Helping Johnny To Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study

Khaled Yakdan  
*University of Bonn*  
Fraunhofer FKIE

Sergej Dechand  
*University of Bonn*

Elmar Gerhards-Padilla  
Fraunhofer FKIE

Matthew Smith  
*University of Bonn*

IEEE Symposium on Security and Privacy 2016
Which code would you rather analyze?

int bar(int a1){
    int v1 = 0;
    int v2 = 1;
    if(a1 > 0){
        do{
            v2 = v2 * foo(v1);
            v1 = v1 + 1;
        } while(v1 < a1);
    }
    return v2;
}

int bar(int max){
    int result = 1;
    for(int i = 0 ; i < max ; i++){
        result = result * foo(i);
    }
    return result;
}
Decompilation

Source Code

```
i = 0;
while(i < size){
    foo(i);
    i = i + 1;
}
```

Decompiled Code

```
v2 = 0;
if(v1 != 0){
    do{
        foo(v2);
        v2 = v2 + 1;
    } while(v2 < v1);
}
```

High-level abstractions are lost

Recovered abstractions
Previous Work on Decompilation

• Do not focus on readability
• Do not include user studies in the evaluation
  • Readability metrics:
    • Compression ratio (smaller is better?)
    • Number of gotos (less is better?)
Our Work on Decompilation

1010010101
0010101010
1001010101
0000100100
0011100101

v2 = 0;
if(v1 != 0){
do{
   foo(v1);
   v2 = v2 + 1;
} while(v2 < v1);
}

for(i = 0; i < size, i++){foo(i);}

This work:
1. Usability extensions to DREAM
2. Malware analysis user study
Usability Extensions to DREAM
Solved Readability Problems

Complex Expressions
- Redundant variables
- Logic expressions
- Pointer arithmetic

Convoluted Control Flow
- Duplicate/Inlined Code
- Complex loop structure

Lack of Semantics
- Unrepresentative variable names
- Named constants
Hex-Rays: Domain generation algorithm (Simda)

```c
void *__cdecl sub_10006390(){
    __int32 v13; // eax@14
    int v14; // esi@15
    unsigned int v15; // ecx@15
    int v16; // edx@16
    char *v17; // edi@18
    bool v18; // zf@18
    unsigned int v19; // edx@18
    char v20; // dl@21
    char v23; // [sp+0h]
    int v30; // [sp+30Ch]
    __int32 v36; // [sp+324h]
    int v37; // [sp+328h]
    int i; // [sp+330h]
    // [...] v30 = "qwrtpsdfghjkllzxcvbnm";
    // [...] v37 = "eyuioa";
    // [...]
    v14 = 0;
    v15 = 3;
}

if ( v13 > 0 ) {
    v16 = 1 - &v23;
    for ( i = 1 - &v23; ; v16 = i ) {
        v20 = v18 ? (*(&v37 + dwSeed / v15 % 6)) : (*(&v30 + dwSeed / v15 % 0x14));
        ++v14;
        v15 += 2;
        if ( v14 >= v36 )
            break;
    }
    // [...]}
```
DREAM++: Domain generation algorithm (Simda)

LPVOID sub_10006390(
{
    char * v1 = "qwrtpsdfghjklzxcvbnm";
    char * v2 = "eyuioa";
    // [...]
    int v13 = 3;
    for(int i = 0; i < num; i++){
        char v14 = i % 2 == 0 ? v1[(dwSeed / v13) % 20] : v2[(dwSeed / v13) % 6];
        v13 += 2;
        v3[i] = v14;
    }
    // [...]
}
Malware Analysis User Study
User Study

- Tested Decompilers
  - DREAM++ (readability improvements)
  - DREAM
  - Hex-Rays
- 6 malware reverse engineering tasks
  - Counterbalanced decompiler order
  - Counterbalanced task order
- User perception after each task
- Feedback at the end of the study
Task Selection

- Independent professional malware analysts
- 6 Tasks
  - Encryption (Stuxnet)
  - Custom Encoding (Stuxnet)
  - Resolving API Dynamically (Cridex)
  - String Parsing (URLZone)
  - Download and execute (Andromeda)
  - Domain generation algorithms (Simda)
Participants

Two groups

1. Students
   - 36 invited
   - 21 completed the study

2. Professional malware analysts
   - 31 invited
   - 17 started the study
   - 9 completed the study
## Results

<table>
<thead>
<tr>
<th>Decompiler</th>
<th>Average Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Students</td>
</tr>
<tr>
<td>DREAM++</td>
<td>70.24</td>
</tr>
<tr>
<td>DREAM</td>
<td>50.83</td>
</tr>
<tr>
<td>Hex-Rays</td>
<td>37.86</td>
</tr>
</tbody>
</table>
Results

Students
• Solved 3 times as many tasks with DREAM++ as with Hex-Rays

Professional malware analysts
• Solved 1.5 times as many tasks with DREAM++ as with Hex-Rays
User Perception

• 8 Questions
  • 6 Usability
  • 2 Trust

• Questions are counterbalanced (positive/negative) to minimize the response bias
User Perception

“The code mostly looks like a straightforward C translation of machine code; besides a general sense about what is going on, I think I'd rather just see the assembly.” - DREAM

“This code looks like it was written by a human, even if many of the variable names are quite generic. But just the named index variable makes the code much easier to read! ” – DREAM++
Final Feedback

- Show code produced by all decompilers side by side
- Scores from 1 (worst) to 10 (best)
Summary and Future Work

• Readability improvements to DREAM
• First malware analysis user study
• Human-centric approach can significantly improve the effectiveness of decompilers
• Focus on other use cases
  • Vulnerability search in binary code