Practical hacking AES using the S-box weakness

Version: 1.0

Date: 22-08-2011

Joost Kremers (J.Kremers@student.ru.nl)

Supervised by Dr. Klaus Kursawe (K.Kursawe@cs.ru.nl)

Foreword

I would like to thank the Radboud University Nijmegen (RUN), for giving me the chance to study what I am interested in. The courses I followed at the RUN were highly useful in the pursuit of this thesis.

Also I would like to a extend a special thanks to my classmates for helping me when I was stuck and did not have any new insights. My family was of great help too: motivating me and assisting with readability.

Last but not least I would like to thank Klaus Kursawe for supervising my thesis. Without his help (and previous research [5]) I would have not been able to complete this thesis.

Summary

In this thesis a theoretical weakness to the Advanced Encryption Standard (AES) is investigated. By modifying the algorithms S-box, the original key can be extracted. This weakness was first investigated in a self-implemented version, where-after the researcher attempted to obtain the key from five widely used programs. No key material was found in this attempt, because all of the programs had counter-measurements for code modification.

Keywords: AES, Advanced, Encryption, Standard, S-box, Rijndael.

Index

1. Introduction

In this chapter the main subject of the thesis will be introduced. First AES and a possible weakness will be described, followed by the main thesis questions. The chapter will end with a justification on why it is interesting to investigate AES on weaknesses.

1.1 Problem-description

Advanced Encryption Standard (AES) is a widely used encryption algorithm, by programs like WinRAR. TrueCrypt, OpenSLL. While many papers
study the mathematical weaknesses of this algorithm, this thesis will focus on an implementation weakness. As Kerins and Kursawe [5] described in their paper, a simple S-box value change, can make the encryption key roll out. In this thesis an attempt will be made on exploiting this weakness in real programs. This leads to the main question:

Does the S-box weakness pose a real threat to programs using AES encryption?

If yes, then it is interesting to see what kind of programs and on what scale AES is used:

What programs are at risk by the S-box weakness?

Another question that arises is:

Do you need real knowledge on computer security/algorithms to perform this hack?

In short: in this thesis the main focus lies on the S-box weakness and the attempts to exploit this in real (widely used) programs.

1.2 Justification

As mentioned before, AES is widely used. Programs use AES to ensure confidentiality and integrity. If AES is weakly implemented in this software and vulnerable to S-box hacks, it may lose confidentiality and integrity: it will lose its purpose. Therefore it is essential that the EYESimplementation is solid, leaving only possible mathematical weaknesses.

For myself this is an interesting topic to start my research in the security sector. Next year I am planning to start my master at the Kerckhoffs Institute, specialized in computer security. To make a jump-start in this sector this seems an excellent research topic.

1.3 Theoretical framework

The AES algorithm has been designed by Joan Daemen and Vincent Rijmen. The algorithm is based on a network called substitution-permution network. [3] In such a network, blocks of the plain-text and the key are taken as input. These are then taken through several rounds of substitution boxes (S-boxes) and permutation boxes (P-boxes).

In AES a 4x4 array of bytes is used, which is named a state.

AES uses the following scheme [3]:

```
1. KeyExpansion: round keys are derived from the key.

2. Initial round
   1. AddRoundKey: state XOR roundkey.

3. Round (repeated 9 times)
   1. SubBytes: S-box.
   2. Shiftrows: each row of the state if moved around.
   3. MixColumns.
   4. AddRoundKey: state XOR roundkey.

4. Final Round:
   1. SubBytes: S-box.
   2. Shiftrows: each row of the state if moved around.
   3. AddRoundKey: state XOR roundkey.
```

In AES, the S-box (Substitution-box) is a look-up table for several encryption-steps. The box is a 16x16 array. When the substitution sets in, the hexadecimal value that is to be encrypted, is the index for the S-box. The new value is the output of the S-box.

The Rijndael S-box has the following values:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF

The implementation weakness introduced by Kerins and Kursawe [5] uses the last roundkey to extract the key. The idea is to change all values in the S-box to 00, which has the following consequences to the algorithm:

Because every SubBytes step now empties the state, the ShiftRows and the MixColumns step can be ignored. The only thing that happens every round is the addition of the (known) round-constants (Rcon) to the round-key.
In Kerins and Kursawe's research [5] a formula is presented for extracting the original key from the new output from the program:

$$k_0 = w_{40} + R_0^{(128)} = w_{40} + D2$$

$$k_1 = w_{41} + R_1^{(128)} = w_{41} + 9C$$

$$k_2 = w_{42} + w_{40} + R_2^{(128)} = w_{42} + w_{40} + CC$$

$$k_3 = w_{43} + w_{41} + R_3^{(128)} = w_{43} + w_{41} + 88$$

Where the original key is $\{k_0, k_1, k_2, k_3\}$, the output of the program is $\{w_{40}, w_{41}, w_{42}, w_{43}\}$. The + operator is a 32 bit XOR operation.

AES-128 and AES-256

AES can operate with a number of key-lengths. AES-128, with a key-length and block-length of 128, is in essence the same as AES-256. Mathematically they are both equally safe, any breakthrough in hacking AES-128 would most certainly lead to a breakthrough in AES-256.

In this thesis the focus will be on AES-128, because this is the most used version. The whole idea behind the S-box weakness also applies to AES-256, where the adjustments have to be made for the length of the blocks and the key.

Cipher modes of operation

There are several block cipher modes of operation that can be used with AES. The basic version is the Electronic CodeBook (ECB) mode. This mode just uses the (round-)key and the plain-text to encrypt the data.

The problem with ECB is that analysts can find patterns: the same plain-text and key will result in the same cipher-text each time. Therefore other cipher modes were introduced: CBC, CFB, OFB, CTR and more. In this thesis only ECB and Cipher-Block Chaining are investigated.

Cipher-Block Chaining (CBC) should make it harder to find patterns in the encryption process. Instead of just encrypting the key with the plain-text, the previous block of cipher-text is also XORed with the plain-text

In normal routines this is a problem with the pattern recognition. But since the S-box changes the value of the state, one small calculation step would make this equal to the ECB-mode.

1.4 Related work

Because of the price of the private key, a lot of research has been done to encryption algorithms and their weaknesses. In this section several methods will be discussed.

One way to retrieve the key is to look at information that can be gained from physical implementation of the algorithm, called side channel attacks. For instance it is possible to measure the time the CPU uses to compute something and with statistical analysis of this data retrieving the key.

Another side channel attack is the differential fault analysis, in which the user inserts unexpected environmental conditions, i.e. voltage overload, magnetic fields etc, to get information about the current encryption state. H. Ziade et al. (2004) [8] present hardware-, software-, simulation- and emulation-based fault injection techniques, which can all be used to test or to hack a system.

Shamir and van Someren (1998) [7] investigated a way to identify RSA-keys in a large set of a data by looking at it's entropy to the rest of the data. Because the key is chosen at random (and the rest of the data is probably not) it shows a large entropy with the rest of the data. By testing the entropy of several blocks of data they were able to find the key in a lot of cases. Shamir and van Someren advised program makers to spread the key over the data, which would make the entropy-density of the program a lot better against entropy based-attacks. This does give some computational overhead.

In the paper presented by Chow et al. [3] a method is presented to hide a key by using tables instead of individual steps, encoding these tables with random bijections and extending the cryptographic boundary beyond the crypto algorithm itself.

Klein (2006) [6] found a way to extract the private key and certificates in RSA by looking for a pattern in the process memory. Each private key and certificate have a known syntax and therefor a search for this syntax can be done. In this thesis it is interesting to look whether the key is outputted to a core-dump if the program crashes.

In the research done by Bogdanov et al [1] a method to retrieve an AES-128 key is shown which reduces the computation complexity from 2^{128} to $2^{126.1}$. Because this still has a large computational complexity, the authors do not think the new method poses any threat to AES.

For most of these attacks you need a lot of security knowledge to a attempt them. The attack presented in this research is a lot simpler and could, in theory, be done by script kiddies.

2. Methods

In this chapter a detailed description of the techniques used to answer the main questions of this thesis, will be given. First a basic implementation will be used to test AES-characteristics. Then a search for programs with an S-box will be done, after which they will be examined in depth.

2.1 Test the hack on a self-implemented version of AES

To understand how AES operates and to test certain changes in the algorithm, the first step of this research is looking at a basic implementation of AES. A basic version implemented by someone else will be used. The code of the program will be verified with the AES specifications [3].

Once the program is running, tests will commence with the following test values:

Plain-text (in ASCII and Hex):

T	w	o		O	n	e		N	i	n	e		T	w	o
54	77	6F	20	4F	6E	65	20	4E	69	6E	65	20	54	77	6F

Key (in ASCII and Hex):

T	h	a	t	s		m	y		K	u	n	g		F	u
54	68	61	74	73	20	6D	79	20	4B	75	6E	67	20	46	75

Following the AES algorithm, the expected output will be:

29	C3	50	5F	57	14	20	F6	40	22	99	B3	1A	02	D7	3A

When the output has been verified the modification can start: the S-box has to be whitened. Because the code is available two options arise: either the S-box is whitened in the code, or an Hex-Editor is used to modify the executable.

Because the S-box has known values, the second option is also very easy. One search for the first four values of the S-box (63 7C 77 7B) and the S-box appears. Replacing all the values in the S-box to 00 finishes this step.

When the original key can be extracted with Kersins and Kursawe's algorithm, the conclusion can be made that the algorithm has an implementation weakness and that developers may have not taken this in account. Widely used programs may have this weakness in them as well and thus they are worth looking in to.

2.2 Find suitable programs to try the hack on

In this thesis the focus lies on widely used programs, therefore a search of the executables on a preinstalled and used computer will give the best results.

To search for S-box's a program called Hex Editor Neo will be used. This program allows you to search in every file on the whole computer. This function is called "Find in Files". The values used for search will be the first four values of the S-box (Hex: 63 7C 77 7B), because it is unlikely that these values occur randomly. The program will search for all .exe and .dll files on the computer: these are the most likely to have S-boxes in them.

Because AES is in a variety of programs, programs for several purposes will be examined.

This thesis will focus on non-system files, because the system files are hard to analyze and editing probably makes the computer crash.

2.3 Attempt to hack the programs

Before actually attempting to hack the found programs a backup of the files will be made: some files may become irreparable. Also all anti-virus and anti spy-ware software must be turned off. This step is just a counter-measurement for possible blockades because of the software alteration.

After making the necessary precautions, the S-box of file will be whitened with an Hex Editor: every value in the S-box (from 63 up to DF) will be put on 00.

After starting the edited file (or file using the .dll), the (possible) output will be analyzed. There are three options:
1) the software starts as if nothing happened and we can find key material in the software itself.
2) the software crashes but gives a buffer output with key material.
3) the software does not give any key material.

Dependent on how the software responses, a test will be created which has the purpose of explaining this behavior. For instance: more (or less) changes with an Hex Editor can be made, Wireshark can be used to analyze network traffic or a core-dump will be investigated. Core-dumps will be created by ADPlus, a Windows Debug tool.

If a key from a widely used program can be obtained this means the software encryption is not safely implemented. Then the weakness that Kerins and Kursawe presented is obviously a weakness to AES.

2.4 Knowledge and hacking

If the exploit is a weakness in widely used programs, other people can use this exploit too. It therefore it is interesting to see whether this is a time-consuming hack or not. Before this research the researcher had little knowledge of AES. He can therefore be seen as a beginner in the hacking scene. The time the researcher invested in this thesis is therefore an measurement for how dangerous this hack may be.

3. Results

In this chapter the results of all steps will be presented.

3.1 Test the hack on a self-implemented version of AES

The first step of the research was to use libraries to implement AES. Unfortunately this was not a success. Java libraries nor C++ libraries functioned as they should have. The C++ library Botan kept on giving Linker errors, while the Crypto++ library gave header errors. After a lot of trial and error, the decision was made to use a full implemented basic version of AES.

The basic version was written by Niyaz PK in C++. After verifying the code, the program was approved for testing purposes. The output received from the known key and known plain-text matched the expected output.

After whitening the S-box in the programming code, the code ran again.

Key (in ASCII and Hex):

T	h	a	t	s		m	y		K	u	n	g		F	u
54	68	61	74	73	20	6D	79	20	4B	75	6E	67	20	46	75

Output (in Hex):

86	68	61	74	EF	20	6D	79	6A	23	14	1A	00	00	2B	0C

It is clearly visible that the output is linked to the Key.

Using the formula supplied by Kerins & Kursawe [5], exactly the original key is retrieved:

$$k_0 = w_{40} + R_0^{(128)} = w_{40} + D2$$
$$k_1 = w_{41} + R_1^{(128)} = w_{41} + 9C$$
$$k_2 = w_{42} + w_{40} + R_2^{(128)} = w_{42} + w_{40} + CC$$
$$k_3 = w_{43} + w_{41} + R_3^{(128)} = w_{43} + w_{41} + 88$$

86 + D2	68	61	74	EF + 9C	20	6D	79
6A + 86 + CC	23 + 68	14 + 61	1A + 74	00 + EF + 88	00 + 20	2B + 6D	0C + 79

=

54	68	61	74	73	20	6D	79	20	4B	75	6E	67	20	46	75

3.2 Find suitable programs to try the hack on

After the Hex Editor Neo's search completed, 340 files appeared to have a S-box. The following file were selected for research:

- WinRAR (WinRAR\Formats\7zxa.dll), a program used for zipping other files.
- FileZilla FTP Client (FileZilla FTP Client\filezilla.exe), a program used to interact with a FTP-server.
- FTD v3.8 (FTDv3.8\FTDv3.exe), a program used for finding binary posts in newsgroups.
- Hitman Pro 3.5 (Hitman Pro 3.5\HitmanPro35_x64.exe), a program specialized in finding trojans and spyware.
- Steam (Steam\steamclient64.dll), a program used for gaming purposes.

These programs fit the description of a widely used program. Most of the the other 335 files were Windows system files. In this research the five programs listed above will be investigated.

Programs that are installed on this PC, but that do not show to have a S-box include: Winamp, TeamSpeak, Mozilla Thunderbird, Mozilla Firefox, OpenOffice, GrabIt, Google Chrome, NetBeans, Skype and Deamon Tools.

3.3 Attempt to hack the programs

WinRAR

After whitening the S-box, the WinRAR does not give us any warning or error that the software may be modified. This may be because the .dlls are not included in a possible check sum.

Because 7zxa.dll is used when handling .7z files, a .7z file is extracted with WinRAR. The extraction went smooth and no errors occurred.

FileZilla FTP Client

When the S-box for FileZilla was edited and the program ran, nothing special happened. When looking around in the program, the change was made to the update feature. Instead of connecting to the update-server, the connection failed:

Error: GnuTLS error -12: A TLS fatal alert has been received.

WireShark says TLS encountered an Alert (20), Bad record MAC.
To understand what happens, the TLS algorithm needs to be taken into account:

```
ClientHello: the client sends a message containing it's TLS-version, a random
number and a list of CipherSuites: it can use.

ServerHello: the server responds with the chosen TLS-version, the chosen
CipherSuite and random number.
```

```
Certificate: the server sends it's certificate.

ServerHelloDone: the handshake message is done.

ClientKeyExchange: this message may contain a PreMasterSecret which is
encrypted with the public key of the server.

MasterSecret: the random numbers, and the PreMasterSecret are used to compute a
common key.

ChangeCipherSpec: the client sends the server the message that every message
from now on will be encrypted with their master secret.

Finished: the client sends an encrypted message to the server, including a hash
and MAC.
```

The point where the software will fail is obviously the last step, because it will encrypt the hash and MAC with the broken S-box. The server tries to decrypt this message, but the output does not match the hypothesis. It is therefore interesting to look at this package and see if key-material can be found. If so, the session can be hijacked.

One of the first interesting things to notice is that TLS_DHE_RSA_WITH_AES_256_CBC_SHA is used. This is extracted from the ServerHello message.

The Finished message is actually sent at the same time as the ChangeCipherSpec-step. Because the key is different each time the authentication is set up, it is impossible to use previous sessions keys. The key material found in the Finished message can be used to hijack the session, if the message can be intercepted before it is sent to the server. Unfortunately this is very hard to do on a Windows PC.

FTD v3.8

The first notable thing when changing the S-box in FTD was that the S-box was not just the basic values. Notice the “FF FF FF FF”-values in the first row:

63	7C	77	7B	F2	6B	6F	C5	30	01	FF	FF	FF	FF	67	2B
FE	D7	AB	76	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF

…

Because the use of the FF's is unknown, another step has been taken to investigate this: first all the values in the S-box were whitened, leaving the FF's intact. This made the program crash, giving a windows system error. Secondly the FF's were whitened as well. The same system error was presented.

In the error box some information about the crash was found and in the additional information possible key material was presented. However, just modifying the S-box's first value to 00, gave the same additional information: meaning no key material was outputted to the error box.

Investigating the core-dump created by ADPlus did not give more insight in why FTD crashed, other than that the KILL command is called straight away. No key material is outputted here either.

Hitman Pro 3.5

After editing the S-box Hitman Pro 3.5 launched like it would normally. Unfortunately no key material was found in the software. Not even using Wireshark when Hitman was communicating with his update server.

Steam

The cloud gaming software did something very sharp: after modifying the S-box in the .dll the software automatically tried to reinstall. The original .dll's were restored in seconds. It is interesting to see how this is achieved: is it a client-side check-sum or maybe the verification of the client with the server failed because of the S-box modification.

Wireshark was used to investigate whether there is a verification-step with the server that fails because of the S-box modification. In the data collected by Wireshark no verification-steps have been found, and therefore no key material could be extracted.

It is highly likely that the client has a check-sum to recognize modifications in the mode. To be sure of this, another alteration of the hex is done. On several occasions several random values were changed. On all occasions the client did an auto reinstall.

3.4 Knowledge and hacking

Understanding AES can be quite time-consuming. But because the actual hack is very easy to perform (+-1 minute) this hack can be very harmful, if the software allows it. Studying the algorithm itself will cost a lot of time if you are not into encryption algorithms.

4. Conclusion

In this chapter the main questions of this thesis will be answered.

By hacking a basic implementation of AES, the theoretical weakness proposed by Kerins and Kursawe is a weakness in practice as well. Actual key material is found from the output of the software. This means that developers have to be aware of this weakness when using the AES algorithm in their software.

Several programs use AES for encryption purposes on a preinstalled computer. Unfortunately none of the software actually outputted key material. Several programs were using check sums (FTD, Steam) to ensure their software was not modified. It is interesting to see how they handle it: FTD's check-sum just makes the software crash, while Steam tries to recover itself by automatically downloading the correct files needed.

WinRAR and Hitman Pro did not give any sign of being aware of the modification made to their software. Not when the program is being used, nor when it is trying to update. Therefor it is very hard to determine if they even use the AES algorithm.

FileZilla is an interesting topic to look at: the encryption using AES is used in the update step. Because it is communicating with a server and a handshake-process is used, it is possible to hijack the session. This is a topic that needs further research.

Other programs like Google Chrome, did not have a visible S-box. It is possible that the developers saw the S-box as a weakness in the code and therefore use AES, but create the S-box in run-time.

The conclusion that can be drawn from this thesis, is that the theoretical weakness presented by Kerins and Kursawe is a real threat in widely used programs. Software developers need to be aware of the risks in code modifications.

5. Discussion

In step one of this research there was some trouble involving the encryption libraries. Most errors seemed to be Windows related, therefore it would be interesting to see whether the libraries can be used on Linux. Other software that is commonly used on Linux is hard to get your hands on in Windows, or completely useless.

Also it is feasible that there are programs that are vulnerable for the S-box hack that are not investigated in this thesis. It may also be interesting to see whether old versions of the investigated software behaved the same.

Combining the issues listed above, it would make sense to redo this research on Linux on other programs.

Many programmers seem to be aware of the fact that the S-box is a weakness to AES, so they try to protect it. One of the biggest flaws found by this research was the unprotected S-box in Filezilla, possibly leading to a TLS-leak. A full research on this topic should be done to see whether TLS can be broken or not.

7. Literature

[1] A. Bogdanov, D. Khovratovich and C. Rechberger. “Biclique Cryptoanalysis of the Full AES”

[2] S. Chow, P. Eisen, H. Johnson and P.C. Van Oorschot. “White-Box Cryptography and an AES Implementation”.

[3] J. Daemen and V. Rijmen. “AES Proposal: Rijndael” (2002) http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf on 25-02-2011

[4] FIPS PUB 197. “Advanced Encryption Standard (AES)" http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf on 25-02-2011(2001)

[5] T. Kerins and K. Kursawe. “A Cautionary Note on Weak Implementations of Block Ciphers”

[6] T. Klein. “All your private keys are belong to us” (2006)

[7] A. Shamir and N. van Someren. “Playing hide and seek with stored keys” (1998)

[8] H. Ziade, R. Ayoubi and R. Velazco. “A Survey on Fault Injection Techniques”. *The International Arab Journal of Information Technology Vol. 1, No. 2*. (2004)