Journal of Computer Applications
计算机应用,2013,33(11):3239-3243
ISSN 1001-9081
CODEN JYIIDU
2013-11-01
http://www.joca.cn

文章编号:1001-9081(2013)11-3239-05
doi:10.11772/j.issn.1001-9081.2013.11.3239

识别恶意软件中的加密函数

蔡建章*,魏　强,祝跃飞
(信息工程大学 网络空间安全学院,郑州 450002)
(*通信作者电子邮箱 cjzh781119@hotmail.com)

摘　要:针对恶意软件通过加密函数规避安全检测和流量分析这一问题,提出了一种识别恶意软件中加密函数的方法。通过识别恶意软件动态执行路径中的循环、循环的输入和输出参数,构建恶意软件的动态循环数据流图,通过循环数据流图提取循环的输入和输出参数集合,设计已知加密函数的参考实现对循环输入集合中的元素进行运算,判断输出是否能够匹配输出集合中的元素从而识别恶意软件中的加密函数。实验证明此分析方法能够分析严重混淆的恶意软件其传输载荷所采用的加密函数。

关键词:加密函数识别;循环的输入和输出参数;循环数据流图;循环输入输出集合;动态二进制插桩
中图分类号:TP309.7　　**文献标志码**:A

Identification of encrypted function in malicious software

CAI Jianzhang*, WEI Qiang, ZHU Yuefei
(*Cyberspace Security Institute, Information Engineering University, Zhengzhou Henan 450002, China*)

Abstract: To resolve that the malware (malicious software) usually avoids security detection and flow analysis through encryption function, this paper proposed a scheme which can identify the encrypted function in malware. The scheme generated the dynamic loop data flow graph by identifying the loop and input/output of the loop in the dynamic trace. Then the sets of input/output were abstracted according to the loop data flow graph, the reference of known encrypted function was designed and the reference whose parameters were elements of the input sets was computed. If the result could match any element of the output sets, then the scheme could conclude the malware encrypts information by the known encrypted function. The experimental results prove that the proposed scheme can analyze the encrypted function of payload in the obfuscated malware.

Key words: encrypted function identification; input/output parameter of loop; loop data flow graph; input/output set in loop; dynamic binary instrument

0　引言

运用加密函数是保障信息安全的重要手段,同时也是恶意代码有效隐藏自身载荷的有效方法,因此有效识别软件中的加密函数、密钥信息能为软件分析破译提供必要的信息,同时也能够有效对恶意代码进行归类。加密函数的特征是有效识别软件中加密函数的重要手段,大多数现有的静态分析技术,如 KANAL(Krypto Analyzer)[1]、DRACA(Draft Crypto Analyzer)[2]、SIGNSRCH(Signatures Search)[3],都基于加密函数在二进制中常量特征或者指令特征,但静态分析技术很难应对恶意软件的加密混淆或者刻意的常量特征和指令特征的隐藏,同时很难对明密文信息和密钥信息进行详细分析。

通过循环迭代的方式对同一类型的数据进行相同的操作是加密函数的典型特征[4],如果二进制软件 p 包含了加密函数,则 p 中包含二进制形式的循环[5-7]。本文提出的恶意软件加密函数的识别方法依赖于恶意软件动态执行路径中的循环输入和输出参数集合,以动态执行路径中循环的输入参数集合中的元素作为已知加密函数参考实现的输入,经计算后的输出匹配动态执行路径中循环输出参数集合中的元素,如果匹配成功,则表明恶意软件在特定的执行路径中实现了已知加密函数。以上识别算法的实现依赖于以下实际应用中属于大概率事件的两个假设:

假设 1　设 f_1、f_2 是加密函数,对应输入 C 和密钥 K,若 $f_1(K,C)=f_2(K,C)=C'$,则可以认为 $f_1=f_2$,即((K,C),C')能够有效识别加密函数。

假设 2　如果软件 p 在特定程序路径上输入 C 和 K 产生了输出 C',根据假设 1,则表明 p 在特定的执行路径上实现了加密函数 f_1。

上述两个假设是本文识别方法的基础,即明文(密文)、密钥相同的情况下,若得到的密文(明文)相同,则表明加密函数相同;若程序 p 在特定的程序路径上输入 C 和 K 产生了输出 C',则表明程序 p 在特定的执行路径上实现加密函数 f_1。

1　相关工作

软件中加密函数的动态识别首先由 Lutz[4] 提出,利用程序执行路径中的三个特征识别加密函数:循环的存在、循环中位算数指令的运算的高比率、循环中代码对数据读写的熵值变化;Wang 等[8] 和 Caballero 等[9] 运用了相近的技术从加密通信中获取解密的数据,通过对污点数据的位算数指令运算的高比率识别软件中的加密函数;Calvet 等[10] 提出了通过动态程序路

收稿日期:2013-05-15;**修回日期**:2013-07-14。　**基金项目**:国家 863 计划项目(2008AA01Z420)。
作者简介:蔡建章(1978-),男,山东潍坊人,博士研究生,主要研究方向:信息安全;　魏强(1980-),男,陕西西安人,博士研究生,主要研究方向:信息安全;　祝跃飞(1962-),男,浙江杭州人,教授,博士生导师,主要研究方向:信息安全、可信计算。

径中的加密函数的输入和输出参数同已知加密函数进行比较，参数的识别通过执行路径中毗邻指令的内存引用操作数，这种方法对变长参数的加密函数如 RC4 无法识别，同时也对混淆技术敏感；Zhao 等[11]同样运用输入和输出关系识别加密函数，其假定的几个程序特征如 xor 指令的比率或者特定类型的函数面对混淆技术也会产生误报。总之混淆技术对上述方法带来很大的挑战，主要是混淆技术使得从程序中识别加密函数变得困难、混淆技术使得指令对数据的操作大幅增加导致提取加密函数的参数也变得非常困难，利用上述方法识别恶意软件中的加密函数同样面临混淆技术的干扰。

2 系统设计

本文提出一种利用循环数据流分析识别恶意软件中加密函数的方法，可以有效地抵抗混淆技术的干扰。利用循环数据流分析识别恶意软件中加密函数，其设计核心思想包括两个部分：1）识别恶意软件动态执行流程中的循环及其输入输出参数集合；2）设计已知加密函数的参考实现对所有输入参数进行运算，比较输出是否匹配动态执行流程中提取的输出参数。整体设计如下：p 为待检测的二进制应用，f 为已知加密函数的参考实现，C 为可选明文输入，K 为可选密钥输入，监测 p 中是否包含已知加密函数 f。

```
CryptDetect(p, f, C, K){
  I = (c, k);
  trace = dbi_trace(p, C, K);
  g = CreateLDF(trace);
  inputset = generateIParam(g);
  outputset = generateOParam(g);
  while(I ∈ inputset[ ]){
    if (f(I) ∈ outputset[ ])
      output(f, I, f(I))
    I = I->next;}}
```

上述函数描述如下：对于给定的软件 p，通过动态执行获得 p 的执行路径 trace；通过对 trace 文件中的循环识别，构建 trace 文件中的循环数据流图 g；从 g 中得到所有的循环输入参数集合、循环输出参数集合；以循环输入集合中的元素为输入，设计已知加密函数的参考实现 f 进行运算，判断输出是否能够匹配循环输出参数集合中的元素，从而实现 p 中的加密函数的识别。上述函数所描述的系统实现如图 1 所示。

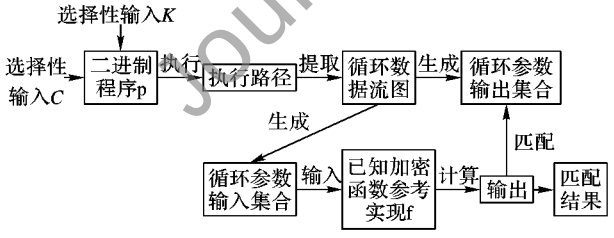

图 1 加密函数识别流程

3 算法实现

3.1 dbi_trace 的实现

本文通过 DBI(Dynamic Binary Instrument)获得程序的动态执行路径，实现了基于 Pin[12]的程序动态执行路径的获取插件，将指令 eip、反汇编指令的内容、操作数特征及其内容构成一个多元组，记录多元组的序列生成 trace 文件；多元组的内容包括指令的 eip，反汇编指令的内容，内存读写地址、操作数长度、操作数指向的地址内容，寄存器名称、寄存器长度、寄存器的内容。程序的执行路径真实地表示了程序在操作系统上的执行序列，本文将这样的多元组描述为动态指令 D(A,I,RM,WM,RR,WR)，因此 trace 文件 T 可以描述为 D 的有限序列，记作 $T = \langle D_1 \cdots D_n \rangle$，$T$ 中的指令序列记作 $T_{/\text{Ins}} = \langle I_1 \cdots I_n \rangle$，图 2 显示一个 trace 文件的动态指令格式。

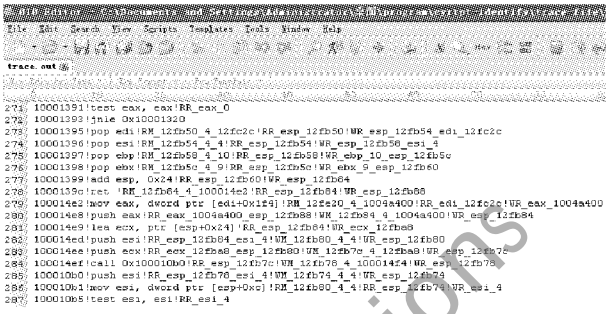

图 2 trace 文件格式

3.2 循环识别

3.2.1 动态执行路径中的循环

循环是加密函数的重要标志之一，即加密函数采用循环迭代的方式对同一个数据集合运用相同的处理方式，现有的二进制程序分析技术有以下三种循环识别方法：

1）利用控制流程图的回溯边识别循环。这种识别方法在文献[5]中提出，用于静态二进制程序分析，利用控制流程图中的回溯边识别循环，即控制流程图中存在特定的目标节点，其任意后继流程中存在返回该目标节点的回路。

2）以跳转地址识别循环。在文献[6]中提出一种循环的动态分析中识别方法，判断循环的存在依据为是否存在多个分支指令的跳转地址指向同一目标地址。

3）以指令序列重复识别循环。Kobayashi[7]将动态执行路径中的循环定义为一个特定指令序列的重复，图 3 显示以指令序列识别的循环。

在 3.1 节生成的动态程序执行路径中，3）所表示的循环明显符合加密函数的迭代操作在动态执行路径中的特征，因此本文给出如下定义：

定义 1 动态执行路径 trace 中的简单循环定义为指令序列的重复，记作 $L_{/\text{Ins}} \in \{I^n \mid I \in \text{X86}^+, n > 1\}$，其中：$I$ 为指令或者指令组合，X86 表示 x86 系统的指令集合。上述的循环可描述为在程序的动态执行路径上存在连续相同的指令或者指令序列。

但上述定义无法刻画如下情况，即存在嵌套循环，且嵌套循环迭代次数变化的指令序列，如图 4 所示。

因此本文采用一个助记标志，将定义 1 中的循环记录生成循环标志集合，记作 L_{id}，如果指令序列中存在 L_{id} 的元素，则用 L_{id} 中元素的助记标志代替其中的简单循环。比如图 4 中“B B B B B B”表示的内部循环生成助记标志 X 写入 L_{id} 集合，并将循环替换为 X，其中 X 的指令或者指令序列为 B，则下次出现“B B B B”时，则查找新构建的循环其指令或者指令集合是否为 B，如果是则采用其助记标志 X 代替，因此上述指令序列最终描述为“A X A X A”，则能够有效识别外围循环“A X”。通过以上分析本文给出循环在动态执行路径中的定义和循环的识别算法如下：

定义 2 动态执行路径 trace 中的循环为 $L_{/\text{Ins}} \in \{I^n \mid I \in$

$(X86 \bigcup L_{id})^+, n>1\}$,其中:$I$为指令或者指令组合,X86表示x86系统的指令集合,L_{id}表示循环标志集合,其元素为指令或者指令序列以及其标志。

```
10001382!jnle 0x10001372
10001372!mov ebp, dword ptr [ecx-0x4]!RM_12fbb0_4_e476a1c2!RR_ecx_12fbb4!WR_ebp_e476a1c2
10001375!sub ecx, 0x4!RR_ecx_12fbb4!WR_ecx_12fbb0
10001378!and ebp, edx!RR_edx_ffffff_ebp_e476a1c2!WR_ebp_76a1c2
1000137a!or ebp, dword ptr
[esi+ecx*1]!RM_12fb6c_4_ce000000!RR_ecx_12fbb0_ebp_76a1c2_esi_fffffffbc!WR_ebp_ce76a1c2
1000137d!dec eax!RR_eax_3!WR_eax_2
1000137e!mov dword ptr [ecx], ebp!RR_ecx_12fbb0_ebp_ce76a1c2!WM_12fbb0_4_ce76a1c2
10001380!test eax, eax!RR_eax_2
10001382!jnle 0x10001372
10001372!mov ebp, dword ptr [ecx-0x4]!RM_12fbac_4_ce20904d!RR_ecx_12fbb0!WR_ebp_ce20904d
10001375!sub ecx, 0x4!RR_ecx_12fbb0!WR_ecx_12fbac
10001378!and ebp, edx!RR_edx_ffffff_ebp_ce20904d!WR_ebp_20904d
1000137a!or ebp, dword ptr
[esi+ecx*1]!RM_12fb68_4_72000000!RR_ecx_12fbac_ebp_20904d_esi_fffffffbc!WR_ebp_7220904d
1000137d!dec eax!RR_eax_2!WR_eax_1
1000137e!mov dword ptr [ecx], ebp!RR_ecx_12fbac_ebp_7220904d!WM_12fbac_4_7220904d
!test eax, eax!RR_eax_1
10001382!jnle 0x10001372
```

图3　trace文件中的循环定义

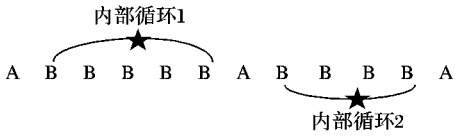

图4　trace中的嵌套循环

算法1　动态执行路径中的循环检测算法。

HistoryLoops表示已经构建好的指令和循环标志的序列,如上例中分析“A X A”;LoopInstance表示一个循环实例;RunningLoops表示按照嵌套循环构建的栈结构,栈顶元素为内层的循环,栈底元素为最下层循环;LoopStorage表示对一个简单循环的存储集合,即循环中的指令及其标志信息L_{id}组成的集合;match()函数采用递归函数匹配当前指令是否为一个循环标志中的指令;createLoops()函数根据当前指令跟history中的指令的匹配情况尝试构建循环实例[13]。

```
detectLoop( tracefile f)
{
  HistoryLoops history; LoopInstance L;
  RunningLoops RL; LoopStorage Storage;
  Tracefile f; Instruction ins;
  While( line = getline( f) ) {
    ins = Instruction( line) ;
    confirmedLoop = None;
    for ( L = getloopfrom( RL) ) {
      if ( match( Storage,  ins,  L, history)  = 1) {
        confirmedLoop = 1;
        break; } }
    if ( confirmedLoop = None) {
      append( ins,  history) ;
      if ( createLoops( Storage, ins, GetAllhisotryInstr( history) ,
        history) )
        append( ins,  Storage, RL)
      else cleanOnGoingLoops( L) } }
}
```

3.3　循环数据流分析

3.3.1　循环输入输出参数

按照3.2节中的定义和算法,取得程序执行路径中所有循环体及其内部结构,本文按照如下定义和算法提取简单循环中的输入输出参数。程序动态执行路径中的循环L的参数定义如下:

定义3

1)对于L的同一个参数,要么在指令内存操作数的内存地址中毗邻要么在L循环体内部的相同指令中指向同一个寄存器操作数,这个定义条件的设计可以将L在低层次上的多个参数合并为高层次上的参数。

2)对于L的同一个参数,L循环体中的同一条指令对其操作方式相同(读、写操作)。

3)输入参数在L的执行序列中读操作要先于写操作,输出参数在L中要进行写操作。

为了能够更直观地刻画简单循环的输入输出参数,本文根据指令操作数的特点引入一个二元组集合concrete(opd, sizeof(opd)),描述指令操作数的字节长度或寄存器长度,引入一个三元组集合var(opd,sizeof(content),content),描述指令操作数的内容长度及其内容。输入输出参数提取算法如下:

1)依据定义3中的前两个条件将以上二元组集合进行赋值及合并,依照第三个条件将二元组集合分为输入二元组和输出二元组集合;

2)根据动态执行路径中的信息,将输入二元组对应的第一次读操作的操作数、操作数指向内容和大小赋值输入三元组,将输出二元组对应的最后一次写操作的操作数、操作数指向内容和大小赋值输出三元组;

3)最后依据操作数的特点生成输入三元组集合IN_{mem}和IN_{reg},输出三元组集合OUT_{mem},OUT_{reg}。

3.3.2　循环数据流图

以上定义和算法描述了简单循环的输入输出参数的提取,但对于像RC4采取多次非嵌套循环实现的加密函数,则无法有效识别其参数,同时对于采用多重加密函数的软件也无法识别其加密函数以及其之间的关系,因此本文采用循环数据流图来实现循环之间输入输出数据流向的分析。为了简化分析,本文只采用IN_{mem}和OUT_{mem}之间的关系构建循环数据流图(IN_{reg}和OUT_{reg}则可能因为循环之间的距离较远而产生变化)。

定义 4 设$\{L_1 \cdots L_n\}$是动态执行路径中的循环序列，如果存在L_j之前执行的L_i满足$OUT_{mem}(L_i) \cap IN_{mem}(L_j)$不为空，则定义$L_i$和$L_j$之间存在输入输出关系$R$，记作$(L_i, L_j) \in R$，循环数据流图则定义为动态执行序列$\{L_1 \cdots L_n\}$上的所有的循环及其输入输出关系$R$，记作$G = (\{L_1 \cdots L_n\}, R)$。

循环数据流图G是一个有向无环图，其所有的有向子图描述了一个可能的加密函数的循环，将G中所有有向子图设为G_k，其根节点和叶子节点分别为$ROOT[G_k]$和$LEAF[G_k]$，循环数据流图中的输入和输出则分别定义为：

$$IN_{mem}[G] = \bigcup_{0<i<n} \left(IN_{mem}(L_i) - \bigcup_{(L_i,L_j)\in R} OUT_{mem}(L_i) \right)$$

$$IN_{reg}[G] = \bigcup_{L_r \in ROOT[G_k]} IN_{reg}(L_r)$$

$$OUT_{mem}[G] = \bigcup_{0<i<n} \left(OUT_{mem}(L_i) - \bigcup_{(L_i,L_j)\in R} IN_{mem}(L_j) \right)$$

$$OUT_{reg}[G] = \bigcup_{L_r \in LEAF[G_k]} OUT_{reg}(L_r)$$

G的输入输出参数可描述为：G的内存引用输入参数为循环序列的每个元素的内存引用输入参数减去同其后继元素存在R关系的当前元素的内存引用输出，G的寄存器输入参数为G中所有子树根节点的寄存器输入参数的总和，G的内存引用输出参数为循环序列的每个元素的内存引用输出参数减去同其后继元素存在R关系的后继元素的内存引用输入，G的寄存器输出参数为G中所有子树叶子节点的寄存器输出参数的总和。如一个简单循环数据流图$G = \{(L_1, L_2, L_3), R\}$，则其子图包括$G_1 = \{(L_1, L_2)\}$，$G_2 = \{(L_2, L_3)\}$，根据定义$L_1$的输出和$L_2$的输入存在交集，$L_2$的输出和$L_3$的输入存在交集，那么：

$$IN_{mem}[G] = IN_{mem}(L_1) + IN_{mem}(L_2) + IN_{mem}(L_3) - OUT_{mem}(L_1) - OUT_{mem}(L_2)$$

$$IN_{reg}[G] = IN_{reg}(L_1) + IN_{reg}(L_2)$$

$$OUT_{mem}[G] = OUT_{mem}(L_1) + OUT_{mem}(L_2) + OUT_{mem}(L_3) - IN_{mem}(L2) - IN_{mem}(L_3)$$

$$OUT_{reg}[G] = OUT_{reg}(L_2) + OUT_{reg}(L_3)$$

3.3.3 循环输入输出参数集合

循环数据流图的构建使得本分析方法能够准确地分析其内部采用多个并列循环的加密函数的输入输出参数，但对于采用多重加密函数以及先压缩再加密依然无法有效识别，其之间的关系也无从分析，为此本文引入两个集合：循环输入输出参数集合$IN_{set}[G]$和$OUT_{set}[G]$，用于改进对多重加密函数以及先压缩再加密的识别。

定义 5 对于给定的循环数据流图$G = (\{L_1 \cdots L_n\}, R)$，则$G$的循环输入集合为$IN_{set}[G] = IN_{par}[G] \cup IN_{par}[G_k] \cup IN_{par}(L_i)$，循环输出集合为$OUT_{set}[G] = OUT_{par}[G] \cup OUT_{par}[G_k] \cup OUT_{par}(L_i)$，其中：$G_k$是$G$的有向子图，$L_i$是$\{L_1 \cdots L_n\}$中的元素。

上述定义将G的输入、G中所有子图的输入、G中所有节点的输入构成一个输入集合$IN_{set}[G]$，将G的输出、G中所有子图的输出、G中所有节点的输出构成一个输出集合$OUT_{set}[G]$。假设循环数据流图$G = \{(L_1, L_2, L_3, L_4), R\}$，其中$L_1$对明文进行压缩，$\{L_2, L_3, L_4\}$对压缩后的数据进行 RC4 加密运算，则采用$G$的输入输出集合能够有效实现$\{L_2, L_3, L_4\}$对应的 RC4 加密运算的识别。

3.4 匹配算法

如何有效地将低层次循环的输入输出参数对应到高层次加密函数的输入输出是匹配函数要解决的核心问题，要解决上述匹配问题主要面临如下挑战：

1）低层次循环参数输入输出参数缺少类型的识别，循环输入输出参数集合提取的参数其内容为内存引用操作数指向的地址的内容或者寄存器指向的内容，其无法提供高层次加密函数所需要的参数类型。

2）参数序列。高层次加密函数进行运算需要提供正确的参数序列，循环输入输出参数集合无法有效识别。

3）参数个数。程序动态执行所需的上下文环境使得高层次加密函数进行运算所需要的参数个数跟低层次循环的参数个数匹配很困难。

4）参数分段传输。对于采用寄存器传输参数的多个并列循环则参数可能进行了分段传输，因此需要重新对低层次的寄存器参数进行组合。

通过上述分析，本文提出以下匹配算法：首先输入参数集合中的每个元素内部的输入参数进行任意组合，对输出参数进行任意组合；依据待匹配的加密函数参数的限制对上述组合形成的参数进行过滤；设计加密函数参考实现 f 对上述输入参数组合进行运算；比较运算结果是否在输出参数集合中。匹配算法如下：

算法 2 软件中的加密函数匹配。

```
compare( f, inputset, outputset) {
  constraint = getconstraint( f) ;
  while( par ∈ inputset[ ]) {
    if ( satisfy( constraint, par) {
      for( j = 0; j < numparof( f) ; j ++)
      var[ j] = combinefrom( par) ;
      if ( f( var0, ..., varj) ∈ output[ ]) {
        mark( f, var0, ..., varj) ; break; } }
    par = par -> next; }
}
```

4 实例分析

本文采用上述识别方法对某恶意软件的传输载荷所采用的加密函数进行了验证。某恶意软件采用 AES（Advanced Encryption Standard）-128 对其传输载荷进行加密，加密过程主要在 * CE. dll 中实现。通过对 * CE. dll 的静态分析和此恶意代码的模拟执行，通过 Pin 插件的运用获得了 AES-128 核心循环（扩展密钥中的 144 字节，对应 AES 算法中间的 9 次循环）的执行路径；经过循环数据流图分析和循环输入输出集合分析，构建的分组信息和密钥信息如图 5 所示，其中 0x12fbb0:16 为需要加密的分组，(0x12fc4c ~ 0x12fccc):16 为核心循环的 144 字节扩展密钥，上述部分为循环的输入集合的元素，0x100014d0 为 * CE. dll 中核心循环的起始地址，0x12fbb0:16 为循环的输出集合的元素。

设计 AES-128 的核心循环的参考实现运算 AES-128 (0x12fbb0:16,(0x12fc4c ~ 0x12fccc):144)，将得到的运算结果同 0x12fbb0:16 进行比较，成功实现了对此恶意代码的载荷所采用的加密函数的识别。

5　结语

本文提出了一种基于循环数据流的恶意软件加密函数识别方法,通过识别恶意软件动态执行路径中的循环输入和输出参数集合,运用已知密码函数的参考实现对输入集合中的元素进行运算,判断输出是否能够匹配输出集合中的元素从而识别恶意软件中的加密函数,实验证明本文方法能够有效应对混淆技术的干扰并能够识别动态执行过程中加密函数对应的详细信息。本文描述的循环参数识别、匹配函数算法有待于进一步改进和扩展,匹配函数实现中拟引入压缩函数的识别。

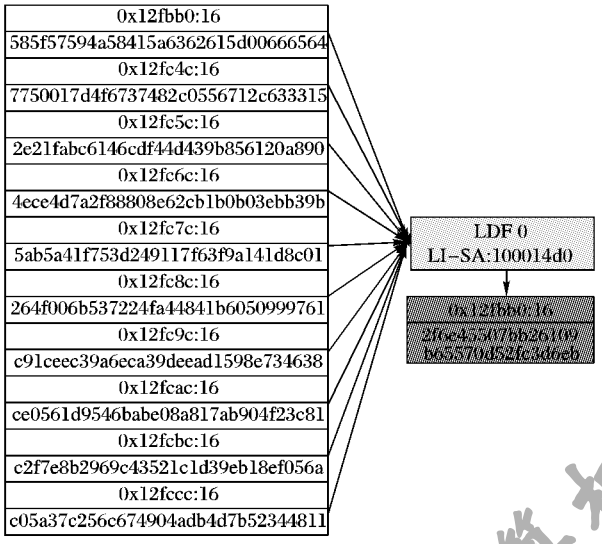

图 5　AES-128 核心循环输入输出

参考文献:

[1] PEiD Krypto Analyzer (KANAL). [2013-02-20]. http://www.softpedia.com/get/Programming/Other-Programming-Files/Kanal.shtml.

[2] LEVIN I O. Draft Crypto Analyzer (DRACA) [EB/OL]. [2003-05-01]. http://www.literatecode.com/draca.

[3] AURIEMMA L. SIGNSRCH tool [EB/OL]. [2013-04-25]. http://aluigi.altervista.org/mytoolz.htm.

[4] LUTZ N. Towards revealing attacker's intent by automatically decrypting network traffic [D]. Ztirich, Switzerland: ETH Zttrich, 2008.

[5] PUNTAMBEKAR A A. Principles of compiler design [M]. Maharashtra, India: Technical Publications, 2009.

[6] TUBELLA J, GONZALEZ A. Control speculation in multithreaded processors through dynamic loop detection [C]// Proceedings of the Fourth International Symposium on High-Performance Computer Architecture. Piscataway: IEEE Press, 1998: 14 - 23.

[7] KOBAYASHI M. Dynamic characteristics of loops [J]. IEEE Transactions on Computers, 1984, 100(2): 125 - 132.

[8] WANG Z, JIANG X X, CUI W D, *et al.* ReFormat: automatic reverse engineering of encrypted messages [M]// ESORICS'09: Proceedings of the 14th European Conference on Research in Computer Security. Berlin: Springer, 2009: 200 - 215.

[9] CABALLERO J, POOSANKAM P, KREIBICH C, *et al.* Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering [C]// Proceedings of the 16th ACM Conference on Computer and Communications Security. New York: ACM Press, 2009: 621 - 634.

[10] CALVET J, DAVIS C R, BUREAU P M. Malware authors don't learn, and that's good! [C]// Proceedings of the 4th International Conference on Malicious and Unwanted Software. Piscataway: IEEE Press, 2009: 88 - 97.

[11] ZHAO R X, GU D W, LI J R, *et al.* Detection and analysis of cryptographic data inside software [M]// ISC 2011: Proceedings of the 14th International Conference on Information Security. Berlin: Springer, 2011: 182 - 196.

[12] LUK C K, COHN R, MUTH R, *et al.* Pin: building customized program analysis tools with dynamic instrumentation [C]// Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2005: 190 - 200.

[13] CALVET J, FERNANDEZ J M, MARION J Y. Aligot: cryptographic function identification in obfuscated binary programs [C]// Proceedings of the 2012 ACM Conference on Computer and Communications Security. New York: ACM Press, 2012: 169 - 182.

(上接第 3182 页)

位置,提高了算法在目标存在尺度变化、旋转、遮挡时的鲁棒性。实验结果表明,该算法可以实现多运动目标的跟踪,且具有较好的鲁棒性。但 SIFT 的引入增加了算法的时间复杂度,今后应进一步优化算法,提高实时性。

参考文献:

[1] 张娟,毛晓波,陈铁军,等. 运动目标跟踪算法研究综述[J]. 计算机应用研究,2009,26(12):4407 - 4410.

[2] YANG H X, SHAO L, ZHENG F, *et al.* Recent advances and trends in visual tracking: a review [J]. Neurocomputing, 2011, 74(18): 3823 - 3831.

[3] YAO A B, LIN X G, WANG G J, *et al.* A compact association of particle filtering and kernel based object tracking [J]. Pattern Recognition, 2012, 45(7): 2584 - 2597.

[4] COMANICIU D, RAMESH V, MEER P. Kernel-based object tracking [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003, 25(5): 564 - 575.

[5] CANNONS K. A review of Visual tracking, CSE-2008-07 [R]. Toronto: York University, 2008.

[6] LOWE D G. Distinctive image features from scale invariant key points [J]. International Journal of Computer Vision, 2004, 60(2): 91 - 110.

[7] ZHU C Y. Video object tracking using SIFT and mean shift [D]. Gothenburg: Chalmers University of Technology, 2011.

[8] CHEN A H, ZHU M, WANG Y H, *et al.* Mean shift tracking combining SIFT [C]// Proceedings of the 9th International Conference on Signal Processing. Piscataway: IEEE Press, 2008: 1532 - 1535.

[9] 翟海涛,吴建,陈建明,等. 基于 SIFT 特征度量的 Mean Shift 目标跟踪算法[J]. 计算机应用与软件,2011,28(6):7 - 50.

[10] 周尚波,胡鹏,柳玉炯,等. 基于改进 Mean-Shift 与自适应 Kalman 滤波的视频目标跟踪[J]. 计算机应用,2010,30(6):1573 - 1576.

[11] 刘继艳,潘建寿,吴亚鹏,等. 结合 Kalman 滤波器的 Mean-Shift 跟踪算法[J]. 计算机工程与应用,2009,45(12):184 - 186.

[12] LINDEBERG T. Scale-space theory: a basic tool for analyzing structures at different scales [J]. Journal of Applied Statistics, 1994, 21(1/2): 225 - 270.