# Obfuscated dechiper routine analysis using theorem prover towards effective trusted computing

Ruo Ando and Kouichi Furukawa
National Institute of Information and Communication Technology,
4-2-1 Nukui-Kitamachi, Koganei,
Tokyo 184-8795 Japan

Keio University, Graduate School of Media and Governance,
Endo 5322 Fujisawa 252-8520 Japan

## Abstract

*Cyber attacks has become more sophisticated, which imposes a great burden on business IT infrastructure. Trusted computing is a promising technology to cope with malicious or illegal access to mission-critical servers. Code obfuscation and encryption are essential techniques for trusted computing. In this paper, we present an obfuscated decipher routine analysis using theorem prover. For detecting parameters of decipher routine, we apply equality substitution techniques called paramodulation and demodulation. Besides, we apply look-ahead strategies to speed up our analysis. Experimental result shows that FoL theorem prover works well for analyzing obfuscated decipher routine, particularly detecting parameters.*

## 1 Introduction

Cyber attacks has become more sophisticated, which imposes a great burden on business IT infrastructure. Trusted computing is a promising technology to cope with malicious or illegal access to mission-critical servers. Besides attestation, code obfuscation and encryption are essential techniques for trusted computing. In this paper, we present an obfuscated decipher routine analysis using theorem prover. For prototyping system, we use FoL (First order Logic) theorem prover OTTER[5].

Table 1 and 2 are the example of code obfuscation. These are functionally equivalent, which execute GetModuleHandleA (Windows API). Code of Table 2 is obfuscated from Table 1.

## 2 Detecting parameters of decipher routine

There are four parameters of decipher routine: address of encrypted data, key, address of loop entry point, and counter. The basic structure of obfuscated decipher routine is as follows:

```
set A address_of_payload
set B key
set C address_loop_start
set D counter

address_loop_start
   payload_transfer(A)
   decryptor(B)
   parload_transfer(A)
   branch(D)
   goto_start(C)
```

When we detect four parameters A-D, the detection is completed.

### 2.1 Code disassemble with theorem prover

In proposal system, binary code is disassembled by demodulation theorem prover. Assembly code consists of operation and operand (argument). For example, we formulate disassembling in the script of theorem prover.

```
list(demodulators).
/* operation disassemble */
c(Y,line)=c(mov(ax),line).
/* operand */
c(X,line)=c(reg(ax,ax),line).
end_of_list.
```

| | |
|---|---|
| 1 | mov dword_3, 6E72654Bh |
| 2 | mov dword_4, 32336C65h |
| 3 | mov dword_5, 0h |
| 4 | push offset dword_3 |
| 5 | call ds:GetModuleHandleA |

**Table 1. Assembly code of GetModuleHandleA API.**

| | |
|---|---|
| 3 | mov dword_1,0h |
| 3 | mov cdx,dword_1 |
| 3 | mov dword_2,edx |
| 3 | mov edp,dword_2 |
| 2 | mov edi,32336C65h |
| 2 | lea eax,[edi] |
| 1 | mov esi,0A624540h |
| 1 | or esi,4670214Bh |
| 2 | lea edi,[eax] |
| 2 | mov dword_4,cid |
| 3 | mov edx,ebp |
| 3 | mov dword_5,edx |
| 1 | mov dwrod_3,esi |
| 4 | mov edx,offset dword_3 |
| 4 | push edx |
| 5 | mov dword_6,offset GetModuleHandleA |
| 5 | push dword_6 |
| 5 | pop dwprd_7 |
| 5 | mov edx,dword_7 |
| 5 | call dword ptr ds:0[edx] |

**Table 2. Obfuscated assembly code of GetModuleHandleA API**

For operand, we formulate all registers. For operation, we formulate some of instructions such as transfer and arithmetic operation.

### 2.2 Paramodulation and demodulation

Paramodulation[15] is one of the techniques of equational reasoning. The purpose of this inference rule is for an equality substitution to occur from one clause to another. In the discussion of completeness and efficiency, paramodulation is often compared with demodulation[14]. Demodulation is mainly designed for canonicalizing information and sometimes more effective than paramodulation. However, demodulation does not have power to cope with clauses as follows:

```
fact: f(g(x),x).
fact: equal(g(a),b).
conclusion f(b,a).
```

That is, paramodulation is a generalization of the substitution rule for equality. For searching parameters of obfuscated decipher routine, we should use both paramodulation and demodulation.

```
fact: equal(data_16e,514Bh).
fact: mov(reg(ah),const(data_16e),
line(63)).
conclusion : mov(reg(ah),const(514Bh),
line(63)).
```

The clauses above is the application of demodulation to deal with constant number defined in the beginning of program. In obfuscating decipher routine, there's another way to hide parameter using mov (data transfer) instruction.

```
fact: mov(reg(ah),const(2Ch),line(162)).
fact: mov(reg(bx),reg(ah),
line(300)).
/* decrypter */
fact: xor(reg(dx),reg(bx),
line(431)).
```

In this case, we insert this clause to occur paramodulation.

```
-mov(reg(x),const(y),line(z)) |
x=const(y,z).
conclusion:
decrypter(reg(dx),key(const(2Ch,162),
line(431)).
```

Conclusion is generated by paramodulation. By using paramodulation, we detect the value of [1]key, [2]address of payload, [3]loop counter (how many times the routine repeats), and [4]entry point of decipher routine.

### 2.3 Applying hot list strategy

In this paper we apply a heuristics to make paramodulation faster. Hot list strategy, proposed by Larry Wos[15], is one of the look ahead strategies. Look-ahead strategy is designed to enable reasoning program to draw conclusions quickly using a list whose elements are processed with each newly retained clause. Mainly, hot list strategy is used for controlling paramodulation. By using this strategy, we can emphasize particular clauses on hot list on paramodulation.

## 3 Numerical results

In this section we validate the effectiveness of our system by numerical output of theorem prover. First, we briefly discuss the result of weighting startegy.

### 3.1 SMEG

In experiment, we use SMEG (Simulated Metamorphic Encryption Generator)[15] to generate sample programs of obfuscated decipher routine. SMEG can generate hundreds of executables including obfuscated decipher routine. There are three types of SMEG mutations as shown in Table 3. Type A and C uses mov and xchg (exchange) to transfer the encrypted data. Type B uses indirect addressing (xor [address] key) to execute payload transfer and decryption at the same time. In type D, stack operation is applied for data transfer (fetch) and loop II (push / retf).

### 3.2 Hot list strategy

To detect parameters, clauses are generated by reasoning program for paramodulation as follows.

```
-mov(reg(x),const(y),z,w) |
x=const(y,z).
```

Clauses on right side are called paramodulant. Pamodulant is used by theorem prover for equality substitution (paramodulation). We make two hot lists. In other words, we set hot list clauses about registers EAX, EBX, ECX EDX and ESI, EDI, EBP, ESP.

```
# hot list group I :
calculation registers
list(hot).
ax=const(x,y). bx=const(x,y).
cx=const(x,y). dx=const(x,y).
end_of_list.

# hot list group II :
memory registers
list(hot).
di=const(x,y). si=const(x,y).
bi=const(x,y). bp=const(x,y).
end_of_list.
```

Table 6, 7, 8 and 9 are result of applying hot lists for four types of SMEG generation. We make 10 hot list (list(hot)) accorging to eight registers and two groups {eax, ecx, ebx, edx} and {edi, esi, ebi, ebp}. Among 8 hot lists (eax, ecx, ebx,edx, edi, esi, ebi, ebp), which hot list increase performance best depends on types of generated code. As a whole, hot list group of calculation registers {eax, ecx, ebx, edx} results in good performance compared with group {edi, esi, ebi, ebp}. In some bad cases hot list of group II increase the number generated clauses compared with no hot list.

Let T(group I) be computing time with hot list group I. Let T(no weight) and T(group II) computing time with no heat and hot list group II. In experiment, our system achieves condition.

| Type A (no weighting) | | Type A (with weighting) | |
|---|---|---|---|
| HOT LIST | all clauses | HOT LIST | all clauses |
| no heat | 915 | no heat | 707 |
| EAX | 677 | EAX | 677 |
| EBX | 670 | EBX | 602 |
| ECX | 799 | ECX | 541 |
| EDX | 756 | EDX | 540 |
| EDI | 1078 | EDI | 822 |
| ESI | 1055 | ESI | 801 |
| EBI | 1055 | EBI | 801 |
| EBP | 1055 | EBP | 801 |
| Group I | 468 | Group I | 366 |
| Group II | 1510 | Group II | 1206 |

**Table 4. Hot list strategies for Type A. Paramodulation for detecting parameters into register E* is speeded up by hot list. We set 10 hot lists for each register and two groups.**

```
T(group I) < T(no weight) < T(group II)
or
T(group I) < T(group II) < T(no weight)
```

Particularly in type A and D, our system achieves this condition.

```
T(group I) *3 < T(no weight)
where T(group II) < T(no weight)
or
T(group I) *3 < T(group II)
where T(no weight) < T(group II)
```

We can conclude that proposed parallel analysis model is effective. Particularly in type A and D, it is shown that we can make deduction system faster without appending hardware computing resources. c

## 4 Conclusion

Cyber attacks has become more sophisticated, which impose a great burden on business IT infrastructure. Trusted computing is expecting a promising technology to cope with malicious or illegal access to mission-critical servers. Code obfuscation and encryption are essential techniques for trusted computing. For detecting parameters of decipher routine, we apply equality substitution techniques called paramodulation and demodulation. Besides, we apply look-ahead strategies called hot list to speed up our analysis. For prototyping our system, we present an obfuscated decipher routing analysis using theorem prover. Experimental result shows that hot list strategy can reduce the computing time.

| | Type A | Type B | Type C | Type D |
|---|---|---|---|---|
| loop I | set loop_start | loop_start | set loop_start | set loop_start |
| transfer I | mov data address | decrypt [address] key | xchg address data | push data / pop data |
| decrypt I | decrypt data key | decrypt [address] key | decrypt data key | decrypt data key |
| transfer II | mov address data | decrypt [address] key | xchg address data | mov address data |
| decrypt II | inc address | inc address | inc address | inc address |
| branch II | dec counter | dec counter | dec counter | dec counter |
| branch | test counter counter | test counter counter | test counter counter | test counter counter |
| loop II | jmp loop_start | jmp loop_start | jmp loop_start | push / retf |

**Table 3. Three kinds of assembly code generated by SMEG**

| Type B (no weighting) | | Type B (with weighting) | |
|---|---|---|---|
| HOT LIST | all clauses | HOT LIST | all clauses |
| no heat | 1592 | no heat | 769 |
| EAX | 915 | EAX | 605 |
| EBX | 1561 | EBX | 494 |
| ECX | 497 | ECX | 490 |
| EDX | 519 | EDX | 593 |
| EDI | 1921 | EDI | 1164 |
| ESI | 1724 | ESI | 843 |
| EBI | 1724 | EBI | 685 |
| EBP | 1724 | EBP | 685 |
| Group I | 463 | Group I | 242 |
| Group II | 2422 | Group II | 1807 |

**Table 5. Hot list strategies for Type B.**

| Type C (no weighting) | | Type C (with weighting) | |
|---|---|---|---|
| HOT LIST | all clauses | HOT LIST | all clauses |
| no heat | 976 | no heat | 604 |
| EAX | 1018 | EAX | 605 |
| EBX | 720 | EBX | 494 |
| ECX | 946 | ECX | 490 |
| EDX | 976 | EDX | 593 |
| EDI | 1592 | EDI | 1164 |
| ESI | 1272 | ESI | 843 |
| EBI | 1114 | EBI | 685 |
| EBP | 1114 | EBP | 685 |
| Group I | 738 | Group I | 463 |
| Group II | 2284 | Group II | 1807 |

**Table 6. Hot list strategies for Type C.**

From results obtained, we can conclude that theorem prover works well for analyzing obfuscated decipher routine, particularly detecting parameters.

## References


[1] Trusted computing: https://www.trustedcomputinggroup.org/home

[2] Peter Szor and Peter Ferrie," Hunting for Metamorphic", Virus Bulletin Conference, 2001.

[3] Larry Wos, Gail W. Pieper, "The Hot List Strategy", Journal of Automated Reasoning, 1999

[4] Simulated Metamorphic Encryption Generator available at http//vx.netlux.org/vx.php?id=es06

[5] OTTER automated deduction system available at http//www.mcs.anl.gov/AR/otter/


| Type D (no weighting) | | Type D (with weighting) | |
|---|---|---|---|
| HOT LIST | all clauses | HOT LIST | all clauses |
| no heat | 1877 | no heat | 801 |
| EAX | 1444 | EAX | 587 |
| EBX | 1675 | EBX | 587 |
| ECX | 870 | ECX | 599 |
| EDX | 1877 | EDX | 737 |
| EDI | 7406 | EDI | 1462 |
| ESI | 2028 | ESI | 876 |
| EBI | 2028 | EBI | 876 |
| EBP | 2028 | EBP | 876 |
| Group I | 563 | Group I | 259 |
| Group II | 8186 | Group II | 1891 |

**Table 7. Hot list strategies for Type D.**