EXPLOITING A SHA1 WEAKNESS IN PASSWORD CRACKING

About me

- Name: Jens Steube
- Nick: atom
- Coding Projects:
 - hashcat / oclHashcat
- Security Research:
 - Searching for exploitable security holes in OSS and non-OSS Software
 - Reported and worked together with the developers to fix them
 - See Bugtraq / Debian Security Advisory
- Work Status: Employed as Coder, but not crypto- or security-relevant
- Weakness found in 1st quarter of 2011

What we should know about SHA1

- SHA1 is processed sequentially
 - Each block of input data that is processed has a fixed size of 512 bit
 - This block is represented as an array of sixteen 32-bit words
 - We will call this array W[]
- The input data is expanded by another 2048 bits of data
 - This expanded data is generated out of the input data
 - We call this phase "Word-expansion"
- Both input and expanded data is used within 80 steps of SHA1 functions
 - These steps and their inclusion of SHA1 specific function is the major part of SHA1
 - We will not focus on them

SHA1 Transform per Instructions

Word-Expansion	Instruction count	t
XOR	3	16 – 79
ROTATE	1	16 – 79
SHA1 Steps	Instruction count	t
SHA1 Step F1	1	0 - 19
SHA1 Step F2	2	20 - 39
SHA1 Step F3	2	40 - 59
SHA1 Step F4	2	60 – 79
Final Steps	Instruction count	t
ADD	4	80

Word-Expansion

- Word-Expansion is a phase of the SHA1 transformation
- Its purpose is to generate a bigger volume of data out of the input data
- This is where the weakness is located in SHA1
- Input data is mixed up using the following set of logical instructions:

$$W[t] = R((W[t-3] \land W[t-8] \land W[t-14] \land W[t-16]), 1)$$

- W[0].. W[15] is filled with the input data
- By iterating t from 16 to 79, 2048 additional bits are generated

4. Dec 2012

Word-Expansion, unrolled view

```
W[16] = R((W[13] \land W[8] \land W[2] \land W[0]), 1)
                                                                       W[30] = R((W[27] \land W[22] \land W[16] \land W[14]), 1)
W[17] = R((W[14] \land W[9] \land W[3] \land W[1]), 1)
                                                                       W[31] = R((W[28] \land W[23] \land W[17] \land W[15]), 1)
W[18] = R((W[15] \land W[10] \land W[4] \land W[2]), 1)
                                                                       W[32] = R((W[29] \land W[24] \land W[18] \land W[16]). 1)
W[19] = R((W[16] \land W[11] \land W[5] \land W[3]), 1)
                                                                       W[33] = R((W[30] \land W[25] \land W[19] \land W[17]), 1)
W[20] = R((W[17] \land W[12] \land W[6] \land W[4]), 1)
                                                                       W[34] = R((W[31] \land W[26] \land W[20] \land W[18]), 1)
W[21] = R((W[18] \land W[13] \land W[7] \land W[5]), 1)
                                                                       W[35] = R((W[32] \land W[27] \land W[21] \land W[19]), 1)
W[22] = R((W[19] \land W[14] \land W[8] \land W[6]), 1)
                                                                       W[36] = R((W[33] \land W[28] \land W[22] \land W[20]), 1)
W[23] = R((W[20] \land W[15] \land W[9] \land W[7]), 1)
                                                                       W[37] = R((W[34] \land W[29] \land W[23] \land W[21]), 1)
W[24] = R((W[21] \land W[16] \land W[10] \land W[8]). 1)
                                                                       W[38] = R((W[35] \land W[30] \land W[24] \land W[22]), 1)
W[25] = R((W[22] \land W[17] \land W[11] \land W[9]), 1)
                                                                       W[39] = R((W[36] \land W[31] \land W[25] \land W[23]), 1)
W[26] = R((W[23] \land W[18] \land W[12] \land W[10]), 1)
                                                                       W[40] = R((W[37] \land W[32] \land W[26] \land W[24]), 1)
W[27] = R((W[24] \land W[19] \land W[13] \land W[11]), 1)
                                                                       W[41] = R((W[38] \land W[33] \land W[27] \land W[25]), 1)
W[28] = R((W[25] \land W[20] \land W[14] \land W[12]), 1)
W[29] = R((W[26] \land W[21] \land W[15] \land W[13]), 1)
                                                                       W[79] = R((W[76] \land W[71] \land W[65] \land W[63]), 1)
```

How to exploit this

- The password candidate generator needs to hold W[1]..W[15] fixed
- Outside the loop precompute W[16]..W[79] ignoring the unknown W[0]
 - We call this precomputed buffer PW[]
- Inside the loop W[0] is changed
 - Since the Word-Expansion process is using XOR, we can apply W[0] to the precomputed buffer at a later stage
 - Using XOR is the root of the problem
 - Logical instructions cannot overflow, but arithmetic ones can
 - If the Word-Expansion had used ADD, it would have been impossible to exploit it
- When iterating W[0] changes is finished, W[1]..W[15] can be changed
- Restart the process with the next precomputed value of W[16]..W[79]

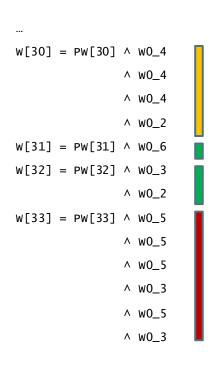
PW[16]..PW[79] in the outer loop

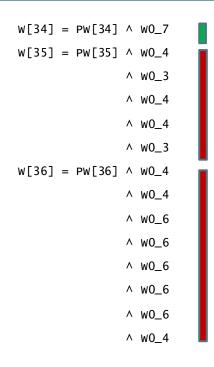
```
PW[30] = R((PW[27] \land PW[22] \land PW[16] \land W[14]), 1)
PW[16] = R((W[13] \land W[8] \land W[2] \land W[0]), 1)
PW[17] = R((W[14] \land W[9] \land W[3] \land W[1]), 1)
                                                                    PW[31] = R((PW[28] \land PW[23] \land PW[17] \land W[15]), 1)
PW[18] = R((W[15] \land W[10] \land W[4] \land W[2]), 1)
                                                                    PW[32] = R((PW[29] \land PW[24] \land PW[18] \land PW[16]). 1)
PW[19] = R((PW[16] \land W[11] \land W[5] \land W[3]), 1)
                                                                    PW[33] = R((PW[30] \land PW[25] \land PW[19] \land PW[17]), 1)
PW[20] = R((PW[17] \land W[12] \land W[6] \land W[4]), 1)
                                                                    PW[34] = R((PW[31] \land PW[26] \land PW[20] \land PW[18]), 1)
PW[21] = R((PW[18] \land W[13] \land W[7] \land W[5]), 1)
                                                                    PW[35] = R((PW[32] \land PW[27] \land PW[21] \land PW[19]), 1)
PW[22] = R((PW[19] \land W[14] \land W[8] \land W[6]), 1)
                                                                    PW[36] = R((PW[33] \land PW[28] \land PW[22] \land PW[20]), 1)
PW[23] = R((PW[20] \land W[15] \land W[9] \land W[7]), 1)
                                                                    PW[37] = R((PW[34] \land PW[29] \land PW[23] \land PW[21]), 1)
PW[24] = R((PW[21] \land PW[16] \land W[10] \land W[8]), 1)
                                                                    PW[38] = R((PW[35] \land PW[30] \land PW[24] \land PW[22]), 1)
PW[25] = R((PW[22] \land PW[17] \land W[11] \land W[9]), 1)
                                                                    PW[39] = R((PW[36] \land PW[31] \land PW[25] \land PW[23]), 1)
PW[26] = R((PW[23] \land PW[18] \land W[12] \land W[10]), 1)
                                                                    PW[40] = R((PW[37] \land PW[32] \land PW[26] \land PW[24]), 1)
PW[27] = R((PW[24] \land PW[19] \land W[13] \land W[11]), 1)
                                                                    PW[41] = R((PW[38] \land PW[33] \land PW[27] \land PW[25]), 1)
PW[28] = R((PW[25] \land PW[20] \land W[14] \land W[12]), 1)
PW[29] = R((PW[26] \land PW[21] \land W[15] \land W[13]), 1)
                                                                    PW[79] = R((PW[76] \land PW[71] \land PW[65] \land PW[63]), 1)
```

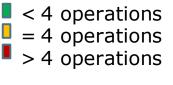
W[0] in the inner loop

```
W0\_1 = R(W[0], 1)
W0_2 = R(W[0], 2)
                                                                                                                                                                                                                                          For 1..20 compute R(W[0], i)
W020 = R(W[0], 20)
W[16] = \frac{R((W[13] \land W[8] \land W[2] \land W[0]), 1)}{R(W[16] \land W[16])} = PW[16] \land W_{0}
W[17] = R((W[14] \land W[9] \land W[3] \land W[1]), 1) = PW[17]
W[18] = R((W[15] \land W[10] \land W[4] \land W[2]), 1) = PW[18]
W[19] = R((W[16] \land W[11] \land W[5] \land W[3]), 1) = PW[19] \land W0_2
W[20] = R((W[17] \land W[12] \land W[6] \land W[4]), 1) = PW[20]
W[21] = R((W[18] \land W[13] \land W[7] \land W[5]), 1) = PW[21]
W[22] = \frac{R((W[19] \land W[14] \land W[8] \land W[6]), 1)}{R(W[19] \land W[14] \land W[8] \land W[6]), 1} = PW[22] \land WO_3
W[23] = R((W[20] \land W[15] \land W[9] \land W[7]), 1) = PW[23]
W[24] = \frac{R((W[21] \land W[16] \land W[10] \land W[8]), 1)}{R(W[21] \land W[16] \land W[10] \land W[
W[25] = \frac{R((W[22] \land W[17] \land W[11] \land W[9]), 1)}{R(W[25] \land WO_4)} = PW[25] \land WO_4
W[26] = R((W[23] \land W[18] \land W[12] \land W[10]), 1) = PW[26]
```

Word-Expansion using precompute







Number of Operations:

$$W[16] = 1$$

 $W[17] = 0$

$$W[33] = 6$$

$$W[75] = 4703$$

...

What we should know about XOR

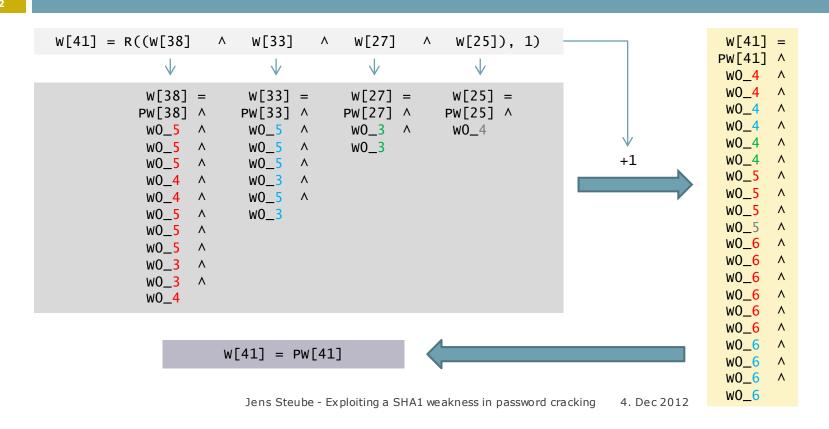
- XORing a value to itself, results in 0
- XORing a value with 0, results in the same value

Conclusion:

- We can ignore many XOR operations in order to optimize the procedure
- We can do this if the sum of a specific value is even

A Perl script to automate this process can be found in the link section

Word-Expansion / XOR zeros



Word-Expansion / XOR groups

```
W[36] = PW[36] \wedge W0_6 \wedge W0_4
W[51] = PW[51] \wedge W0_6 \wedge W0_4
W[62] = PW[62] \wedge W0_6 \wedge W0_4 \wedge W012 \wedge W0_8
const int w0_6_{--}w0_4 = w0_6 \wedge w0_4
W[36] = PW[36] \wedge W0_6__W0_4
W[51] = PW[51] \wedge W0_6_W0_4
W[62] = PW[62] \wedge W0_6 W0_4 \wedge W012 \wedge W0_8
```

Final optimized Word-Expansion

Reference Impl.

```
 w[16] = R((w[13] \land w[8] \land w[2] \land w[0]), 1) 
 w[17] = R((w[14] \land w[9] \land w[3] \land w[1]), 1) 
 w[18] = R((w[15] \land w[10] \land w[4] \land w[2]), 1) 
 w[19] = R((w[16] \land w[11] \land w[5] \land w[3]), 1) 
 w[20] = R((w[17] \land w[12] \land w[6] \land w[4]), 1) 
 w[21] = R((w[18] \land w[13] \land w[7] \land w[5]), 1) 
 w[22] = R((w[19] \land w[14] \land w[8] \land w[6]), 1) 
 w[23] = R((w[20] \land w[15] \land w[9] \land w[7]), 1) 
 w[24] = R((w[21] \land w[16] \land w[10] \land w[8]), 1) 
 w[25] = R((w[22] \land w[17] \land w[11] \land w[9]), 1) 
 w[26] = R((w[23] \land w[18] \land w[12] \land w[10]), 1) 
 w[27] = R((w[24] \land w[19] \land w[13] \land w[11]), 1) 
 w[28] = R((w[25] \land w[20] \land w[14] \land w[12]), 1) 
 w[29] = R((w[26] \land w[21] \land w[15] \land w[13]), 1 
 w[30] = R((w[27] \land w[22] \land w[16] \land w[14]), 1)
```

Optimized Impl.

```
W[16] = PW[16] ^ W0_1
W[17] = PW[17]
W[18] = PW[18]
W[19] = PW[19] ^ W0_2
W[20] = PW[20]
W[21] = PW[21]
W[22] = PW[22] ^ W0_3
W[23] = PW[23]
W[24] = PW[24] ^ W0_2
W[25] = PW[25] ^ W0_4
W[26] = PW[26]
W[27] = PW[27]
W[28] = PW[28] ^ W0_5
W[29] = PW[29]
W[30] = PW[30] ^ W0_4 ^ W0_2
```

SHA1 instruction count; Unoptimized

Section	Instruction count	t
Word-Expansion	256	16 - 79
SHA1 Step F1	140	0 - 19
SHA1 Step F2	160	20 - 39
SHA1 Step F3	160	40 - 59
SHA1 Step F4	160	60 - 79
Final Add	4	80

tal	880
-----	-----

SHA1 instruction count; Known optimizations

Section	Instruction count	t
Word-Expansion	240	16 - 75
SHA1 Step F1	140	0 - 19
SHA1 Step F2	160	20 - 39
SHA1 Step F3	160	40 - 59
SHA1 Step F4	128	60 - 75

Total	828
-------	-----

SHA1 instruction count; Exploiting SHA1's XOR weakness

Section	Instruction count	t
Word-Expansion	106	16 - 75
SHA1 Step F1	140	0 - 19
SHA1 Step F2	160	20 - 39
SHA1 Step F3	160	40 - 59
SHA1 Step F4	128	60 - 75

Total	694
-------	-----

Final comparision

Section	Instruction count	Optimization
Unoptimized	880	0 %
- Known optimizations	828	5.1 %
- This weakness, exploited	694	21.1 %

Files for download

Download here: https://hashcat.net/p12/

- This presentation
- XORzero generator Perl script
- Full code results from slides

Questions?

Feel free to contact me!

- via Twitter: @hashcat
- via Hashcat Forum: https://hashcat.net/forum/
- via IRC: Freenode #hashcat
- via Email: atom at hashcat.net