Decompiler User Survey 1

October, 2020

1 Survey

1.1 Baseline Questions

What is your highest level of education?

- ○ School Graduation
- ○ Bachelor
- ○ Master
- ○ PhD

How often do you reverse malware?

- ○ Monthly
- ○ Weekly
- ○ Daily
- ○ Less

For how long are you reversing malware?

- ○ >5 years
- ○ 3 years
- ○ 2 years
- ○ <1 years

How much experience do you have in .. ?

	1	2	3	4	5	6	7	8	9	10
Reversing	○	○	○	○	○	○	○	○	○	○
Malware Analysis	○	○	○	○	○	○	○	○	○	○
Writing C-code	○	○	○	○	○	○	○	○	○	○

Which binary code decompiler have you used before? (multiple answers are possible)

- ☐ Hex-Rays
- ☐ Ghidra
- ☐ BinaryNinja HLIL
- ☐ Boomerang
- ☐ REC
- ☐ DISC
- ☐ Other

1.2 Part A

1.2.1 Part A Our decompiler

Please consider the following decompiled function and answer the questions below. Feel free to use the editor as you would normally do (e.g. by renaming variables).

```
unsigned long A(int arg1, int arg2) {
    unsigned long var_0;

    if (arg1 == 0) {
        var_0 = arg2;
    }
    else {
        var_0 = arg2;
        while(true) {
            arg2 = (arg1 + -0x1) & 0xffffffff;
            if ((unsigned int)var_0 == 0) {
                if (arg2 == 0) {
                    var_0 = 0x1;
                    break;
                }
                arg1 = arg2;
                var_0 = 0x1;
                continue;
            }
            var_0 = A(arg1, ((unsigned int) var_0) + 0xffffffff);
            if (arg2 == 0) {
                break;
            }
            arg1 = arg2;
        }
    }
    return ((unsigned int) var_0) + 0x1;
}
```

	strongly disagree	disagree	weakly disagree	unsure	weakly agree	agree	strongly agree
The used control-flow structures are appropriate.	○	○	○	○	○	○	○
The control-flow is strangely restructured.	○	○	○	○	○	○	○
It was easy to understand the code.	○	○	○	○	○	○	○
It took much effort to understand the code.	○	○	○	○	○	○	○
It seems that there are no unused instructions.	○	○	○	○	○	○	○
There are too much unused instructions.	○	○	○	○	○	○	○
I think the code is correctly decompiled.	○	○	○	○	○	○	○
The decompiled code seems to be incorrect.	○	○	○	○	○	○	○
The conditions are too complex.	○	○	○	○	○	○	○
The code contains too many intermediate results.	○	○	○	○	○	○	○
The line length is too long	○	○	○	○	○	○	○
Many variables have useless copies.	○	○	○	○	○	○	○
There are no useless copies of variables.	○	○	○	○	○	○	○
The variable types seem to be reasonable.	○	○	○	○	○	○	○
Some variable types seem to be wrong.	○	○	○	○	○	○	○
There are no variables that only store unnecessary intermediate constants.	○	○	○	○	○	○	○
There are too many variables that just store intermediate constants.	○	○	○	○	○	○	○

What is the output of the function for the parameters

	0	1	2	3	4	5	I do not know
arg_1 = 0 and arg_2 = 1	○	○	○	○	○	○	○
arg_1 = 1 and arg_2 = 0	○	○	○	○	○	○	○

Which computation needs more recursive calls?

○ arg_1 = 1 and arg_2 = 2

○ arg_1 = 2 and arg_2 = 1

What do you like or dislike about the above sample?

1.2.2 Part A IDA

Please consider the following decompiled function and answer the questions below. Feel free to use the editor as you would normally do (e.g. by renaming variables).

```
__int64 __fastcall A(__int64 arg_1, int arg_2){
    unsigned int var_0;

    if ( (_DWORD)arg_1 ) {
        do{
            while ( 1 ) {
                var_0 = arg_1 - 1;
                if ( !arg_2 ) {
                    break;
                }
                arg_2 = A(arg_1, (unsigned int)(arg_2 - 1));
                arg_1 = var_0;
                if ( !var_0 ) {
                    return (unsigned int)(arg_2 + 1);
                }
            }
            arg_2 = 1;
            arg_1 = var_0;
        }while ( var_0 );
    }
    return (unsigned int)(arg_2 + 1);
}
```

	strongly disagree	disagree	weakly disagree	undecided	weakly agree	agree	strongly agree
The used control-flow structures are appropriate.	○	○	○	○	○	○	○
The control-flow is strangely restructured.	○	○	○	○	○	○	○
It was easy to understand the code.	○	○	○	○	○	○	○
It took much effort to understand the code.	○	○	○	○	○	○	○
It seems that there are no unused instructions.	○	○	○	○	○	○	○
There are too much unused instructions.	○	○	○	○	○	○	○
I think the code is correctly decompiled.	○	○	○	○	○	○	○
The decompiled code seems to be incorrect.	○	○	○	○	○	○	○
The conditions are too complex.	○	○	○	○	○	○	○
The code contains too many intermediate results.	○	○	○	○	○	○	○
The line length is too long	○	○	○	○	○	○	○
Many variables have useless copies.	○	○	○	○	○	○	○
There are no useless copies of variables.	○	○	○	○	○	○	○
The variable types seem to be reasonable.	○	○	○	○	○	○	○
Some variable types seem to be wrong.	○	○	○	○	○	○	○
There are no variables that only store unnecessary intermediate constants.	○	○	○	○	○	○	○
There are too many variables that just store intermediate constants.	○	○	○	○	○	○	○

What is the output of the function for the parameters

	0	1	2	3	4	5	I do not know
arg_1 = 0 and arg_2 = 1	○	○	○	○	○	○	○
arg_1 = 1 and arg_2 = 0	○	○	○	○	○	○	○

Which computation needs more recursive calls?

○ arg_1 = 1 and arg_2 = 2

○ arg_1 = 2 and arg_2 = 1

What do you like or dislike about the above sample?

1.2.3 Part A Ghidra

Please consider the following decompiled function and answer the questions below. Feel free to use the editor as you would normally do (e.g. by renaming variables).

```
ulong A(ulong arg_1,ulong arg_2){
    int var_0;
    uint var_1;

    arg_2 = arg_2 & 0xffffffff;
    var_0 = (int)arg_2;
    var_1 = (uint)arg_1;
    while (var_1 != 0) {
        while( true ) {
            var_1 = (int)arg_1 - 1;
            if ((int)arg_2 != 0) {
                break;
            }
            arg_2 = 1;
            var_0 = 1;
            arg_1 = (ulong)var_1;
            if (var_1 == 0) {
                goto Label_1;
            }
        }
        arg_2 = A(arg_1,(ulong)((int)arg_2 - 1));
        var_0 = (int)arg_2;
        arg_1 = (ulong)var_1;
    }
Label_1:
    return (ulong)(var_0 + 1);
}
```

	strongly disagree	disagree	weakly disagree	undecided	weakly agree	agree	strongly agree
The used control-flow structures are appropriate.	○	○	○	○	○	○	○
The control-flow is strangely restructured.	○	○	○	○	○	○	○
It was easy to understand the code.	○	○	○	○	○	○	○
It took much effort to understand the code.	○	○	○	○	○	○	○
It seems that there are no unused instructions.	○	○	○	○	○	○	○
There are too much unused instructions.	○	○	○	○	○	○	○
I think the code is correctly decompiled.	○	○	○	○	○	○	○
The decompiled code seems to be incorrect.	○	○	○	○	○	○	○
The conditions are too complex.	○	○	○	○	○	○	○
The code contains too many intermediate results.	○	○	○	○	○	○	○
The line length is too long	○	○	○	○	○	○	○
Many variables have useless copies.	○	○	○	○	○	○	○
There are no useless copies of variables.	○	○	○	○	○	○	○
The variable types seem to be reasonable.	○	○	○	○	○	○	○
Some variable types seem to be wrong.	○	○	○	○	○	○	○
There are no variables that only store unnecessary intermediate constants.	○	○	○	○	○	○	○
There are too many variables that just store intermediate constants.	○	○	○	○	○	○	○

What is the output of the function for the parameters

	0	1	2	3	4	5	I do not know
arg_1 = 0 and arg_2 = 1	○	○	○	○	○	○	○
arg_1 = 1 and arg_2 = 0	○	○	○	○	○	○	○

Which computation needs more recursive calls?

- ○ arg_1 = 1 and arg_2 = 2
- ○ arg_1 = 2 and arg_2 = 1

What do you like or dislike about the above sample?

1.3 Part B

1.3.1 Part B - Q1

Please consider the following three samples and order them from your favourite to least favourite:

Sample 1 (our):

```c
int test1() {
  int var_0;

  var_0 = rand();
  rand();
  switch(var_0) {
  case 1:
    var_0 = 0;
    break;
  case 5:
    var_0 = var_0 + 1;
  case 10:
    var_0 = var_0 << 1;
    break;
  }
  return var_0;
}
```

Sample 2 (Ghidra):

```c
int test1(void) {
  int var_0;

  var_0 = rand();
  rand();
  if (var_0 != 10) {
    if (10 < var_0) {
      return var_0;
    }
    if (var_0 == 1) {
      return 0;
    }
    if (var_0 != 5) {
      return var_0;
    }
    var_0 = 6;
  }
  return var_0 << 1;
}
```

Sample 3 (IDA):

```
int test1() {
  int var_0;

  var_0 = rand();
  rand();
  if ( var_0 == 10 ) {
    goto Label_1;
  }
  if ( var_0 <= 10 ) {
    if ( var_0 == 1 ) {
      return 0;
    }
    if ( var_0 == 5 ) {
      var_0 = 6;
Label_1:
      var_0 *= 2;
      return var_0;
    }
  }
  return var_0;
}
```

Sample 1	
Sample 2	
Sample 3	

Do you prefer switch or if-else when there are at most 2 or 3 cases?

○ switch

○ if-else

If the value of a variable is known due to a condition, i.e., Branch or Switch, should it be propagated?

○ yes, the value should be propagated

○ no, the value should not be propagated

What did you like most about your favourite sample?

What did you dislike about your least favourite sample?

1.3.2 Part B - Q2

Please consider the following three samples and order them from your favourite to least favourite:

Sample 1 (Ghidra):

```c
undefined4 test2(void) {
  int var_0 [3];

  printf("Enter any number: ");
  __isoc99_scanf(0x0804a01f,var_0);
  if (var_0[0] < 1) {
    if (var_0[0] < 0) {
      if (var_0[0] < 0) {
        printf("%d is negative.",var_0[0]);
      }
    }
    else {
      printf("%d is zero.",var_0[0]);
    }
  }
  else {
    if (var_0[0] >= 1) {
      printf("%d is positive.",var_0[0]);
    }
  }
  return 0;
}
```

Sample 2 (our):

```c
int test2() {
  int * var_0;
  int * var_1;

  printf("Enter any number: ");
  var_1 = &(var_0);
  __isoc99_scanf(0x804a01f, var_1);
  switch((unsigned int) (byte) (var_0 & 0xffffff00) || (var_0 > 0)) {
    case 0:
      switch((unsigned int) (unsigned byte) var_0 u>> 31) {
        case 0:
```

```
12          printf("%d is zero.", var_0);
13          break;
14        case 1:
15          printf("%d is negative.", var_0);
16          break;
17        }
18        break;
19      case 1:
20        printf("%d is positive.", var_0);
21        break;
22    }
23    return 0;
24  }
```

Sample 3 (IDA):

```
1  int test2() {
2    int var_0[3];
3  
4    printf("Enter any number: ");
5    __isoc99_scanf("%d", var_0);
6    if ( var_0[0] <= 0 ) {
7      if ( var_0[0] >= 0 ) {
8        printf("%d is zero.", var_0[0]);
9      }
10     else if ( var_0[0] < 0 ) {
11       printf("%d is negative.", var_0[0]);
12     }
13   }
14   else if ( var_0[0] > 0 ) {
15     printf("%d is positive.", var_0[0]);
16   }
17   return 0;
18 }
```

Sample 1	
Sample 2	
Sample 3	

Do you prefer switch or if-else in a switch-case?

○ switch

○ if-else

○ if-else if switch has at most 2 cases and switch otherwise

Do you prefer "else if" as one expression, if possible, (as in sample 3 lines 10 and 14) or separated into two expressions (as in sample 1 lines 16/17)?

○ one expression

○ separated expressions

What did you like most about your favourite sample?

What did you dislike about your least favourite sample?

1.3.3 Part B - Q3

Please consider the following three samples and order them from your favourite to least favourite:

Sample 1 (IDA):

```c
int test3() {
  int var_0;
  int var_1;
  int i;

  printf("Please enter a number:");
  __isoc99_scanf("%d", &var_0);
  while ( var_0 <= 99 ) {
    var_1 = rand();
    for ( i = 0; i < var_1; ++i ) {
      var_0 += i;
      if ( var_0 > 100 ) {
        return printf("The result is %d\\n", var_0);
      }
    }
  }
  return printf("The result is %d\\n", var_0);
}
```

Sample 2 (Ghidra):

```c
void test3(void) {
  int var_0;
  int var_2;
  int var_1;

  printf("Please enter a number:");
  __isoc99_scanf(0x0804a01f,&var_0);
  while (var_0 < 100) {
    var_2 = rand();
    var_1 = 0;
    while (var_1 < var_2) {
      var_0 = var_1 + var_0;
      if (100 < var_0) {
        goto Label_1;
      }
      var_1 = var_1 + 1;
    }
  }
Label_1:
  printf("The result is %d\\n",var_0);
  return;
}
```

Sample 3 (our):

```c
int test3() {
    int exit_4;
    int var_0;
    int var_1;
    int * var_2;
    int var_3;
    int var_4;

    printf("Please enter a number:");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a01f, var_2);
    while(var_0 <= 99) {
        var_4 = rand();
        var_3 = 0;
        while(true) {
            if (var_3 >= var_4) {
                exit_4 = 0;
                break;
            }
            var_1 = var_3 + var_0;
            if (var_3 + var_0 > 100) {
                var_0 = var_1;
                exit_4 = 1;
                break;
            }
            var_3 = var_3 + 1;
            var_0 = var_1;
        }
        if (exit_4 != 0) {
            break;
        }
    }
    var_3 = printf("The result is %d\\n", var_0);
    return var_3;
}
```

Sample 1	
Sample 2	
Sample 3	

Do you think goto is a good choice when breaking out of the inner loop?

○ yes, goto is a good choice

○ no, goto should not be used

Do you think it is a good idea to duplicate (return-)statements to avoid complicated conditions or gotos?

- ○ copy statements to simplify the structure
- ○ never copy a statement

What did you like most about your favourite sample?

What did you dislike about your least favourite sample?

1.3.4 Part B - Q4

Please consider the following three samples and order them from your favourite to least favourite:

Sample 1 (our):

```
int test4() {
  int * i;
  int var_0;
  int var_1;
  int var_2;
  int * var_3;

  var_2 = rand();
  printf("Enter an number: ");
  var_3 = &(var_0);
  __isoc99_scanf(0x804a01f, var_3);
  var_3 = 0;
  for (i = 0; i < var_0; i = i + 1) {
    while(var_2 + i > var_0) {
      if (var_2 - i > var_0) {
        var_0 = i + var_0;
        continue;
      }
      var_1 = i + var_0;
      if (var_2 == i + var_0) {
        var_0 = var_2;
        break;
      }
      var_0 = var_1;
    }
    if (var_2 == var_0) {
      var_3 = i;
    }
    if (var_3 > 0) {
      printf("The random number is %d\\n", var_3);
      break;
    }
    var_0 = var_0 - 1;
  }
  return 0;
}
```

Sample 2 (IDA):

```
int test4() {
  int var_0;
  int var_1;
  int var_2;
  int var_3;

  var_1 = rand();
  printf("Enter an number: ");
  __isoc99_scanf("%d", &var_0);
  var_3 = 0;
  var_2 = 0;
  while ( var_3 < var_0 ) {
    while ( var_3 + var_1 > var_0 ) {
      if ( var_1 - var_3 <= var_0 ) {
        var_0 += var_3;
        if ( var_1 == var_0 ) {
          var_0 = var_1;
          break;
        }
      }
      else {
        var_0 += var_3;
      }
    }
    if ( var_1 == var_0 ) {
      var_2 = var_3;
    }
    if ( var_2 > 0 ) {
      printf("The random number is %d\\n", var_2);
      return 0;
    }
    --var_0;
    ++var_3;
  }
  return 0;
}
```

Sample 3 (Ghidra):

```
undefined4 test4(void) {
  int var_0;
  int var_1;
  int var_2;
  int var_3;

  var_1 = rand();
  printf("Enter an number: ");
  __isoc99_scanf(0x0804a01f,&var_0);
  var_3 = 0;
  var_2 = 0;
  do {
```

```
  if (var_0 <= var_3) {
   return 0;
  }
  do {
   while( true ) {
    if (var_1 + var_3 <= var_0) {
     goto Label_1;
    }
    if (var_1 - var_3 <= var_0) {
     break;
    }
    var_0 = var_3 + var_0;
   }
   var_0 = var_3 + var_0;
  } while (var_1 != var_0);
  var_0 = var_1;
Label_1:
  if (var_1 == var_0) {
   var_2 = var_3;
  }
  if (0 < var_2) {
   printf("The random number is %d\\n",var_2);
   return 0;
  }
  var_0 = var_0 + -1;
  var_3 = var_3 + 1;
 } while( true );
}
```

Sample 1	
Sample 2	
Sample 3	

Do you prefer while(true) or dowhile(true)?

◯ while

◯ do-while

Do you prefer continue-statements and only one if-statement or no continue-statement and an if-else-statement?

◯ continue and if

◯ if-else

What did you like most about your favourite sample?

What did you dislike about your least favourite sample?

1.3.5 Part B - Q5

Please consider the following three samples and order them from your favourite to least favourite:

Sample 1 (IDA):

```c
int test5() {
  int var_0;
  int var_1[2];
  int var_2;
  int j;
  int i;

  printf("Enter a number: ");
  __isoc99_scanf("%d", var_1);
  printf("Enter a second number: ");
  __isoc99_scanf("%d", &var_0);
  printf("You chosed the numbers %d and %d\\n", var_1[0], var_0);
  var_2 = 0;
  var_1[1] = 0;
  for ( i = 0; i < var_1[0]; ++i ) {
    for ( j = 0; j < var_0; ++j ) {
      if ( var_1[0] - i == var_0 - j ) {
        var_2 = 1;
      }
    }
    if ( !var_2 ) {
      if ( var_1[0] >= var_0 ) {
        var_0 = 2 * var_0 - var_1[0];
      }
      else{
        var_0 -= var_1[0];
      }
    }
    var_2 = 0;
  }
  return printf("The numbers are %d and %d\\n", var_1[0], var_0);
}
```

Sample 2 (our):

```c
int test5() {
  int i;
  int j;
  int var_0;
  int var_1;
  int * var_2;
  int var_3;

  printf("Enter a number: ");
  var_2 = &(var_1);
  __isoc99_scanf(0x804a01f, var_2);
  printf("Enter a second number: ");
  var_2 = &(var_0);
  __isoc99_scanf(0x804a01f, var_2);
  printf("You chosed the numbers %d and %d\\n", var_1, var_0);
  for (i = 0; i < var_1; i = i + 1) {
    var_3 = 0;
    for (j = 0; j < var_0; j = j + 1) {
      if (var_1 - i == var_0 - j) {
        var_3 = 1;
      }
    }
    if (var_3 == 0) {
      if (var_1 < var_0) {
        var_0 = var_0 - var_1;
      }
      else {
        var_0 = (var_0 + var_0) - var_1;
      }
    }
  }
  var_3 = printf("The numbers are %d and %d\\n", var_1, var_0);
  return var_3;
}
```

Sample 3 (Ghidra):

```
void test5(void) {
  int var_1;
  int var_0;
  undefined4 var_2;
  int var_3;
  int var_4;
  int var_5;

  printf("Enter a number: ");
  __isoc99_scanf(0x0804a01f,&var_0);
  printf("Enter a second number: ");
  __isoc99_scanf(0x0804a01f,&var_1);
  printf("You chosed the numbers %d and %d\\n",var_0,var_1);
  var_2 = 0;
  var_5 = 0;
  while (var_3 = 0, var_5 < var_0) {
    var_4 = 0;
    while (var_4 < var_1) {
      if (var_0 - var_5 == var_1 - var_4) {
        var_3 = 1;
      }
      var_4 = var_4 + 1;
    }
    if (var_3 == 0) {
      if (var_0 < var_1) {
        var_1 = var_1 - var_0;
      }
      else {
        var_1 = var_1 * 2 - var_0;
      }
    }
    var_5 = var_5 + 1;
  }
  printf("The numbers are %d and %d\\n",var_0,var_1);
  return;
}
```

Sample 1	
Sample 2	
Sample 3	

Do you prefer the condition if(var == 0) or if(!var)?

- ○ if(var == 0)
- ○ if(!var)

What did you like most about your favourite sample?

What did you dislike about your least favourite sample?

1.3.6 Part B - Q6

Please consider the following three samples and order them from your favourite to least favourite:

Sample 1 (Ghidra):

```
undefined4 test6(void) {
  int var_4;
  int var_0;
  int var_1;
  int var_2;
  int var_3;

  printf("Find prime numbers between 1 to : ");
  __isoc99_scanf(0x0804a01f,&var_0);
  printf("All prime numbers between 1 to %d are:\\n",var_0);
  var_3 = 2;
  do {
    if (var_0 < var_3) {
      return 0;
    }
    var_1 = 1;
    var_2 = 2;
    while (var_2 <= var_3) {
      var_4 = dividable(var_3,var_2);
      if (var_4 != 0) {
        var_1 = 0;
        break;
      }
      var_2 = var_2 + 1;
    }
    if (var_1 == 1) {
      printf("%d, ",var_3);
    }
    var_3 = var_3 + 1;
  } while( true );
}
```

Sample 2 (IDA):

```
int test6() {
  int var_0;
  int var_1;
  int j;
  int i;

  printf("Find prime numbers between 1 to : ");
  __isoc99_scanf("%d", &var_0);
  printf("All prime numbers between 1 to %d are:\\n", var_0);
  for ( i = 2; i <= var_0; ++i ) {
    var_1 = 1;
    for ( j = 2; j <= i; ++j ) {
      if ( dividable(i, j) ) {
        var_1 = 0;
        break;
      }
    }
    if ( var_1 == 1 ) {
      printf("%d, ", i);
    }
  }
  return 0;
}
```

Sample 3 (our):

```
int test6() {
  int i;
  int var_0;
  int * var_1;
  int var_3;
  unsigned int var_4;

  printf("Find prime numbers between 1 to : ");
  var_1 = &(var_0);
  __isoc99_scanf(0x804a01f, var_1);
  printf("All prime numbers between 1 to %d are:\\n", var_0);
  for (i = 2; i <= var_0; i = i + 1) {
    var_3 = 2;
    while(true) {
      if (var_3 > i) {
        var_3 = 1;
        break;
      }
      var_4 = dividable(i, var_3);
      if (var_4 != 0) {
        var_3 = 0;
        break;
      }
      var_3 = var_3 + 1;
    }
    if (var_3 == 1) {
      printf("%d, ", i);
    }
  }
  return 0;
}
```

Sample 1	
Sample 2	
Sample 3	

Do you prefer for-loops or while-loops?

○ for-loops

○ while-loops

What did you like most about your favourite sample?

What did you dislike about your least favourite sample?

1.4 Part C

1.4.1 Part C Q1: expression propagation limits

We have decompiled the same function with different expression propagation limits. Please choose your favourite version with respect to the most optimal nesting complexity of instructions. .tab

Sample 1:

```
int palindrom() {
    int var_0;
    int var_1;
    int var_2;
    int var_3;
    int var_4;
    int var_5;
    int * var_6;
    long var_7;
    printf("Enter an integer: ");
    var_6 = &(var_0);
    __isoc99_scanf(0x804a028, var_6);
    var_1 = var_0;
    var_2 = 0x0;
    while(true) {
        var_3 = var_0;
        if (var_3 == 0) {
            break;
        }
        var_3 = var_0;
        var_7 = var_3 * 0x66666667;
        var_4 = var_7 >> 0x20;
        var_4 = var_4 >> 0x2;
        var_5 = var_3 >> 0x1f;
        var_5 = var_4 - var_5;
        var_4 = var_5 << 0x2;
        var_4 = var_4 + var_5;
        var_4 = var_4 + var_4;
        var_3 = var_3 - var_4;
        var_4 = var_2 << 0x2;
        var_2 = var_4 + var_2;
        var_2 = var_2 + var_2;
        var_3 = var_3 + var_2;
        var_4 = var_0;
        var_7 = var_4 * 0x66666667;
        var_2 = var_7 >> 0x20;
        var_2 = var_2 >> 0x2;
        var_4 = var_4 >> 0x1f;
        var_2 = var_2 - var_4;
        var_0 = var_2;
        var_2 = var_3;
    }
    if (var_1 == var_2) {
        printf("%d is a palindrome.", var_1);
    }
    else {
        printf("%d is not a palindrome.", var_1);
    }
    return 0x0;
}
```

Sample 2:

```
int palindrom() {
    int i;
    int var_1;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    int var_6;
    long var_7;
    printf("Enter an integer: ");
    var_2 = &(var_1);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_1; i != 0; i = var_4 - var_5) {
        var_7 = i * 0x66666667;
        var_4 = var_7 >> 0x20;
        var_4 = var_4 >> 0x2;
        var_6 = i >> 0x1f;
        var_5 = var_4 - var_6;
        var_4 = var_4 - var_6;
        var_4 = var_4 << 0x2;
        var_4 = var_4 + var_5;
        var_4 = var_4 + var_4;
        var_5 = var_3 << 0x2;
        var_3 = var_5 + var_3;
        var_3 = var_3 + var_3;
        var_4 = i - var_4;
        var_3 = var_4 + var_3;
        var_7 = i * 0x66666667;
        var_4 = var_7 >> 0x20;
        var_4 = var_4 >> 0x2;
        var_5 = i >> 0x1f;
    }
    if (var_1 != var_3) {
        printf("%d is not a palindrome.", var_1);
    }
    else {
        printf("%d is a palindrome.", var_1);
    }
    return 0x0;
}
```

Sample 3:

```
int palindrom() {
    int i;
    int var_1;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_1);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_1; i != 0; i = var_4 - (i >> 0x1f)) {
        var_5 = ((i * 0x66666667) >> 0x20) >> 0x2;
        var_4 = (var_5 - (i >> 0x1f)) << 0x2;
        var_4 = var_4 + (var_5 - (i >> 0x1f));
        var_3 = (var_3 << 0x2) + var_3;
        var_3 = var_3 + var_3;
        var_3 = (i - (var_4 + var_4)) + var_3;
        var_4 = ((i * 0x66666667) >> 0x20) >> 0x2;
    }
    if (var_1 == var_3) {
        printf("%d is a palindrome.", var_1);
    }
    else {
        printf("%d is not a palindrome.", var_1);
    }
    return 0x0;
}
```

Sample 4:

```c
int palindrom() {
    int i;
    int var_1;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_1);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_1; i != 0; i = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) {
        var_5 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_4 = ((var_4 << 0x2) + var_5) + ((var_4 << 0x2) + var_5);
        var_3 = ((var_3 << 0x2) + var_3) + ((var_3 << 0x2) + var_3);
        var_3 = (i - var_4) + var_3;
    }
    if (var_1 == var_3) {
        printf("%d is a palindrome.", var_1);
    }
    else {
        printf("%d is not a palindrome.", var_1);
    }
    return 0x0;
}
```

Sample 5:

```c
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_4 = (((((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) << 0x2) + var_4;
        var_3 = (i - (var_4 + var_4)) + (((var_3 << 0x2) + var_3) + ((var_3 << 0x2) +
        ↪  var_3));
    }
    if (var_0 != var_3) {
        printf("%d is not a palindrome.", var_0);
    }
    else {
        printf("%d is a palindrome.", var_0);
    }
    return 0x0;
}
```

○ limit 1 (Sample 1)

- ○ limit 3 (Sample 2)
- ○ limit 5 (Sample 3)
- ○ limit 7 (Sample 4)
- ○ limit 10 (Sample 5)

1.4.2 Part C Q2: common subexpression elimination

Please take a look at the following function and evaluate it with respect to long subexpressions that occur multiple times. Choose the most appropriate one.

Sample 1:

```c
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = var_4) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_5 = (var_3 << 0x2) + var_3;
        var_3 = (var_4 << 0x2) + var_4;
        var_3 = (i - (var_3 + var_3)) + (var_5 + var_5);
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

Sample 2:

```c
int palindrom() {
    int i;
    int var_1;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_1);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_1; i != 0; i = var_4) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_5 = (var_4 << 0x2) + var_4;
        var_3 = (i - (var_5 + var_5)) + (((var_3 << 0x2) + var_3) + ((var_3 << 0x2) +
         ↪  var_3));
    }
    if (var_1 == var_3) {
        printf("%d is a palindrome.", var_1);
    }
    else {
        printf("%d is not a palindrome.", var_1);
    }
    return 0x0;
}
```

Sample 3:

```
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_4 = (((((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) << 0x2) + var_4;
        var_3 = (i - (var_4 + var_4)) + (((var_3 << 0x2) + var_3) + ((var_3 << 0x2) +
         ↪  var_3));
    }
    if (var_0 != var_3) {
        printf("%d is not a palindrome.", var_0);
    }
    else {
        printf("%d is a palindrome.", var_0);
    }
    return 0x0;
}
```

○ Sample 1

○ Sample 2

○ Sample 3

1.4.3 Part C Q3: mix of config options

We have decompiled the same function with different configurations. Please choose up to three samples that are most comprehensible to you. .tab

Sample 1:

```c
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    int var_6;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = (var_4 >> 0x2) - var_5) {
        var_4 = (i * 0x66666667) >> 0x20;
        var_5 = i >> 0x1f;
        var_6 = (var_4 >> 0x2) - var_5;
        var_6 = (var_6 << 0x2) + var_6;
        var_3 = (var_3 << 0x2) + var_3;
        var_6 = i - (var_6 + var_6);
        var_3 = var_6 + (var_3 + var_3);
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

Sample 2:

```c
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    int var_6;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = var_4 - var_5) {
        var_5 = i >> 0x1f;
        var_4 = ((i * 0x66666667) >> 0x20) >> 0x2;
        var_6 = var_4 - var_5;
        var_6 = (var_6 << 0x2) + var_6;
        var_3 = (var_3 << 0x2) + var_3;
        var_3 = (i - (var_6 + var_6)) + (var_3 + var_3);
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

Sample 3:

```
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = var_4) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_5 = (var_4 << 0x2) + var_4;
        var_3 = (var_3 << 0x2) + var_3;
        var_5 = var_5 + var_5;
        var_3 = var_3 + var_3;
        var_3 = (i - var_5) + var_3;
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

Sample 4:

```
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = var_4) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_5 = ((var_4 << 0x2) + var_4) + ((var_4 << 0x2) + var_4);
        var_3 = ((var_3 << 0x2) + var_3) + ((var_3 << 0x2) + var_3);
        var_3 = (i - var_5) + var_3;
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

Sample 5:

```
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    int var_5;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = var_4) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_5 = (var_4 << 0x2) + var_4;
        var_3 = ((var_3 << 0x2) + var_3) + ((var_3 << 0x2) + var_3);
        var_3 = (i - (var_5 + var_5)) + var_3;
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

Sample 6:

```
int palindrom() {
    int i;
    int var_0;
    int * var_2;
    int var_3;
    int var_4;
    printf("Enter an integer: ");
    var_2 = &(var_0);
    __isoc99_scanf(0x804a028, var_2);
    var_3 = 0x0;
    for (i = var_0; i != 0; i = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) {
        var_4 = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f);
        var_4 = (((((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) << 0x2) + var_4;
        var_3 = ((var_3 << 0x2) + var_3) + ((var_3 << 0x2) + var_3);
        var_3 = (i - (var_4 + var_4)) + var_3;
    }
    if (var_0 == var_3) {
        printf("%d is a palindrome.", var_0);
    }
    else {
        printf("%d is not a palindrome.", var_0);
    }
    return 0x0;
}
```

- ☐ Sample 1
- ☐ Sample 2
- ☐ Sample 3
- ☐ Sample 4
- ☐ Sample 5
- ☐ Sample 6

Please enter your comment here:

1.4.4 Part C Q4: Complexity of for-loop conditions

Please take a look at the following function that was decompiled either using a for-loop or using a while-loop. Choose the output that you prefer.

Sample 1:

```
int first_last_digit() {
    int i;
    int var_0;
    int var_1;
    int var_2;
    int * var_3;
    __x86.get_pc_thunk.bx();
    printf("Enter any number to find sum of first and last digit: ");
    var_3 = &(var_0);
    __isoc99_scanf(0x1693, var_3);
    var_2 = (((var_0 * 0x66666667) >> 0x20) >> 0x2) - (var_0 >> 0x1f);
   var_2 = (((((var_0 * 0x66666667) >> 0x20) >> 0x2) - (var_0 >> 0x1f)) << 0x2) + var_2;
    for (i = var_0; i > 9; i = (((i * 0x66666667) >> 0x20) >> 0x2) - (i >> 0x1f)) {}
    ↪  var_1 = i;
    printf("Sum of first and last digit = %d", (var_0 - (var_2 + var_2)) + var_1);
    return 0x0;
 }
```

Sample 2:

```
int fisrt_last_digit() {
    int var_0;
    int var_1;
    int var_2;
    int * var_3;
    __x86.get_pc_thunk.bx();
    printf("Enter any number to find sum of first and last digit: ");
    var_3 = &(var_0);
    __isoc99_scanf(0x1693, var_3);
    var_2 = (((var_0 * 0x66666667) >> 0x20) >> 0x2) - (var_0 >> 0x1f);
    var_2 = (((((var_0 * 0x66666667) >> 0x20) >> 0x2) - (var_0 >> 0x1f)) << 0x2) + var_2;
    var_1 = var_0;
    while(var_1 > 9) {
        var_1 = (((var_1 * 0x66666667) >> 0x20) >> 0x2) - (var_1 >> 0x1f);
    }
    printf("Sum of first and last digit = %d", (var_0 - (var_2 + var_2)) + var_1);
    return 0x0;
}
```

○ for-loop (Sample 1)

○ while-loop (Sample 2)

1.4.5 Part C Q5: byte constants representation

The output contains various of byte constant (from printable ASCII range) representation in the decompiler output. Rank them from most to least favourite.

Sample 1:

```
int is_consonant_or_vowel() {
    byte var_0;
    byte * var_1;
    __x86.get_pc_thunk.bx();
    printf("Enter any character: ");
    var_1 = &(var_0);
    __isoc99_scanf(0xe8b, var_1);
    if ((var_0 != 'a') && (var_0 != 'e') && (var_0 != 'i') && (var_0 != 'o') && (var_0
    ↪ != 'u') && (var_0 != 'A') && (var_0 != 'E') && (var_0 != 'I') && (var_0 != 'O')
    ↪ && (var_0 != 'U')) {
        if (((var_0 <= 'z') && (var_0 > '`')) || ((var_0 <= 'Z') && (var_0 > '@'))) {
            printf("\'%c\' is Consonant.", (int) var_0);
        }
        else {
            printf("\'%c\' is not an alphabet.", (int) var_0);
        }
    }
    else {
        printf("\'%c\' is Vowel.", (int) var_0);
    }
    return 0x0;
}
```

Sample 2:

```
int is_consonant_or_vowel() {
    byte var_0;
    byte * var_1;
    __x86.get_pc_thunk.bx();
    printf("Enter any character: ");
    var_1 = &(var_0);
    __isoc99_scanf(0xe8b, var_1);
    if ((var_0 != 'a'/*97*/) && (var_0 != 'e'/*101*/) && (var_0 != 'i'/*105*/) && (var_0
    ↪  != 'o'/*111*/) && (var_0 != 'u'/*117*/) && (var_0 != 'A'/*65*/) && (var_0 !=
    ↪  'E'/*69*/) && (var_0 != 'I'/*73*/) && (var_0 != 'O'/*79*/) && (var_0 !=
    ↪  'U'/*85*/)) {
        if (((var_0 <= 'z'/*122*/) && (var_0 > '`'/*96*/)) || ((var_0 <= 'Z'/*90*/) &&
        ↪  (var_0 > '@'/*64*/))) {
            printf("\'%c\' is Consonant.", (int) var_0);
        }
        else {
            printf("\'%c\' is not an alphabet.", (int) var_0);
        }
    }
    else {
        printf("\'%c\' is Vowel.", (int) var_0);
    }
    return 0x0;
}
```

Sample 3:

```
int is_consonant_or_vowel() {
    byte var_0;
    byte * var_1;
    __x86.get_pc_thunk.bx();
    printf("Enter any character: ");
    var_1 = &(var_0);
    __isoc99_scanf(0xe8b, var_1);
    if ((var_0 != 97) && (var_0 != 101) && (var_0 != 105) && (var_0 != 111) && (var_0 !=
    ↪  117) && (var_0 != 65) && (var_0 != 69) && (var_0 != 73) && (var_0 != 79) &&
    ↪  (var_0 != 85)) {
        if (((var_0 <= 122) && (var_0 > 96)) || ((var_0 <= 90) && (var_0 > 64))) {
            printf("\'%c\' is Consonant.", (int) var_0);
        }
        else {
            printf("\'%c\' is not an alphabet.", (int) var_0);
        }
    }
    else {
        printf("\'%c\' is Vowel.", (int) var_0);
    }
    return 0x0;
}
```

Sample 4:

```
int is_consonant_or_vowel() {
    byte var_0;
    byte * var_1;
    __x86.get_pc_thunk.bx();
    printf("Enter any character: ");
    var_1 = &(var_0);
```

```c
    __isoc99_scanf(0xe8b, var_1);
    if ((var_0 != 97/*'a'*/) && (var_0 != 101/*'e'*/) && (var_0 != 105/*'i'*/) && (var_0
    ↪  != 111/*'o'*/) && (var_0 != 117/*'u'*/) && (var_0 != 65/*'A'*/) && (var_0 !=
    ↪  69/*'E'*/) && (var_0 != 73/*'I'*/) && (var_0 != 79/*'O'*/) && (var_0 !=
    ↪  85/*'U'*/)) {
        if (((var_0 <= 122/*'z'*/) && (var_0 > 96/*'`'*/)) || ((var_0 <= 90/*'Z'*/) &&
        ↪  (var_0 > 64/*'@'*/))) {
            printf("\'%c\' is Consonant.", (int) var_0);
        }
        else {
            printf("\'%c\' is not an alphabet.", (int) var_0);
        }
    }
    else {
        printf("\'%c\' is Vowel.", (int) var_0);
    }
    return 0x0;
}
```

char (Sample 1)	
char + decimal as comment (Sample 2)	
decimal (Sample 3)	
decimal + char as comment (Sample 4)	

Please enter your comment here:

1.4.6 Part C Q6

Please consider the following variable naming schemes and rate them from favourite to least favourite:

```c
// naming scheme 1
int var_0, var_1, var_2, var_3, var_4;

// naming scheme 2
int v1, v2, v3, v4, v5;

// naming scheme 3
int dog, cat, wolf, bear;

// naming scheme 4
int a, b, c, d, e, f;

// naming scheme 5
int loc_1, loc_2, loc_3;
```

Scheme 1 (var_0,...) (Sample 1)	
Scheme 2 (v1,...) (Sample 2)	
Scheme 3 (dog,...) (Sample 3)	
Scheme 4 (a,...) (Sample 4)	
Scheme 5 (loc_1,...) (Sample 5)	

Do you have any better ideas for variable naming schemes?

1.4.7 Part C Q7

Please consider the following parameter naming schemes and rate them from favourite to least favourite:

```
// naming scheme 1
int param_1, param_2, param_3;

// naming scheme 2
int a1, a2, a3, a4, a5;

// naming scheme 3
// <animal_emojis>

// naming scheme 4
int arg_1, arg_2, arg_3;
```

Scheme 1 (Sample 1)	
Scheme 2 (Sample 2)	
Scheme 3 (Sample 3)	
Scheme 4 (Sample 4)	

Do you have any better ideas for parameter naming schemes?

1.4.8 Part C Q8

Please consider the following two code snippets. Do you prefer reusing variable names for successive for-loops?

Sample 1:

```c
int foo(int n, int m){
    for (i = 0; i < n; i++){
        do_important_stuff(i);
    }
    for (i = 0; i < m; i++){
        do_other_important_stuff(i);
    }
    return 0;
}
```

Sample 2:

```c
int foo(int n, int m){
    for (i = 0; i < n; i++){
        do_important_stuff(i);
    }
    for (j = 0; j < m; j++){
        do_other_important_stuff(j);
    }
    return 0;
}
```

○ I prefer Reuse (Sample 1)

○ I prefer No Reuse (Sample 2)

1.5 Part D

What other optimization/de-optimization features would you like to see in your decompiled code? (for example, string/ip address does not appear correctly, thus hard to trace origin, etc.)

What quality of life features (for example, GUI interactivity, etc.) do you think would help you use a decompiler much more effectively?

Is there any particular malware sample (or even a specific function), family, or functionality you would like to see as a decompiled output for the next survey?