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

universitätbonn Fraunhofer FKIE

Which code would you rather analyze?

```
080483f9 <bar>:
 80483f9: push   %ebp
 80483fa: mov    %esp,%ebp
 80483fc: sub    $0x10,%esp
 80483ff: movl   $0x0,-0x4(%ebp)
 8048406: movl   $0x1,-0x8(%ebp)
 804840d: cmpl   $0x0,0x8(%ebp)
 8048411: jle    8048435 <bar+0x3c>
 8048413: pushl  -0x4(%ebp)
 8048416: call   80483cb <foo>
 804841b: add    $0x4,%esp
 804841e: mov    %eax,%edx
 8048420: mov    -0x8(%ebp),%eax
 8048423: imul   %edx,%eax
 8048426: mov    %eax,-0x8(%ebp)
 8048429: addl   $0x1,-0x4(%ebp)
 804842d: mov    -0x4(%ebp),%eax
 8048430: cmp    0x8(%ebp),%eax
 8048433: jl     8048413 <bar+0x1a>
 8048435: mov    -0x8(%ebp),%eax
 8048438: leave
 8048439: ret
```

```
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

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

Source Code

Compiler

High-level abstractions are lost

```
1010010101
0010101010
1001010101
0000100100
0011100101
```

Binary Code

Decompiler

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

Decompiled Code

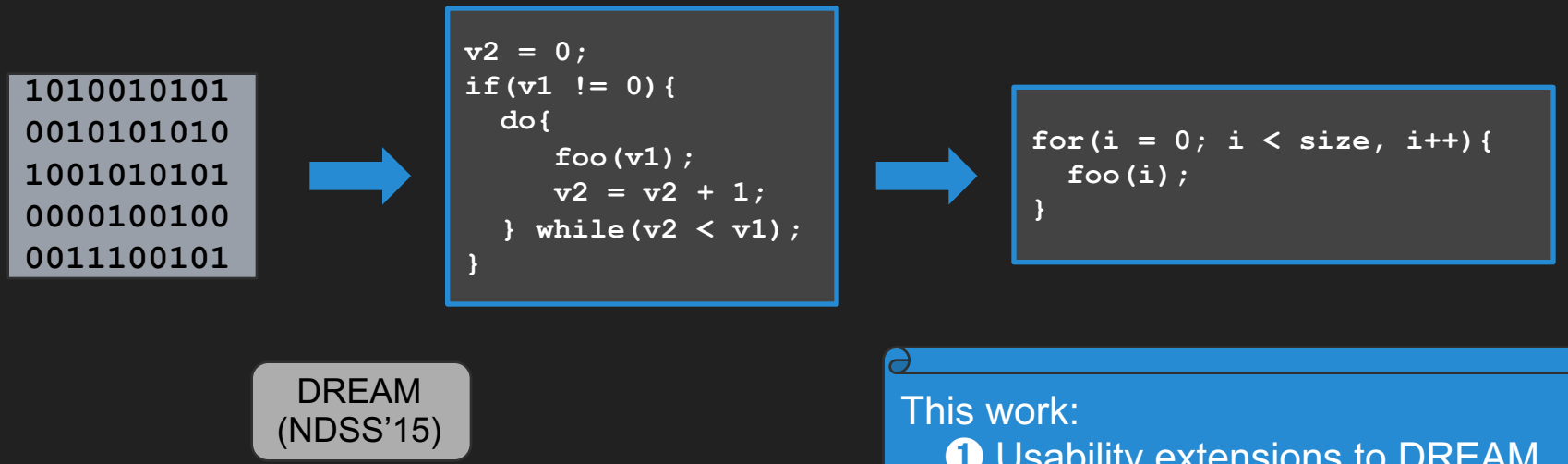
Recovered abstractions

universitätbonn Fraunhofer FKIE

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



universitätbonn Fraunhofer FKIE

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)

```
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; // ed
  char v20; // dl@21
  char v23; // [sp+0h]
  int v30; // [sp+30Ch]
  __int32 v36; // [sp+324h] [bp-14h]@14
  int v37; // [sp+328h] [bp-10h]@1
  int i; // [sp+330h] [bp-8h]@1
  // [...]
  v30 = *"qwrtpsdfghjklzxcvbnm";
  v37 = *"eyuioa";
  // [...]
  v14 = 0;
  v15 = 3;
```

Complex logic

Pointer Arithmetic

Many Variables

```
  if ( v13 > 0 )
  {
    v16 = 1 - &v23;
    for ( i = 1 - &v23; ; v16 = i )
    {
                                  4;
                                 14 + v16) & 0x80000001;
      if ( (v19 & 0x80000000) != 0 )
        v18 = ((v19 - 1) | 0xFFFFFFFE) == -1;
      v20 = v18 ? *(&v37 + dwSeed / v15 % 6)
                : *(&v30 + dwSeed / v15 % 0x14);
      ++v14;
      v15 += 2;
      *v17 = v20;
      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;
  }
  // [...]
}
```

universitätbonn Fraunhofer FKIE

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

universitätbonn Fraunhofer FKIE

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

universitätbonn Fraunhofer FKIE

Results

Decompiler	Average Score (%)	
	Students	Experts
DREAM++	70.24	84.72
DREAM	50.83	79.17
Hex-Rays	37.86	61.39

universitätbonn Fraunhofer FKIE

Results

Students

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

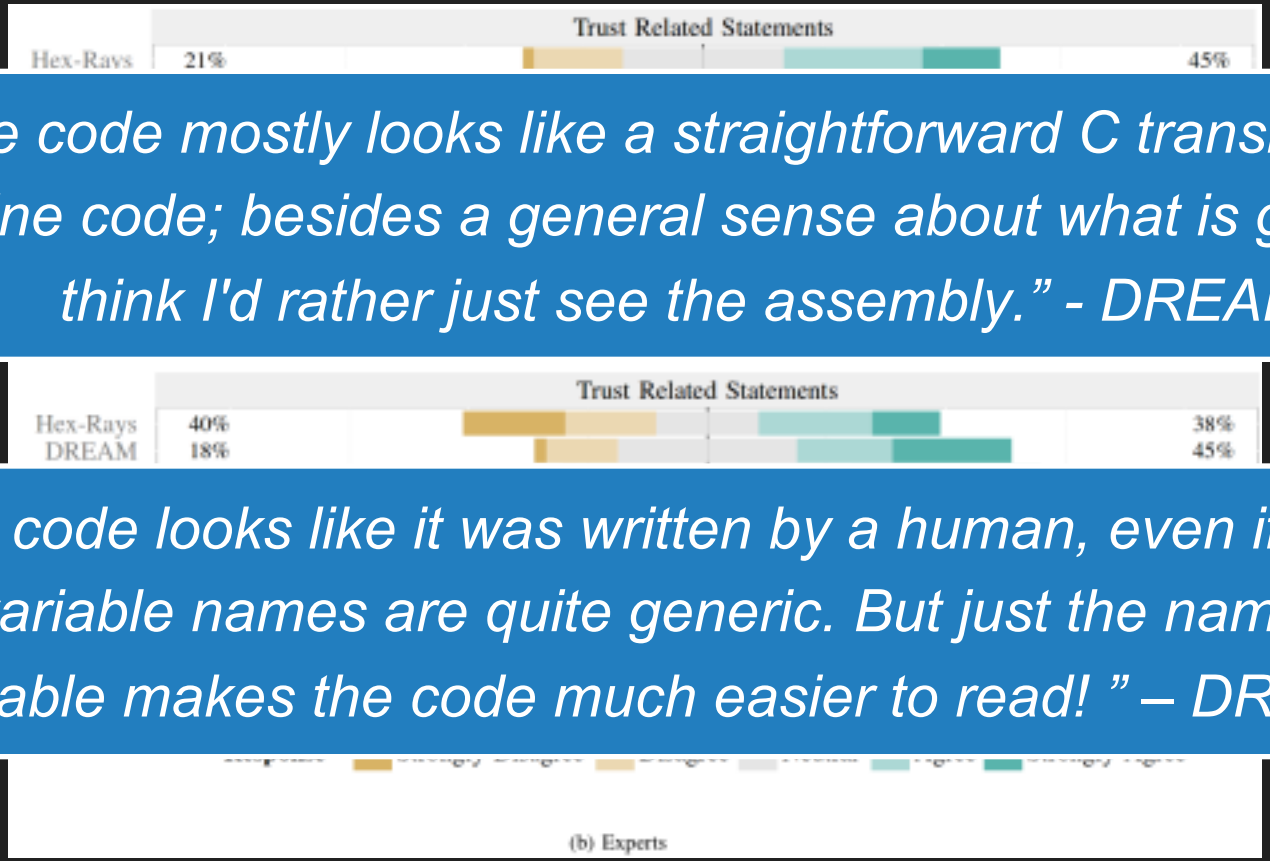
Professional malware analysts

- 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

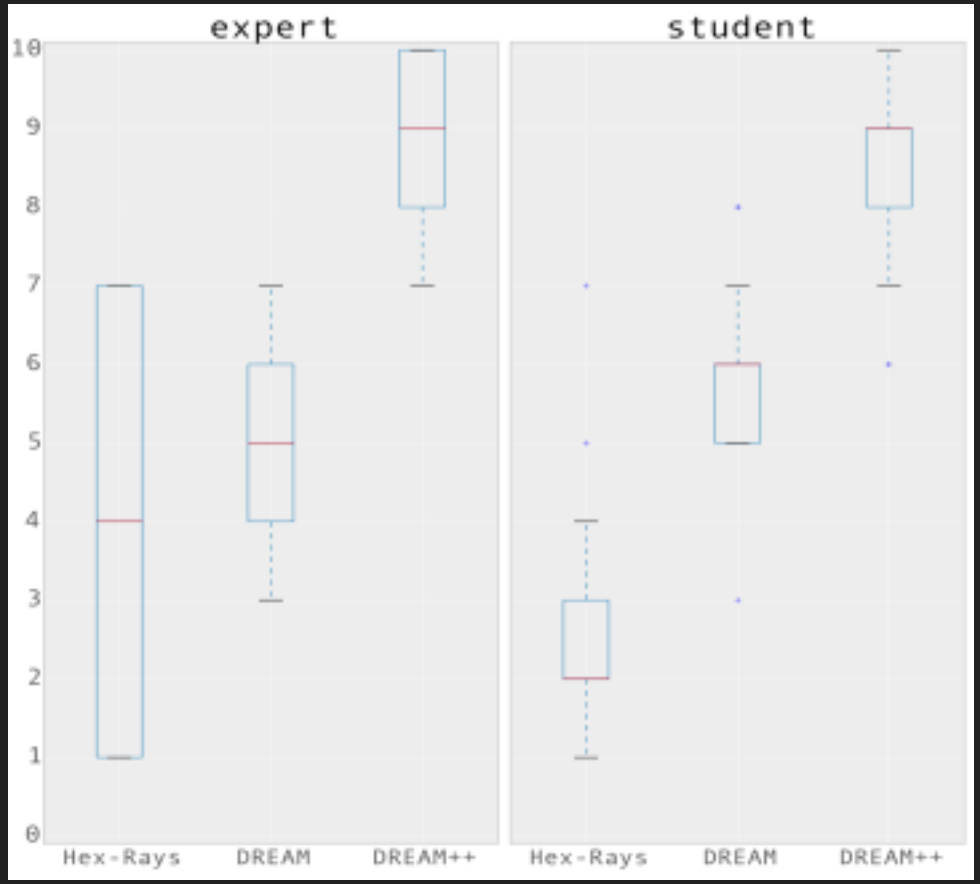


“The code mostly looks like a straightforward C translation of machine code; besides a general sense about what is going on, I think I'd rather just see the assembly.” - DREAM

“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)



universitätbonn Fraunhofer FKIE

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