Results

Survey 968184

Number of records in this query:	44
Total records in survey:	44
Percentage of total:	100.00%

Field summary for lastsurvey

Did you take part in the decompiler survey last year?

Answer	Count	Percentage
Yes (Y)	17	38.64%
No (N)	27	61.36%
No answer	0	0.00%

Field summary for c

How much time (approx.) did you spend working with C code? (Development, Reviews, Learning, ...)

Answer	Count	Percentage
None (C0)	2	4.55%
A few hours (C1)	6	13.64%
Serveral days (C2)	12	27.27%
More than a year (C3)	10	22.73%
On a regular basis (C4)	14	31.82%
No answer	0	0.00%

Field summary for reversing

How much time did you spend reversing executables before?

Answer	Count	Percentage
None (R0)	3	6.82%
A few hours (R1)	10	22.73%
Serveral days (R2)	15	34.09%
More than a year (R3)	8	18.18%
On a regular basis (R4)	8	18.18%
No answer	0	0.00%

Field summary for purpose

What does this function return? Please note that you cannot change the answer after selecting it.

Answer	Count	Percentage
No Idea (A1)	6	13.64%
Manipulates the system time (A2)	8	18.18%
Allows Virtual Machine Detection (A3)	1	2.27%
Generates random domains (AX)	27	61.36%
Encryptes memory regions (A4)	2	4.55%
Comments	17	38.64%
No answer	0	0.00%

ID	Response
1	Obviuously does not encrypt or manipulate system time. Function does not obtain enough information to do virtual machine detection. ASCII printable text in result most likely points to generation of random domains.
4	it does something with the system time apparently. Maybe a repeatedly called generator function?
6	for loop changes first 8 bytes of output, switch statement last 8 bytes. Both changes depend on system time.
7	It returns some random (not really) values generated from system time and tick count. Probably uses it to generate random domains.
16	DGA
17	I don't know how you can dereferance 'var_4', but I guess it's x86?
33	Missing cast when var4 is initialized.
51	Returns a string with eight chars + tld and memory ends with 0x00 e.g.: ? = random printable char ????????.com ????????.to ????????.net ????????.ru
53	To replace the code with RUST since it's far better to use RUST.
58	First I thought it might be used to generate "random" looking domains in order to communicate with a c&c center of a botnet or something similar, but since the generator results depend on the system up time it can not be in sync with another instance of that generator on another machine without communicating the tickcount which would defeat the purpose.
64	var_0 contains the result. Translates first 8 characters of var_0 from GetSystemTime to presumably an alphabetic string using GetTickCount(). Sets 9th character to some kind of checksum value based on GetTickCount(). Sets 10th character to null byte (presumably string terminator).
67	seems to generate a random string of characters using the time
69	getting system time and performing multiple and and or operation
75	Compile the code and run
92	Buffer Overflow var_0[i]: 0-23; 0x00 - 0xE Switch(0-6)
94	Look like always return "a"
105	The char * seems to be a `SYSTEMTIME`struct. The loop manipulates the first four WORDS: year, month, week, day byte wise. The var_4 int points to hour and minute and changes this values in the switch statement. The allocated pointer to the time struct is returned. Since this does not really change the system time and the values are out of valid time values, I have no

idea.

Field summary for tlds

Please type all utilized top-level-domains (TLDs)

Answer	Count	Percentage
Answer	39	88.64%
No answer	5	11.36%

ID	Response
1	.com, .to, .net, .ru
4	.com, .to, .net, .ru
5	.com, .to, .net, .ru
6	.to, .net, .ru, .com
7	.com .net .ru .to
10	.to, .net, .com
9	.com, .to, .net, .ru
14	.com, .to, .net, .ru
16	.com, .ru, .net, .to
17	.com .to .net .ru
18	.com, .to, .net, .ru
28	.com, .to, .net, .ru
33	.com, .to, .net, .ru
31	.com, .to, .net, .ru
41	.com, .to, .net, .ru
47	.idk
51	.com, .to, .net, .ru
53	Rust
57	.com, .to, .net, .ru
58	.com .to .net .ru
64	.com, .to, .net, .ru
65	.com, .to, .net, .ru
67	.com, .to, .net, .ru
69	.com, .de, .sg, .org,.edu, .int, .gov, .ney, .mil, .my, .ae, .af, .ag
70	.to, .ten, .ur, .moc
74	.com, .to, .net, .ru
75	.com, .net, .to, .ru
79	.com,.to,.net,.ru
81	.com, .to, .net, .ru
85	.to, .net, .ru,.com
87	.com, .org, .gov, .net
88	.com, .to, .net, .ru
92	.ru, .net, .com
93	.a
94	.a
101	.com, .to, .net, .ru
102	.com,.to,.net,.ru
105	.com, .to, .net, .ru
106	.com, .ru, .neu

Field summary for frequent

Which Top-Level domain will be utilized the most?

Answer	Count	Percentage
Answer	37	84.09%
No answer	7	15.91%

ID	Response
1	.com
4	.com
5	.com
6	.com
7	.com
10	.to
9	.com
14	.com
16	.com
17	.com
18	.com
28	.com
33	.com
31	.com
41	.com
51	.com
57	.com
58	.com
64	.com
65	.com
67	.com
69	.com
70	.moc
74	.com
75	.com
79	.com
81	.com
85	.com
87	.com
88	.com
92	.ru
93	.moc
94	.a
101	.com
102	.com
105	.com
106	.com

Field summary for domains(SX001)

Which of these second-level domains can potentially be generated by the function? [simpmpfp]

Answer	Count	Percentage
No (A1)	3	6.82%
Yes (A2)	31	70.45%
No answer	10	22.73%

Field summary for domains(SX002)

Which of these second-level domains can potentially be generated by the function? [xfbcbcic]

Answer	Count	Percentage
No (A1)	3	6.82%
Yes (A2)	31	70.45%
No answer	10	22.73%

Field summary for domains(SX003)

Which of these second-level domains can potentially be generated by the function? [facebook]

Answer	Count	Percentage
No (A1)	10	22.73%
Yes (A2)	25	56.82%
No answer	9	20.45%

Field summary for domains(SZ004)

Which of these second-level domains can potentially be generated by the function? [squzuzfz]

Answer	Count	Percentage
No (A1)	21	47.73%
Yes (A2)	13	29.55%
No answer	10	22.73%

Field summary for domains(SZ005)

Which of these second-level domains can potentially be generated by the function? [rlpmpmgmjdh]

Answer	Count	Percentage
No (A1)	33	75.00%
Yes (A2)	2	4.55%
No answer	9	20.45%

Field summary for entropy

How many domains can potentially be generated by this function per day?

Calculation	Result
Count	43
Sum	1439745345753.0000000000
Standard deviation	112635412323.32
Average	33482449901.23
Minimum	0.0000000000
1st quartile (Q1)	24
2nd quartile (Median)	345600
3rd quartile (Q3)	86400000
Maximum	440301256704.0000000000

Null values are ignored in calculations
Q1 and Q3 calculated using minitab method

Field summary for positive

Which aspects in the decompiled codes above is especially favorable to you?

Answer	Count	Percentage
Answer	43	97.73%
No answer	1	2.27%

ID	Response
1	hexrays: Automatic renaming of some variables, creation of structs. ghira: As above. dewolf: Seems to be the cleanest. Likely not as "aggressive" in analysis, but that is favourable in many situations. snowman: Useful information, but with drawbacks. retdec: likewise.
4	hexrays: type info ghidra: type info binaryninja: - dewolf: looks simple and uncluttered snowman: - retdec: - jeb: type info
5	binaryninja: the switch-statement is represented as if-statement. Values, especially numbers, are shown as hexdecimals. hexrays: de-obfuscation of variables, that's neat. ghidra: type-casting is shown. dewolf: proper for-loop. type-casting shown. Values are well very represented and only then shown as characters where appropriate. snowman: - retdec: usage of indentation jeb: the best switch-statement of all shown decompilations: no unnecessary brackets, a default statement, ... -> allows a clear track of flow here. Also, good type deduction for the systemTime.
6	hexray, snowman, retdec: addresses, registers as variable names, or in comments (guess I prefer it as comment as I can rename vars) dewolf, retdec: short lines
7	hex-rays, cleanest code with the min amt of variables used. variable names makes more sense.
10	Using strings resolved e.g: lpSystemTime->wYear in line 16
9	hexrays, ghidra: variable name guessing, including struct members jeb: variable name guessing
14	Hexrays: structure aware (e.g. LPSYSTEMTIME), renames variables based on what is known from the windows APIs (e.g. GetTickCount())
16	* default case for switch statement: hexrays, jeb * struct resolution: e.g. &lpSystemTime->wSecond: hexrays, ghidra, (jeb) * correct variable type cast, e.g. LPSYSTEMTIME lpSystemTime: hexrays, ghidra, jeb * DGA algorithm easily readable in: dewolf * switch case possibilities: TickCount % 8 in hexrays,
17	Best one is not surprisingly IDA it seems both very accurate and quite comprehensible. Only thing is that both '1836016430' and '97' should probably be in hex, but there probably is an option to default to hex. Ghidra struggles with the '%' alot but otherwise it seems pretty much on par. Dewolf and jeb have rather basic but pretty clean code. They are not trying to be as smart (not trying to infer the LPSYSTEMTIME) which kinda helps in this case as there is an intentional type confusion.
18	hexrays, ghidra, retdec, jeb: - recognize LPSYSTEMTIME type -> also struct fields recognizable (e.g. lpSystemTime->wYear) (had to look this up via docs.microsoft.com) generally: viewer lines to read is (mostly) better

28	Heyrays: Automated annotation of the stack layout Heyrays, Ghidra, binaryninja, dewolf, jeb: Annoate name of GetSystemTime parameter dewolf: Very clean decompilation. Proper placement of var_0[i] instead of pointer castings
33	HexRays: Reasonable variable names and datatype inference, accurate decompilation, and no redundant typecasts. Optimization to detect modulo arithmetic. Looks the 'cleanest' Ghidra: Correctly optimizes away the default case in switch statement. Few redundant typecasts. Quite clean Dewolf: Similar to hexrays, slightly less accurate, but optimizes away the redundant default case in the switch statement. Clean look. JEB: Casts the malloc to LPSYSTEMTIME correctly. Relatively clean look. retdec: The comments with addresses is unnecessary bloat
31	hexrays: 1) Automatically inferring of the return type of windows API. 2) Renaming of the variables based on the inferred types. dewolf: 1) Representing the 0x61 constant as 'a'. 2) The for loop that sets the 2nd level domain is much more comprehensible compared to all other because it has less noisy type casting.
41	hexrays is the cleanest.
47	In HexRays, it detects the LPSYSTEMTIME struct, and the rest of the code interacting with it is in struct form. Ghidra does the same, albeit in a more standard C way.
50	hexrays
51	dewolf: - nice and clean switch case structure - good variable definition -
53	dewolf
57	dewolf looks very clean and is easy to read hexrays looks also very clean, you get an idea of the stack trough the comments of esp and ebp if I would ignore the comments in retdec, its also good to read jeb is ok, nothing too bad, but would not prefer about bn I like, that the variable names enumerate the used registers
58	It really helps to see the last part of the code as a switch case instead of nested if-statements. While it might be useful to see all the typecasts / types explicitly I like it more to see more "clean" or shorter statements. Therefore I liked the way the while/for loop was presented by all the decompilers except for snowman. I liked the most: dewolf / retdec / jeb (no particular order)
64	Hexrays: The ability to break down the structure of lpSystemTime without requiring me to search for the documentation --> good vraiable naming.
65	hexrays, ghidra: meaningful variable and field names dewolf: appears closest to actual written code
67	- useful comments (hexrays) - recognized mod 24 (as decimal int) and the "a" as char (dewolf)
69	dewolf has the most accurate decompiled codes.
70	looks more like C code
74	Hexrays: Naming of the buffer variable with parameter and applying type is usually helpful, although not so much in this case. Code is concise. Switch statement is the simplest amongst the options. Ghidra: Naming of the buffer variable with parameter and applying type is usually helpful. Reasonably concise decompilation. Binary Ninja: n/a Dewolf: Basic but fairly clean and concise decompilation. Not applying the lpSystemTime struct happens to be helpful here, making the loop simpler to read. Switch parameter is simpler than some of the others. Snowman: n/a Retdec: Loop is easy to read, not applying the struct happens to be helpful as well. Jeb: Mostly simple and clean decompilation.
75	You can compile hexrays, ghidra and dewolf code easily to dynamically debug them .
78	hexrays / ghidra: + STRUCT TYPE + var naming + easy switch
79	hexrays: types and struckts, variable names
81	At first glance, dewolf will be most favourable because the previous exercise has required the

	need to go through the same set of information. However if an unbiased mind is to review the codes from the start again, Ghidra would be especially favourable due to the consistency of the values being primarily resolved to hex values.
85	hexrays: naming of variables, type hints, clean switch case, nice for loop dewolf: very nice for loop, somewhat clean switch case, not resolving lpSystemTime fields (since they are misused anyway)
87	hexrays has better variable naming
88	dewolf decompiled the codes in a cleaner and more understandable manner compared to the other decompilers.
90	Hexrays managed to provide meaningful name(s) for variables, e.g. p_wHour, TickCount, wHour, wSecond
92	hexrays, retdec
93	using switch and case for options
94	Switch and choice options
101	I like the variable naming of hexrays as it already contains some meaning. I like that binaryninja, retdec, and jeb combine variable declaration and definition where possible. Especially jeb has only a short variable declaration/definition section at the beginning and moves the definition of loop variables into the loop. This helps keeping the code compact and doesn't force me to remember lots of variables. In general, I like that hexrays and ghidra seem to know the structure of LPSYSTEMTIME and use its field accessors (->wYear, ->wHour). Although in this particular example this doesn't really help or make sense, in general I think this would be useful. This is my favorite loop header "for(char i = 0; i < 8; ++i) {" (jeb). The for loop is definitely better that all while loops. From an overall perspective, dewolfs output looks "most natural" to me. I. e. this could be code as written by a human. The output is rather uncluttered (no cast chains, magic numbers, or special notations).
102	hexrays: struct resolution implies reuse of fields for other purpose (wHour,wMin for TLD), inconsistent number display (dec, hex) ghidra: unlucky comparions (local_5 < '\b'), but somewhat consistent use of hex numbers. logic expressions are a bit harder to understand (bitmasking, ...) binja: explicit registers instead of abstraction makes it harder to comprehend. dewolf: looks clean, but inconsistent number display (dec, hex) snowman: too much casting, no simplification through resolved aliasing, inconsistent number display (dec, hex) retdec: VAs as comments clutter the code, this code be solved better with highlighting in a UI jeb: assignment GetTickCount->DWORD->char makes it somewhat more obvious there are only 256 values possible
105	The correct typing of the LPSYSTEMTIME (hexrays, ghidra, jeb) variable in this case is not helpful, in other cases it may be though. hexrays: switch statement seems clearer dewolf, retdec: clean layout
106	ease of understanding, jeb.
110	dwolf code make it easy to analyze

Field summary for negative

Which aspects of the presented decompiled functions do you deem unhelpful or hamper your understanding of the code?

Answer	Count	Percentage
Answer	41	93.18%
No answer	3	6.82%

ID	Response
1	ghidra: Not as good at handling loops. (e.g, line 14) binary ninja: Nasty way of naming variables. snowman: type-casting is useful when deep diving, but for quick glances, it only adds to clutter the viewport. (hotkey to toggle?) retdec: Addresses on comments only add to the clutter. jeb: Type-casting seems off
4	hexrays: I don't care about registers ghidra: numbers represented as char strings, all those () and * and shit binaryninja: I don't care about registers, not even in variable names dewolf: - snowman: are we using registers as variable names again? also, what is this casting nonsense retdec: What are those comments about jeb: all those () and *
5	binaryninja: the for-loop is shown as while loop. Casts are sometimes difficult to read. hexrays: the default in the switch-statement bugs me and it is redundant. Usage of decimal values seems inappropriate here. ghidra: strange naming of variables. local_5 is shown as character, which confuses more than it helps. dewolf: no indentation in switch-statement snowman: not clear what intrinsic does. Casts clutter the lines. Very bad naming convention (variable names should be longer) retdec: those comments... really? Fall-through cases are less clear with these brackets jeb: too many casts (line 8).
6	I lack the experience to say what I see as unhelpful other than the opposit of question 1
7	binja's (and snowman) variables being named after their registers is a huge pain to read, I also have no idea what is sx.d etc.
10	In line 19, &lpSystemTime->wHour replaces the address calculation. May be confusing.
9	binaryninja, snowman: very basic disassembly, looking at their assembly codes might be more useful.
14	ghidra: Converts the variable to a char (local_5) in the while loop. Not structure-aware as well (e.g. &lpSystemTime->wYear + (int)local_5): have to manually calculate the offset)
16	* weird type castings that clutter the code: ghidra, snowman (!), jeb * IR: Binja IR is okayish but doesnt even come close to fully decompiled code like in hexrays or jeb * retdec code formatting and comments (??) * weird ass loop condition: ghidra (while local_5 < '\b', the heck is that..
17	1) The crazy C++ cast Situation in snowman, also not understanding a cdq before a mod is kinda weird. 2) Binja is not really a decompiler its more of a visualization of their bytecode. 3) Retdec is (I think) a command line tool so the comments at the end of the line make sense, but are really verbose. They would probably be less annoying if they were aligned and way to the right. 4) In Jeb it seems like some of the Casts should be able to get folded. '(unsigned int *)(char *) = (unsinged int *)' and '(char *)(int + (int)ptr) = (char *)ptr + int'
18	snowman: - uses C++ style casts which is a bit strange - inline assembly (e.g. cdq) should be avoided -> use casts for that binaryninja: - poor type recognition, also sx.q, zx.d etc. not nice to read if you don't know what it means

hexrays vs. dewolf:
- `TickCount % 8` vs `(((int) var_1) % 0x8) - 0x1)` -> this is IMHO not the same condition... but +1 for the modulo vs OR representation

ghidra:
- loop recognition and variable usage

```
 while (local_5 < '\b') {
   *(byte *)((int)&lpSystemTime->wYear + (int)local_5) =
        (byte)(*(byte *)((int)&lpSystemTime->wYear + (int)local_5) ^ (byte)DVar1) % 0x18 +
0x61;
   local_5 = local_5 + '\x01';
 }
```

pretty messed up for a simple for loop that counts from 0 to 7 :D

binaryninja:
- if vs. switch... switch is better to read

hexrays:
- has a redundant default: return lpSystemTime -> +1 for dewolf

snowman:
- goto 0x40112d; -> no idea where that is ^^
- C++ casts clutter everything and have no real "advantage" if this isn't a C++ program

dewolf:
- i don't like the GetSystemTime(lpSystemTime: var_0); lpSystemTime

jeb:
- inconsistent use of hex representations e.g.

```
    *ptr0 = 1836016430; // decimal
    *(ptr0 + 1) = 0; // decimal
    int v2 = (int)v1 & 0x80000007; // hex
```

ghidra and others that do not use the % operator:
- this is way to complicated to be easily recognized:

```
 uVar3 = (int)(char)(byte)DVar1 & 0x80000007;
 if ((int)uVar3 < 0) {
   uVar3 = (uVar3 - 1 | 0xfffffff8) + 1;
 }
 switch(uVar3) {
```

28	Heyrays: Integers "randomly" being in decimal Heyrays, Ghidra: The default type of the 16 byte array being LPSYSTEMTIME and all access then being treated like it is that type Snowman: The casts hinder readability Ghidra, binaryninja, snowman : No for loop detection Ghidra, binaryninja, snowman, retdec, jeb: Don't understand "TickCount % 8" and produce hard to comprehend code for it snowman:__asm__("cdq "); and gotos binaryninja: Too similar naming scheme and things like sx.q and mods.dp.d(edx_4:eax_6, 0x18) dewolf: "- 0x1" on the switch which causes the -1 case which is a bit more confusing compared to IDA Ghidra: Random usage of `\b` and `\x01` instead of hex numbers
33	Ghidra: Redundant casts calculating uVar3. No mod 8 optimization. Binary Ninja: sx.q, zx.d, mods.dp.d etc. are alien syntax to people used to C. No mod 8 optimization. Dewolf: Missing typecast on var4 made the surrounding code very confusing Snowman: reinterpret_cast / static_cast bloat the screen and hide the other more important code. Unimplemented instruction cdq. goto without a matching label. No mod 8 optimization. JEB: A few weird typecasts on the TickCount
31	binja: Things like "*(eax + sx.d(var_5)) = (mods.dp.d(edx_4:eax_6, 0x18)).b + 0x61" is not C and so it's hard to understand
41	Too much casting, and no API type information.

47	In BinaryNinja, the variable names are chosen by the register they were found in, which makes reading the code far harder than with other decompilers. In Dewolf, the type decompilation was far off from other decompilers, making analysis difficult. In Snowman, the use of C++ type casts makes the block of code very cluttered and difficult to understand.
50	the comments beside the local variable declaration sometimes are not helpful
51	dewolf: -> + 'a'; irritated me at first -> switch((((int) var_1) % 0x8) - 0x1)
53	.
57	snowman has this string "reinterpret_cast
58	It was quite easy to determine which domain is used most due to the way the switch case was presented. BinaryNinja does not present that part of the code as a switch case but rather nested if statements which is also fine but - at least to me - harder to wrap my head around it. The while loop from the snowman generated code looks so awful that it encurages me to ignore it the very second I see it.
64	retdec: comments corresponding to address of equivalent instruction bytes make it harder to process the main logic visually. snowman: excessive display of static_cast and reinterpret_cast
65	binaryninja: bad variable names snowman: too many casting functions, incomplete decompilation with asm snippets remaining
67	- for loop not recognized (ghidra, binaryninja, snowman) - comparison with char where it does not make sense (ghidra) - chained type cast (ghidra) - reinterpret_cast ??? (snowman) - switch statement not recognized (binaryninja) - goto (snowman) - unnecessary many returns (jeb)
69	In snowman, overuse of reinterpret_cast, static_cast can be confusing
74	Hexrays: Applying the struct type happens to make the loop harder to read, but it is not a huge issue. Ghidra: The excess casts can be slightly annoying. Bit manipulation of the switch argument is a bit difficult to parse. Binary Ninja: The IL is not particularly helpful in my opinion. Would prefer to read the assembly instead. Dewolf: No particular complaints. Switch could be slightly simplified (see Hexrays version) but is not a huge issue. Snowman: Excessive casting along with verbose C++-like casting syntax is off-putting. The bit of inline assembly appearing in the loop is not helpful. Bit manipulation of switch argument is a bit difficult to parse. I have no idea what __intrinsic() is supposed to mean. Retdec: Decompiled code could be more concise. Arithmetic and bit manipulation of switch argument is difficult to parse. Jeb: Casting in loop makes it slightly harder to read. Bit manipulation of switch argument is a bit difficult to parse.
75	binaryninja decompiler seems to be very convoluted
79	snowman: casts are confusing ghidra/binaryninja: everything
81	snowman decompiler presented a decompiled code that was too messy to be picked up at first glance, binarywolf decomipler would have been useful for memory diving, but not when determining the use of said code in example The additional "*(undefined4 *)" presented by Ghidra would have thrown me off the table if a prior exercise had not been done.
85	binaryninja: while loop with horrible body, no switch case snowman: verbose type casting retdec: many lines with useless arithmatic expressions
87	snowman has many casts
88	binaryninja and snowman seems to be too complicated to understand what has been decompiled
90	Going towards assembly (binaryninja), e.g. eax, edx does not seem to be helpful to

	understand the intent of the code
92	binaryninja, dewolf
93	Using cpu register name in the code
94	using cpu register name in the code
101	Basically, the things that make the code feel "unnatural": I don't like long cast chains. In jeb, there is the part "(unsigned char)((unsigned int)*(char*)((int)i + (int)lpSystemTime)". Here it's quite easy to miss the i in the middle. This gets worse when we use C++ casts as snowman does. This makes it really hard to find actual code in between the cast operations. I also don't like special notation as binaryninja uses. There we have this .b and .d and an sx function which are not really C. You need special knowledge to find out what they mean and I'd prefer not having to know this when reading the output of a debugger. Especially, since they don't help in understanding the code in this example. The same holds for some magic constants which are present in retdec for example (lines 13-14). I'd have a hard time understanding why they are there and what they do. Finally, I don't like that ghidra and binaryninja use hex notation also for small integers. In the malloc call I definitely prefer 16 over 0x10.
102	while probably most accurate, too fine-grained output (binja) is a bit contraproductive for fast analysis (might as well look at assembly then). too much casting clutters the output (snowman). automated application of structs and fields is usually very helpful, here it's potentially slightly confusing because of the "unintended" use (hexrays,ghidra).
105	hexray: switching between hex und dec representation of numbers. ghidra: while loop with char limit mixed with hex increment hexrays, ghidra, jeb: in this case, the correct type LPSYSTEMTIME and the resulting byte casting in the loop binaryninja: weired style snowman: cluttering c++ casts dewolf, retdec: incorrect types sometimes retdec, jeb: brackets in switch cases may lead to false assumptions
106	require a longer time to understand the codes, snowman
110	Snowman and gihidra code can be hard to read

Field summary for feedback

Thank you for getting this far! If you would like to leave us any feedback about the survey, please use the lines below to help us improve ourselves.

Answer	Count	Percentage
Answer	14	31.82%
No answer	30	68.18%

ID	Response
1	More code decompilation examples would be helpful.
10	NIL
16	all gucci
18	This is the first time I actually saw a DGA in C/Pseudo C :D So my results will not be biased be former experience of reversing DGAs. The usage of available API information (data types, names, struct defs) is one of the most important things to quickly understand what's going on. Complex expression with low level shifting, bitwise operations should be converted to conditions that are easier to understand (e.g., modulo vs ... | ...) dewolf + API information would be pretty close to hexrays IMHO The length of the survey was very manageable and thus was quite entertaining ;)
53	Give me more Rust
58	After taking a break and resuming the survey I was able to change my first answer because it was not selected anymore.
75	very interesting survey
79	The code editor did not let me select-to-copy, that was anoying.
81	It is not fair to the other decompiles if one of the contestant's output is used as the precursor exercise.
92	nil
93	No feedback at the moment
94	Not at the moment
101	This was more fun that the last one! ;-) When asked again which aspect of compiler output I like, it would be cool to have the ability to comment inline in the output as e. g. in a code review. But I'm not sure if this is technically possible. Also, it would be nice to have the different decompiler outputs side-by-side so that I don't have to click through the tabs. Can we get the solutions?
110	Nil