Showing differences between disassembled functions

Aureliano Calvo*
Core Security Technologies
ITBA phD program

ABSTRACT

In this paper we describe how to show the differences between two basic-block graphs in a single graph using similar conventions as the ones used to show differences in text. All the previously known tools show two graphs side-by-side with the old and new graphs and some information on how to match the blocks in the original graph and the blocks in the new one.

1. INTRODUCTION

Penetration testing software (such as Core Impact, Immunity Canvas and Metasploit) need to be kept up up to date with the new public security vulnerabilities published by different vendors. But the information published is usually deficient and a reverse engineering effort is needed to be able to actually exploit the new vulnerability.

Exploiting the vulnerabilities is the task of the exploit writing team. When a vendor releases a new version of a product that solves a security problem they begin their work. Their first task is to understand how to exploit such vulnerability using information of the exploit (usually published as an advisory) together with the old and new versions of the vulnerable software.

1.1 Current state of the art tools

Extracting information from the old and new software is not an easy task. In order to do that they may use several tools (such as BinDiff [1], PatchDiff[4] or TurboDiff [7]) to compare the functions changed between the two versions by matching basic blocks[1] that are shared between the old and new version and showing changes within these blocks and the blocks that are not matched.

These tools share a similar approach to show differences. The old and new graphs are shown side by side and annotations are made when a difference is found (see figures 2 and 3). BinDiff goes one step further and highlights the matching block on the other graph.

Showing the differences in this way has several drawbacks. Jumping between the old and new graph is required to analyze its differences and if the layout is not similar it even forces the user to look for a basic block in the other graph in a sequential manner. Another problem is that it uses extra space, showing twice the matched elements.

2. DESIGN PRINCIPLES

Our solution addresses these drawbacks. Instead of showing each graph in a separated pane, it merges the old and the new graph in a merged graph and denotes the changes using a uniform convention. While this approach is used in basic-blocks graphs, we believe that the general idea behind this solution can be applied to other kinds of graphs (such as, for example, network graphs and attack graphs).

We also strive to use preattentive attributes [8, 9] to show the graph differences in order to aid the analysis. To achieve this goal, a single convention to note all the types of differences is needed. The differences can be assembler code inside a node, nodes added or deleted and changes in the graph edges.

3. OUR SOLUTION

The merged graph has three different types of basic blocks:

Matched: A matched block appears both in the old and in the new graph.

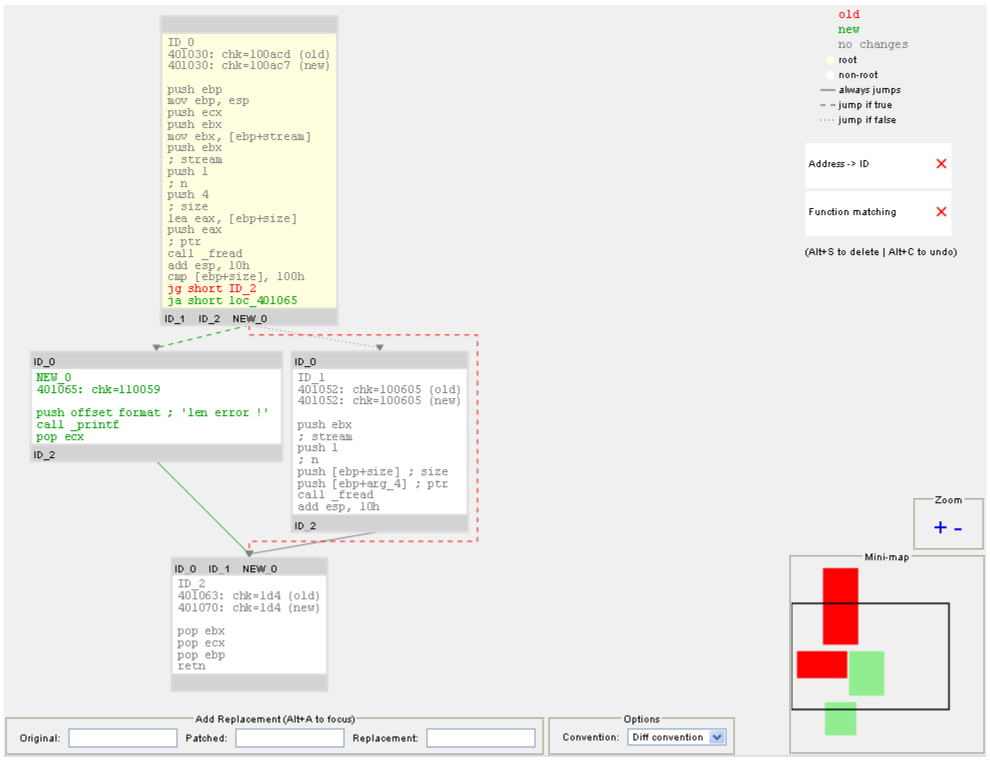
Old: An old block appears in the old basic-block graph but does not match any basic block in the new graph.

New: A new block appears in the new basic-block graph but does not match any basic block in the old graph.

The edges of the merged graph correspond to the edges of the old and new graph and also have three types:

Matched: A matched edge represents an edge in the old graph and a an edge in the new graph such as that they both go from the same matched node to the same matched node and have the same type (always, then, else)

*aurelianocalvo@coresecurity.com
[1]A basic block is a sequence of instructions that has a single entry point and does not branch [6].


```
ID_0
401030: chk=100acd (old)
401030: chk=100ac7 (new)

push ebp
mov ebp, esp
push ecx
push ebx
mov ebx, [ebp+stream]
push ebx
; stream
push 1
; n
push 4
; size
lea eax, [ebp+size]
push eax
; ptr
call _fread
add esp, 10h
cmp [ebp+size], 100h
jg short ID_2
ja short loc_401065
```
ID_1 ID_2 NEW_0

ID_0
```
NEW_0
401065: chk=110059

push offset format ; 'len error !'
call _printf
pop ecx
```
ID_2

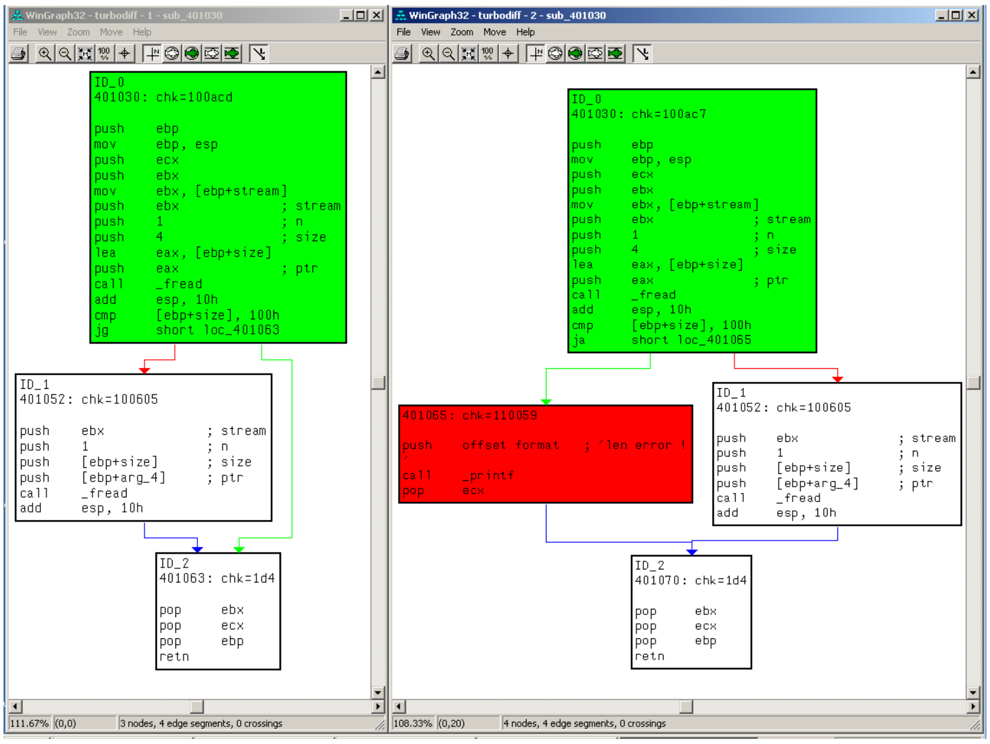
ID_0
```
ID_1
401052: chk=100605 (old)
401052: chk=100605 (new)

push ebx
; stream
push 1
; n
push [ebp+size] ; size
push [ebp+arg_4] ; ptr
call _fread
add esp, 10h
```
ID_2

ID_0 ID_1 NEW_0
```
ID_2
401063: chk=1d4 (old)
401070: chk=1d4 (new)

pop ebx
pop ecx
pop ebp
retn
```

Figure 1: Our implementation shows the differences between two versions of the same function in a single graph


WinGraph32 - turbodiff - 1 - sub_401030
```
ID_0
401030: chk=100acd

push     ebp
mov      ebp, esp
push     ecx
push     ebx
mov      ebx, [ebp+stream]
push     ebx                  ; stream
push     1                    ; n
push     4                    ; size
lea      eax, [ebp+size]
push     eax                  ; ptr
call     _fread
add      esp, 10h
cmp      [ebp+size], 100h
jg       short loc_401063
```
```
ID_1
401052: chk=100605

push     ebx                  ; stream
push     1                    ; n
push     [ebp+size]           ; size
push     [ebp+arg_4]          ; ptr
call     _fread
add      esp, 10h
```
```
ID_2
401063: chk=1d4

pop      ebx
pop      ecx
pop      ebp
retn
```
111.67% (0,0) 3 nodes, 4 edge segments, 0 crossings

WinGraph32 - turbodiff - 2 - sub_401030
```
ID_0
401030: chk=100ac7

push     ebp
mov      ebp, esp
push     ecx
push     ebx
mov      ebx, [ebp+stream]
push     ebx                  ; stream
push     1                    ; n
push     4                    ; size
lea      eax, [ebp+size]
push     eax                  ; ptr
call     _fread
add      esp, 10h
cmp      [ebp+size], 100h
ja       short loc_401065
```
```
401065: chk=110059

push     offset format    ; 'len error !
'
call     _printf
pop      ecx
```
```
ID_1
401052: chk=100605

push     ebx                  ; stream
push     1                    ; n
push     [ebp+size]           ; size
push     [ebp+arg_4]          ; ptr
call     _fread
add      esp, 10h
```
```
ID_2
401070: chk=1d4

pop      ebx
pop      ecx
pop      ebp
retn
```
108.33% (0,20) 4 nodes, 4 edge segments, 0 crossings

Figure 2: Turbodiff with wingraph showing the differences between two versions of the same function in two different graphs

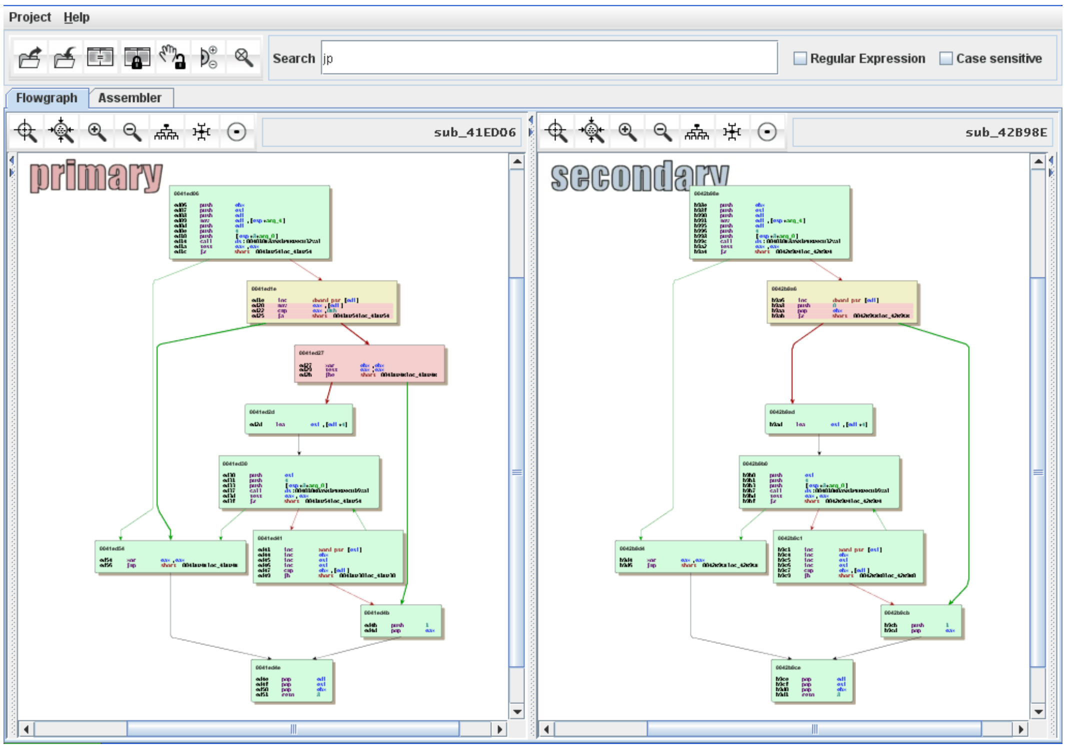
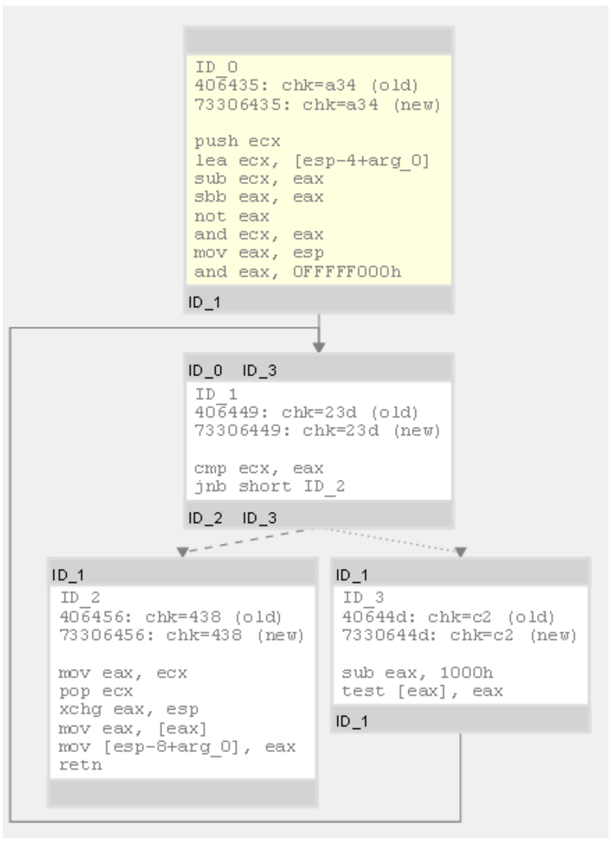


Figure 3: Bindiff shows the differences between two versions of the same function using two different graphs



```
ID_0
406435: chk=a34 (old)
73306435: chk=a34 (new)

push ecx
lea ecx, [esp-4+arg_0]
sub ecx, eax
sbb eax, eax
not eax
and ecx, eax
mov eax, esp
and eax, 0FFFFF000h
```
```
ID_1
406449: chk=23d (old)
73306449: chk=23d (new)

cmp ecx, eax
jnb short ID_2
```
```
ID_2
406456: chk=438 (old)
73306456: chk=438 (new)

mov eax, ecx
pop ecx
xchg eax, esp
mov eax, [eax]
mov [esp-8+arg_0], eax
retn
```
```
ID_3
40644d: chk=c2 (old)
7330644d: chk=c2 (new)

sub eax, 1000h
test [eax], eax
```

Figure 4: A loop can be easily seen because the upwards arrows go on the left.

Old: An old edge is an edge in the old graph that can not be matched with an edge in the new graph.

New: A new edge is an edge in the new graph that can not be matched with an edge in the old graph.

The merged graph is composed of all the nodes and edges defined in this section.

We have chosen to show the differences between nodes by using colors. Old information (things that appear in the first graph exclusively) is shown in red whereas new elements are shown in green. Matching information is shown in grey. This convention is also used to show differences within a matching basic block. This color selection is chosen because the diff between source code shown as text is displayed with these colors in several tools[2].

Different kinds of edges in the merged graphs are shown using the line texture. If the conditional jump is taken it is shown with a dashed line. Given it is not taken a dotted line is drawn instead. In case of an unconditional jump or the basic blocks ends without a jump instruction and it does not return to the calling function it is shown with a solid line. The line texture was chosen because we need to note three different options on lines, these lines can have any orientation and the colors are taken. Jumps are also colored with green, red or grey to show if the jump is in the old graph, the new graph or both.

[2]Trac and Eclipse are two tools that use this color convention.

The graph layout chosen for the merged graph attempts to mimic the layout used in turbodiff. The merged graph is walked in DFS order starting from the root nodes of the old and new basic-block graphs[3] and the induced tree is used to generate the layout. The graph layout algorithm chosen separates the nodes in levels of a tree and then each level is drawn under the previous one.

Edges that go from level n to level $n+1$ are drawn as straight lines and the rest of the edges are drawn following Manhattan paths[4]. The edges that go upward are drawn at the left of the graph, whereas the lines that go downward are located at the right, making it easier to detect loops in the function (see figure 4).

We used a yellow background to show the root nodes of the old and new graphs.

Memory addresses of the matching nodes that appear on top of each basic block are not shown in red and green when they change because these differences are not important in order to analyze the changes of a function. Instead of it they are shown in grey and noted. This change was requested by the tool users.

When a memory address matches the start address of a basic block, the address that appears in the assembler instructions is replaced with the matching node id. This change was also requested by the tool users.

Another request made by the exploit writers was to be able to tell the tool that something that is textually different in the old and new graphs is actually the same thing in both. For instance, references to EBP+constant (local variables) or different registers may be changes introduced by the compiler that are not interesting for them. In order to fulfill this request, we introduced a general search and replace tool that looks for some text in the old graph, another text in the new graph and replaces both of them with a new label.

Another enhancement to the reverse engineer is the minimap, located on the lower right corner. It was added to keep a high level view of the code. Also, several navigation tools were added: pan, zoom and ids links on top and bottom of the basic blocks. Those are used to jump to the connected basic block with the corresponding id.

4. EXAMPLES

Figures 1 and 2 show how a simple change is shown using both the usual approach and the one proposed in this paper. In figure 1 red entities represent things that where in the original graph but not in the new one, green entities represent new things and grey entities are things that are matched. It can be seen that showing changes in the same graph saves space and it is easier to see the changes in the initial basic block.

We can also see that the problem stated in the introduction is solved and that design principles presented in section 2 are respected. In figure 1 a single graph is shown, the differences are displayed using colors and this convention is used inside the nodes and in the edges. Because color is a preattentive attribute it is very easy to see the differences in the function structure and it is easy to see the difference context. By contrast, in figure 2 it is difficult to see differences inside a basic block and also it is more difficult to understand the changes in the graph structure.

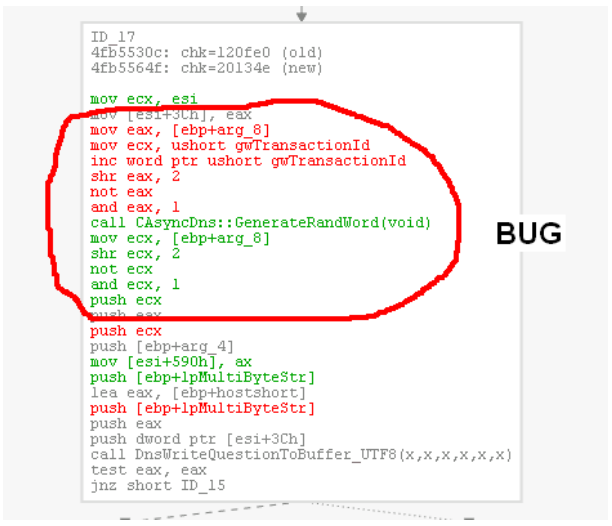
The original C code of the function being diffed can be seen in figure 5.

The other example is a real world usage. Figure 6 shows a "silent patch" introduced by the Microsoft Bulletin MS10-024[5] . It can be clearly seen that a call to the function CAsyncDns::GenerateRandWord was added changing the transaction id from a sequential number to a random number. Further details can be seen in the advisory that publishes this vulnerability[6].

5. ALTERNATE CONVENTION

```
file_reader ( FILE *f , char buffer [ 256 ] ) {
    unsigned int len;
    fread ( &len , sizeof ( int ) , 1 , f );
    if ( len <= 256 ) {
        fread ( buffer , len , 1 , f );
    } else {
        printf ( "len error !" );
    }
}
```

Figure 5: Source code for the function being diffed in the example (green=new code)



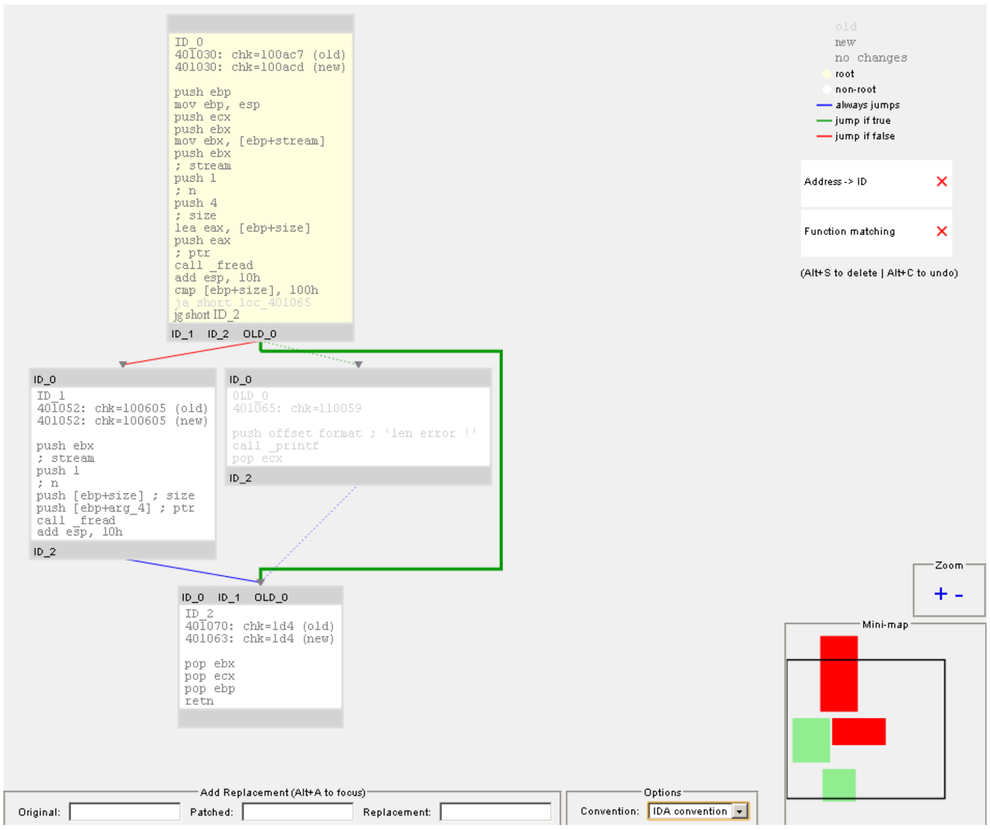
```
ID_17
4fb5530c: chk=120fe0 (old)
4fb5564f: chk=20134e (new)

mov ecx, esi
mov [esi+3Ch], eax
mov eax, [ebp+arg_8]
mov ecx, ushort gwTransactionId
inc word ptr ushort gwTransactionId
shr eax, 2
not eax
and eax, 1
call CAsyncDns::GenerateRandWord(void)
mov ecx, [ebp+arg_8]
shr ecx, 2
not ecx
and ecx, 1
push ecx
push eax
push ecx
push [ebp+arg_4]
mov [esi+590h], ax
push [ebp+lpMultiByteStr]
lea eax, [ebp+hostshort]
push [ebp+lpMultiByteStr]
push eax
push dword ptr [esi+3Ch]
call DnsWriteQuestionToBuffer_UTF8(x,x,x,x,x,x)
test eax, eax
jnz short ID_15
```

Figure 6: Patch ms10-024 changes a sequential number with a random generated one in dns.

[3]In our experience, in most cases it will be a single node in the merged graph. This is because usually the root node in the old graph matches the root node in the new graph.
[4]Using horizontal and vertical lines.

[5]http://www.microsoft.com/technet/security/Bulletin/MS10-024.mspx
[6]http://www.coresecurity.com/content/CORE-2010-0424-windows-smtp-dns-query-id-bugs



```
ID_0
401030: chk=100ac7 (old)
401030: chk=100acd (new)

push ebp
mov ebp, esp
push ecx
push ebx
mov ebx, [ebp+stream]
push ebx
; stream
push 1
; n
push 4
; size
lea eax, [ebp+size]
push eax
; ptr
call _fread
add esp, 10h
cmp [ebp+size], 100h
ja short loc_401065
jg short ID_2
ID_1   ID_2   OLD_0
```

```
ID_0
ID_1
401052: chk=100605 (old)
401052: chk=100605 (new)

push ebx
; stream
push 1
; n
push [ebp+size] ; size
push [ebp+arg_4] ; ptr
call _fread
add esp, 10h
ID_2
```

```
ID_0
OLD_0
401065: chk=110059

push offset format ; 'len error !'
call _printf
pop ecx
ID_2
```

```
ID_0   ID_1   OLD_0
ID_2
401070: chk=1d4 (old)
401063: chk=1d4 (new)

pop ebx
pop ecx
pop ebp
retn
```

Convention: IDA convention

Figure 7: Our implementation using the IDA convention

IDA Pro[2] uses colors to mark the different types of jumps instead of texture like we did. Its color convention is green for conditional jump taken, red for conditional jump not taken and blue for the other cases.

If we intend to use this convention for jumps then we can not use the color convention for changes. Then we propose a posible alternate convention using the intensity of the color to show if something is old, new or matched:

Old: Low color intensity (light grey on text).

Matched: Medium color intensity (grey on text).

New: High color intensity (bold on text).

In figure 7 we can see the example used in the previous section with this convention. As we can see, the alternate convention also fulfills the guidelines given in sections 1 and 2 but our experience shows that the colored lines are too distracting and it is more difficult to easily see the differences.

6. IMPLEMENTATION

The implementation is plugged into TurboDiff [7] as an alternate viewer. It uses the .gdl files that made by TurboDiff (that contain the old and new graph) and generates an .html file that is shown with a standards compliant browser[7].

[7]Tested with Firefox, Opera and Google Chrome

All the layout logic, text diffing and data visualization has been implemented in JavaScript using protovis [5] to show the visualization and jsdifflib [3] to calculate the text differences inside a matched basic block.

Several support scripts have been written in python.

The tool will be available as GPL licensed software.

7. CONCLUSIONS AND FUTURE WORK

Being at the prototype stage several improvements can be made to this project, such as:

- Integration with other diffing tools (such as BinDiff or PatchDiff).
- Assisted basic block matching. Letting the user match and unmatch some basic blocks and then matching all the others using this information.
- Closer integration with IDA.
- Alternate layouts and the ability to collapse, expand and move nodes in the graph.

The exploit writers did find useful to see the differences between the two basic-call graphs merged in a single graph.

This approach may also be used to visualize differences between other types of graphs such as network graphs, attack graphs, etc. We also believe that the idea of showing differences in graphs by merging the graph and annotating the differences can be used to show differences in several contexts and it is worth to explore this aproach for other types of data.

8. ACKNOWLEDGEMENTS AND CREDITS

I would like to thank Alberto Pose for working endless hours with me in the implementation, Nicolas Economou for his invaluable collaboration and sugestions while making aureliax, the exploits and research teams at Core for help, ideas and sugestions, Jamie Love for modifying protovis to support strokeDasharray and to the protovis team and mailing list for being extra-helpful in helping us troubleshoot several issues during the implementation.

9. REFERENCES

[1] Bindiff. `http://www.zynamics.com/bindiff.html`.

[2] Ida pro. `http://www.hex-rays.com/idapro/`.

[3] jsdifflib. `http://snowtide.com/jsdifflib`.

[4] Patchdiff2. `http://cgi.tenablesecurity.com/tenable/patchdiff.php`.

[5] protovis. `http://vis.stanford.edu/protovis/`.

[6] F.E. Allen. Control flow analysis. In *Proceedings of a symposium on Compiler optimization*, volume 5, pages 1–19. ACM New York, NY, USA, 1970.

[7] Nicolás Economou. Turbodiff. `http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=turbodiff`.

[8] S. Few. *Information dashboard design: the effective visual communication of data*. O'Reilly Media, Inc., 2006.

[9] C. Ware. *Information visualization: perception for design*. Morgan Kaufmann, 2004.