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

Jens Steube - Exploiting a SHA1 weakness in password cracking  4. Dec 2012
## SHA1 Transform per Instructions

<table>
<thead>
<tr>
<th>Word-Expansion</th>
<th>Instruction count</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>3</td>
<td>16 – 79</td>
</tr>
<tr>
<td>ROTATE</td>
<td>1</td>
<td>16 – 79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SHA1 Steps</th>
<th>Instruction count</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1 Step F1</td>
<td>1</td>
<td>0 – 19</td>
</tr>
<tr>
<td>SHA1 Step F2</td>
<td>2</td>
<td>20 – 39</td>
</tr>
<tr>
<td>SHA1 Step F3</td>
<td>2</td>
<td>40 – 59</td>
</tr>
<tr>
<td>SHA1 Step F4</td>
<td>2</td>
<td>60 – 79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Final Steps</th>
<th>Instruction count</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>4</td>
<td>80</td>
</tr>
</tbody>
</table>
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:
  \[ W[t] = R((W[t-3] \oplus W[t-8] \oplus W[t-14] \oplus W[t-16]), 1) \]
- \( W[0] \ldots W[15] \) is filled with the input data
- By iterating \( t \) from 16 to 79, 2048 additional bits are generated
Word Expansion, unrolled view

\begin{align*}
W[30] &= R((W[27] \land W[22] \land W[16] \land W[14]), 1) \\
W[31] &= R((W[28] \land W[23] \land W[17] \land W[15]), 1) \\
W[34] &= R((W[31] \land W[26] \land W[20] \land W[18]), 1) \\
W[36] &= R((W[33] \land W[28] \land W[22] \land W[20]), 1) \\
W[37] &= R((W[34] \land W[29] \land W[23] \land W[21]), 1) \\
W[40] &= R((W[37] \land W[32] \land W[26] \land W[24]), 1) \\
W[41] &= R((W[38] \land W[33] \land W[27] \land W[25]), 1) \\
\ldots \\
W[79] &= R((W[76] \land W[71] \land W[65] \land W[63]), 1)
\end{align*}
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

\[
\begin{align*}
PW[31] &= R((PW[28] \land PW[23] \land PW[17] \land W[15]), 1) \\
PW[34] &= R((PW[31] \land PW[26] \land PW[20] \land PW[18]), 1) \\
PW[36] &= R((PW[33] \land PW[28] \land PW[22] \land PW[20]), 1) \\
PW[37] &= R((PW[34] \land PW[29] \land PW[23] \land PW[21]), 1) \\
PW[40] &= R((PW[37] \land PW[32] \land PW[26] \land PW[24]), 1) \\
PW[41] &= R((PW[38] \land PW[33] \land PW[27] \land PW[25]), 1) \\
&\vdots \\
PW[79] &= R((PW[76] \land PW[71] \land PW[65] \land PW[63]), 1)
\end{align*}
\]
W[0] in the inner loop

For 1..20 compute R(W[0], i)

w0_1 = R(w[0], 1)
w0_2 = R(w[0], 2)
...
w0_{20} = R(w[0], 20)

Word Expansion using precompute

\[ \begin{align*}
W[30] &= \text{PW}[30] \wedge w_0.4 \\
& \wedge w_0.4 \\
& \wedge w_0.4 \\
& \wedge w_0.2 \\
W[31] &= \text{PW}[31] \wedge w_0.6 \\
W[32] &= \text{PW}[32] \wedge w_0.3 \\
& \wedge w_0.2 \\
W[33] &= \text{PW}[33] \wedge w_0.5 \\
& \wedge w_0.5 \\
& \wedge w_0.5 \\
& \wedge w_0.3 \\
& \wedge w_0.5 \\
& \wedge w_0.3
\end{align*} \]

\[ \begin{align*}
W[34] &= \text{PW}[34] \wedge w_0.7 \\
W[35] &= \text{PW}[35] \wedge w_0.4 \\
& \wedge w_0.3 \\
& \wedge w_0.4 \\
& \wedge w_0.4 \\
& \wedge w_0.3 \\
W[36] &= \text{PW}[36] \wedge w_0.4 \\
& \wedge w_0.4 \\
& \wedge w_0.6 \\
& \wedge w_0.6 \\
& \wedge w_0.6 \\
& \wedge w_0.4
\end{align*} \]

\[ \begin{align*}
< 4 \text{ operations} \\
= 4 \text{ operations} \\
> 4 \text{ operations}
\end{align*} \]

Number of Operations:

\[ \begin{align*}
W[16] &= 1 \\
W[17] &= 0 \\
W[33] &= 6 \\
W[43] &= 308 \\
W[75] &= 4703 \\
\end{align*} \]

Jens Steube - Exploiting a SHA1 weakness in password cracking  4. Dec 2012
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**

\[ W[41] = R((W[38] \land W[33] \land W[27] \land W[25]), 1) \]

- \[ PW[38] \land PW[33] \land PW[27] \land PW[25] \land \]
- \[ w0_5 \land w0_5 \land w0_3 \land w0_4 \]
- \[ w0_5 \land w0_5 \land w0_3 \land \]
- \[ w0_4 \land w0_5 \land \]
- \[ w0_5 \land w0_3 \land \]
- \[ w0_3 \land \]
- \[ w0_4 \land \]

\[ W[41] = PW[41] \]

+1

Jens Steube - Exploiting a SHA1 weakness in password cracking

4. Dec 2012
... w[36] = PW[36] ^ W0_6 ^ W0_4
w[51] = PW[51] ^ W0_6 ^ W0_4
w[62] = PW[62] ^ W0_6 ^ W0_4 ^ W012 ^ W0_8
...

const int W0_6____W0_4 = W0_6 ^ W0_4

...

w[36] = PW[36] ^ W0_6____W0_4
w[51] = PW[51] ^ W0_6____W0_4
w[62] = PW[62] ^ W0_6____W0_4 ^ W012 ^ W0_8
...
**Final optimized Word-Expansion**

Reference Impl.

\[
\begin{align*}
\end{align*}
\]

Optimized Impl.

\[
\begin{align*}
W[16] &= PW[16] \land W_{0,1} \\
W[18] &= PW[18] \\
W[19] &= PW[19] \land W_{0,2} \\
W[22] &= PW[22] \land W_{0,3} \\
W[23] &= PW[23] \\
W[24] &= PW[24] \land W_{0,2} \\
W[25] &= PW[25] \land W_{0,4} \\
W[26] &= PW[26] \\
W[27] &= PW[27] \\
W[28] &= PW[28] \land W_{0,5} \\
W[29] &= PW[29] \\
W[30] &= PW[30] \land W_{0,4} \land W_{0,2}
\end{align*}
\]
### SHA1 instruction count; Unoptimized

<table>
<thead>
<tr>
<th>Section</th>
<th>Instruction count</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-Expansion</td>
<td>256</td>
<td>16 – 79</td>
</tr>
<tr>
<td>SHA1 Step F1</td>
<td>140</td>
<td>0 – 19</td>
</tr>
<tr>
<td>SHA1 Step F2</td>
<td>160</td>
<td>20 – 39</td>
</tr>
<tr>
<td>SHA1 Step F3</td>
<td>160</td>
<td>40 – 59</td>
</tr>
<tr>
<td>SHA1 Step F4</td>
<td>160</td>
<td>60 – 79</td>
</tr>
<tr>
<td>Final Add</td>
<td>4</td>
<td>80</td>
</tr>
</tbody>
</table>

**Total**: 880

---

Jens Steube - Exploiting a SHA1 weakness in password cracking  4. Dec 2012
### SHA1 instruction count; Known optimizations

<table>
<thead>
<tr>
<th>Section</th>
<th>Instruction count</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-Expansion</td>
<td>240</td>
<td>16 – 75</td>
</tr>
<tr>
<td>SHA1 Step F1</td>
<td>140</td>
<td>0 – 19</td>
</tr>
<tr>
<td>SHA1 Step F2</td>
<td>160</td>
<td>20 – 39</td>
</tr>
<tr>
<td>SHA1 Step F3</td>
<td>160</td>
<td>40 – 59</td>
</tr>
<tr>
<td>SHA1 Step F4</td>
<td>128</td>
<td>60 – 75</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>828</strong></td>
<td></td>
</tr>
</tbody>
</table>
### SHA1 instruction count; Exploiting SHA1's XOR weakness

<table>
<thead>
<tr>
<th>Section</th>
<th>Instruction count</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-Expansion</td>
<td>106</td>
<td>16 – 75</td>
</tr>
<tr>
<td>SHA1 Step F1</td>
<td>140</td>
<td>0 – 19</td>
</tr>
<tr>
<td>SHA1 Step F2</td>
<td>160</td>
<td>20 – 39</td>
</tr>
<tr>
<td>SHA1 Step F3</td>
<td>160</td>
<td>40 – 59</td>
</tr>
<tr>
<td>SHA1 Step F4</td>
<td>128</td>
<td>60 – 75</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>694</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Final comparison

<table>
<thead>
<tr>
<th>Section</th>
<th>Instruction count</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unoptimized</td>
<td>880</td>
<td>0 %</td>
</tr>
<tr>
<td>- Known optimizations</td>
<td>828</td>
<td>5.1 %</td>
</tr>
<tr>
<td>- This weakness, exploited</td>
<td>694</td>
<td>21.1 %</td>
</tr>
</tbody>
</table>

Jens Steube - Exploiting a SHA1 weakness in password cracking  
4. Dec 2012
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