Research on RSA Padding Identification Method in IoT Firmwares

Chao Mu\textsuperscript{1,2,3*}, Ming Yang\textsuperscript{1,2,3}, Zhenya Chen\textsuperscript{1,2,3} and Biao Wang\textsuperscript{1,2,3}

\textsuperscript{1} Qilu University of Technology (Shandong Academy of Sciences), Jinan, Shandong 250014, China
\textsuperscript{2} Shandong Computer Science Center (National Supercomputer Center in Jinan), Jinan, Shandong 250014, China
\textsuperscript{3} Shandong Provincial Key Laboratory of Computer Networks, Jinan, Shandong 250014, China

*Corresponding author’s e-mail: muchao@sdas.org

Abstract. Cryptography is a fundamental mechanism for security, various cryptographic algorithms are widely used in IoT firmwares, and identifying cryptographic algorithms is paramount for IoT security. RSA algorithm is the most popular asymmetric key cryptographic algorithm in use. Insecure RSA padding mode may result in padding oracle attack, causing certificate leakage, random number prediction, and plaintext recovery. In this paper, we show how to extract binaries from IoT firmwares, then describe 7 padding modes in RSA algorithm, and identify RSA padding in two methods -- function-name matching and IR expression constant analysis. We use Angr VEX model to translate specific code block into intermediate representation, and analyze their constant value to track RSA parameter int padding. We collect and filter 159 firmwares from 6 different vendors, analyze 7 kinds of RSA paddings using function-name matching, and 4 kinds of RSA paddings in RSA functions which contain parameter int padding using IR expression constant analysis. Then we identify 335 RSA paddings, and find out that flawed padding is still broadly in use in IoT environment.

1. Introduction

IoT firmware contains many cryptographic algorithms, many security mechanisms such as authentication, storage, transmission and communication are hardly to achieve without them. However, many IoT firmwares choose inappropriate cryptographic algorithms, causing some critical problems like Mirai botnet\cite{1}, brute-force attack, encryption key recovery, and leakage of personal private information\cite{2}. For instance, firmwares using vulnerable versions of the OpenSSL\cite{3} software may leak secret key and account information under HeartBleed attack\cite{4}. It is not guaranteed that firmwares using the protection of cryptographic algorithms are secure enough, the misuse of cryptographic algorithms still cause serious security problems. So it is necessary and important to identify misused cryptographic algorithms in firmwares.

To investigate defects of cryptographic algorithms in firmwares and improve security, first of all we need to identify which particular cryptographic algorithm is used in firmware binary. Unfortunately, IoT firmwares have different architectures such as X86/64, ARM and MIPS. Their assembly languages have a great difference. And some algorithms are not standard cryptographic algorithms, such as self-defined algorithms or modified algorithms by developer. To solve this
problem, ReFormat[5] and Dispatcher[6] identify cryptographic algorithm using the execution statistics of bitwise and arithmetic instructions method, this method consumes a lot of time and has some false positive results.

The good thing is that most firmwares use standard cryptographic API, for instance, PKCS#1[7] is a standard API contains RSA algorithm functions, provides recommendations for the implementation of public-key cryptography based on the RSA algorithm, covering cryptographic primitives, encryption schemes, signature schemes with appendix, and ASN.1 syntax for representing keys and for identifying the schemes. RSA padding refers to a number of distinct practices in cryptography. Bleichenbacher attack[8] bases on a padding oracle revealing whether or not a decrypted message is correctly padded according to PKCS#1 v1.5. This attack finds the whole plaintext, even when it is unpadded or padded under another scheme, to prevent this attack it is enough to adopt RSA OAEP padding. According to CWE-780[9], software uses the RSA algorithm but does not incorporate Optimal Asymmetric Encryption Padding (OAEP)[10], which might weaken the encryption. The other padding mode besides OAEP like PKCS1 Padding does not use random bytes, makes the plaintext more predictable, thus attacker can take less work to decrypt the data.

In this paper, we focus on how to identify RSA padding in IoT firmwares. First of all we download firmwares from the Internet, extract and filter all binary executable files, then we propose two methods to identify RSA padding: function-name matching and IR expression constant analysis. We chose IDA Pro[11] and Angr[12][13] as our analysis tools, to eliminate the problem of different firmware architectures.

2. Method

As showing in Fig.1, our system uses web crawler to download firmwares from the Internet, then use firmware reversing engineer tools to extract firmware images, filter all binary executable files, load all binaries including ARM or MIPS or X86/64 architectures, identify all RSA paddings in two methods - function-name matching and IR expression constant analysis.

![Figure 1. An overview of RSA padding identification method](image)

2.1 Preprocess

We develop a web crawler to download firmware images from vendor official websites, ftp sites and forum sites. Most of firmware images are .bin format, and we use Binwalk[14] and Firmware-ModKit[15] to extract files from these images. Binwalk and FMK are both very famous and powerful tool for reverse engineering, they can extract different kinds of images into document files, and they support file formats such as squashfs, cramfs, cromfs etc.

Each extracted file is a part of a small operating system component. These files are divided into binary executable file, configuration file (ini/xml etc.), web page file (html/php/js/css etc.), link files etc. The format and file header of these files are not quite identical, in the light of this feature we build a filter command model to choose all binary executable files.

2.2 RSA Padding Identification

Some RSA functions contain padding mode information in their name, like RSA_padding_add_PKCS1_type_1(), RSA_padding_check_PKCS1_type_2(), RSA_padding_check_PKCS1_OAEP() etc. Using function-name matching method we can detect RSA padding value correctly and efficiently. We call this kind of RSA function like RSA_padding_*() as Type I Function.
Some RSA functions contain parameter ‘int padding’, and this value is passing from their caller functions. To track the value of parameter int padding, at first we locate function like RSA(args, int padding), then check the cross-reference functions using reversing engineer tool like IDA Pro, and locate its calling code block address. Then we use Angr VEX module which borrowed from Valgrind[16], convert this code block into IR expression, the VEX IR abstracts away several architecture differences when dealing with different architectures, allowing a single analysis to be run on all of them. After that we analyze the IR expressions, filter out the expressions containing a constant value, and compare with the defined RSA padding interval value, if this constant value is in the range of 0x1-0x7, and its operation of IR expression is read or write a register or memory value, then we mark this IR expression as a suspicious instruction. Finally we analyze the operand and opcode of this instruction, and analyze transfer path of the parameter in code block, if this instruction meets the conditions of parameter assignment, then the specific value of this parameter can be determined. In our analysis method, we call this kind of RSA function like RSA(args, int padding) as Type II Function, and its caller function as Type III Function.

3. Results

3.1 Preprocess
We download 552 firmwares from 6 different vendors on the Internet, in which 159 of them can be filtered and analyzed further. Running filter command model we obtain all 159 firmwares which contain 976 binary executable files. Firmware analysis statistics is showing in Table 1.

| Vendor   | Firmware Download | Firmware Filter | Binary Executable File |
|----------|-------------------|-----------------|------------------------|
| 360      | 26                | 15              | 115                    |
| Belkin   | 107               | 36              | 189                    |
| Tenda    | 72                | 22              | 86                     |
| Netcore  | 9                 | 3               | 29                     |
| TP-Link  | 298               | 64              | 397                    |
| Linksys  | 40                | 19              | 160                    |
| Total    | 552               | 159             | 976                    |

3.2 RSA Padding Identification
At first, we load all binaries into IDA Pro, refer to the definition of RSA function in the standard encryption library, and write IDAPython[17] script to perform identification. Then we export all Type I&II Function results, Type II&III Function call relations and several important code block addresses. We use Angr VEX model to convert certain code blocks into IR expressions and analyze further.

3.2.1 Function-name Matching. We take OpenSSL library as research object. It defines several padding modes in RSA algorithm, and the padding value definitions are written in OpenSSL/rsa.h.

```c
/* RSA Padding definitions in rsa.h */
#define RSA_PKCS1_PADDING        1
#define RSA_SSLV23_PADDING       2
#define RSA_NO_PADDING           3
#define RSA_PKCS1_OAEP_PADDING   4
#define RSA_X931_PADDING         5
#define RSA_PKCS1_PSS_PADDING    6
#define RSA_PKCS1_WITH_TLS_PADDING 7
```

OpenSSL standard library defines RSA functions that contain padding values in their name directly. Part of Type I Function definitions are written in OpenSSL/include/OpenSSL/rsa.h, which are showed as below.
We collect 16 Type I Functions in rsa.h, output a function list result, as the basis of search keywords written by IDAPython script.

3.2.2 IR Expression Constant Analysis. Searching all RSA function which has parameter int padding in OpenSSL APIs. Type II Function definition in Openssl/rsa.h as show below.

We collect 4 Type II Functions, combined with 16 Type I Functions, write IDAPython identification script, and obtain all 335 RSA padding statistics as shown in Table 2, including 267 RSA padding values from Type I Function and 68 RSA functions from Type II Function.

Table 2. Type I &II Function Statistics.

| Vendor | Binaries | PKCS1 | SSLV23 | NO | OAEP | X931 | PSS | TLS | Type II Function |
|--------|----------|-------|--------|----|------|------|-----|-----|-----------------|
| 360    | 115      | 0     | 0      | 0  | 0    | 0    | 0   | 0   | 0               |
| Belkin | 189      | 0     | 0      | 0  | 0    | 0    | 0   | 0   | 2               |
| Tenda  | 86       | 4     | 2      | 2  | 2    | 1    | 0   | 0   | 4               |
| Netcore| 29       | 0     | 0      | 0  | 0    | 0    | 0   | 0   | 0               |
| TP-Link| 397      | 68    | 34     | 34 | 34   | 0    | 0   | 0   | 52              |
| Linksys| 160      | 20    | 10     | 10 | 10   | 0    | 0   | 0   | 10              |
| Total  | 976      | 92    | 46     | 46 | 36   | 1    | 0   | 68  |                 |

Based on the statistical results in table 2, we need to analyze all 68 Type II Function further and identify each RSA padding value. For instance, TP-Link firmware binary file ‘httpd’ has Type II Function RSA_public_decrypt(), we check cross-reference to get its Type III Function RSA_verify(), then locate its code block start address 0x5757AC and translate this block into IR expression. At address 0x5757B0 in Line 06, PUT(r2) = 0x00000001, a constant value and a register write operation, which represents updating r2 register with the value 0x1, and r2 register storing the fifth parameter ‘int padding’ after our parameter transfer analysis.

/* TP-Link httpd 0x5757AC    Part IR Expression in RSA_verify() */
00 | ------ IMark(0x5757ac, 4, 0) ------
01 | t14 = GET;i32(bp)
05 | ------ IMark(0x5757b0, 4, 0) ------
06 | PUT(r2) = 0x00000001
07 | PUT(pc) = 0x005757b4
.....
37 | PUT(lr) = 0x005757d8
38 | ------ IMark(0x5757d4, 4, 0) ------
NEXT: PUT(pc) = t33; Ijk_Call
RSA_verify() function is defined in OpenSSL source code openssl/cryptorsa/rsasign.c as shown below. Line 351-352 RSA_verify() calls RSA_public_decrypt() and its parameter ‘int padding’ equals constant value RSA_PKCS1_PADDING, and equals 1 which defined in rsa.h. It is proved that this IR expression constant analysis method is feasible.

```c
/* openssl/crypto/rsa/rsasign.c line 351-352 */
len = RSA_public_decrypt((int)siglen, sigbuf, decrypt_buf, rsa, RSA_PKCS1_PADDING);
```

In Tenda ucloud file, Type III Function sub_47874C calls Type II Function RSA_public_decrypt(), we translate code block start at 0x478864 into IR expression and find that at address 0x47886C in Line 08, PUT(r2) = 0x3, which is a suspicious constant value.

```c
/* Tenda ucloud 0x478864 Part IR Expression in sub_47874C() */
00 | ------ IMark(0x478864, 4, 0) ------
01 | PUT(pc) = 0x00478868
07 | ------ IMark(0x47886c, 4, 0) ------
08 | PUT(r2) = 0x00000003
09 | PUT(pc) = 0x00478870
......
30 | t23 = GET;132(r17)
31 | PUT(r7) = t23
```

Through decompiling MIPS instruction to pseudo code using Ghidra[18] tool, we find that parameter int padding equals 0x3, thus this method is still practicable to recognize int padding in nonstandard sub functions.

```c
/* ucloud sub_47874C() function pseudo code */
if (iVar1 != 0) {
    iVar1 = RSA_public_decrypt(param_3,param_2,*(*uchar**)(type + 0x1c),rsa,3);
}
```

We identify all RSA padding values in Type II Function and list detailed information as show in Table 3. RSA_verify() calls RSA_public_decrypt() and RSA_sign() calls RSA_private_encrypt() which are the most common application scenario in several different binary files. Int padding = RSA_PKCS1_PADDING is still broadly used in IoT firmwares, which may cause security risks.

| Vendor   | Binary File   | Type II Function                     | Type III Function | Int padding |
|----------|--------------|--------------------------------------|------------------|-------------|
| belkin   | transmissio 1| RSA_private_encrypt                  |                  | 1           |
| n-remote |              | RSA_public_decrypt                   |                  | 1           |
| dctcs    | 1            | RSA_private_encrypt                  | RSA_sign         | 1           |
|          | 2            | RSA_public_decrypt                   | RSA_verify       | 1           |
| tenda    | ucloud       | RSA_private_encrypt                  | RSA_sign         | 1           |
|          |              | RSA_public_decrypt                   |                  | 1           |
| netcore  | httpd        | RSA_private_encrypt                  |                  | 1           |
|          |              | RSA_public_decrypt                   |                  | 1           |
| tp-link  | WTP          | RSA_private_encrypt                  |                  | 1           |
|          |              | RSA_public_decrypt                   |                  | 1           |
|          |              | RSA_private_encrypt                  |                  | 1           |
|          |              | RSA_public_decrypt                   |                  | 1           |
| linksys  | selfsign     | RSA_private_encrypt                  |                  | 1           |
|          |              | RSA_public_decrypt                   |                  | 1           |
| openssl  |              | RSA_private_encrypt                  |                  | 1           |

We identify all RSA padding values in Type II Function and list detailed information as show in Table 3. RSA_verify() calls RSA_public_decrypt() and RSA_sign() calls RSA_private_encrypt() which are the most common application scenario in several different binary files. Int padding = RSA_PKCS1_PADDING is still broadly used in IoT firmwares, which may cause security risks.

Table 3. Parameter Int Padding Identification In Type II&III Function.
4. Conclusions
In this paper, we extract and filter IoT firmwares from different vendors, propose two RSA padding identification methods, including function-name matching and IR expression constant analysis. We utilize Angr VEX model to translate specific code block into IR expression and track parameter int padding. Using above methods, we obtain 267 RSA paddings using function-name matching method and 68 RSA paddings in parameter using IR expression constant analysis method. Our methods can identify RSA padding correctly and efficiently, and the results show that flawed RSA padding is still used in IoT firmwares, which represents a security risk.

Acknowledgment
We would like to thank anonymous reviews for their valuable comments and helpful suggestions. The work is supported by the Fundamental Research Fund of Shandong Academy of Sciences (NO.2018:12-16).

References
[1] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., ... Zhou, Y. (2017) Understanding the mirai botnet. In: USENIX Security Symposium. Vancouver. pp. 1093-1110.
[2] Zhao, K., Ge, L. (2013) A Survey on the Internet of Things Security. In: International Conference on Computational Intelligence and Security. Emeishan. pp. 663-667.
[3] OpenSSL. (2020) OpenSSL Cryptography and SSL/TLS Toolkit. https://www.openssl.org/.
[4] Durumeric, Z., Payer, M., Paxson, V., Kasten, J., Beekman, J. (2014) The Matter of Heartbleed. In: ACM Internet Measurement Conference. Vancouver. pp. 475-488.
[5] Wang, Z., Jiang, X., Cui, W., Wang, X., Grace, M. (2009) ReFormat: Automatic Reverse Engineering of Encrypted Messages. In: European Symposium on Research in Computer Security. Saint-Malo. pp. 200-215.
[6] Caballero, J., Poosankam, P., Kreibich, C., Song, D.X. (2009) Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In: ACM Conference on Computer and Communications Security, CCS 2009. Chicago. pp. 621-634.
[7] RFC 3447. (2003) Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. https://tools.ietf.org/html/rfc3447.
[8] Bleichenbacher, D. (1998) Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: International Cryptology Conference. Santa Barbara. pp. 1-12.
[9] CWE 780. (2012) Use of RSA Algorithm without OAEP. https://cwe.mitre.org/data/definitions/780.html.
[10] Fujisaki, E., Okamoto, T., Pointcheval, D., Stern, J. (2001) RSA-OAEP Is Secure under the RSA Assumption. In: International Cryptology Conference. Santa Barbara. pp. 260-274.
[11] Hex-Rays. (2020) About IDA. https://www.hex-rays.com/products/ida/.
[12] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Vigna, G. (2016) SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In: IEEE Symposium on Security and Privacy. San Jose. pp. 138-157.
[13] angr. (2016) angr.io. https://angr.io.
[14] Craig Heffner. (2010) BINWALK FIRMWARE EXTRACTION TOOL IN THE WORLD. https://www.refirmlabs.com/binwalk/.
[15] rampageX. (2013) firmware-mod-kit. https://github.com/rampageX/firmware-mod-kit/.
[16] Valgrind. (2019) Valgrind Home. http://valgrind.org/.
[17] idapython. (2019) IDA Python project for Hex-Ray's IDA Pro. https://github.com/idapython/src.
[18] Research Directorate for NSA's cybersecurity mission. (2019) Ghidra. https://www.nsa.gov/resources/everyone/ghidra/.