Automatic Identification of Cryptographic Primitives in Software

—or: cage fighting with Rice’s Theorem—

27th of December 2010
27c3 in Berlin, Germany
Felix Gröbert <felix@groebert.org>
ATTENTION!!!!!!

ALL YOUR PERSONAL FILES WERE ENCRYPTED
WITH A STRONG ALGORITHM RSA-1024
AND YOU CAN'T GET AN ACCESS TO THEM
WITHOUT MAKING OF WHAT WE NEED!

READ 'HOW TO DECRYPT' TXT-FILE
ON YOUR DESKTOP FOR DETAILS

JUST DO IT AS FAST AS YOU CAN!

REMEMBER: DON'T TRY TO TELL SOMEONE
ABOUT THIS MESSAGE IF YOU WANT TO GET
YOUR FILES BACK! JUST DO ALL WE TOLD.
Motivation: Cryptography in Malware

- **GpCode**: AES-ECB-256 and “STRONG ALGORYTHM RSA-1024”
- **ShadowBot**: own implementation of MD5, obfuscation 8-bit-XOR
- **Conficker**: OpenSSL SHA1, reference implementation of MD6, RSA with 1024 bit, later 4096 bit, for signature verification
- **Waledac**: OpenSSL AES-CBC with zero IV, key exchange protocol with MITM vulnerability, JPEG obfuscation/steganography
- **Mebroot / Torpig / Sinowal**: BASE64 XOR obfuscation, symmetric cipher with self-designed 58-round Feistel network with 32 bit key, IV-modified SHA1
- **Agobot**: IRC over SSL
- **Storm**: P2P/FastFlux subnode authentication with 56 bit RSA, static XOR obfuscation
- **Nugache**: RSA key exchange, AES-256, RSA-4096 signed MD5 hashes of C&C

» How to help analyst finding cryptographic usage?
Motivation: System Verification

- “A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.”
  – Kerckhoffs, 1883

- “The enemy knows the system.”
  – Shannon, 1948

- “Any security software design that doesn’t assume the enemy possesses the source code is already untrustworthy.”
  – Raymond, 2004

- Extended Version: “Any security software design that doesn’t assume the enemy is able to reverse engineer the source code is already untrustworthy.”

- Security evaluation: determine the used cryptographic primitives and their composition: what, where, how, when

- Difficult if design or code is not public: Custom DRM and application protocols, malware protocols... is it a secure cryptographic design or just secure by obscurity?
Proposed Solution

1. **Run Binary**
   - Execute the software.
2. **Trace**
   - Generate traces of the software execution.
3. **Analyze Trace**
   - Examine the traces for cryptographic primitives.
4. **Report**
   - Summarize the findings.

**Software Execution Process**
- `software.exe`
- **Run**
- **Trace**
- **Analyze**
- **Report**
Thesis & Contributions

- If a standardized cryptographic primitive with its input and output is present in an execution trace, an algorithm exists to identify and verify the instance of the primitive including its parameters.

- Assumptions for the master thesis proof:
  - Not obfuscated or self-modifying code (except one-stage packers)
  - Not just-in-time-compiled or interpreted code
  - Limited to only the cryptographic primitive:
    - No mode-of-operation detection
    - No plaintext encoding or padding detection
    - No compression detection
  - Only Win32-based x86 code
## The Moving Targets

<table>
<thead>
<tr>
<th>Reference</th>
<th>Version</th>
<th>Algo</th>
<th>Mode</th>
<th>Compiler</th>
<th>Key</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beecrypt</td>
<td>4.1.2</td>
<td>AES</td>
<td>ECB</td>
<td>VC dll</td>
<td>128 bit</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>Beecrypt</td>
<td>4.1.2</td>
<td>MD5</td>
<td>-</td>
<td>VC dll</td>
<td>-</td>
<td>4096B</td>
<td>16B</td>
</tr>
<tr>
<td>Crypto++</td>
<td>5.6.0</td>
<td>AES</td>
<td>CFB</td>
<td>VC static</td>
<td>128 bit</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>Crypto++</td>
<td>5.6.0</td>
<td>DES</td>
<td>CFB</td>
<td>VC static</td>
<td>64 bit</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>Crypto++</td>
<td>5.6.0</td>
<td>RC4</td>
<td>-</td>
<td>VC static</td>
<td>128B</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>Crypto++</td>
<td>5.6.0</td>
<td>MD5</td>
<td>-</td>
<td>VC static</td>
<td>-</td>
<td>4096B</td>
<td>16B</td>
</tr>
<tr>
<td>Crypto++</td>
<td>5.6.0</td>
<td>RSA</td>
<td>-</td>
<td>VC static</td>
<td>1024 bit</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>Gladman</td>
<td>07-10-08</td>
<td>AES</td>
<td>CBC</td>
<td>VC static</td>
<td>128 bit</td>
<td>128B</td>
<td>144B</td>
</tr>
<tr>
<td>custom</td>
<td>custom</td>
<td>XOR</td>
<td>-</td>
<td>VC static</td>
<td>128B</td>
<td>256B</td>
<td>256B</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>0.9.8g</td>
<td>AES</td>
<td>CFB</td>
<td>MinGW, VC static</td>
<td>128 bit</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>0.9.8g</td>
<td>DES</td>
<td>ECB</td>
<td>MinGW, VC static</td>
<td>64 bit</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>0.9.8g</td>
<td>RC4</td>
<td>-</td>
<td>MinGW, VC static</td>
<td>128B</td>
<td>128B</td>
<td>128B</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>0.9.8g</td>
<td>MD5</td>
<td>-</td>
<td>MinGW, VC static</td>
<td>-</td>
<td>4096B</td>
<td>16B</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>1.0.0-beta3</td>
<td>RSA</td>
<td>-</td>
<td>VC dll</td>
<td>512 bit</td>
<td>128B</td>
<td>192B</td>
</tr>
</tbody>
</table>
## Related Work: Static Approaches

- **Tools require unpacked binary**
- **Byte-orientated signatures**
- **Evaluation:**
  - All detect MD5
  - No tool detects RSA
  - RC4 is only detected once by SnD
  - No tool detects dynamically linked cryptographic code

**Table of Algorithms and Tools**

<table>
<thead>
<tr>
<th>Name</th>
<th>Author(s)</th>
<th>Platform</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Krypto Analyzer (KANAL)</td>
<td>Several</td>
<td>PEiD</td>
<td>2.92</td>
</tr>
<tr>
<td>Findcrypt plugin</td>
<td>Ilfak Guilfanov</td>
<td>IDA Pro</td>
<td>2</td>
</tr>
<tr>
<td>SnD Crypto Scanner</td>
<td>Loki</td>
<td>OllyDBG</td>
<td>0.5b</td>
</tr>
<tr>
<td>Crypto Searcher</td>
<td>x3chun</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash &amp; Crypto Detector (HCD)</td>
<td>Mr Paradox, AT4RE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRACA</td>
<td>Ilya O. Levin</td>
<td>standalone</td>
<td>0.5.7b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>KANAL</th>
<th>Findcrypt</th>
<th>SnD</th>
<th>x3chun</th>
<th>HCD</th>
<th>DRACA</th>
</tr>
</thead>
<tbody>
<tr>
<td>gladman aes</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>cryptopp aes</td>
<td>+</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>openssl aes</td>
<td>+</td>
<td>6</td>
<td>+</td>
<td>3</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>cryptopp des</td>
<td>+</td>
<td>3</td>
<td>+</td>
<td>2</td>
<td>+</td>
<td>3</td>
</tr>
<tr>
<td>openssl des</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>cryptopp rc4</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>openssl rc4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cryptopp md5</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>openssl md5</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>openssl rsa</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cryptopp rsa</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>3</td>
</tr>
</tbody>
</table>

+ = algorithm found  
number = number of false-positives  
number = number of found algorithms

<table>
<thead>
<tr>
<th>File</th>
<th>KANAL</th>
<th>Findcrypt</th>
<th>SnD</th>
<th>x3chun</th>
<th>HCD</th>
<th>DRACA</th>
</tr>
</thead>
<tbody>
<tr>
<td>beecrypt.dll</td>
<td>11</td>
<td>18</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>libeay.dll</td>
<td>126</td>
<td>14</td>
<td>17</td>
<td>13</td>
<td>20</td>
<td>7</td>
</tr>
</tbody>
</table>

Monday, December 27, 2010
Related Work: Dynamic Approaches

- Wang et al. (PoC with Agobot) 2008
  - Crypto operations on tainted data differ vastly from other types of modifications (high percentage of bitwise arithmetic instructions)
  - Cumulative bitwise instruction percentage is used to determine turning point between encryption phase and other processing phases

- Caballero et al. (PoC with MegaD) 2009
  - Encrypted block processing » rather use a function/block-wise bitwise instruction percentage instead of a cumulative

- Noé Lutz (PoC with Kraken) 2008
  - Determines whether the read/write set inside a loop decreases information entropy of tainted memory
Related Work: Adjacent Approaches

- Key search in data
  - Shamir, Van Someren, Janssens: RSA in bit strings
  - Halderman et al.: “Coldboot Attack”
  - Stevens: XORsearch
  - Boldewin: OfficeMalScanner

- Loosely related reverse code engineering approaches
  - BinCrowd: collaborative reverse engineering
  - REGoogle: IDA plugin to codesearch for imports and constants
Our Approach: Dynamic Instrumentation

- software.exe
- Dynamic Binary Instrumentation
- Trace
- Analysis
- Report
Execution Tracing

- Dynamically instrument target binary code using PIN tool
  - PIN is a free-of-charge dynamic binary instrumentation framework by Intel
  - Can be extended by custom PIN tools (C++)
- Optionally filter by DLL or thread ID
- Start trace after a specific number of instructions
- Record compressed trace file
- Dynamic approach constraint: code must be executed
- Dynamic approach advantage: data can be examined
Execution Tracing Example

[...]

R|32|0022F948=0x22f9a4
0x7c9111f3|@1|@2|0x0016|0|mov esi, dword ptr [ebp+0x8]|esi=0x22f9a4
[...]
# Execution Tracing Example

Memory Access Mode | **Size** | **Address** = 0x22f9a4

[...]

R | 32 | 00 | 02 | F9 | 48 | 0x22f9a4
0x7c9111f3 | @1 | @2 | 0x0016 | 0 | mov esi, dword ptr [ebp+0x8] | esi = 0x22f9a4

[...]
Tracing Framework = Exchangeable

- software.exe
- Dynamic Binary Instrumentation
- Trace
- Analysis
- Report
Tracing Framework = Exchangeable
Implementation: Analysis

- Parse
- Filter
- Queue
- Structure
- Identify

Interactive Console
Control Flow Graph to PDF

Trace → Report
BBL & CFG

• Basic block (BBL) generation
  - Two pass over sequential instruction trace:
    (1) determine starts and ends of BBLs
    (2) populate data structure
  - Advantage of dynamic analysis: known targets of indirect branches

• Control flow graph (CFG) generation
  - Single pass over sequentially executed BBLs
  - Use Graphviz to export visualized CFGs
Loop Detection

- Based on Tubella et al.: “Control Speculation in Multithreaded Processors through Dynamic Loop Detection”
- Detects a loop by multiple executions of the same addresses
- Exposes the following features (unlike CFG-based Lengauer-Tarjan algorithm):
  - Number of loop executions
  - Number of loop iterations per loop execution (min/avg/max/total)
  - Set of instructions belonging to a loop body
  - Hierarchy of nested loops
Loop Detection: Example

target code

```c
for(i = 1; i < x; i++) {
    c = c + i;
    for(j = 0; j < i; j++) {
        if(j % 5 < 3)
            dosomething(c,j);
    }
}
```

CFG

result

Outer yellow loop was iterated twice. Red loop was executed twice, first with one, then with two iterations. Inner blue loop was executed three times, each with one iteration.
Memory Reconstruction from Trace

- Cryptographic data is generally larger than CPU supported register size. Thus the reconstruction of blocks of relating data is beneficial to key reconstruction.

- For an instruction+memory trace:
  - A recursive search checks whether at the next address (last address + length) a similar memory value has been accessed.
  - The search continues if the structure is continued (eg two byte access) at the next address. A search may be split if there are multiple matching next values.

- The search also checks whether the access occurred nearby the last access in the current block in the instruction trace.

<table>
<thead>
<tr>
<th>Address</th>
<th>Values (eg only reads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>001a0000</td>
<td>31313131</td>
</tr>
<tr>
<td>001a0004</td>
<td>32323232</td>
</tr>
<tr>
<td>001a0008</td>
<td>33333333</td>
</tr>
<tr>
<td>001a000c</td>
<td>34343434</td>
</tr>
<tr>
<td>001a0010</td>
<td>35353535 01</td>
</tr>
<tr>
<td>001a0011</td>
<td>02</td>
</tr>
<tr>
<td>001a0012</td>
<td>03</td>
</tr>
<tr>
<td>001a0013</td>
<td>04 05 60</td>
</tr>
<tr>
<td>001a0014</td>
<td>36363636 05 08</td>
</tr>
<tr>
<td>001a0015</td>
<td>06 0d</td>
</tr>
<tr>
<td>001a0016</td>
<td>07 15</td>
</tr>
</tbody>
</table>

Nearby (in the trace) the access to 03, the access to 04 and 05 occurred, but not the access to 60.
Implementation: Analysis

Trace → Parse → Filter → Queue → Structure → Identify → Report
Identification Observations

Cryptographic code makes excessive use of bitwise arithmetic instructions

```
BBL 0x4018a0  _DES_encrypt1 (49):
push ebx
push ebp
push esi
push edi
mov ecx, dword ptr ss:[esp+0x14]
mov edx, dword ptr ds:[ecx+0x4]
mov eax, dword ptr ds:[ecx]
mov ecx, edx
shr ecx, 0x4
xor ecx, eax
and ecx, 0xf0f0f0f
xor eax, ecx
shl ecx, 0x4
xor edx, ecx
mov ecx, eax
shr ecx, 0x10
xor ecx, edx
and ecx, 0xffff
xor edx, ecx
shl ecx, 0x10
xor eax, ecx
mov ecx, edx
shr ecx, 0x2
xor ecx, eax
and ecx, 0x33333333
```
Identification Observations

Constants and sequences of mnemonics indicate the type of cryptographic algorithm

<table>
<thead>
<tr>
<th>beecrypt</th>
<th>cryptopp</th>
<th>openssl</th>
</tr>
</thead>
<tbody>
<tr>
<td>rol ecx, 0x7</td>
<td>rol ecx, 0x7</td>
<td>rol edx, 0x7</td>
</tr>
<tr>
<td>add ecx, edi</td>
<td>add ecx, edi</td>
<td>add edx, ebp</td>
</tr>
<tr>
<td>mov ebp, esi</td>
<td>mov ebp, esi</td>
<td>mov edi, ebx</td>
</tr>
<tr>
<td>xor ebp, edi</td>
<td>xor ebp, edi</td>
<td>xor edi, ebp</td>
</tr>
<tr>
<td>and ebp, ecx</td>
<td>and ebp, ecx</td>
<td>and edi, edx</td>
</tr>
<tr>
<td>xor ebp, esi</td>
<td>xor ebp, esi</td>
<td>xor edi, esi</td>
</tr>
<tr>
<td>add ebp, ebx</td>
<td>add ebp, ebx</td>
<td>add edi, esi</td>
</tr>
<tr>
<td>lea edx, ptr [edx+ebp*1-0x173848aa] mov ebx, dword ptr ds:[eax+0x18] rol edx, 0xc add edx, ecx mov ebp, edi xor ebp, ecx and ebp, edx xor ebp, edi add ebp, ebx</td>
<td>mov dword ptr ss:[esp+0x54], ebx mov ebx, dword ptr ds:[eax+0x4] add ebp, ebx lea edx, ptr [edx+ebp*1-0x173848aa] rol edx, 0xc add edx, ecx mov ebp, edi xor ebp, ecx and ebp, edx xor ebp, edi add ebp, ebx</td>
<td>mov esi, dword ptr ds:[ecx+0xc] lea esi, ptr [edi+esi*1-0x173848aa] mov edi, ebp xor edi, edx rol esi, 0xc add esi, edx and edi, esi xor edi, ebp add edi, dword ptr ds:[eax-0x30]</td>
</tr>
</tbody>
</table>
Identification Observations

• **Cryptographic code contains loops**
  - Similar set of operations is commonly applied to the state with a different round key
  - Unrolling of loops is used to optimize algorithms

• **Input and output to cryptographic code have a predefined, verifiable relation**
  - If algorithm is known, the relation can be verified
  - Blinded RSA has (somewhat) non-deterministic relation
Implemented Analysis Modules

- Signature identification methods
  - sigAPI
  - constants in memory
  - mnemonic sequences
  - (mnemonic, constant) tuples

- Related work
  - Caballero (bitwise-instruction-percentage for functions and BBLs)
  - Wang (turning-point in cumulative bitwise-instruction-percentage)
  - Lutz (entropy-based)

- Generic identification methods
  - xor detection
  - loop differ
  - data verifier
(Mnemonic,Constant)-Tuples

- Foreach implementation, build a set of bitwise instructions with static constants, eg (rol, 0x14)
- Foreach algorithm, build a intersection and a unique set
- For an unknown set of instructions from a trace, the match degree is the percentage of found signature tuples in the unknown set

**Example:**
- **cryptopp aes**
  - (Mnemonic,Constant)-Tuples
  - intersection md5
  - unique md5
- **openssl md5**
  - cmd 56
  - cmp 24
  - and 255
- **beecrypt md5**
  - rol 4
  - rol 6
  - or 4
  - xor 4
- **openssl rc4**
  - rol 4
  - rol 6
  - and 4
  - xor 4

**Sets:**
- small set, strong relation to algorithm
- broad set, loose relation to algorithm
- high relation to type of implementation

**Monday, December 27, 2010**
### AES does not have a unique set

#### RC4 unique set only has two tuples

<table>
<thead>
<tr>
<th></th>
<th>beecrypt</th>
<th>beecrypt</th>
<th>cryptopp</th>
<th>cryptopp</th>
<th>cryptopp</th>
<th>cryptopp</th>
<th>gladman</th>
<th>openssl</th>
<th>openssl</th>
<th>openssl</th>
<th>openssl</th>
<th>openssl</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc4 unique</td>
<td>0 %</td>
<td>0 %</td>
<td>100 %</td>
<td>100 %</td>
<td>100 %</td>
<td>100 %</td>
<td>0 %</td>
<td>0 %</td>
<td>100 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>des unique</td>
<td>0 %</td>
<td>0 %</td>
<td>44 %</td>
<td>44 %</td>
<td>44 %</td>
<td>22 %</td>
<td>33 %</td>
<td>100 %</td>
<td>11 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>rsa unique</td>
<td>22 %</td>
<td>8 %</td>
<td>58 %</td>
<td>61 %</td>
<td>50 %</td>
<td>46 %</td>
<td>89 %</td>
<td>34 %</td>
<td>18 %</td>
<td>1 %</td>
<td>7 %</td>
<td>1 %</td>
</tr>
<tr>
<td>md5 unique</td>
<td>0 %</td>
<td>100 %</td>
<td>6 %</td>
<td>29 %</td>
<td>100 %</td>
<td>6 %</td>
<td>12 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>100 %</td>
<td>0 %</td>
</tr>
<tr>
<td>rc4 intersect</td>
<td>68 %</td>
<td>68 %</td>
<td>100 %</td>
<td>100 %</td>
<td>100 %</td>
<td>100 %</td>
<td>95 %</td>
<td>64 %</td>
<td>77 %</td>
<td>77 %</td>
<td>68 %</td>
<td>100 %</td>
</tr>
<tr>
<td>aes intersect</td>
<td>100 %</td>
<td>82 %</td>
<td>100 %</td>
<td>100 %</td>
<td>82 %</td>
<td>82 %</td>
<td>94 %</td>
<td>100 %</td>
<td>100 %</td>
<td>88 %</td>
<td>88 %</td>
<td>71 %</td>
</tr>
<tr>
<td>des intersect</td>
<td>56 %</td>
<td>51 %</td>
<td>87 %</td>
<td>100 %</td>
<td>77 %</td>
<td>77 %</td>
<td>82 %</td>
<td>51 %</td>
<td>74 %</td>
<td>100 %</td>
<td>64 %</td>
<td>46 %</td>
</tr>
<tr>
<td>rsa intersect</td>
<td>34 %</td>
<td>28 %</td>
<td>71 %</td>
<td>71 %</td>
<td>63 %</td>
<td>57 %</td>
<td>93 %</td>
<td>41 %</td>
<td>35 %</td>
<td>24 %</td>
<td>29 %</td>
<td>16 %</td>
</tr>
<tr>
<td>md5 intersect</td>
<td>40 %</td>
<td>100 %</td>
<td>60 %</td>
<td>67 %</td>
<td>100 %</td>
<td>52 %</td>
<td>62 %</td>
<td>26 %</td>
<td>45 %</td>
<td>43 %</td>
<td>100 %</td>
<td>38 %</td>
</tr>
<tr>
<td>rc4 cryptopp</td>
<td>13 %</td>
<td>14 %</td>
<td>83 %</td>
<td>82 %</td>
<td>100 %</td>
<td>57 %</td>
<td>16 %</td>
<td>17 %</td>
<td>16 %</td>
<td>15 %</td>
<td>11 %</td>
<td>31 %</td>
</tr>
<tr>
<td>rc4 openssl</td>
<td>60 %</td>
<td>58 %</td>
<td>68 %</td>
<td>63 %</td>
<td>58 %</td>
<td>55 %</td>
<td>65 %</td>
<td>38 %</td>
<td>55 %</td>
<td>53 %</td>
<td>50 %</td>
<td>100 %</td>
</tr>
<tr>
<td>aes beecrypt</td>
<td>100 %</td>
<td>33 %</td>
<td>35 %</td>
<td>34 %</td>
<td>27 %</td>
<td>27 %</td>
<td>58 %</td>
<td>62 %</td>
<td>41 %</td>
<td>29 %</td>
<td>27 %</td>
<td>26 %</td>
</tr>
<tr>
<td>aes gladman</td>
<td>41 %</td>
<td>12 %</td>
<td>27 %</td>
<td>28 %</td>
<td>23 %</td>
<td>22 %</td>
<td>45 %</td>
<td>100 %</td>
<td>21 %</td>
<td>17 %</td>
<td>13 %</td>
<td>11 %</td>
</tr>
<tr>
<td>aes cryptopp</td>
<td>12 %</td>
<td>13 %</td>
<td>100 %</td>
<td>73 %</td>
<td>64 %</td>
<td>62 %</td>
<td>59 %</td>
<td>15 %</td>
<td>16 %</td>
<td>14 %</td>
<td>14 %</td>
<td>10 %</td>
</tr>
<tr>
<td>aes openssl</td>
<td>52 %</td>
<td>34 %</td>
<td>56 %</td>
<td>55 %</td>
<td>47 %</td>
<td>47 %</td>
<td>62 %</td>
<td>40 %</td>
<td>100 %</td>
<td>45 %</td>
<td>37 %</td>
<td>30 %</td>
</tr>
<tr>
<td>des cryptopp</td>
<td>12 %</td>
<td>14 %</td>
<td>74 %</td>
<td>100 %</td>
<td>65 %</td>
<td>62 %</td>
<td>53 %</td>
<td>15 %</td>
<td>15 %</td>
<td>15 %</td>
<td>15 %</td>
<td>10 %</td>
</tr>
<tr>
<td>des openssl</td>
<td>26 %</td>
<td>22 %</td>
<td>36 %</td>
<td>38 %</td>
<td>29 %</td>
<td>30 %</td>
<td>36 %</td>
<td>22 %</td>
<td>32 %</td>
<td>100 %</td>
<td>29 %</td>
<td>20 %</td>
</tr>
<tr>
<td>rsa cryptopp</td>
<td>12 %</td>
<td>9 %</td>
<td>48 %</td>
<td>43 %</td>
<td>39 %</td>
<td>36 %</td>
<td>72 %</td>
<td>14 %</td>
<td>11 %</td>
<td>9 %</td>
<td>9 %</td>
<td>6 %</td>
</tr>
<tr>
<td>rsa openssl</td>
<td>22 %</td>
<td>19 %</td>
<td>47 %</td>
<td>47 %</td>
<td>42 %</td>
<td>38 %</td>
<td>62 %</td>
<td>28 %</td>
<td>23 %</td>
<td>17 %</td>
<td>20 %</td>
<td>11 %</td>
</tr>
<tr>
<td>md5 beecrypt</td>
<td>45 %</td>
<td>100 %</td>
<td>50 %</td>
<td>56 %</td>
<td>74 %</td>
<td>41 %</td>
<td>58 %</td>
<td>26 %</td>
<td>38 %</td>
<td>35 %</td>
<td>73 %</td>
<td>35 %</td>
</tr>
<tr>
<td>md5 cryptopp</td>
<td>11 %</td>
<td>22 %</td>
<td>74 %</td>
<td>76 %</td>
<td>100 %</td>
<td>72 %</td>
<td>57 %</td>
<td>14 %</td>
<td>15 %</td>
<td>13 %</td>
<td>23 %</td>
<td>10 %</td>
</tr>
<tr>
<td>md5 openssl</td>
<td>34 %</td>
<td>66 %</td>
<td>49 %</td>
<td>53 %</td>
<td>71 %</td>
<td>41 %</td>
<td>55 %</td>
<td>25 %</td>
<td>37 %</td>
<td>41 %</td>
<td>100 %</td>
<td>27 %</td>
</tr>
</tbody>
</table>
Implementations

Cell value > 70%

Beecrypt AES signature hits Beecrypt AES code (1 to 1)

MD5 CryptoPP signature hits MD5 with 100% and with 70%<x<100% RC4, AES, DES
Unique
Cell value > 70%

RC4 unique signature very small: three false-positives

0 false-positives

No unique set for AES

0 false-positives

0 false-positives
Generic Method: Loop Differ

- Foreach Loop:
  - List of values for executions, iterations, instructions
  - Testing values for
    - XOR relation
    - Counter heuristic
    - S/P-box relation
    - Entropy heuristic
    - ...

- Evaluation:
  - Finds counters for almost all implementations
  - Finds XOR relation for most of the CFB/CBC mode symmetric ciphers
Generic Method: Data Verifier

- Use memory reconstruction to filter blocks above a specific size

- Generate key, plaintext, ciphertext candidates of a specific size
**Generic Method: Data Verifier**

**Trace**

AES Code:
- 8 bit Read
- 256 bit Read
- 128 bit Read
- 128 bit Read
- 32 bit Read
- 128 bit Write
- 128 bit Write
- 32 bit Read
- 32 bit Read

**Key**

AES reference implementation

**Plaintext**

**Ciphertext**

**AES detected**

**no AES detected**

**equals?**

yes

no
General Performance

<table>
<thead>
<tr>
<th>#instructions</th>
<th>kilobytes</th>
<th>seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL RC4</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Beecrypt AES</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Gladman AES</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>OpenSSL AES</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>OpenSSL DES</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Custom XOR</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>OpenSSL MD5</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Beecrypt MD5</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Crypto++ RC4</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Crypto++ AES</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Crypto++ MD5</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Crypto++ DES</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>Crypto++ RSA</td>
<td>100000</td>
<td>10000</td>
</tr>
<tr>
<td>OpenSSL RSA</td>
<td>100000</td>
<td>10000</td>
</tr>
</tbody>
</table>
Real World Experiments

- Packed XOR testing application with ASPack 2.12
  - Trace size increased by factor 17; Analysis still found loops and xor
- curl HTTPS session: AES-256-CBC with OpenSSL 0.9.8l
  - Trace: 7 minutes, 45 MB; Analysis: 9 minutes
  - Identification of plain-, ciphertext and key in 94% of all blocks

<table>
<thead>
<tr>
<th>Method</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>xorNotNullAndMov()</td>
<td>only false-positives / unknown results</td>
</tr>
<tr>
<td>symmetricCipherDataTester()</td>
<td>detected 94% of AES instances including parameters</td>
</tr>
<tr>
<td>loopDiffer()</td>
<td>detected AES counters, some false-positives</td>
</tr>
<tr>
<td>sigAPI()</td>
<td>detected cryptographic functions</td>
</tr>
<tr>
<td>constmemory()</td>
<td>detected AES, one false-positive</td>
</tr>
<tr>
<td>chains()</td>
<td>detected AES and RSA, including implementation</td>
</tr>
<tr>
<td>constmnemonic()</td>
<td>detected AES implementation, one false-positive</td>
</tr>
<tr>
<td>wang()</td>
<td>no results</td>
</tr>
<tr>
<td>caballero()</td>
<td>detected core AES basic blocks</td>
</tr>
<tr>
<td>lutz()</td>
<td>detected core AES loops</td>
</tr>
</tbody>
</table>
ATTENTION!!!!!!

ALL YOUR PERSONAL FILES WERE ENCRYPTED WITH A STRONG ALGORITHM RSA-1024 AND YOU CAN'T GET AN ACCESS TO THEM WITHOUT MAKING OF WHAT WE NEED!

READ 'HOW TO DECRYPT' TXT-FILE ON YOUR DESKTOP FOR DETAILS

JUST DO IT AS FAST AS YOU CAN!

REMEMBER: DON'T TRY TO TELL SOMEONE ABOUT THIS MESSAGE IF YOU WANT TO GET YOUR FILES BACK! JUST DO ALL WE TOLD.
Attention!!!
All your personal files (photo, documents, texts, databases, certificates, kwm-files, video) have been encrypted by a very strong cypher RSA-1024. The original files are deleted. You can check this by yourself - just look for files in all folders.
There is no possibility to decrypt these files without a special decrypt program! Nobody can help you - even don't try to find another method or tell anybody. Also after n days all encrypted files will be completely deleted and you will have no chance to get it back.
We can help to solve this task for 120$ via wire transfer (bank transfer SWIFT/IBAN). And remember: any harmful or bad words to our side will be a reason for ingoring your message and nothing will be done.
For details you have to send your request on this e-mail (attach to message a full serial key shown below in this 'how to..' file on desktop):
datafinder@fastmail.fm
28D94212E021265919101292613D9240D78A6828D61ED51A26FFF6F386C81D46BD9E3AF9E7
2BC422AE7DOFA336706348956EFA337EA260F451676ED22936846
71604E249B2D830B273F5C18596587B4A3B032539B7E846794D27AC1385F5A93B3DDE516EB7304EA0A4E1C0CC6C10403210F8F03D3E68C6A994FFFEFE11F2B37
GpCode Malware 2010

0000000: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000010: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000020: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000030: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000040: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000050: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000060: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000070: 4444 4444 3333 3333 4444 4444 3333 3333 DDDD3333DDDD3333
0000080: 0300 0000

2010-12-05 20:48:17,080 Analysis.py:symmetricCipherDataTester@623 [DEBUG] found aes 'DDDD3333DDDD3333'
'\xa8\xc2HM\xa21K\x96\x89\xce\x7fG\x13Q\xe1\xa4\x80\xf7:\xc1\x18\xe4u\x1f%f\x85X\x8c\xaaxb6m\xda'
'\xea\x0b\xe8\x1c h\xe7\xc4\x89\xcac\xa2\x1d\x02\x98u\x91'
Further Work

- Implement data relation checker in dedicated PIN tool or ptrace tool
  - Proof-of-Concept with Skype
- Reduce trace/analysis time and space requirements: Switch from PIN to BitBlaze
- Adopt machine-learning methods to signatures
- Research on detection of padding, compression, encoding and eventually be able to detect complete cryptographic composition

» http://code.google.com/p/kerckhoffs
Summary

• If you use a insecure cryptographic composition you fail (static key)

• Automatic identification of crypto code is feasible

• Applications of the proposed methods will find interesting results in malware, DRM systems and closed-source/obfuscated software

• Machine learning, dynamic binary trace systems and formats will help to further advance the described methods
Questions?

Thanks!

special thanks: carsten willems & thorsten holz
greets: joernchen, aimster, buzze, d0mber, cw aka iceman, opti, sp, l1tt, foolabs, fluxfingers crew, gynvael, dfa, hawkes, jln, robert, taviso, liquid-k, asirap, novocainated, sirdarckcat, headhntr, greg, ths, max

http://code.google.com/p/kerckhoffs
© felix@groebert.org