Code Snippets

Khaled Yakdan

Institute of Computer Science 4
University of Bonn, Germany

This document presents the decompiled output generated by DREAM [1] and Hex-Rays for selected functions. The presented functions contain functions from malware as well as benign programs. The reader can get a personal impression on the readability improvements offered by DREAM compared to Hex-Rays.

1 Functions taken from malware samples

Here we show code snippets from our experiments of decompiling three samples from the ZeusP2P, Cridex, and SpyEye families.

1.1 Cridex 1

Hex-Rays generated the following code

```
int __usercall sub_408A70<eax>(int a1<ebx>, int a2){
  int v2; // ebp@1
  int v3; // esi@1
  unsigned int v4; // eax@1
  int v5; // esi@1
  int v6; // edi@2
  int v7; // ecx@3
  int v8; // edx@5
  int v9; // eax@7
  int v10; // edi@10
  int v11; // esi@11
  char v12; // al@12
  int v13; // edx@14
  int v14; // eax@16
  bool v15; // cf@24
  int result; // eax@25
  int v17; // [sp+Ch] [bp-Ch]@2
  int v18; // [sp+10h] [bp-8h]@1
  unsigned int v19; // [sp+14h] [bp-4h]@1
  int v20; // [sp+1Ch] [bp+4h]@1

  v2 = a2;
  v3 = *(_DWORD *)(*(_DWORD *)(a2 + 60) + a2 + 120);
  v4 = *(_DWORD *)(v3 + a2 + 24);
  v5 = a2 + v3;
  v18 = v5;
  v20 = 0;
  v19 = v4;
  if ( v4 ){
    v6 = v2 + *(_DWORD *)(v5 + 32);
    v17 = v2 + *(_DWORD *)(v5 + 32);
    while ( 1 ){
      v7 = v2 + *(_DWORD *)v6;
      if ( v7 && a1 ){
        v8 = 0;
        if ( *(_BYTE *)v7 ){
          do
            ++v8;
          while ( *(_BYTE *)(v8 + v7) );
        }
```

```
        v9 = 0;
        if ( *(_BYTE *)a1 ){
          do
            ++v9;
          while ( *(_BYTE *)(v9 + a1) );
        }
        if ( v8 == v9 )
          break;
      }
LABEL_24:
      v6 += 4;
      v15 = v20++ + 1 < v19;
      v17 = v6;
      if ( !v15 )
        goto LABEL_25;
    }
    v10 = v8 + v7;
    if ( v7 < (unsigned int)(v8 + v7) ){
      v11 = a1 - v7;
      do{
        v12 = *(_BYTE *)v7;
        if ( *(_BYTE *)v7 < 'A' || v12 > 'Z' )
          v13 = v12;
        else
          v13 = v12 + 32;
        LOBYTE(v14) = *(_BYTE *)(v11 + v7);
        if ( (char)v14 < 'A' || (char)v14 > 'Z' )
          v14 = (char)v14;
        else
          v14 = (char)v14 + 32;
        if ( v13 != v14 ){
          v5 = v18;
          v6 = v17;
          goto LABEL_24;
        }
        ++v7;
      }while ( v7 < (unsigned int)v10 );
      v5 = v18;
    }
    result = v2 + *(_DWORD *)(*(_DWORD *)(v5 + 28) + 4 * *(_WORD *)(*(_DWORD *)(v5 + 36) + 2
    * v20 + v2) + v2);
  }
  else{
LABEL_25:
    result = 0;
  }
  return result;
}
```

DREAM generated the following code

```
void* sub_408A70(char* str1, void* a1){
    int v1 = *(*(a1 + 0x3c) + a1 + 0x78);
    int v2 = *(v1 + a1 + 0x18);
    void* v3 = v1 + a1;
    int i = 0;
    if(v2){
        int v4 = *(v3 + 0x20);
        void* v5 = v4 + a1;
        void* v6 = v4 + a1;
        do{
            v6 = v5;
            char* str = *(v6) + a1;
            if(str && str1){
```

```
                size_t len = strlen(str);
                if(len == strlen(str1)){
                    bool cond1 = str >= len +str;
                    int result;
                    if(!cond1)
                        result = strcmpi(str, str1);
                    if(cond1 || !result)
                        return *(*(v3 + 0x1c) + *(*(v3 + 0x24) + i * 2 + a1) * 4 + a1) + a1;
                }
            }
            i++;
            v5 = v6 + 4;
            v6 += 4;
        }while(i < v2);
    }
    return 0;
}
```

1.2 Cridex 2

Hex-Rays generated the following code

```
signed int __usercall encrypt_session_key<eax>(int a1<edi>, HCRYPTKEY hKey, HCRYPTKEY
    hExpKey)
{
  signed int v3; // esi@1
  BYTE *v4; // eax@1
  int v5; // eax@4
  DWORD v6; // esi@4
  LPVOID lpMem; // [sp+4h] [bp-Ch]@1
  DWORD pdwDataLen; // [sp+8h] [bp-8h]@2
  int v10; // [sp+Ch] [bp-4h]@2

  v3 = 0;
  v4 = (BYTE *)HeapAlloc(hHeap, 8u, 0x8Cu);
  lpMem = v4;
  if ( v4 )
  {
    pdwDataLen = 140;
    v10 = 0;
    if ( CryptExportKey(hKey, hExpKey, SIMPLEBLOB, 0, v4, &pdwDataLen) )
    {
      if ( pdwDataLen != 12 )
      {
        v5 = a1;
        v6 = pdwDataLen - 12;
        do
        {
          *(_BYTE *)v5 = *((char *)lpMem + v5 - a1 + 12);
          ++v5;
          --v6;
        }
        while ( v6 );
      }
      v3 = 1;
    }
    HeapFree(hHeap, 0, lpMem);
  }
  return v3;
}
```

DREAM generated the following code

```
encrypt_session_key(HCRYPTKEY hKey, HCRYPTKEY hExpKey, char* destination){
```

```
    int result = 0;
    LPVOID pbData = HeapAlloc(hHeap, 8, 0x8c);
    if(pbData){
        DWORD pdwDataLen = 0x8c;
        if(CryptExportKey(hKey, hExpKey, 1, 0, pbData, &(pdwDataLen))){
            memcpy(destination, pbData + 0xc, pdwDataLen - 0xc);
            result = 1;
        }
        HeapFree(hHeap, 0, pbData);
    }
    return result;
}
```

1.3 Cridex 3

Hex-Rays generated the following code

```
int __usercall ftp_pop_1_1<eax>(int a1<eax>, int a2<edx>, int a3)
{
  unsigned int v3; // ecx@1
  int result; // eax@4
  int v5; // esi@6
  int v6; // edi@11
  int v7; // eax@11

  v3 = 0;
  if ( !a1 )
    goto LABEL_4;
  while ( *(_BYTE *)(v3 + a2) != 32 )
  {
    ++v3;
    if ( v3 >= a1 )
      goto LABEL_4;
  }
  if ( (signed int)v3 <= 0 )
    goto LABEL_4;
  v5 = v3 + 1;
  *(_DWORD *)a3 = a2;
  *(_DWORD *)(a3 + 4) = v3;
  if ( (signed int)(v3 + 1) >= a1 )
    goto LABEL_4;
  while ( *(_BYTE *)(v5 + a2) == 32 )
  {
    ++v5;
    if ( v5 >= a1 )
      return 0;
  }
  if ( v5 < a1 && (v6 = v5 + a2, v7 = sub_401F90(v5 + a2), v7 > 0) )
  {
    *(_DWORD *)(a3 + 8) = v6;
    *(_DWORD *)(a3 + 12) = v7;
    result = v7 + v5 + 2;
  }
  else
  {
LABEL_4:
    result = 0;
  }
  return result;
}
```

DREAM generated the following code

```
ftp_pop_1_1(int a1, int a2, void* a3){
```

```
    if(!a1)
        return 0;
    int i = 0;
    do{
        if(*(i + a2) == 0x20){
            if(i <= 0)
                return 0;
            int j = i + 1;
            *(a3) = a2;
            *(a3 + 4) = i;
            if(j >= a1)
                return 0;
            do{
                if(*(j + a2) != 0x20){
                    if(j >= a1)
                        return 0;
                    int v1 = j + a2;
                    int v2 = sub_401F90(v1);
                    if(v2 <= 0)
                        return 0;
                    else{
                        *(a3 + 8) = v1;
                        *(a3 + 0xc) = v2;
                        return v2 + j + 2;
                    }
                }
                j++;
            }while(j < a1);
            return 0;
        }
        i++;
    }while(i < a1);
    return 0;
}
```

1.4 Cridex 4

Hex-Rays generated the following code

```
int __usercall sub_403850<eax>(int a1<esi>)
{
  int v1; // edi@1
  int v2; // eax@1
  int v3; // ecx@2

  v1 = 0;
  v2 = get_process_list();
  if ( v2 )
  {
    v3 = v2;
    if ( *(_DWORD *)(v2 + 68) == a1 )
    {
LABEL_5:
      v1 = *(_DWORD *)(v3 + 4);
    }
    else
    {
      while ( *(_DWORD *)v3 )
      {
        v3 += *(_DWORD *)v3;
        if ( *(_DWORD *)(v3 + 68) == a1 )
          goto LABEL_5;
      }
```

```
    }
    HeapFree(hHeap, 0, (LPVOID)v2);
  }
  return v1;
}
```

DREAM generated the following code

```
sub_403850(int a1){
    int result = 0;
    void* lpMem = get_process_list();
    if(lpMem){
        void* v1 = lpMem;
        bool cond1 = *(lpMem + 0x44) == a1;
        int i;
        if(!cond1)
            do{
                i = *(v1);
                if(i)
                    v1 += *(v1);
            }while(*(v1 + 0x44) != a1 && i);
        if(cond1 || i)
            result = *(v1 + 4);
        HeapFree(hHeap, 0, lpMem);
    }
    return result;
}
```

1.5 ZeusP2P

Hex-Rays generated the following code

```
int __cdecl sub_43E100(int a1, unsigned int a2)
{
  int v2; // ecx@1
  unsigned int v3; // esi@1
  unsigned int v4; // edx@1
  int result; // eax@1
  unsigned int v6; // ecx@2

  v2 = a1 + *(_DWORD *)(a1 + 60);
  v3 = *(_WORD *)(v2 + 6);
  v4 = 0;
  result = *(_WORD *)(v2 + 20) + v2 + 24;
  if ( *(_WORD *)(v2 + 6) )
  {
    while ( 1 )
    {
      v6 = *(_DWORD *)(result + 12);
      if ( a2 >= v6 && a2 < v6 + *(_DWORD *)(result + 8) )
        break;
      ++v4;
      result += 40;
      if ( v4 >= v3 )
        goto LABEL_5;
    }
  }
  else
  {
LABEL_5:
    result = 0;
  }
  return result;
}
```

DREAM generated the following code

```
sub_43E100(void* a1, int a2){
    void* v1 = *(a1 + 0x3c) + a1;
    int v2 = *(v1 + 0x14);
    int v3 = *(v1 + 6);
    int i = 0;
    void* result = v2 + v1 + 0x18;
    int v5;
    int v4;
    if(v3)
        do{
            v4 = *(result + 0xc);
            if(a2 >= v4)
                v5 = *(result + 8) + v4;
            if(a2 < v5 && a2 >= v4)
                break;
            i++;
            result += 0x28;
        }while(i < v3);
    if(!v3 || a2 < v4 || a2 >= v5)
        result = 0;
    return result;
}
```

1.6 SpyEye

Here Hexrays produces a decompilation error since local variables v19 and v21 are used in line 21 before being initialized in the function. Hexrays generated the following code

```
signed int __cdecl sub_4638BB(unsigned int a1, unsigned int a2, unsigned int a3, unsigned
    int *a4, unsigned int *a5, int a6, int a7, int a8)
{
  HANDLE v8; // eax@1
  unsigned int v9; // eax@1
  unsigned int v10; // edi@1
  signed int v12; // eax@3
  signed int v13; // eax@5
  HANDLE v14; // eax@8
  HANDLE v15; // eax@9
  bool v16; // zf@12
  HANDLE v17; // eax@16
  HANDLE v18; // eax@17
  DWORD v19; // [sp+0h] [bp-14h]@0
  DWORD v20; // [sp+0h] [bp-14h]@1
  SIZE_T v21; // [sp+4h] [bp-10h]@0
  void *v22; // [sp+4h] [bp-10h]@1
  int v23; // [sp+Ch] [bp-8h]@3
  int v24; // [sp+10h] [bp-4h]@3

  v8 = GetProcessHeap();
  v9 = (unsigned int)HeapAlloc(v8, v19, v21);
  v10 = v9;
  if ( !v9 )
    return -4;
  v23 = 0;
  v12 = sub_4634E2(a4, a3, a1, 0x101u, (int)dword_4571A0, (int)dword_457220, a6, a8, (int)&
    v23, v9);
  v24 = v12;
  if ( v12 )
  {
    v16 = v12 == -4;
    goto LABEL_14;
```

```
  }
  if ( !*a4 )
    goto LABEL_16;
  v13 = sub_4634E2(a5, a3 + 4 * a1, a2, 0, (int)dword_4572A0, (int)dword_457318, a7, a8, (
    int)&v23, v10);
  v24 = v13;
  if ( v13 )
  {
    if ( v13 == -4 )
      goto LABEL_15;
    v16 = v13 == -5;
LABEL_14:
    if ( !v16 )
      goto LABEL_16;
LABEL_15:
    v24 = -3;
    goto LABEL_16;
  }
  if ( *a5 || a1 <= 0x101 )
  {
    v14 = GetProcessHeap();
    if ( HeapValidate(v14, 0, (LPCVOID)v10) )
    {
      v15 = GetProcessHeap();
      HeapFree(v15, v20, v22);
    }
    return 0;
  }
LABEL_16:
  v17 = GetProcessHeap();
  if ( HeapValidate(v17, 0, (LPCVOID)v10) )
  {
    v18 = GetProcessHeap();
    HeapFree(v18, v20, v22);
  }
  return v24;
}
```

DREAM generated the following code

```
sub_4638BB(int a1, int a2, int a3, int a4, void* a5, void* a6, int a7, int a8){
    LPCVOID lpMem = HeapAlloc(GetProcessHeap(), 8, 0x481);
    if(!lpMem)
        return -4;
    int v1 = 0;
    int v2 = sub_4634E2(a1, a2, 0x101, &(dword_4571A0), &(dword_457220), a3, a4, &(v1),
    lpMem);
    bool cond1 = v2;
    bool v3 = false;
    if(cond1)
        v3 = v2 == -4;
    bool cond3 = false;
    int v4;
    bool cond2 = !*(a5);
    if(!cond1 && !cond2){
        v4 = sub_4634E2(a1 + a2 * 4, a7, 0, &(dword_4572A0), &(dword_457318), a8, a4, &(v1),
    lpMem);
        v2 = v4;
        if(v4){
            cond3 = v4 == -4;
            if(!cond3)
                v3 = v4 == -5;
        } else if(*(a6) != v4 || a2 <= 0x101){
            if(HeapValidate(GetProcessHeap(), 0, lpMem))
```

```
                HeapFree(GetProcessHeap(), 0, lpMem);
            return 0;
        }
    }
    if((cond1 || v4) && (cond1 || !cond2) && (cond3 || !v3) && (!cond1 || !v3))
        v2 = -3;
    if(!HeapValidate(GetProcessHeap(), 0, lpMem))
        return v2;
    HeapFree(GetProcessHeap(), 0, lpMem);
    return v2;
}
```

2 Some nonmalware examples

Here are some functions we wrote to illustrates DREAM ability to structure certain cases in cases where Hex-Rays cannot.

2.1 For loop

Hex-Rays generated the following code

```
int __cdecl for_1(int a1, int a2, int a3)
{
  char v3; // al@7
  int v5; // [sp+4h] [bp-Ch]@1
  int v6; // [sp+8h] [bp-8h]@1
  int v7; // [sp+Ch] [bp-4h]@1

  v5 = 0;
  v6 = 0;
  v7 = 0;
  while ( 1 )
  {
    v3 = v6 + 1 >= a3 && v5 >= a3 ? 0 : 1;
    if ( !v3 )
      break;
    v7 += v5 + v6;
    if ( !(v5 & 1) )
      v5 += 2;
    ++v5;
    ++v6;
  }
  return v7;
}
```

DREAM generated the following code

```
for_1(int a1){
    int result = 0;
    int i;
    int j;
    for(i = 0, j = 0; j + 1 < a1 || i < a1; i++, j++){
        result += j + i;
        if(!(i & 1))
            i += 2;
    }
    return result;
}
```

2.2 Overlapping loop

Overlapping loops share parts of there bodies without being contained in one another.

Hex-Rays generated the following code

```
int __cdecl multi_exit_loop(int a1, int a2)
{
  int v3; // [sp+4h] [bp-Ch]@1
  int v4; // [sp+8h] [bp-8h]@1
  int v5; // [sp+Ch] [bp-4h]@1

  v3 = 0;
  v4 = 0;
  v5 = 0;
LABEL_2:
  ++v3;
  do
  {
    v5 += v3;
    if ( v3 < a1 )
      goto LABEL_2;
    v5 += v4;
    v3 = 0;
    ++v4;
  }
  while ( v4 < a2 );
  return v5;
}
```

DREAM generated the following code

```
multi_exit_loop(int a1, int a2){
    int v1 = 0;
    int i = 0;
    int result = 0;
    bool cond1 = false;
    do{
        v1++;
        do{
            result += v1;
            cond1 = v1 < a1;
            if(cond1)
                break;
            result += i;
            i++;
            v1 = 0;
        }while(i < a2);
    }while(cond1);
    return result;
}
```

References

1. Khaled Yakdan, Sebastian Eschweiler, Elmar Gerhards-Padilla, and Matthew Smith. No More Gotos: Decompilation Using Pattern-Independent Control-Flow Structuring and Semantics-Preserving Transformations. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS)*, 2015.