Finding Cryptography in Object Code

Jason L. Wright
Cyber Security Researcher
Idaho National Laboratory
jason.wright@inl.gov

Security Education Conference
Toronto October 8, 2008
(SECTor08)
Current Work

• Malware is using cryptography

• When reverse-engineering, crypto algorithm must be found and identified

• The Bad Guys use many obfuscation techniques, but they usually do not change the underlying crypto algorithm
Mifare: Little Security, Despite Obscurity

• Henryk Plötz/Karsten Nohl presented at 24C3 (Dec'07) and CanSecWest 2008 (also USENIX Security 08)
• Hardware hack of a smartcard crypto algorithm
• “Focus on interesting-looking parts:
  – strings of flip-flops (registers)
  – XOR
  – Units around the edges that sparsely [connect] to the rest of the chip”

• More:
  http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html
FindCrypt: locating constants

• Many algorithms have constants that are unique (MD4, CAST-128, etc).
• Easy enough... just search for them:
  – data segment
  – text segment
• This is findcrypt from Datarescue:
  http://hexblog.com/2006/01/findcrypt.html
  http://hexblog.com/2006/02/findcrypt2.html
• Also KANAL (plug in for PeiD)
  http://peid.has.it/
Example:

```c
void MD5Init(MD5_CTX *ctx) {
    ctx->count = 0;
    ctx->state[0] = 0x67452301;
    ctx->state[1] = 0xefcdab89;
    ctx->state[2] = 0x98badcfe;
    ctx->state[3] = 0x10325476;
}
```

Becomes:

```assembly
<MD5Init>:
    push %ebp
    mov %esp,%ebp
    mov 0x8(%ebp),%eax
    movl $0x0,0x10(%eax)
    movl $0x0,0x14(%eax)
    movl $0x67452301,(%eax)
    movl $0xefcdab89,0x4(%eax)
    movl $0x98badcfe,0x8(%eax)
    movl $0x10325476,0xc(%eax)
    leave
    ret
```

Most crypto algorithms have these magic constants
Example (cont):

```c
void MD5Init(MD5_CTX *ctx) {
    ctx->count = 0;
    ctx->state[0] = 0x67452301;
    ctx->state[1] = 0xefcdab89;
    ctx->state[2] = 0x98badcfe;
    ctx->state[3] = 0x10325476;
}
```

```
00000000 <MD5Init>:
0: 55       push %ebp
1: 89 e5    mov %esp,%ebp
3: 8b 45 08 mov 0x8(%ebp),%eax
6: c7 40 10 00 00 00 00 movl $0x0,0x10(%eax)
d: c7 40 14 00 00 00 00 movl $0x0,0x14(%eax)
14: c7 00 01 23 45 67 movl $0x67452301,(%eax)
1a: c7 40 04 89 ab cd ef movl $0xefcdab89,0x4(%eax)
21: c7 40 08 fe dc ba 98 movl $0x98badcfe,0x8(%eax)
28: c7 40 0c 76 54 32 10 movl $0x10325476,0xc(%eax)
2f: c9       leave
30: c3       ret
```

```c
void MD5Init(MD5_CTX *ctx) {
    ctx->count = 0;
    ctx->state[0] = 0x67452301;
    ctx->state[1] = 0xefcdab89;
    ctx->state[2] = 0x98badcfe;
    ctx->state[3] = 0x10325476;
}
```

```
0000000000000000 <MD5Init>:
0: 03 19 d1 48 sethi %hi(0x67452000), %g1
4: 05 3b f3 6a sethi %hi(0xefcd8000), %g2
8: c0 72 20 10 clr x [ %o0 + 0x10 ]
c: 82 10 63 01 or %g1, 0x301, %g1
10: 84 10 a3 89 or %g2, 0x389, %g2
14: c2 22 00 00 st %g1, [ %o0 ]
18: c4 22 20 04 st %g2, [ %o0 + 4 ]
c: 03 26 2e b7 sethi %hi(0x98badc00), %g1
20: 05 04 0c 95 sethi %hi(0x10325400), %g2
24: 82 10 60 fe or %g1, 0xfe, %g1
28: 84 10 a0 76 or %g2, 0x76, %g2
2c: c2 22 20 08 st %g1, [ %o0 + 8 ]
30: 81 c3 e0 08 rel 1
34: c4 22 20 0c st %g2, [ %o0 + 0xc ]
```
A Better Way? Look for behavior...

- Don't search for concrete constants
- Look for properties unique to cryptographic functions
- We'll search for these properties
Look for...

• Heavy use of integer operations:
  – AND/OR \( a \land b \quad a \lor b \)
  – ADD/SUB
  – XOR \( a \oplus b \)

• No floating point operations

• XOR is interesting by itself, it's a part of:
  – DES, SKIPJACK, RC4, BLOWFISH, AES
  – It has a C operator: \( c = a \land b \);
  – Other languages have it too (well, COBOL doesn't)
Looking for XOR...

- Not just any XOR will do, however...
  - `XOR %eax, %eax`
  - On many arch's, including IA32, this is very commonly used to “zero” a register (e.g. `eax`)
  - 93% of XOR usage is “zeroing” in Linux `glibc`
- Constant/immediate form: `xor $0x103, %eax`
  - Not usually interesting
- But memory references and registers are...
Rotation (ROR, ROL, not ROTFL)

- Commonly “diffusion” part of “confusion and diffusion” (Claude Shannon)
- No C operator, so...
  - Compiler optimization (libc)
  - Inline assembler (OpenSSL)
Example: OpenBSD libc

- Has quite a bit of crypto:
  SHA1, MD4/MD5, RMD160, CAST-128, SKIPJACK,
  DES/3DES, BLOWFISH
- 1379 functions total (4.3)
- Versions tested all the way back to 2.5 (May 1999!) (Remember a.out?)
- Compiler seems to have little effect (gcc 2.8.1, 2.95.2, 3.3.5)
- Nor do compiler options (O0 to O3)
Example: Linux libc

- Not much crypto... just DES
  i.e. `crypt()`
Example: CALC.EXE

- Shouldn't have any crypto

... and doesn't appear to...
Scoring...

- Initially tried a density...

\[ \text{density} = \frac{\text{score}}{\text{num of instructions}} \]
Hazards

• Changing/obscurring constants (*findcrypt* can't deal with this)
• FindCrypto not immune to dead/obscured code (combine with dead code analysis?)

Notably missing

• Asymmetric algorithms (DH, RSA, etc.)
  Macro operations on big numbers (>=512bit)
• RC4 (not Feistel cipher)

\[
\begin{align*}
c &= m^e \mod n \\
\quad m &= c^d \mod n
\end{align*}
\]
Questions/Comments?

00000000 <_exit>:
  0: 8b 5c 24 04 mov 0x4(%esp),%ebx
  4: b8 fc 00 00 00 mov $0xfc,%eax
  9: ff 15 00 00 00 00 call *0x0
  f: b8 01 00 00 00 mov $0x1,%eax
 14: cd 80 int $0x80
 16: f4 hlt