

CTO (Call Tree Overviewer)

Yet Another Function Call Tree Viewer

Hiroshi Suzuki

Malware & Forensic Analyst

Internet Initiative Japan Inc.

Who Am I

- Hiroshi Suzuki is from “Internet Initiative Japan Inc.” that is called “IIJ” for short.
 - IIJ is a Japanese ISP.
 - We are the first commercial ISP in Japan.
 - I belong to “IIJ-SECT”, which is the CSIRT team of our company.
 - I’m a malware analyst, a forensic investigator, an incident responder and a security researcher.
- I have been a Black Hat Briefing speaker (USA, Europe and Asia). And I have also been a trainer at Black Hat (USA and Japan).



Internet Initiative Japan



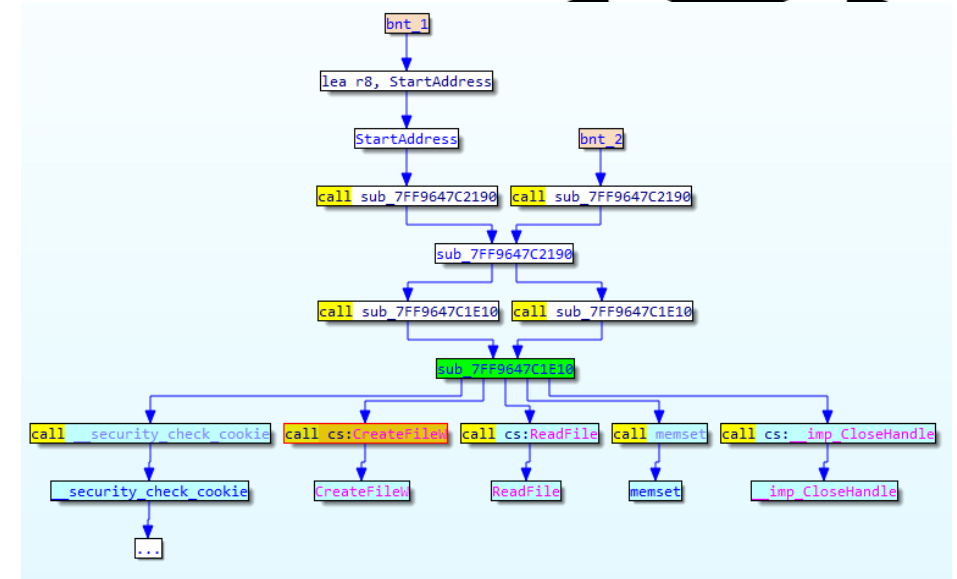
IIJ-SECT
IIJ Group Security Coordination Team



What is CTO?



- CTO is an IDA Pro plugin for visualizing function call tree.
- It can also:
 - Summarize function information such as:
 - Internal function calls
 - API calls
 - Static linked library function calls
 - Unresolved function calls
 - String references
 - Structure member accesses
 - Comments
 - Find paths to/from a function.
 - Get arguments of unresolved calls by applying type information.
 - Collect other tools result such as ironstrings and fincrypt.py.
 - And so on.



Motivation

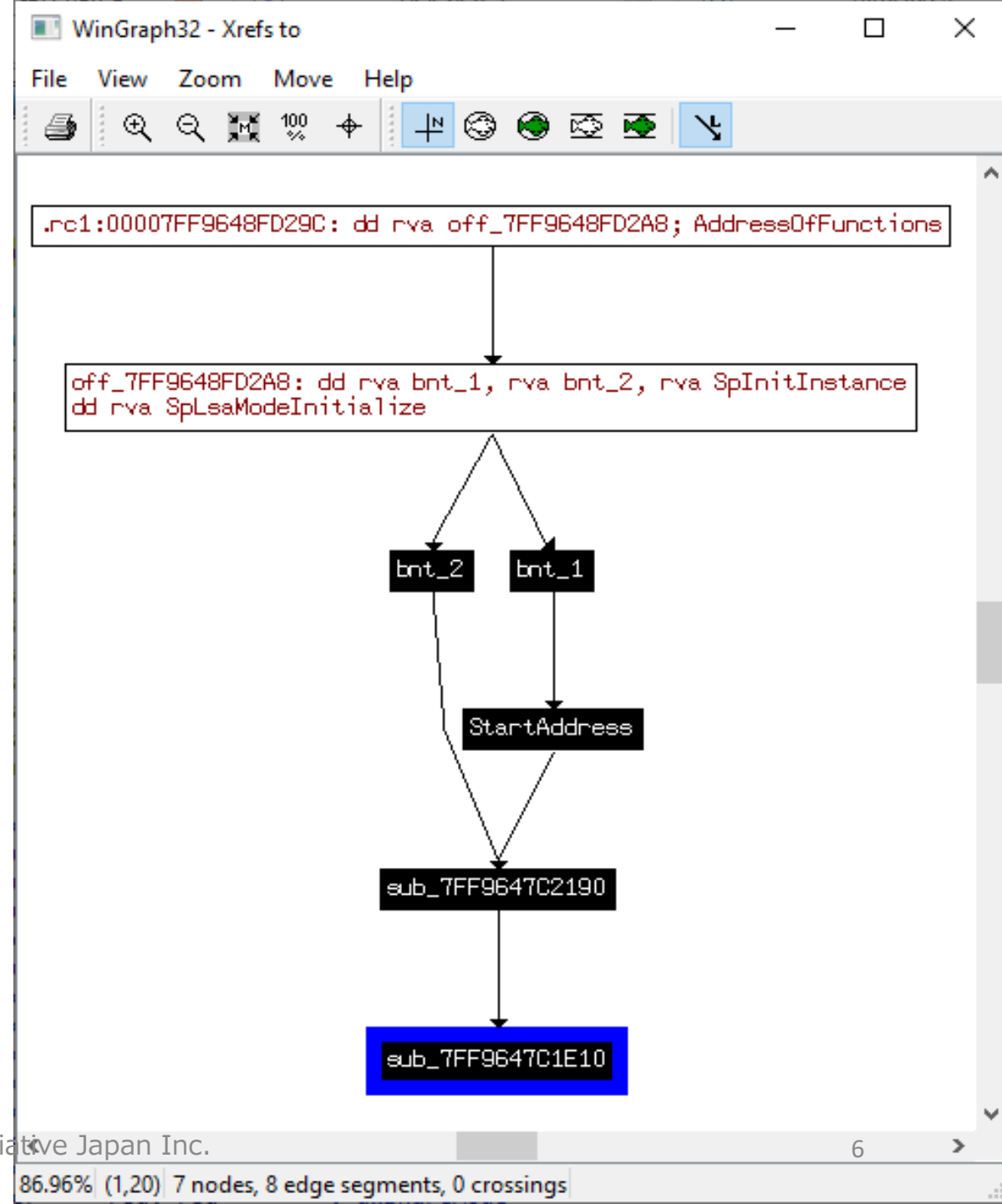
- Why did I develop CTO?

Motivation - Why did I develop CTO? (1)

- There are already two features related to function call tree graph in IDA Pro.
- One is called the "Graphs" or "Chart" feature, and another one is called "Proximity Browser".
- Despite of them, why did I decide to develop this plugin? Let me explain about it.

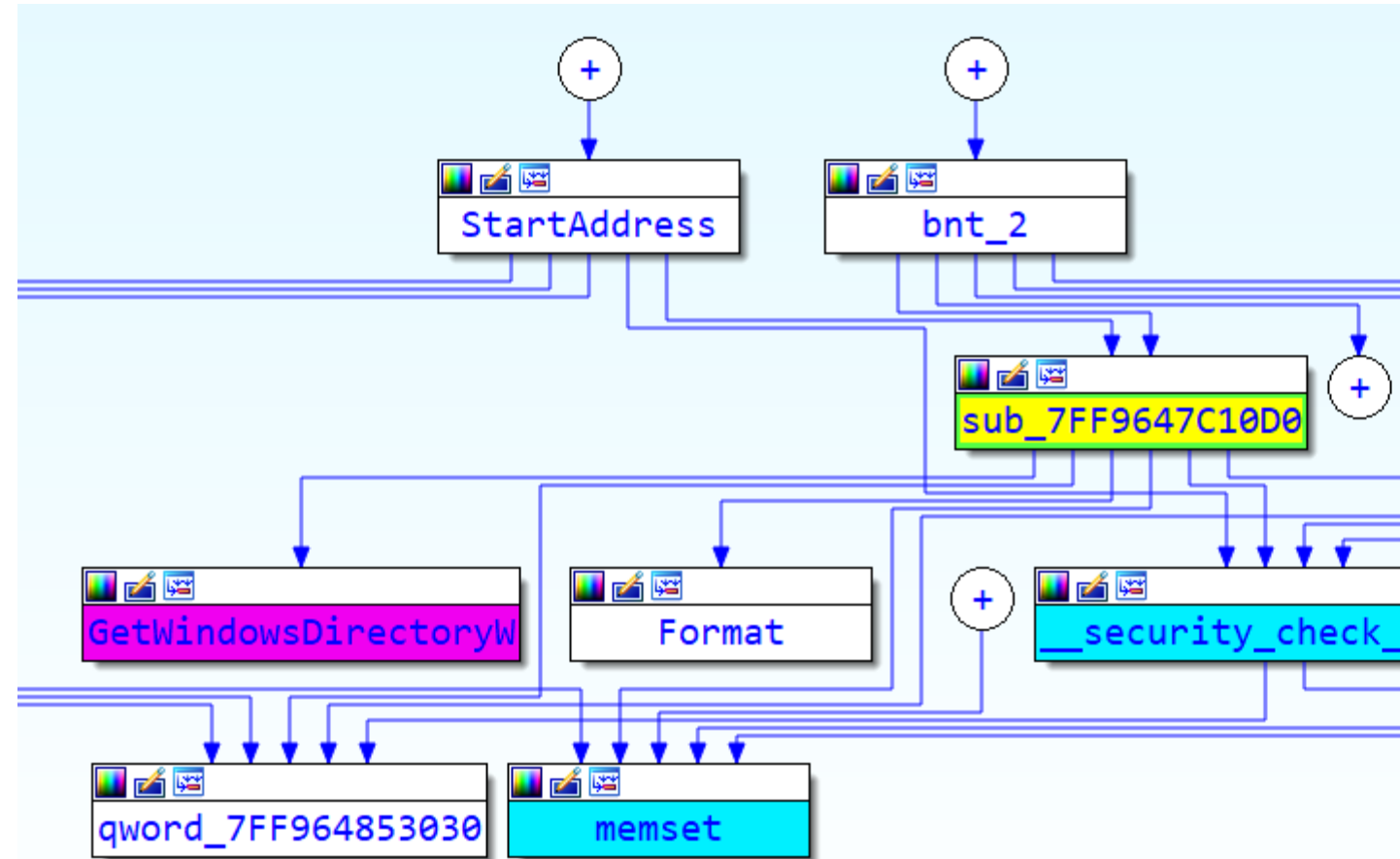
Motivation - Why did I develop CTO? (2)

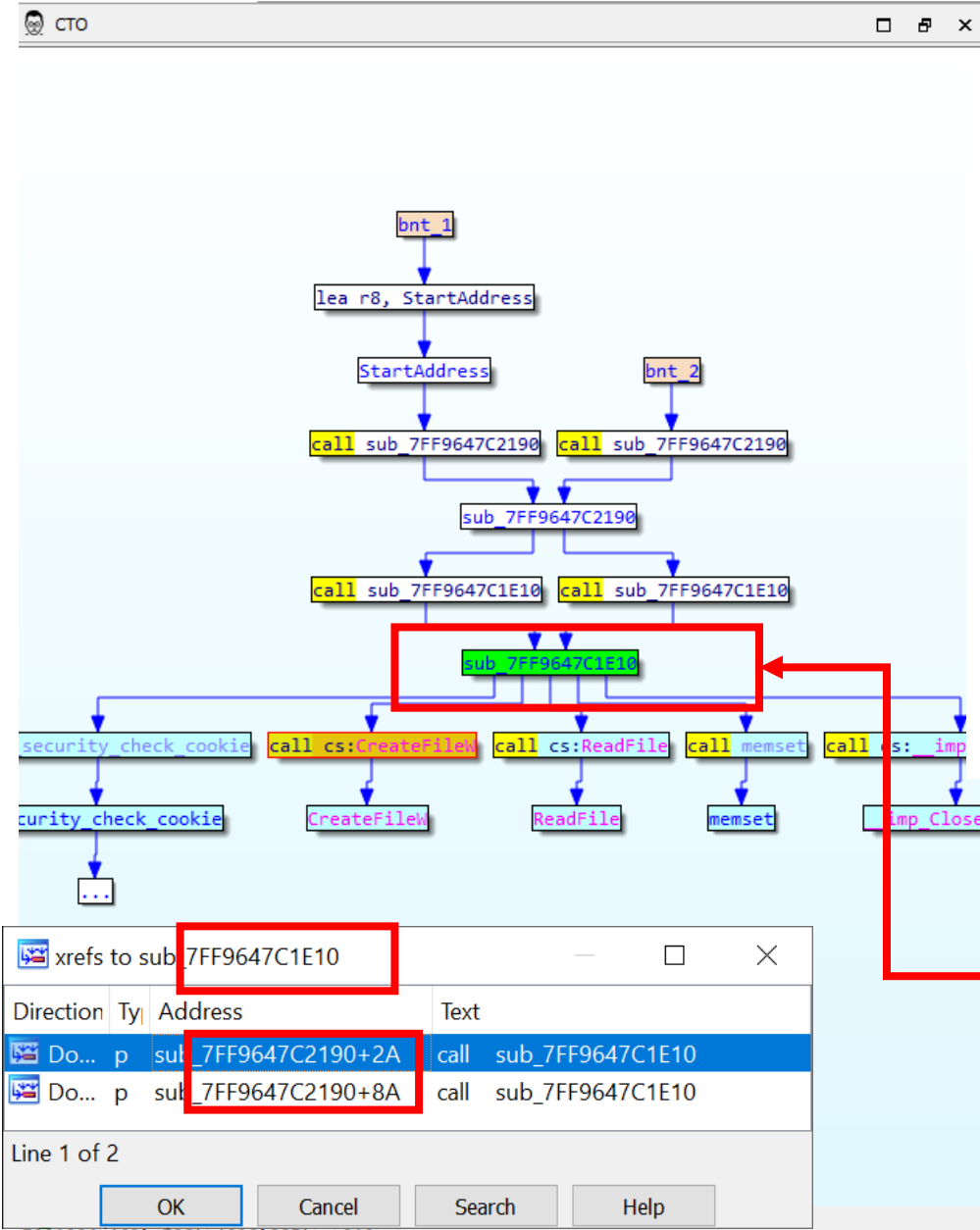
- The "Graphs" or "Chart" feature does **NOT** generate **clickable graphs** and lacks of path filter feature because it generates a graph with a modified version of WinGraph.



Motivation - Why did I develop CTO? (3)

- On the other hand, “Proximity Browser” is more sophisticated feature than the former one.
- You can click nodes and there are filter feature and path discovery feature as well.

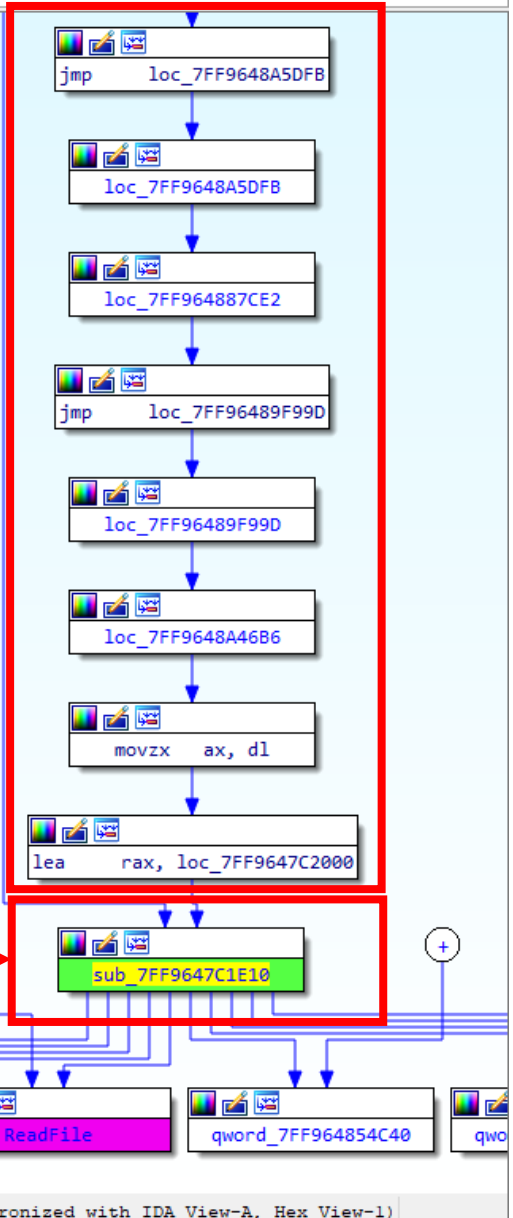




However, it tends to follow unnecessary paths and xrefs, and an area per node is large because of the menu bar on every node.

This is an example of a result comparing CTO (on the left) with Proximity Browser (on the right) at the same address.

Compared to CTO's result, Proximity Browser traced another path that is not listed in the xrefs of the function.



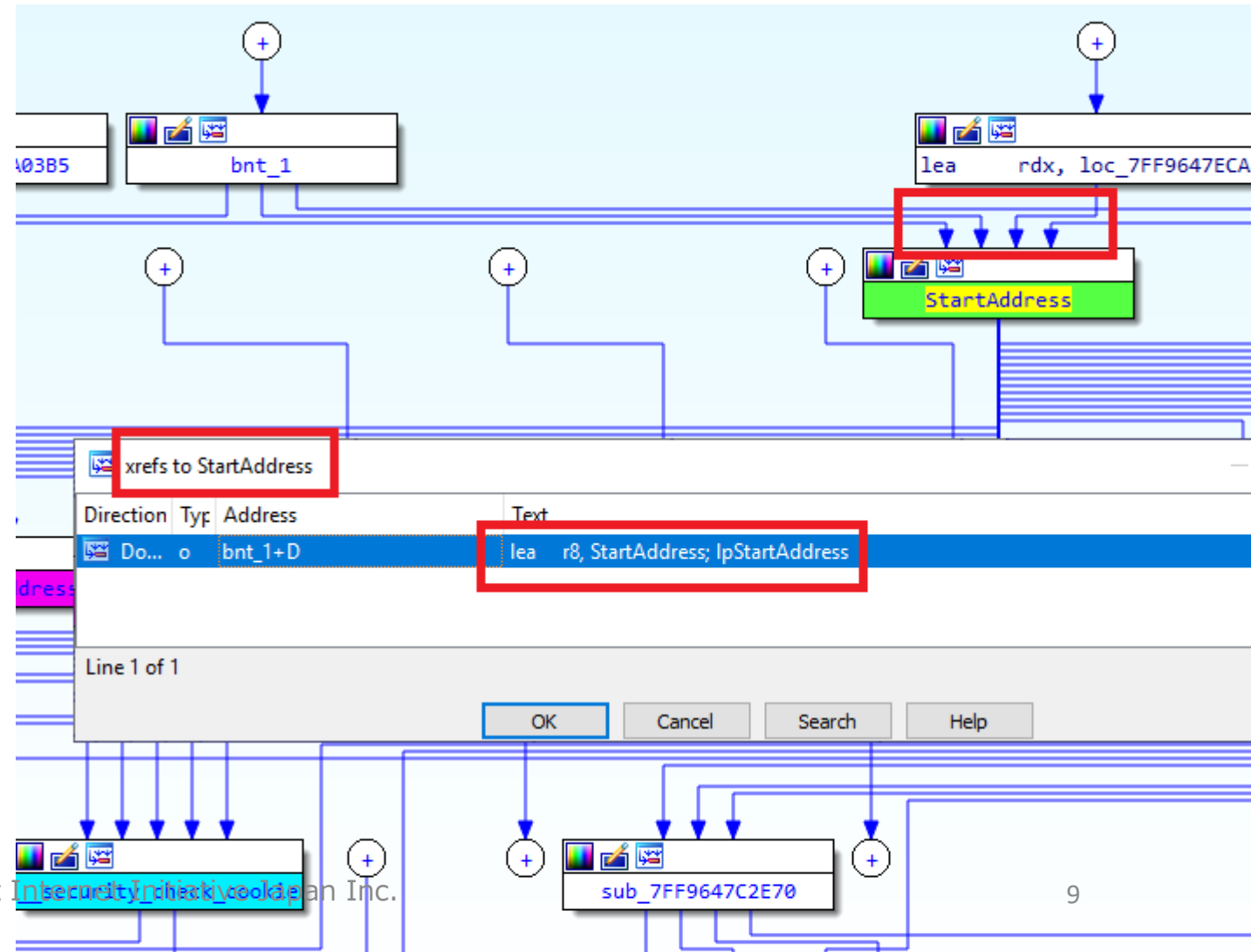
Direction	Type	Address	Text
Do...	p	sub_7FF9647C2190+2A	call sub_7FF9647C1E10
Do...	p	sub_7FF9647C2190+8A	call sub_7FF9647C1E10

Line 1 of 2

OK Cancel Search Help

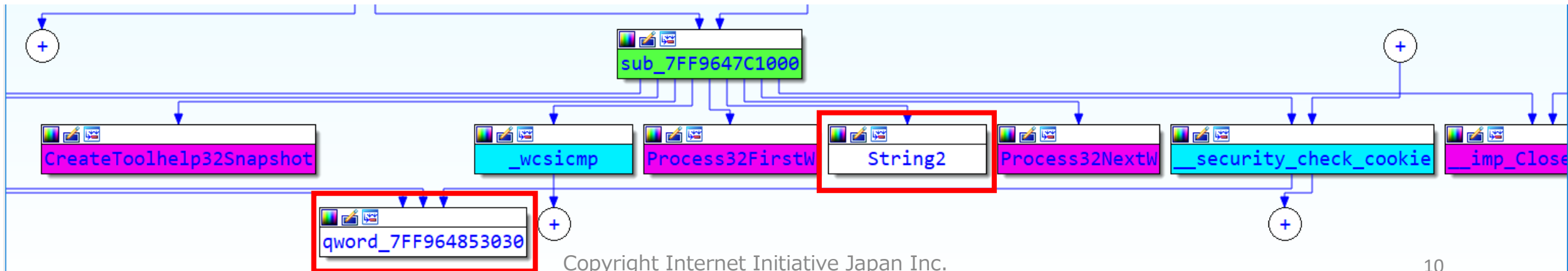
Motivation - Why did I develop CTO? (5)

- You can also see Proximity Browser shows four paths, which you can see four arrows are connected to “StartAddress” function, while IDA’s xrefs window for it shows only one xref.



Motivation - Why did I develop CTO? (6)

- Furthermore, Proximity browser always displays all types of nodes.
- The figure below shows us Proximity Browser displays a string node and a global variable node.
- Large number of nodes makes us difficult to understand the relationships due to the complexity.
- These disadvantages made me to develop CTO.



Introducing CTO

Introducing CTO (1) CTO's Widgets

The screenshot displays the CTO (Control Flow Transformer) interface, which is used for analyzing and transforming control flow graphs. The interface is divided into several main sections:

- CTO function lister:** Located on the left, it shows a tree view of functions. The selected function is `sub_7FF9647C1E10`, which is a child of `sub_7FF9647C1E10`. The list includes various system calls like `cs:CreateFileW`, `cs:ReadFile`, `memset`, and `imp_CloseHandle`.
- Assembly code:** The central pane shows assembly instructions for the selected function. The code includes instructions for setting up registers, calling `cs:CreateFileW`, and `cs:ReadFile`. A red arrow points from the `call cs:CreateFileW` instruction in the assembly to the corresponding entry in the function lister.
- Control Flow Graph (CFG):** On the right, a CFG is displayed. It shows a sequence of basic blocks connected by control flow edges. The graph starts with `bnt 1`, followed by `lea r8, StartAddress`, `StartAddress`, `bnt 2`, and several `call` instructions to `sub_7FF9647C2190`, `sub_7FF9647C1E10`, and `sub_7FF9647C1E18`. The final block is `sub_7FF9647C1E18`, which branches to `call security check cookie`, `call cs:CreateFileW`, `call cs:ReadFile`, `call memset`, and `call cs: imp_CloseHandle`.

CTO function lister

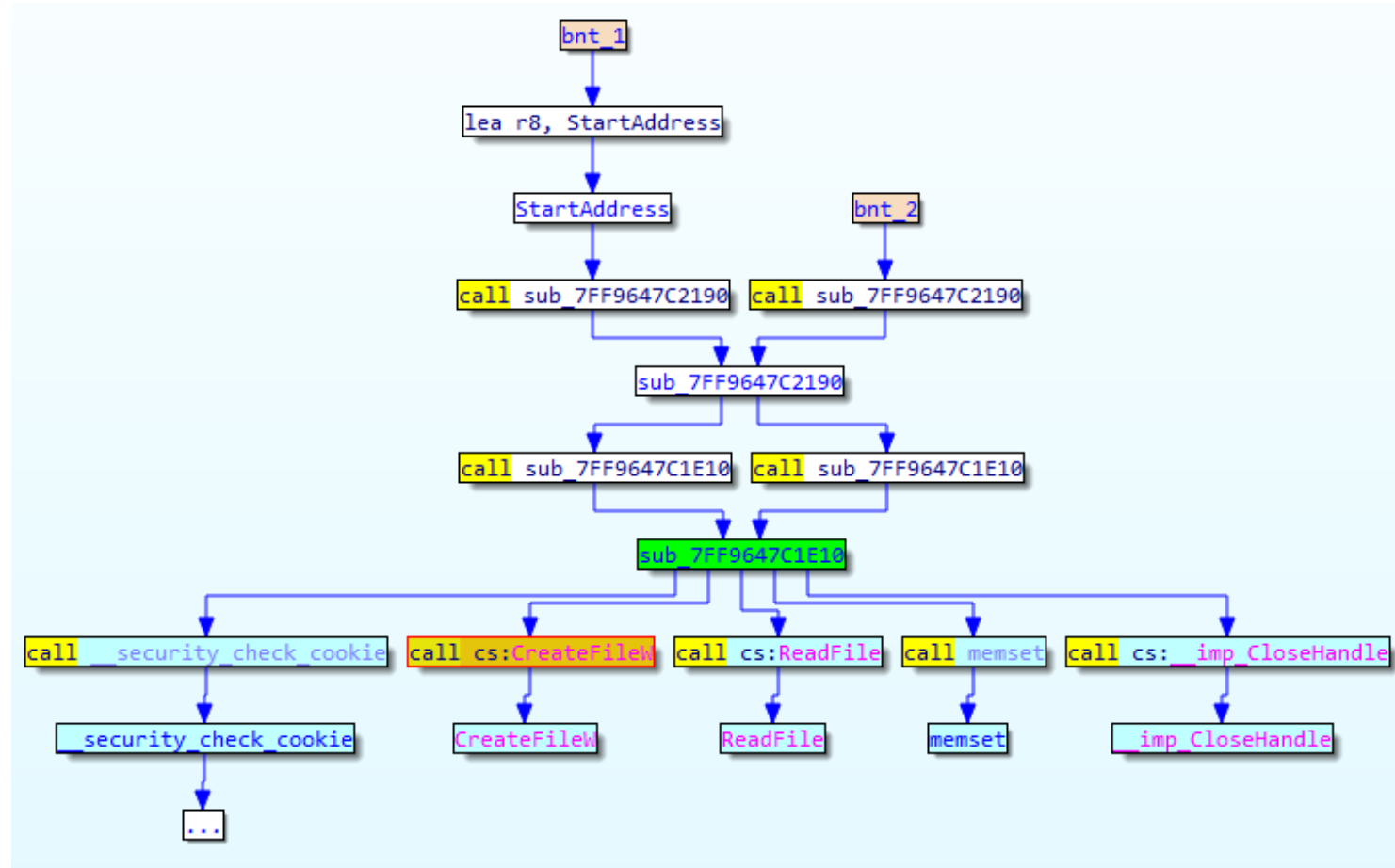
CTO's main window

Copyright Internet Initiative Japan Inc.

12

Introducing CTO (2) Main Window

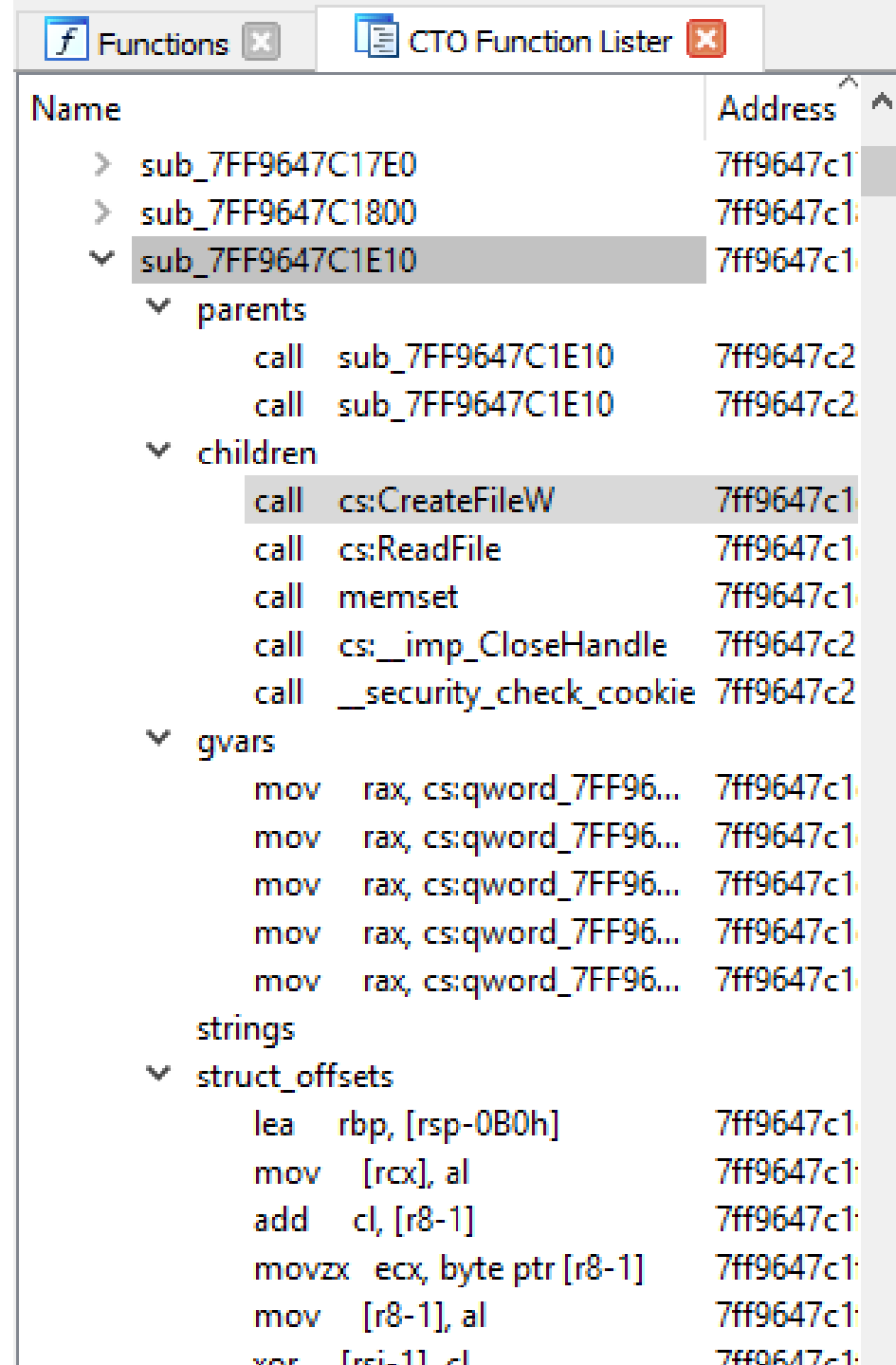
- Graphical function call tree viewer.
- It is aimed to:
 - Grasp function relationships.
 - Check important information in a function.
 - Functions
 - Strings
 - Global/Static variables
 - Structure offsets
 - Comments
 - Find paths to/from a function
 - Handle IDA such as
 - Renaming a function/variable
 - Applying a structure to an offset
 - Applying a function definition to a indirect function call



Introducing CTO (3)

CTO Function Lister

- Enhanced function list.
- It is aimed to:
 - Check/Find important information in a function.
 - Functions
 - Strings
 - Global/Static variables
 - Structure offsets
- You can use a regex filter to find nodes with a specific pattern.



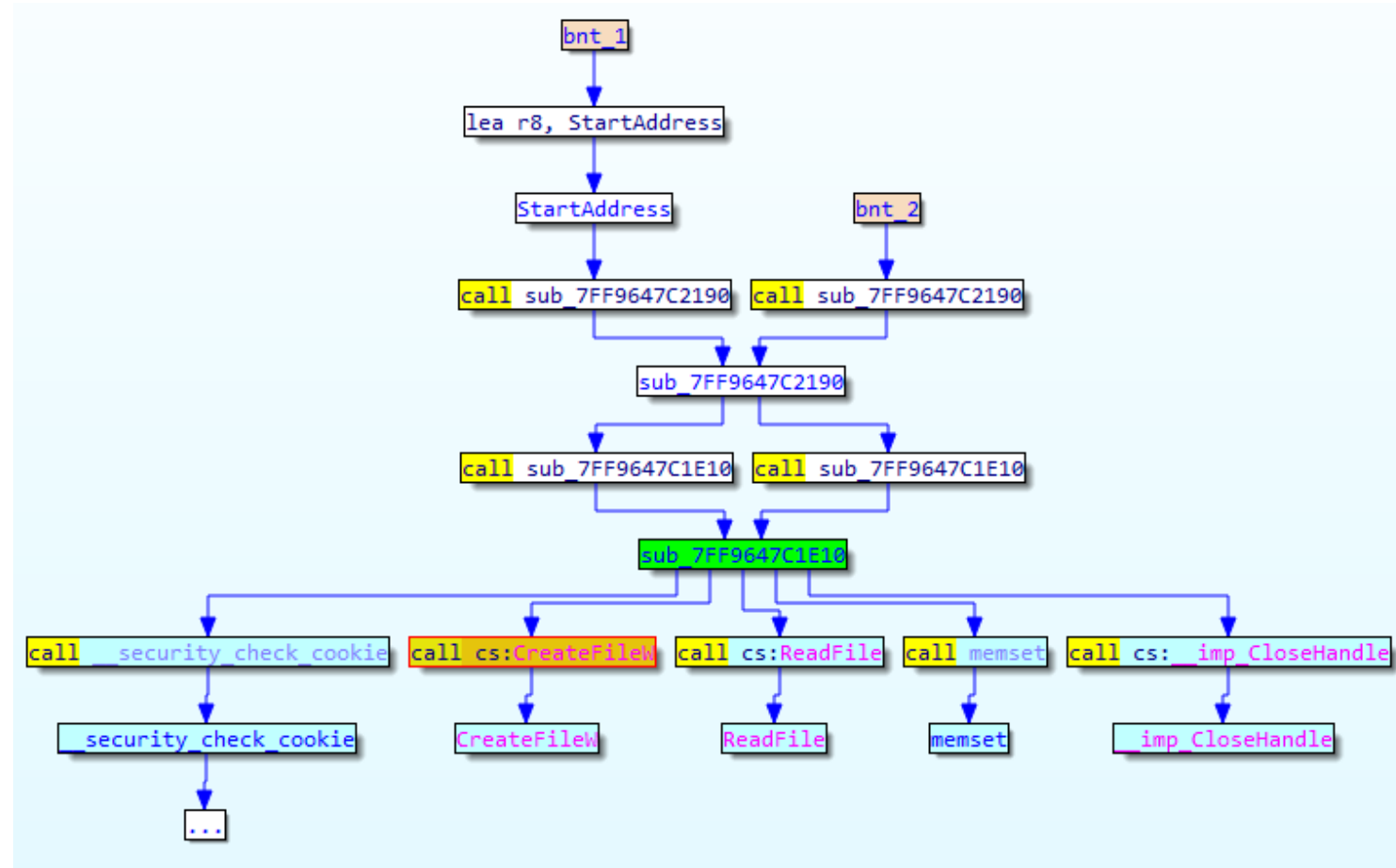
The screenshot shows the 'CTO Function Lister' window with a tree view of function calls and offsets for the function `sub_7FF9647C1E10`. The tree is expanded to show the following structure:

Name	Address
> sub_7FF9647C17E0	7ff9647c1
> sub_7FF9647C1800	7ff9647c1
▼ sub_7FF9647C1E10	7ff9647c1
▼ parents	
call sub_7FF9647C1E10	7ff9647c2
call sub_7FF9647C1E10	7ff9647c2
▼ children	
call cs:CreateFileW	7ff9647c1
call cs:ReadFile	7ff9647c1
call memset	7ff9647c1
call cs:_imp_CloseHandle	7ff9647c2
call __security_check_cookie	7ff9647c2
▼ gvars	
mov rax, cs:qword_7FF96...	7ff9647c1
mov rax, cs:qword_7FF96...	7ff9647c1
mov rax, cs:qword_7FF96...	7ff9647c1
mov rax, cs:qword_7FF96...	7ff9647c1
mov rax, cs:qword_7FF96...	7ff9647c1
strings	
▼ struct_offsets	
lea rbp, [rsp-0B0h]	7ff9647c1
mov [rcx], al	7ff9647c1
add cl, [r8-1]	7ff9647c1
movzx ecx, byte ptr [r8-1]	7ff9647c1
mov [r8-1], al	7ff9647c1
xor [r8-1], cl	7ff9647c1

Main Features of CTO (1)

Simple But Sufficient Tree (1)

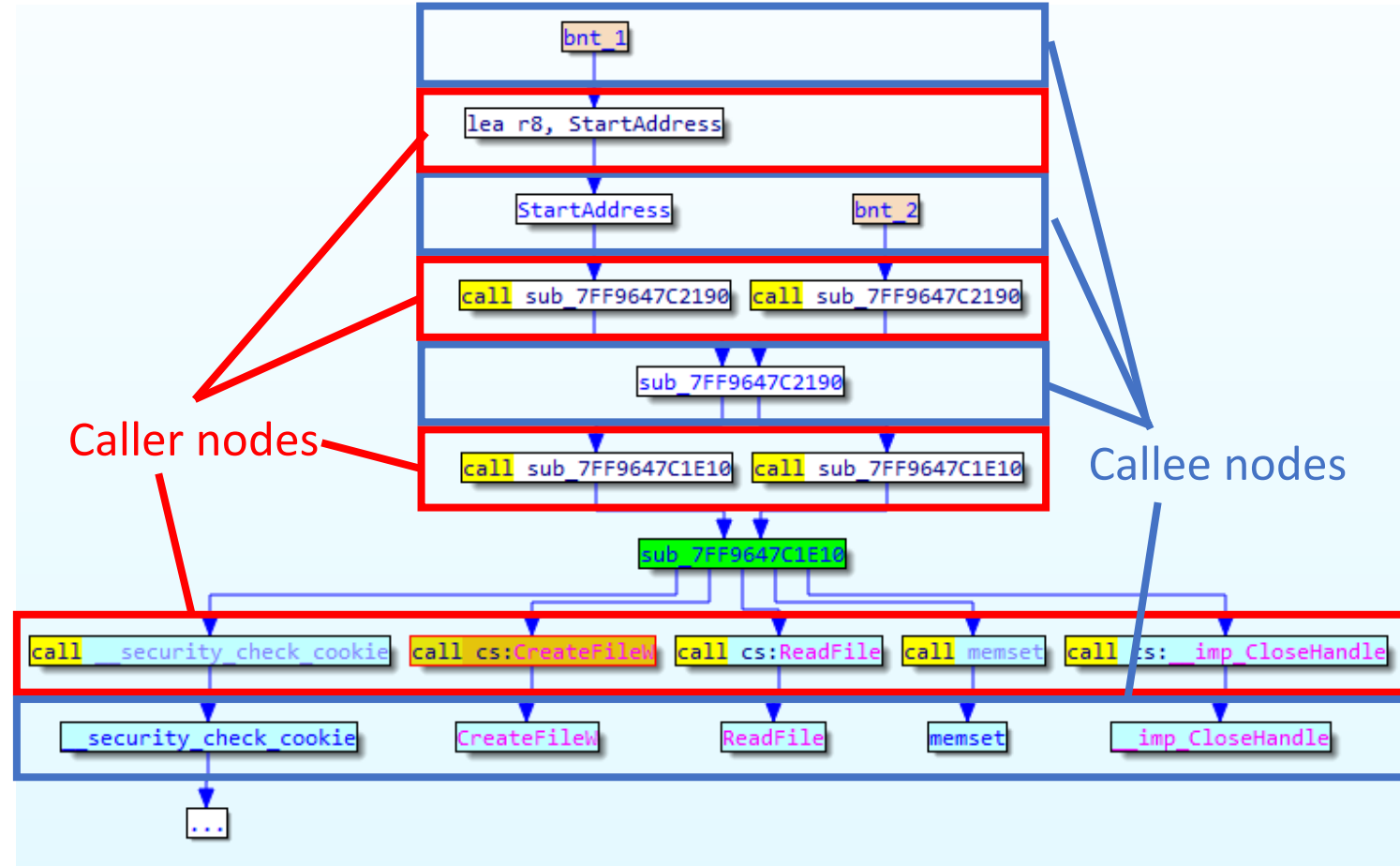
- Only function callees (function pointers) and callers (references) are displayed by default.
 - You can omit even caller nodes as well.
 - Other nodes can be displayed with a shortcut key.
- Stop tracing if CTO finds a static linked-library
- Omit child nodes in parent functions



Main Features of CTO (1)

Simple But Sufficient Tree (2)

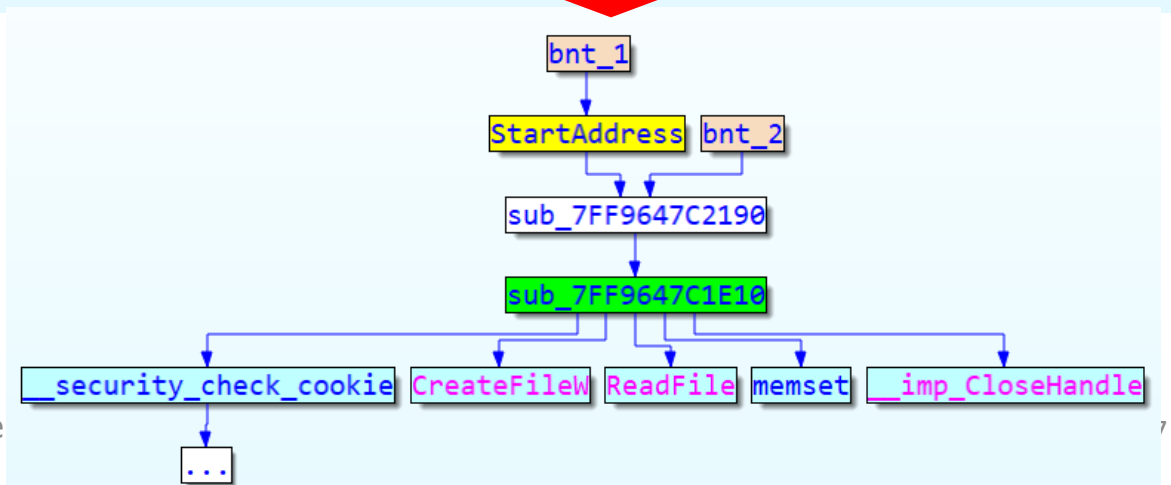
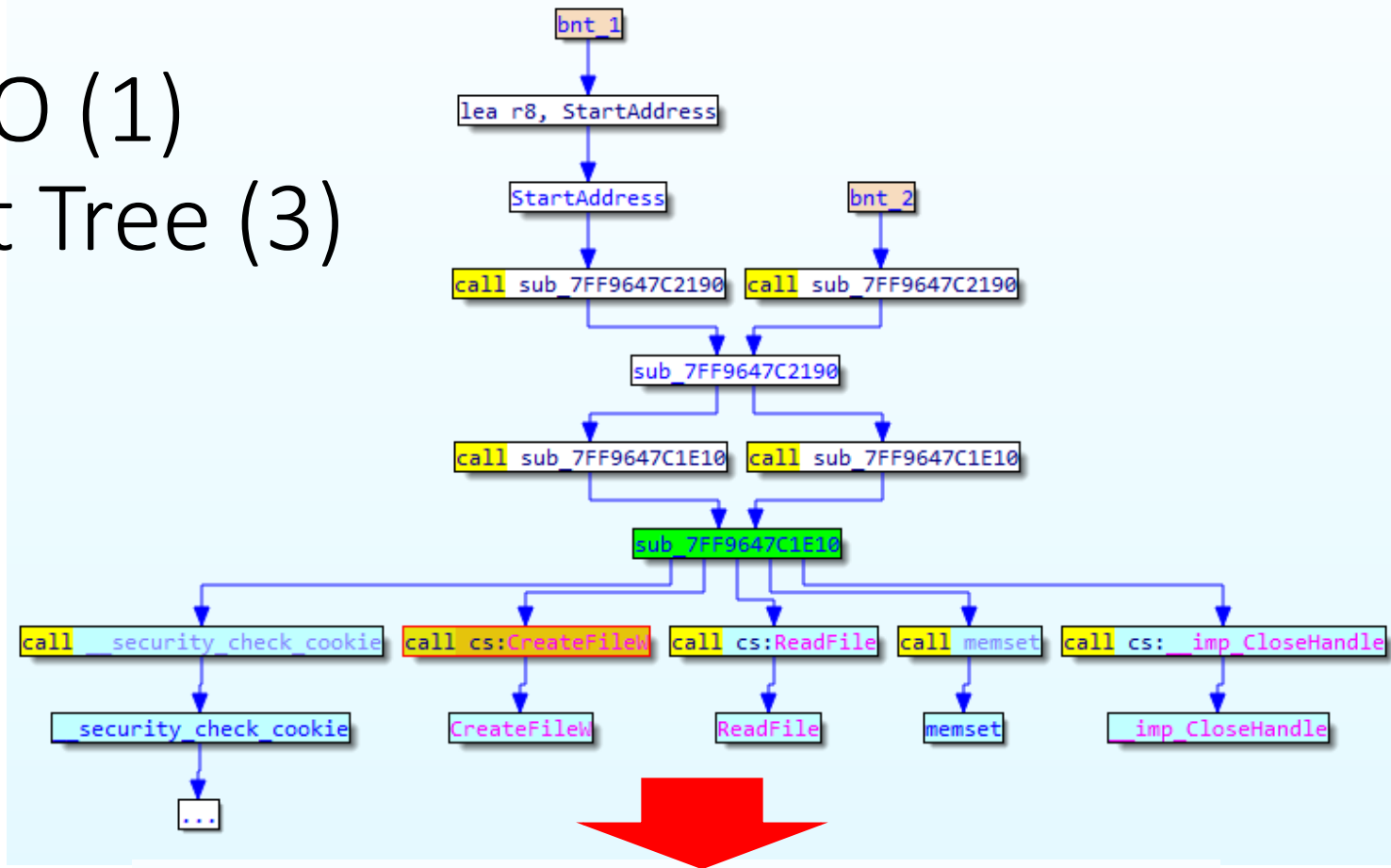
- **Only function callees (function pointers) and callers (references) are displayed by default.**
 - You can omit even caller nodes as well.
 - Other nodes can be displayed with a shortcut key.
- Stop tracing if CTO finds a static linked-library
- Omit child nodes in parent functions



Main Features of CTO (1)

Simple But Sufficient Tree (3)

- Only function callees (function pointers) and callers (references) are displayed by default.
 - **You can omit even caller nodes as well.**
 - Other nodes can be displayed with a shortcut key.
- Stop tracing if CTO finds a static linked-library
- Omit child nodes in parent functions



Main Features of CTO (1)

Simple But Sufficient Tree (4)

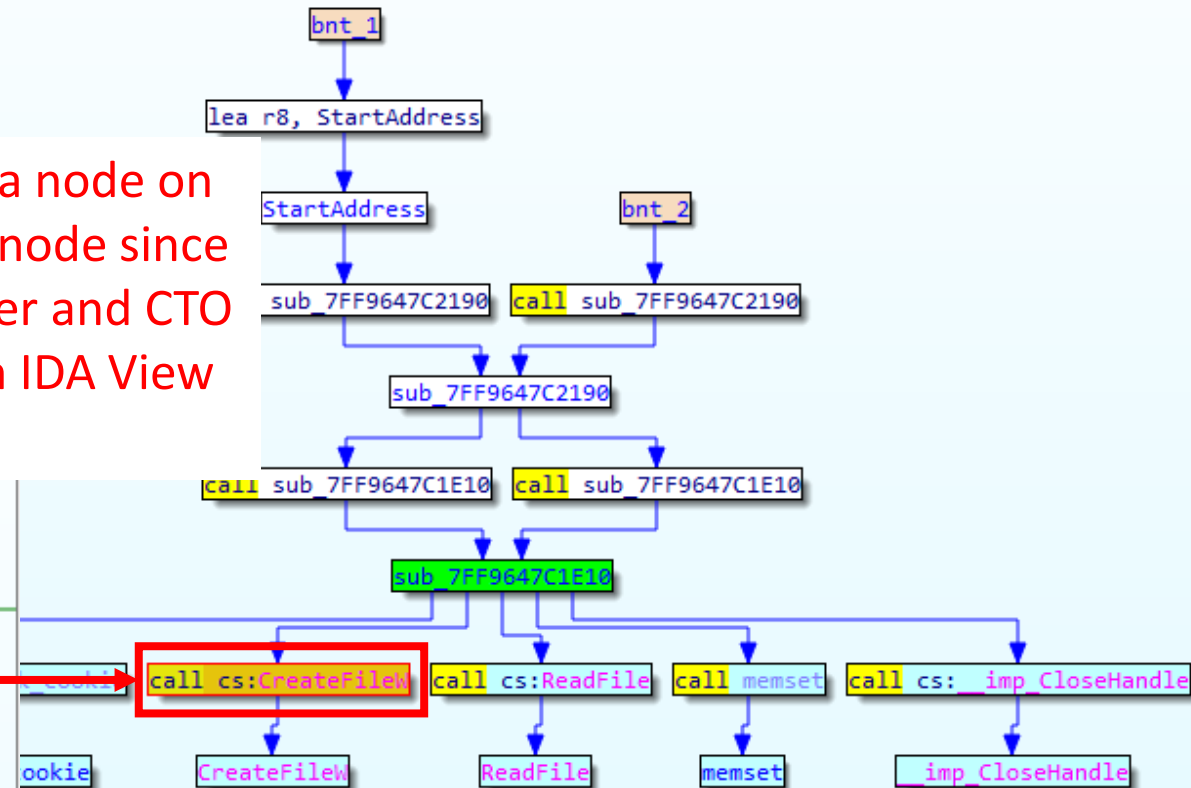
- Why do I include callers by default?

You can check code around a node on CTO when you click a caller node since this is not a callee but a caller and CTO synchronizes the address on IDA View with a node on CTO.

```
IDA View-A
mov     [rsp+1C0h+hTemplateFile], 0 ; h
xor     r9d, r9d ; lpSecurityAtt
mov     edx, 80000000h ; dwDesiredAcce
xor     r8d, r8d ; dwShareMode
mov     [rsp+1C0h+dwFlagsAndAttributes]
or      ebx, 0FFFFFFFh
mov     [rsp+1C0h+dwCreationDisposition]
call    cs:CreateFileW
mov     rdi, rax
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      loc_7FF9647C2167

lea     r9, [rsp+1C0h+NumberOfBytesRead] ; lpNumberOfBytesRead
mov     r8d, 1AA0h ; nNumberOfBytesToRead
mov     rdx, rsi ; lpBuffer
mov     rcx, rax ; hFile
mov     qword ptr [rsp+1C0h+dwCreationDisposition], 0 ; lpOverlapped
call    cs:ReadFile
test    eax, eax
jz      loc_7FF9647C215E

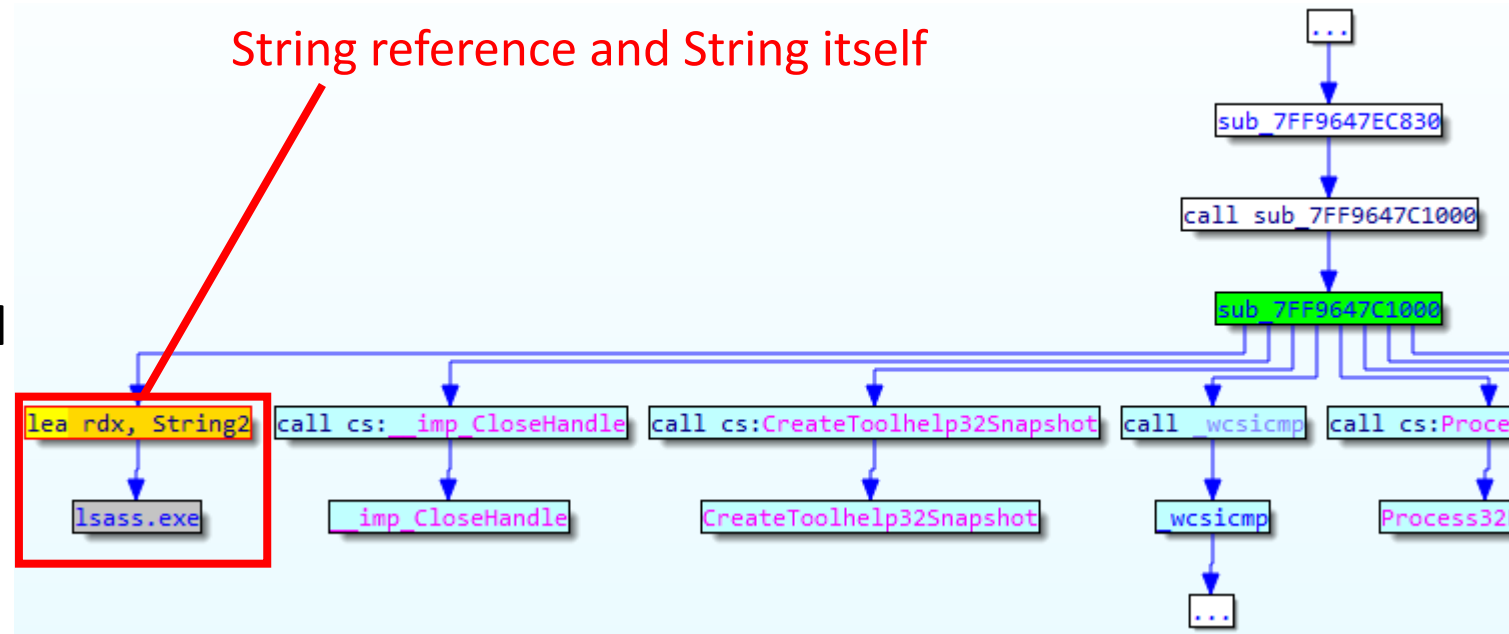
cmp     [rsp+1C0h+NumberOfBytesRead], 1AA0h
jnz     loc_7FF9647C215E
```



Main Features of CTO (1)

Simple But Sufficient Tree (5)

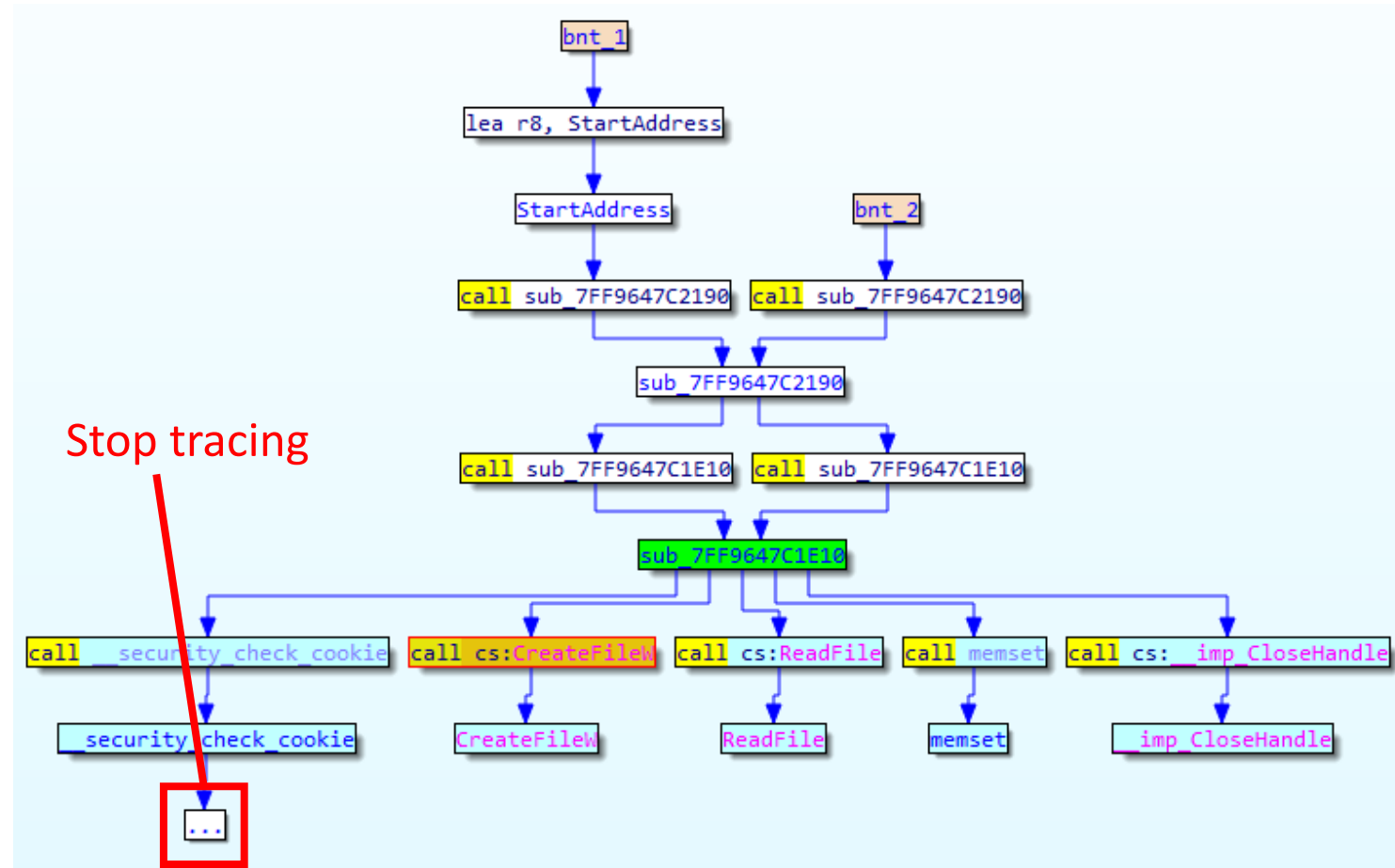
- Only function callees (function pointers) and callers (references) are displayed by default.
 - You can omit even caller nodes as well.
 - **Other nodes can be displayed with a shortcut key.**
- Stop tracing if CTO finds a static liked-library
- Omit child nodes in parent functions



Main Features of CTO (1)

Simple But Sufficient Tree (6)

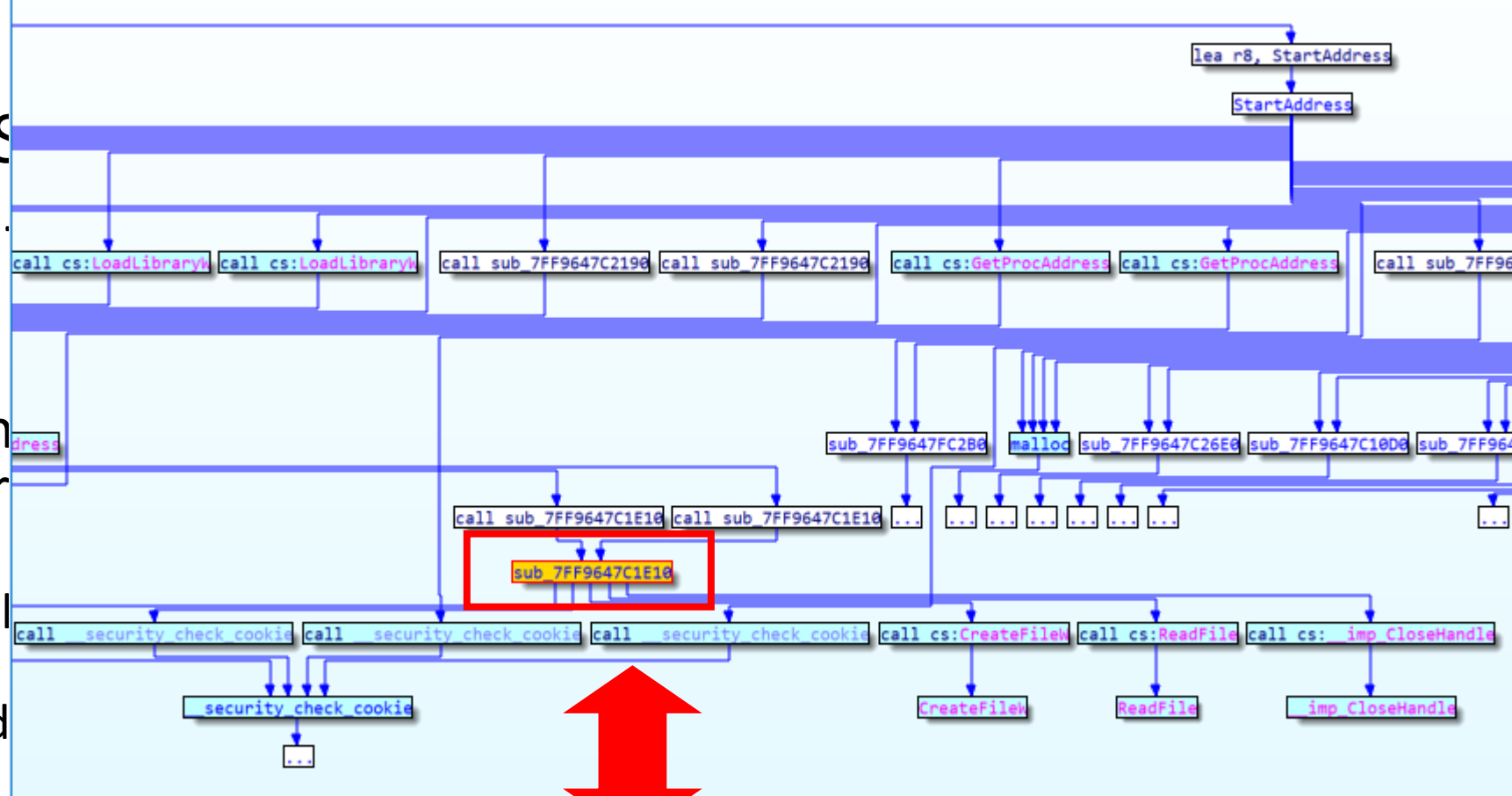
- Only function callees (function pointers) and callers (references) are displayed by default.
 - You can omit even caller nodes as well.
 - Other nodes can be displayed with a shortcut key.
- **Stop tracing if CTO finds a static liked-library**
- Omit child nodes in parent functions



Main Features

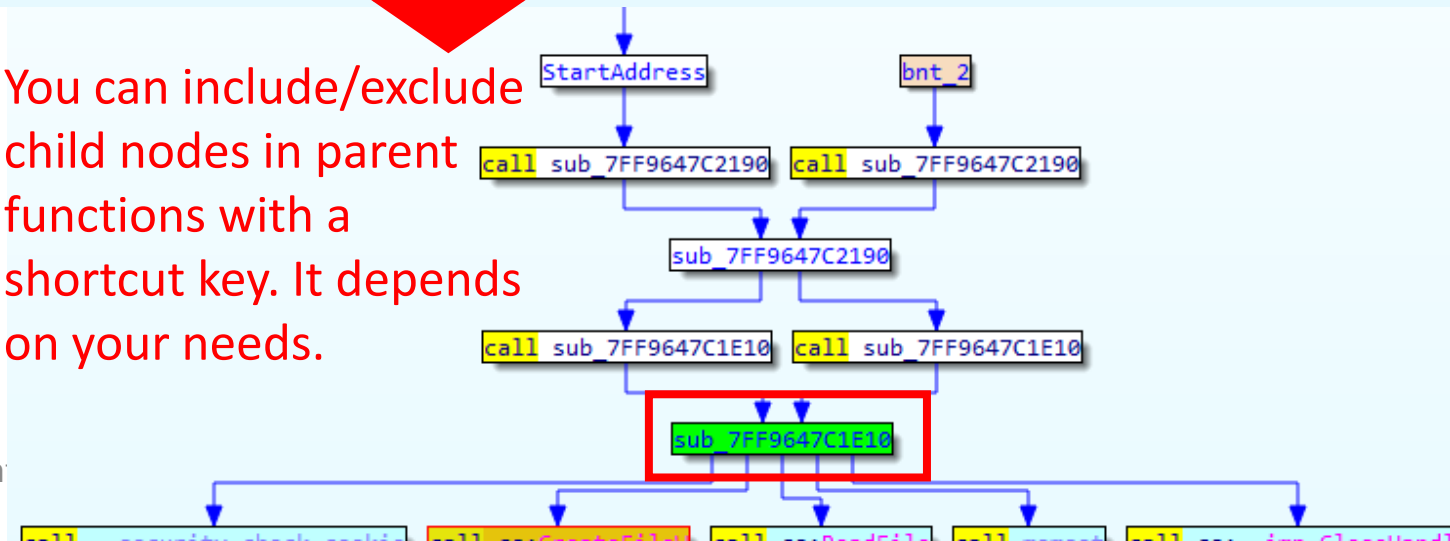
Simple But Su

- Only function callees (function pointers) and callers (references) are displayed by default.
 - You can omit even callers as well.
 - Other nodes can be displayed with a shortcut key.



- Stop tracing if CTO finds a static linked-library
- **Omit child nodes in parent functions**

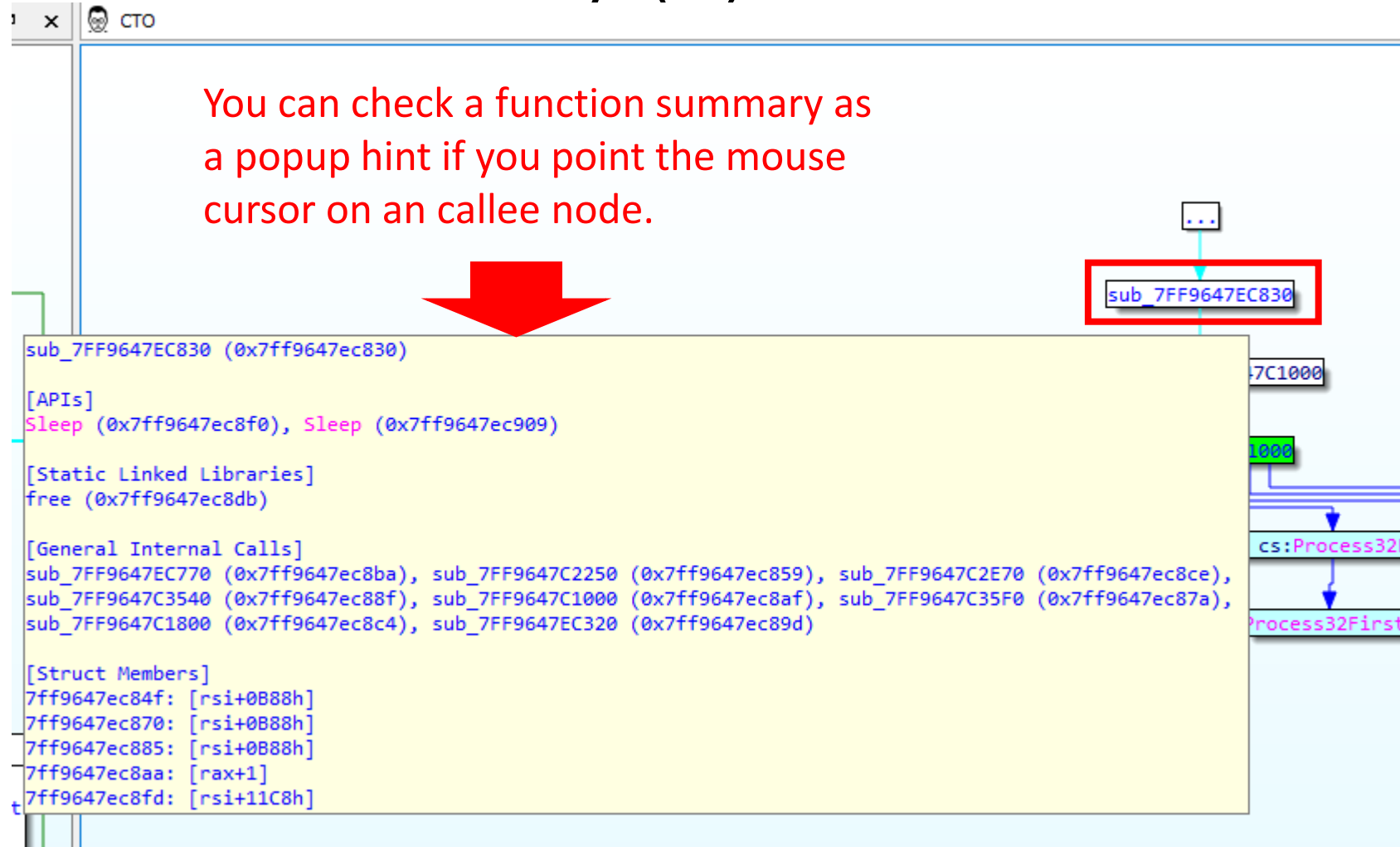
You can include/exclude child nodes in parent functions with a shortcut key. It depends on your needs.



Main Features of CTO (2)

Function Summary (1)

You can check a function summary as a popup hint if you point the mouse cursor on an callee node.



The screenshot shows a call graph with a callee node labeled `sub_7FF9647EC830` highlighted by a red box. A red arrow points from the text above to a yellow popup window that displays the function's details:

```
sub_7FF9647EC830 (0x7ff9647ec830)
[APIs]
Sleep (0x7ff9647ec8f0), Sleep (0x7ff9647ec909)
[Static Linked Libraries]
free (0x7ff9647ec8db)
[General Internal Calls]
sub_7FF9647EC770 (0x7ff9647ec8ba), sub_7FF9647C2250 (0x7ff9647ec859), sub_7FF9647C2E70 (0x7ff9647ec8ce),
sub_7FF9647C3540 (0x7ff9647ec88f), sub_7FF9647C1000 (0x7ff9647ec8af), sub_7FF9647C35F0 (0x7ff9647ec87a),
sub_7FF9647C1800 (0x7ff9647ec8c4), sub_7FF9647EC320 (0x7ff9647ec89d)
[Struct Members]
7ff9647ec84f: [rsi+0B88h]
7ff9647ec870: [rsi+0B88h]
7ff9647ec885: [rsi+0B88h]
7ff9647ec8aa: [rax+1]
7ff9647ec8fd: [rsi+11C8h]
```

sub_7FF9647EC830 (0x7ff9647ec830)

[APIs]
Sleep (0x7ff9647ec8f0), Sleep (0x7ff9647ec909)

[Static Linked Libraries]
free (0x7ff9647ec8db)

[General Internal Calls]
sub_7FF9647EC770 (0x7ff9647ec8ba), sub_7FF9647C2250 (0x7ff9647ec859), sub_7FF9647C2E70 (0x7ff9647ec8ce),
sub_7FF9647C3540 (0x7ff9647ec88f), sub_7FF9647C1000 (0x7ff9647ec8af), sub_7FF9647C35F0 (0x7ff9647ec87a),
sub_7FF9647C1800 (0x7ff9647ec8c4), sub_7FF9647EC320 (0x7ff9647ec89d)

[Struct Members]
7ff9647ec84f: [rsi+0B88h]
7ff9647ec870: [rsi+0B88h]
7ff9647ec885: [rsi+0B88h]
7ff9647ec8aa: [rax+1]
7ff9647ec8fd: [rsi+11C8h]

sub_7FF9647EC830 (0x7ff9647ec830)

[APIs]
Sleep (0x7ff9647ec8f0), Sleep (0x7ff9647ec909)

[Static Linked Libraries]
free (0x7ff9647ec8db)

[General Internal Calls]
sub_7FF9647EC770 (0x7ff9647ec8ba), sub_7FF9647C2250 (0x7ff9647ec859), sub_7FF9647C2E70 (0x7ff9647ec8ce),
sub_7FF9647C3540 (0x7ff9647ec88f), sub_7FF9647C1000 (0x7ff9647ec8af), sub_7FF9647C35F0 (0x7ff9647ec87a),
sub_7FF9647C1800 (0x7ff9647ec8c4), sub_7FF9647EC320 (0x7ff9647ec89d)

[Struct Members]
7ff9647ec84f: [rsi+0B88h]
7ff9647ec870: [rsi+0B88h]
7ff9647ec885: [rsi+0B88h]
7ff9647ec8aa: [rax+1]
7ff9647ec8fd: [rsi+11C8h]



Output

sub_7FF9647EC830 (0x7ff9647ec830)

[APIs]
Sleep (0x7ff9647ec8f0), Sleep (0x7ff9647ec909)

[Static Linked Libraries]
free (0x7ff9647ec8db)

[General Internal Calls]
sub_7FF9647EC770 (0x7ff9647ec8ba), sub_7FF9647C2250 (0x7ff9647ec859), sub_7FF9647C2E70 (0x7ff9647ec8ce),
sub_7FF9647C3540 (0x7ff9647ec88f), sub_7FF9647C1000 (0x7ff9647ec8af), sub_7FF9647C35F0 (0x7ff9647ec87a),
sub_7FF9647C1800 (0x7ff9647ec8c4), sub_7FF9647EC320 (0x7ff9647ec89d)

[Struct Members]
7ff9647ec84f: [rsi+0B88h]
7ff9647ec870: [rsi+0B88h]
7ff9647ec885: [rsi+0B88h]
7ff9647ec8aa: [rax+1]
7ff9647ec8fd: [rsi+11C8h]

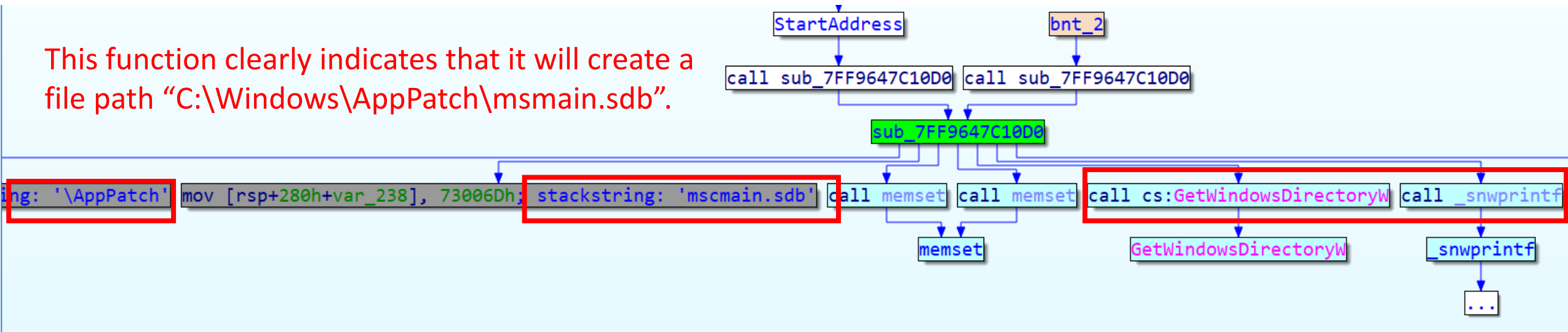
If you want to access the information, you can use "Print hint feature". It dumps the same information to the output window with a shortcut key.

Main Features of CTO (3)

3rd Party Tool Corroboration

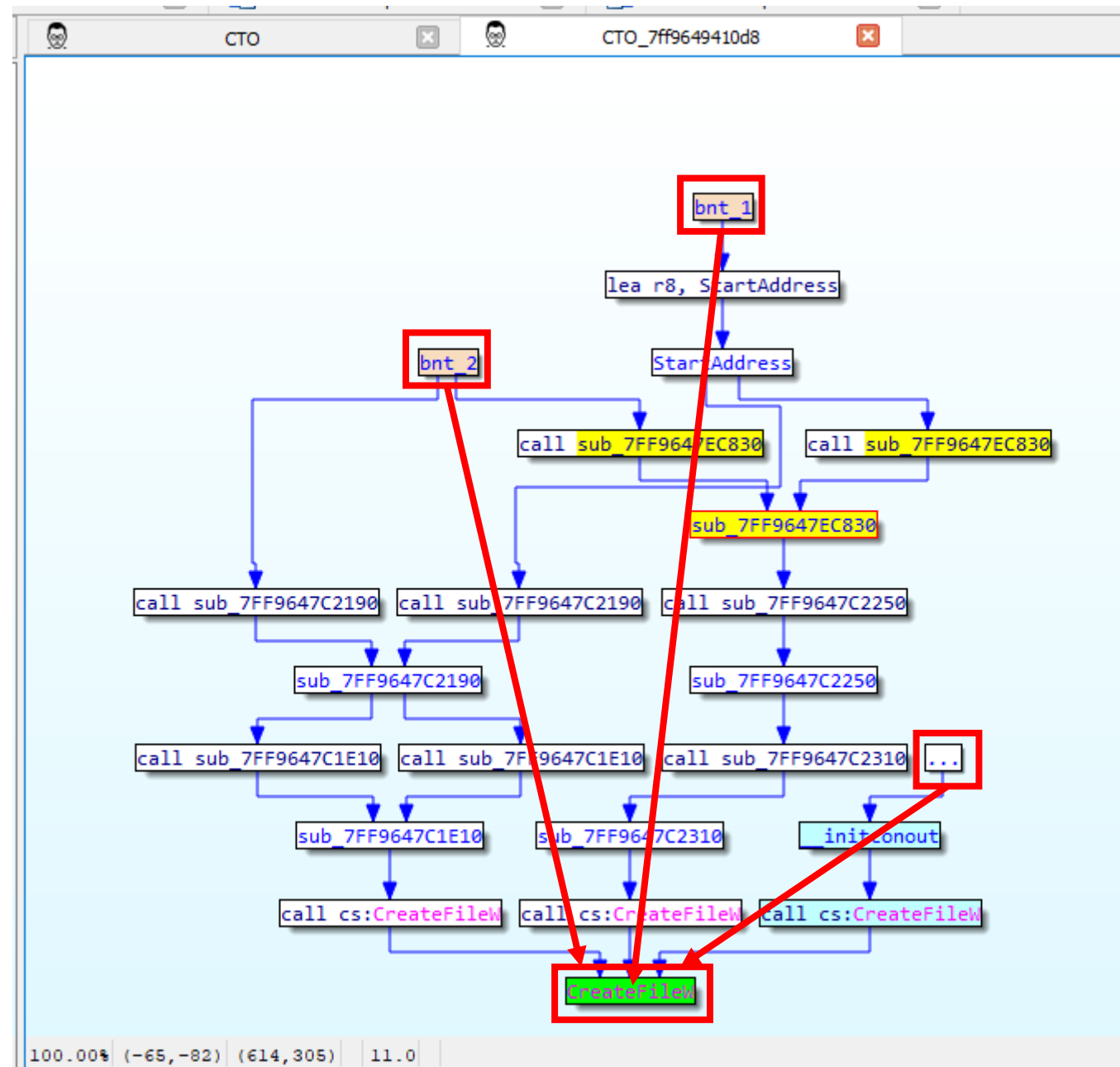
- CTO collects repeatable comments and specific regular comments, which are outputted from several tools. They are also useful to identify functions' roll.

This function clearly indicates that it will create a file path "C:\Windows\AppPatch\msmain.sdb".



Main Features of CTO (4) Find Paths

- You can find paths to/from a node including an API pointer on the import table.
- This figure on the right is an example of the result of “Find Paths” feature for CreateFileW.
 - You can easily understand there are three paths and one is from inside a static linked library and the others are from two entry points.



Main Features of CTO (5)

IDA's Shortcuts Redirection

- Even if you focus the CTO window, you can still use several IDA's shortcuts such as renaming a function/variable (N), finding xrefs (x), commenting (: or ;) since CTO redirects them to IDA.

The screenshot shows the IDA Pro interface with the CTO (Control Transfer Object) window active. The CTO window displays a call instruction: `call sub_7FF9647C1800`. A red arrow points from this instruction to a dialog box titled "xrefs to sub_7FF9647C1800". The dialog box contains a table with the following data:

Direction	Type	Address	Text
Do...	p	sub_7FF9647EC830+94	call sub_7FF9647C1800

Below the table, the dialog box shows "Line 1 of 1" and buttons for "OK", "Cancel", "Search", and "Help".

In the background, the CTO window shows a flowchart with nodes: `sub_7FF9647EC830`, `call sub_7FF9647C1800`, and `sub_7FF9647C1800`. The `sub_7FF9647C1800` node is highlighted with a red box, and a red arrow points from it to the dialog box.

If you press "x" on the node, you will see the xrefs like this.

Main Features of CTO (6)

Applying Function Definition to an Indirect Call

```
mov     edx, edi           ; uBytes
call    cs:LocalAlloc
mov     r9d, edi
mov     rdx, r12
mov     rcx, rsi
mov     r8, rax
mov     qword ptr [rsp+3D0h+!IPProtect], 0
mov     rbx, rax
call    r14
```

Before

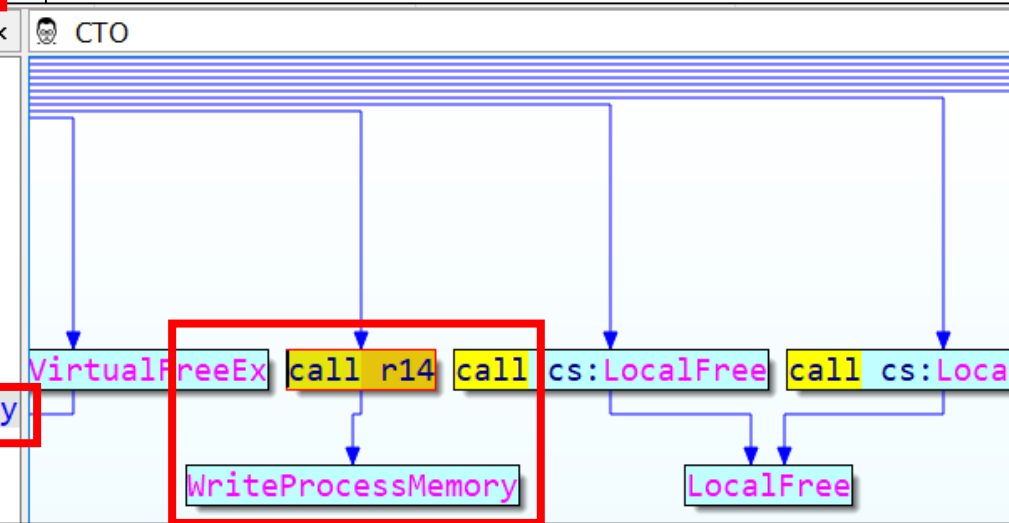
If you set an API name as a comment on an unresolved indirect call, CTO will set its arguments on IDA View.

These two figures show us before and after putting a comment on a call instruction. You can see IDA recognized the arguments after setting the comment on the figure below.

```
ecx, 40n ; @ ; uFlags
edx, edi ; uBytes
call    cs:LocalAlloc
mov     r9d, edi           ; nSize
mov     rdx, r12           ; lpBaseAddress
mov     rcx, rsi           ; hProcess
mov     r8, rax            ; lpBuffer
mov     qword ptr [rsp+3D0h+!IPProtect], 0 ;
mov     rbx, rax
call    r14                ; WriteProcessMemory
mov     r9d, 8000h         ; dwFreeType
xor     r8d, r8d           ; dwSize
```

After

Setting a comment



How CTO works - Inside CTO

How CTO works - Inside CTO (1)

Core Structures of CTO

- In order to synchronize with IDA View, CTO utilizes two hooks.
 - UI hooks (**UI_Hooks** class)
 - View hooks (**View_Hooks** class)
- UI hooks and view hooks are in **ida_kernwin.py**.
- CTO also inherits graph viewer class (**GraphViewer** class) in **ida_graph.py**.
- Since IDA's GUI related APIs are in these two modules, if you want to create a GUI-based IDA plugin, you should look into them first.

How CTO works - Inside CTO (2)

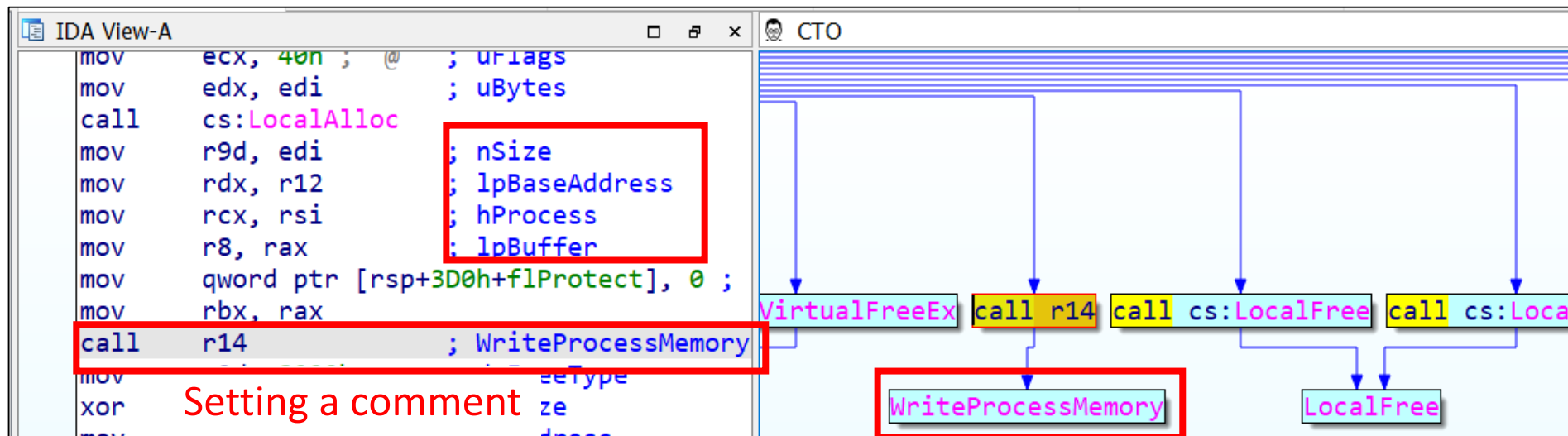
UI_Hooks Class (1)

- You can catch all UI events, which are called “actions” by Hex-Rays, by inheriting this class.
 - For example, the “MakeName” action will be issued if you press “N” key on a variable name.
 - You can check all defined actions by executing `get_registered_actions()` API on `ida_kernwin.py`.
- CTO inherits this class to update node information on CTO by overwriting these methods (See `sync_ui.py`).
 - `preprocess_action()`
 - `postprocess_action()`
 - `updating_actions()`

How CTO works - Inside CTO (3)

UI_Hooks Class (2)

- For example, if you set an API name as a comment, CTO will create an additional node on CTO. And CTO will also make IDA to set its arguments as comments. This feature is implemented by catching the “MakeComment” event.



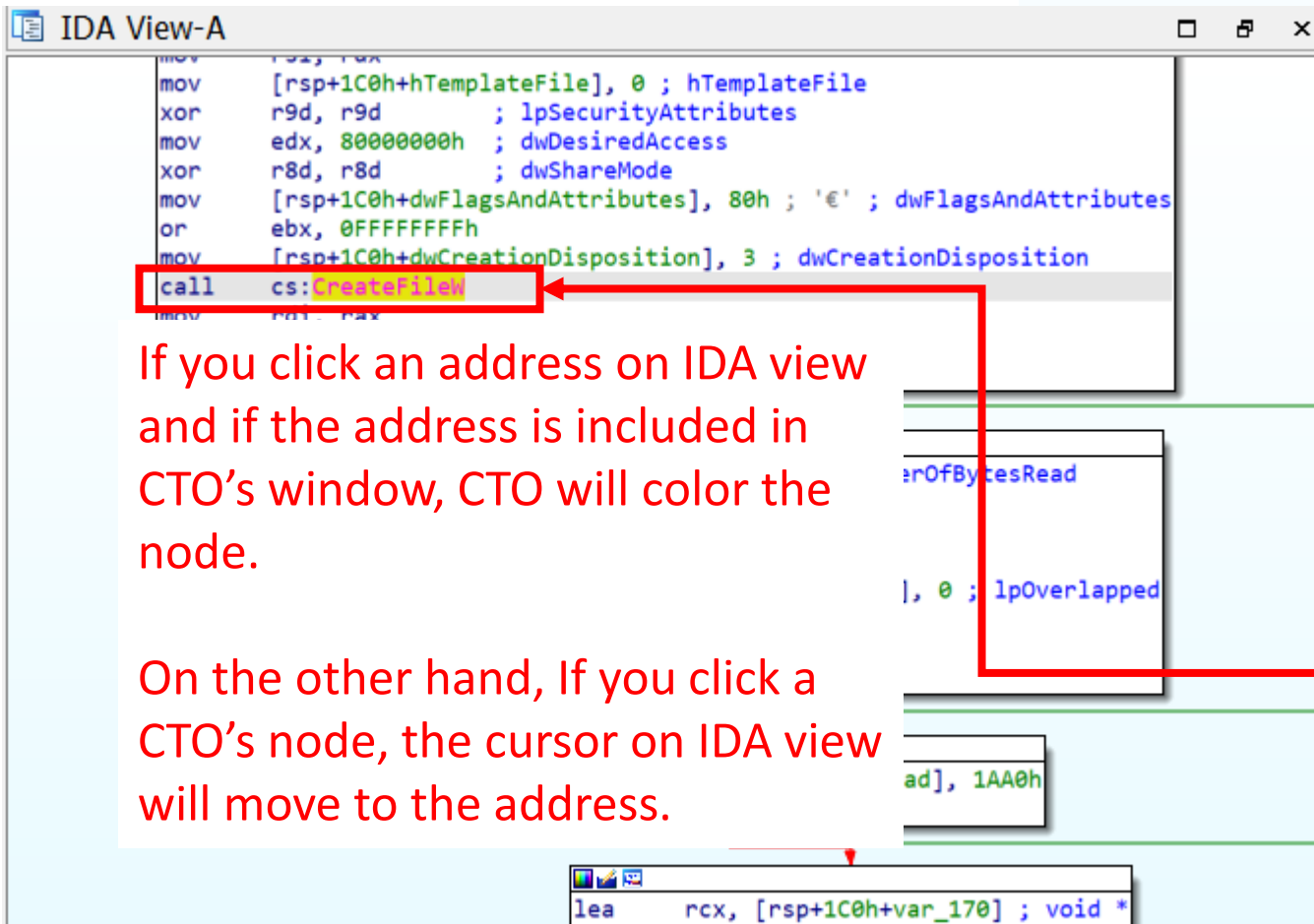
How CTO works - Inside CTO (4)

View_Hooks Class (1)

- CTO utilizes the “**View_Hooks**” class to synchronize a CTO’s node with the address in IDA View.
- CTO observes location change events by overwriting **view_loc_changed()** method in the class. If the address on IDA View is changed, CTO colors the corresponded node on CTO. On the other hand, if a different node on CTO is selected, CTO changes the location on IDA View with `jumpto()` API.

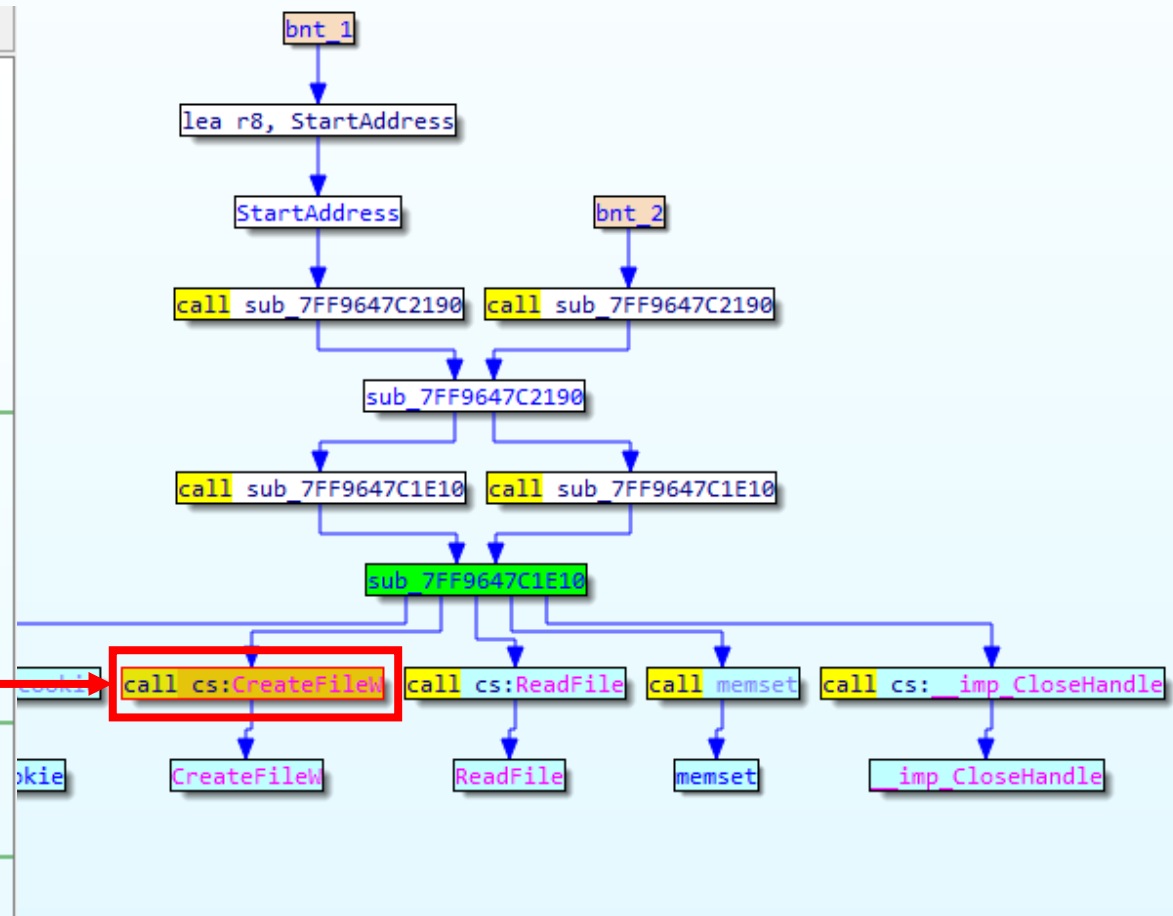
How CTO works - Inside CTO (5)

View_Hooks Class (2)



If you click an address on IDA view and if the address is included in CTO's window, CTO will color the node.

On the other hand, If you click a CTO's node, the cursor on IDA view will move to the address.



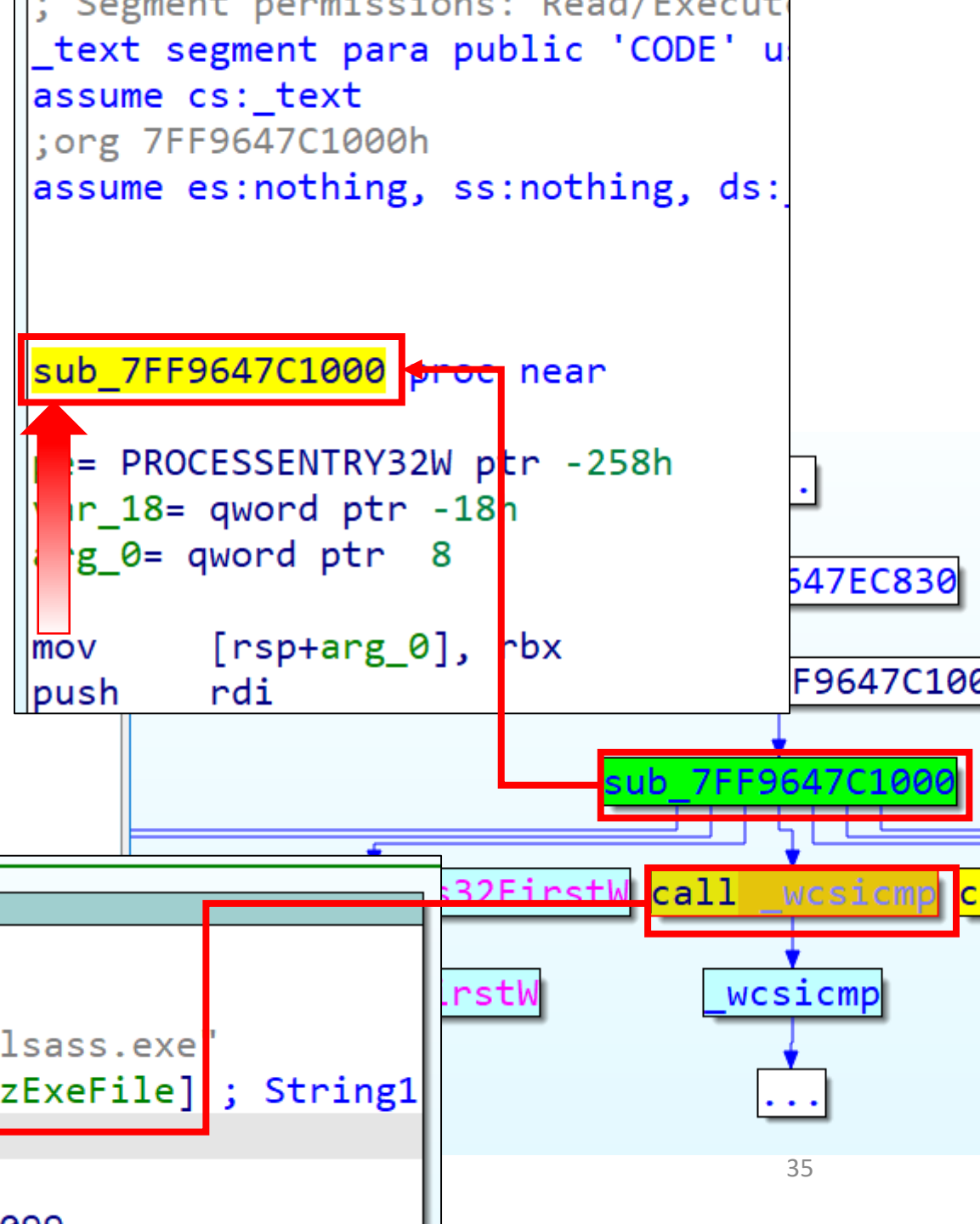
How CTO works - Inside CTO (6)

GraphViewer Class

- In order to create a custom graph like CTO, inheriting the GraphViewer class is the simplest way. You can create a graph by overwriting the **AddNode()** method to add a box called a “node”, and the **AddEdge()** method to add an arrow connector called an “edge”.
- You can also hook many events on your widget as well. For example,
 - Keyboard events
 - OnViewKeyDown()
 - Mouse events
 - OnClick() for click events
 - OnDbClick() for double-click events
 - OnPopup() for right-click events
 - OnHint() for on-mouse-over events for nodes
 - OnEdgeHint() for on-mouse-over events for edges

How CTO works - Inside CTO (7) jumpto API (1)

- CTO tweaks the cursor position on IDA View when a node is clicked.
- For example, if a callee node is clicked, CTO will change the cursor position to a function name, not the first instruction.
- On the other hand, a caller node is clicked, CTO will change the position to a variable name in a instruction, not the head of the instruction.
- It's necessary if a user wants to rename the function.



```
loc_7FF9647C1070:  
lea    rdx, String2    ; "lsass.exe"  
lea    rcx, [rsp+278h+pe.szExeFile] ; String1  
call   _wcsicmp  
test   eax, eax  
jz     short loc_7FF9647C1000
```

How CTO works - Inside CTO (8)

jumpto API (2)

- In order to tweak the cursor position, CTO utilizes two variants of jumpto() APIs. According to ida_kernwin.py, there are two definitions of jumpto() APIs.

```
jumpto(ea, opnum=-1, uijmp_flags=0x0001) -> bool  
jumpto(custom_viewer, place, x, y) -> bool
```

- The former definition is used for jumping into in the middle of an instruction on the given address. If you pass the second argument “opnum”, that is an operand number, you can easily tweak the horizontal position of the cursor.
- If you want to move the cursor vertically and/or horizontally, you can use the second definition of the API.

Closing Remarks

Summary



- CTO is an IDA Pro plugin for visualizing function call tree.

IDA Pro interface showing the CTO plugin. The left pane displays the function list, the middle pane shows assembly code, and the right pane shows a call tree graph. The graph illustrates the function call structure, with the root node 'sub_7FF9647C1E10' highlighted in green. The graph shows the function calling 'security_check_cookie', 'CreateFileW', 'ReadFile', 'memset', and '_imp_CloseHandle'.

```
graph TD
    bnt1[bnt 1] --> StartAddress[StartAddress]
    StartAddress --> call1[call sub_7FF9647C2190]
    StartAddress --> call2[call sub_7FF9647C2190]
    call1 --> sub_7FF9647C2190[sub_7FF9647C2190]
    call2 --> sub_7FF9647C2190
    sub_7FF9647C2190 --> call3[call sub_7FF9647C1E10]
    sub_7FF9647C2190 --> call4[call sub_7FF9647C1E10]
    call3 --> sub_7FF9647C1E10
    call4 --> sub_7FF9647C1E10
    sub_7FF9647C1E10 --> call5[call security_check_cookie]
    sub_7FF9647C1E10 --> call6[call cs:CreateFileW]
    sub_7FF9647C1E10 --> call7[call cs:ReadFile]
    sub_7FF9647C1E10 --> call8[call memset]
    sub_7FF9647C1E10 --> call9[call cs:_imp_CloseHandle]
    call5 --> security_check_cookie[security_check_cookie]
    call6 --> CreateFileW[CreateFileW]
    call7 --> ReadFile[ReadFile]
    call8 --> memset[memset]
    call9 --> _imp_CloseHandle[_imp_CloseHandle]
```

Future Work

- Collaborating with some more tools
- Collecting more instructions to be observed
- Implementing more efficient way to collect paths
- Enhancing the speed
- Improving stability

FAQ

- Where will you disclose the source code?
 - I will do it on the following URL.
 - <https://github.com/herosi/CTO>
- Will you create CTO for Ghidra or other RE tools?
 - No, I won't. I'm already an IDA Pro user. You can port it to Ghidra or other tools since I will disclose the source code.

Thank you for
watching my presentation!

@herosi_t