hund, Carsten Willems, Carsten Paar, and Thorsten Holz. Don't trust satellite phones: A security analysis of two satphone standards. In *33rd IEEE Symposium on Security and Privacy (S&P 2012)*, pages 128–142. IEEE Computer Society, 2012.

[71] Roel Verdult, Flavio D. Garcia, and Barış Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *22nd USENIX Security Symposium (USENIX Security 2013)*. USENIX Association, 2013.

[72] David Oswald, Daehyun Strobel, Falk Schellenberg, Timo Kasper, and Christof Paar. When reverse-engineering meets side-channel analysis–digital lockpicking in practice. In *20th International Conference on Selected Areas in Cryptography (SAC 2013)*, Lecture Notes in Computer Science. Springer-Verlag, 2013.

[73] Daehyun Strobel, Benedikt Driessen, Timo Kasper, Gregor Leander, David Oswald, Falk Schellenberg, and Christof Paar. Fuming acid and cryptanalysis: Handy tools for overcoming a digital locking and access control system. In *33rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2013)*, volume 8042 of *Lecture Notes in Computer Science*, pages 147–164. Springer-Verlag, 2013.

[74] Michael Weiner, Maurice Massar, Erik Tews, Dennis Giese, and Wolfgang Wieser. Security analysis of a widely deployed locking system. In *20th ACM Conference on Computer and Communications Security (CCS 2013)*, pages 929–940. ACM, 2013.

[75] Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult. Wirelessly lockpicking a smart card reader. *International Journal of Information Security*, 13(5):403–420, 2014.

[76] Roel Verdult. *The (in)security of proprietary cryptography*. PhD thesis, Radboud University, The Netherlands and KU Leuven, Belgium, April 2015.

[77] Hovav Shacham, E Buchanan, R Roemer, and S Savage. Return-oriented programming: Exploits without code injection. *Black Hat USA Briefings (August 2008)*, August, 2008.

[78] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[79] T Lomas, Li Gong, J Saltzer, and R Needhamn. Reducing risks from poorly chosen keys. *ACM SIGOPS Operating Systems Review*, 23(5):14–18, 1989.

[80] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. *NIST Special Publication (800-22)*, 22:1–152, 2001.

[81] Steve Christey and Chris Wysopal. Responsible vulnerability disclosure process. http://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00, 2002. RFC draft.