Diffing and Merging Databases with IDA Teams

Last updated on June 15, 2022 — v1.0

1. Overview

IDA 7.8 introduces IDA Teams - a mechanism that provides revision control for your IDA database files. Perhaps the most essential feature of this new product is the ability to natively diff and merge databases directly within IDA, allowing multiple reverse engineers to manage work on the same IDA database.

This document discusses in detail the steps involved when diffing and merging IDA databases.

Before continuing, you might want to take a quick look at the tutorial for hvui[TODO link], the GUI client for IDA Teams' revision control functionality. It will be referenced multiple times in this document, although here we will focus specifically on the merging functionality.

2. Inspecting changes

After having done some reverse-engineering work on an IDA database, it is possible to view those changes in a special mode in IDA: right-click, and choose the diff action:

Here a new instance of IDA will be launched in a special "diff" mode:
2.1. IDA’s diff mode

This new IDA mode lets the user compare two databases, in a traditional "diff" fashion: essentially a two-panel window, showing the unmodified file on the left and the version with your changes on the right.

2.1.1. The "Progress" widget

Represents the current step in the diff process.
2.1.2. The left panel

Shows the "untouched" version of the database (i.e., the one without your changes)
2.1.3. The right panel

<table>
<thead>
<tr>
<th>C:/temp/vutests/screenshots-misc_differing_and_merging/workspace/santazon.i64</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data:0000000000005001 00</td>
</tr>
<tr>
<td>.data:0000000000005002 00</td>
</tr>
<tr>
<td>.data:0000000000005003 00</td>
</tr>
<tr>
<td>.data:0000000000005004 00</td>
</tr>
<tr>
<td>.data:0000000000005005 00</td>
</tr>
<tr>
<td>.data:0000000000005006 00</td>
</tr>
<tr>
<td>.data:0000000000005007 00</td>
</tr>
<tr>
<td>.data:0000000000005008 00</td>
</tr>
<tr>
<td>.data:0000000000005009 00</td>
</tr>
<tr>
<td>.data:000000000000500a 00+</td>
</tr>
<tr>
<td>.data:000000000000500b 00 00 00+</td>
</tr>
<tr>
<td>.data:000000000000500c 00 00 00</td>
</tr>
<tr>
<td>.data:000000000000500d 00</td>
</tr>
<tr>
<td>.data:000000000000500e 00</td>
</tr>
<tr>
<td>.data:000000000000500f 00</td>
</tr>
<tr>
<td>.data:0000000000005010 00</td>
</tr>
</tbody>
</table>

Show your version of the database (i.e., featuring your changes)
2.1.4. Diff region details

Notice how both panels have a little area at the bottom, that is labeled "Details".

Details are available on certain steps of the diffing process, and provide additional information about the change that is currently displayed.
2.1.5. The "diffing" toolbar

The actions in the toolbar are:

- Previous chunk
- Center chunk
- Next chunk
- Proceed to the next step
- Toggle 'Details'

Using actions in the toolbar, you can now iterate through the differences between the two databases, with each change shown in context as if viewed through a normal IDA window.

The ability to view changes in context was a major factor in the decision to use IDA itself as the diffing/merging tool for IDA Teams.

Diff mode IDA’s toolbar actions

Previous chunk
Move to the previous change

Center chunk
Re-center the panels to show the current chunk (useful if you navigated around to get more context)

Next chunk
Move to the next change

Proceed to the next step
Move to the next step in the diffing process.

Toggle 'Details'
Toggle the visibility of the "Details" widgets in the various panels (note that some steps do not provide details, so even if the "Details" are requested, they might not be currently visible.)
2.2. Terminology

It is important to note the difference between the terms "diff" and "merge".

This document will sometimes use the two terms interchangeably. This is because to IDA, a diff is just a specialized merge. Both diffing and merging are handled by IDA’s “merge mode”, which involves up to 3 databases, one of which can be modified to contain the result of the merge.

A diff is simply a merge operation that involves only 2 databases, neither of which are modified.

This is why often times you will see the term "merge" used in the context of a diff. In this case "merge" is referring to IDA’s “merge mode”, rather than the process of merging multiple databases together into a combined database.

2.3. Using IDA as a diffing tool

We must stress the fact that performing a merge between two IDA databases is quite different than performing a merge between, say, two text files. A change in a chunk of text file will not have an impact over another chunk.

IDA databases are not so simple. A change in one place in an idb will often have an impact on another place. For example, if a structure `mystruct` changed between two databases, it will have an impact not only on the name of the structure, but on cross-references to structure members, function prototypes, etc.

This is why IDA’s merge mode is split into a strict series of "steps":

![Image of IDA's merge mode]

Within a single step it is possible to go forward & backward between different chunks. But because of possible inter-dependencies between steps, it is not possible to move backwards between steps, you can only go forward:

![Image showing IDA's merge mode sequence]

Since IDA’s diff mode is just a variation of its merge mode, diffing databases is also subject to this sequential application of steps in order to view certain bits of information. That is why, in some steps (e.g., the "Disassembly/Items") IDA might not report some changes that were performed at another level.

For instance, if a user marked a function as `noret`, the listings that will be shown in "Disassembly/Items" step, will not advertise that there was a change at that place (even though the "Attributes: noreturn" is visible in the left-hand listing), only the changes to the instructions (and data, ...) are visible in the current step:
The change will, however, be visible at a later step (i.e., "Functions/Registry"): 
NOTE

The changes applied during the "diff" process are only temporary. Exiting IDA (at any moment) will not alter the files being compared.
2.4. Merging concurrent modifications (conflicts)

As with any collaborative tool, it may happen that two coworkers work on the same dataset (e.g., IDA database), and make modifications to the same areas, resulting in "conflicts". Conflicts must be "resolved" prior to committing.

To do that, right-click and pick one of the "resolve" options:

IDA Teams provides the following merge strategies.

2.4.1. Interactive merging

If the option that was chosen (e.g., Interactive merge mode) requires user interaction due to conflicts, IDA will show in 3-pane "merge" mode.
When a conflict is encountered, you'll have the ability to pick, for all conflicts, which change should be kept (yours, or the other). Every time you pick a change (and thus resolve a conflict), IDA will proceed with the merging, applying all the non-conflicting changes it can, until the next conflict - if any. When all conflicts are resolved, you can leave IDA, and the new resulting file is ready to be submitted.

3. Appendix A

3.1. Merge Steps

This section provides a detailed overview of the steps involved in the merge process. The list of predefined merge steps is defined in `merge.hpp` of the IDASDK:

```
enum merge_kind_t
{
    MERGE_KIND_NETNODE,    // netnode (no merging, to be used in idbunits)
    MERGE_KIND_AUTOQ,      // auto queues
    MERGE_KIND_INF,        // merge the inf variable (global settings)
    MERGE_KIND_ENCODINGS,  // merge encodings
    MERGE_KIND_ENCODINGS2, // merge default encodings
    MERGE_KIND_SCRIPTS2,   // merge scripts common info
    MERGE_KIND_SCRIPTS,    // merge scripts
    MERGE_KIND_CUSTDATA,   // merge custom data type and formats
    MERGE_KIND_STRUCTS,    // merge structs (globally: add/delete structs entirely)
    MERGE_KIND_STRTMEM,   // merge struct members
    MERGE_KIND_ENUMS,      // merge enums
    MERGE_KIND_TILS,       // merge type libraries
    MERGE_KIND_TINFO,      // merge tinfo
    MERGE_KIND_UDTMEM,     // merge UDT members (local types)
    MERGE_KIND_SELECTORS,  // merge selectors
    MERGE_KIND_STT,        // merge flag storage types
    MERGE_KIND_SEGMENTS,   // merge segments
    MERGE_KIND_SEGREGS     // merge segment registers
};
```
The list of merge steps is not final. If for example there is a conflict in structure members then the new merge phase to resolve this conflict will be created. The same is hold for UDT, functions, frames and so on. In other words in general case the exact number of merge steps is undefined and depends on the databases.

We provide examples for some of the critical merge steps in the following sections.

### 3.1.1. Global settings/Database attributes

Global `idainfo` database attributes:
3.1.2. Global settings/Processor specific

Global `idpflags` processor settings:
3.1.3. Encodings/Registry and Encodings/Settings

This is an example of what you might see when merging registered string encodings:
### Encodings

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>Encoding</td>
</tr>
<tr>
<td>&lt;deleted #5&gt;</td>
<td>SBSS_k1</td>
</tr>
<tr>
<td>UTF-16LE</td>
<td>UTF-16LE</td>
</tr>
<tr>
<td>UTF-32LE</td>
<td>UTF-32LE</td>
</tr>
<tr>
<td>UTF-8</td>
<td>UTF-8</td>
</tr>
<tr>
<td>windows-1251</td>
<td>windows-1252</td>
</tr>
<tr>
<td>windows-1252</td>
<td>windows-1252</td>
</tr>
</tbody>
</table>

### Settings

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>default 8-bit</td>
<td>windows-1251</td>
</tr>
<tr>
<td>default 16-bit</td>
<td>UTF-16LE</td>
</tr>
<tr>
<td>default 32-bit</td>
<td>UTF-32LE</td>
</tr>
<tr>
<td>output_encoding</td>
<td>&lt;default&gt;</td>
</tr>
<tr>
<td>Default 8-bit</td>
<td>windows-1252</td>
</tr>
<tr>
<td>default 16-bit</td>
<td>UTF-16LE</td>
</tr>
<tr>
<td>default 32-bit</td>
<td>UTF-32LE</td>
</tr>
<tr>
<td>output_encoding</td>
<td>&lt;default&gt;</td>
</tr>
</tbody>
</table>
3.1.4. Scripts/Registry and Scripts/Settings

When merging scripts (or snippets) saved in the database, the script name/language is displayed, and the "Detail" pane contains the script source with the highlighted differences:

![](image)

3.1.5. Types/ Enums

Merging assembly level enums, which are not the ghost copy of local type.

As an example of merging enums, consider the following:

```
local_idb
;--------------------------
; enum enum_1, mappedto_1
A    = 0
B    = 1

remote_idb
;--------------------------
; enum enum_1, mappedto_1
A    = 0
;--------------------------
; enum enum_2, mappedto_2
B    = 1
```

In both idbs, enum constant "B" is present. However in the remote idb, "B" has a different parent enum, "enum_2".

When merging IDA will display both enum_1 and enum_2 in the Remote pane, indicating that the difference between Local and Remote corresponds to two separate enums, but is only considered a to be single difference location. The "Detail" pane will display the full enum definitions, with the highlighted differences:
3.1.6. Types/Structs

When merging assembly level structures, the structs are matched by name, structure tid and size, as well as the structure attributes. Note however that structure members are handled in a different step. If members of the same structure differ, the conflict may be resolved later during the Types/Struct members/… merge phase.

In the UI, IDA will display a list of structure names, with the "Detail" pane showing the structure attributes:
3.1.7. Types/Local types

Merging local types is similar to structures, in that types are matched by their attributes, and the members are handled in a different step. Local types are compared by type name, ordinal number, and base type. If the members differ then conflicts may be resolved later during the Types/Local types members/... merge step.

IDA uses the standard "Local types" widget to display the merged types. The "Detail" pane display the type definition and its attributes:
At the end of this merge step, new steps might be added if there are conflicting members:
3.1.8. Addressing/Segmentation

When merging the segments layout IDA separates segments in databases to non intersecting ranges.

For example, the following segmentations:

```plaintext
local_idb
  seg000:00000000
  ...
  seg000:00000020
  ...
remote_idb
  seg000:00000000
  ...
  seg001:00000010
  ...
  seg001:00000020
```

will result in a single difference:

The “Detail” pane displays segments in range with the attributes.

While merging IDA tries to move segment boundaries when possible to preserve segment data. Otherwise the conflicted segments will be deleted and new ones will be created.

There also may be orphan bytes that not belong to any segment:
Also byte values in a given segment may differ (for example if debug segment was saved in database) the merge step displays the differences using the standard "IDA-View" widget. The "Detail" pane will display the conflicted byte values:
3.1.9. Disassembly/Items

When merging, IDA compares each disassembly item in the analysis. IDA compares disassembly items by length, flags, opinfo, name, comment, and netnode information (NALT_* and NSUP_* flags).

This merge step uses the standard widget “IDA-View” so that items can be viewed in-context. For example:

![IDA-View screenshot showing disassembly items comparison]

3.1.10. Functions/Registry

Function definitions are merged using the standard “Functions” widget, with a “Detail” pane that displays function attributes:

![Functions widget screenshot showing function attributes]
3.1.11. Function frames

Merging function frames works similarly to types, in that global frames and frame members are handled in separate steps.

The **Frames (global)** step compares the global list of frames, which can be added or deleted:
If members of a matched function frame differ, the conflict may be resolved later during Functions/Frame/... merge phase.

For example, if IDA detects conflicts in the following function frames:

- Functions/Frame/sub_401200 at 401200
- Functions/Frame/_main at 4014E0

Each frame will be assigned its own merge step, with the “Details” pane showing the detailed information about the frame members:
3.1.12. Processor specific/…

Each processor plugin creates its own merge steps to handle the processor plugin's specific data.

For example, the PC processor plugin adds the following merge steps:

- Processor specific/Analyze ea for a possible offset
- Processor specific/Frame pointer info
- Processor specific/Pushinfo
- Processor specific/VXD info 2
- Processor specific/Callee EA/AH value
- …
<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.ZI</code>return_intcPc</td>
<td><code>.ZI</code>return_intcPc</td>
</tr>
<tr>
<td>prolog: [8048475]</td>
<td>prolog: [8048475]</td>
</tr>
<tr>
<td>bpltc: 1</td>
<td>bpltc: 1</td>
</tr>
<tr>
<td>spoiled: 0000003</td>
<td>spoiled: 0000003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 ExQueueWorkItem</td>
<td>18 ExQueueWorkItem</td>
</tr>
<tr>
<td>spoff: 8</td>
<td>spoff: 8</td>
</tr>
<tr>
<td>comment: <strong>inplace</strong></td>
<td>comment: remote_c__</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 ExQueueWorkItem</td>
<td>18 ExQueueWorkItem</td>
</tr>
<tr>
<td>spoff: 8</td>
<td>spoff: 8</td>
</tr>
<tr>
<td>comment: <strong>inplace</strong></td>
<td>comment: remote_c__</td>
</tr>
</tbody>
</table>
3.1.13. Plugins/Decompiler/…

Merging decompiler data starts with the global configuration parameters from hexrays.cfg:

To handle the decompilation of specific functions, IDA stores the decompilation data in a database netnode named Hexrays node.

The merge step Plugins/Decompiler/Hexrays nodes adds or deletes netnodes, indicating which functions have or haven't been decompiled between the two databases:
Decompilation data for matching functions is compared using the following attributes:

- Plugins/Decompiler/.../Numforms
- Plugins/Decompiler/.../mflags
- Plugins/Decompiler/.../User-defined funcargs
- Plugins/Decompiler/.../User-defined variable mapping
- Plugins/Decompiler/.../User-defined Ivar info
- Plugins/Decompiler/.../Ivar settings
- Plugins/Decompiler/.../IFLAGS
- Plugins/Decompiler/.../User labels
- Plugins/Decompiler/.../User unions
- Plugins/Decompiler/.../User comments
- Plugins/Decompiler/.../User-defined call

Each comparison criteria will be assigned its own merge step if a difference is found, and each step uses the standard "Pseudocode" widget so that differences can be viewed in-context with the full pseudocode.
### User-defined variable mapping

#### User-defined lvar info

<table>
<thead>
<tr>
<th>Line 107 of 125</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
</tr>
<tr>
<td>Remote</td>
</tr>
</tbody>
</table>

- `__int16 v3: // [ebp+30h] [ebp+10h]
- `__int128 v14[4]: // [ebp+70h] [ebp+70h] OVERLAPPED BYREF
- `dev_t a3: // x2 ISARG MAPDST

<table>
<thead>
<tr>
<th>Line 2 of 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details</td>
</tr>
</tbody>
</table>

- `__int16 v3: // [ebp+30h] [ebp+30h]
  - `cmt: lvar comment for v6 (local)
  - `__int128 v14[4]: // [ebp+70h] [ebp+70h] OVERLAPPED BYREF
  - `dev_t a3: // x2 ISARG MAPDST

<table>
<thead>
<tr>
<th>Line 1 of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote</td>
</tr>
</tbody>
</table>

- `__int64 v3: // [ebp+30h] [ebp+30h]
  - `cmt: lvar comment for v6 (remote)
  - `type: __int64
  - `__int64 v14[7]: // [ebp+70h] [ebp+70h] BYREF
  - `Unsinged __int64 a3: // x2 ISARG MAPDST

### User-defined lvar info

<table>
<thead>
<tr>
<th>Line 121 of 128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
</tr>
<tr>
<td>Remote</td>
</tr>
</tbody>
</table>

- `stckoff delta 0x0
- `l4v Flags:
### Line 1 of 2

<table>
<thead>
<tr>
<th>Address</th>
<th>Op</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>15444</td>
<td>If</td>
<td>15444</td>
</tr>
</tbody>
</table>

### Line 111 of 125

```c
char *v7; // x21
signed __int64 v10; // x21
const char *v11; // x22
__int64 v13; // x24
v11 = *(const char **)(v2 + 48);
if (v11)
    return 1LL;

v8 = a1;
if (v8 <= (unsigned __int64)v11 && (unsigned __int64)v11 <
     return 1LL;
v8 = v8[v16][9];
if (v8)
    }
v7 = **
        goto LOCAL_13;
    
CONFLICT:LOCAL_5:
es = v8->a88.u8.0e7.c7;
if (es)
    goto LOCAL_13;
while (v1)
{
    if (v7)
        goto CONFLICT_10;
    if (v7)
        break;
if (108->a88.u8.u8[0])
{
    v13 = strlen(v11) + 1LL;
v8 = 0LL;
        goto LABEL_21;

CONFLICT:LOCAL_10:
v7 = add_one(strlen(v11));
if (v7)
    00015444 save_abbr27 (15448)

CONFLICT:USER:
v8 = v8->0e7[1].c7;
if (108->a88.u8.u8[0])
    goto LABEL_13;
while (1)
{
    if (v8)
        goto CONFLICT_10;
    if (v8 == c7)
        break;
if (108->a88.u8.u8[0])
{
    v13 = strlen(v11) + 1LL;
v8 = 0LL;
        goto REMOTE_21;

CONFLICT:USER:
v7 = inc(strlen(v8));
if (v7)
    00015448 save_abbr27 (15448)
```
3.1.14. Loader data merge phases

The merge steps that correspond to loader data depend on which loader was used, for example:

- Loader/PE file/…
- Loader/NE file/…
- Loader/ELF file/…
- Loader/TLS/…
- Loader/ARM segment flags/…
### 3.1.15. Debugger data merge phases

To handle the differences in debugger data the following merge steps may be created:

- Debugger/pin
- Debugger/gdb
- Debugger/xnu
- Debugger/ios
- Debugger/bochs
- Debugger/windbg
- Debugger/rmac_arm
- Debugger/lmac_arm
- Debugger/rmac
- Debugger/lmac

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger/pin/PIN_PATH:</td>
<td>Debugger/pin/PIN_PATH:</td>
</tr>
<tr>
<td>Debugger/pin/PIN_TOOL_PATH:</td>
<td>Debugger/pin/PIN_TOOL_PATH:</td>
</tr>
<tr>
<td>Debugger/pin/PIN_TRACING: 4</td>
<td>Debugger/pin/PIN_TRACING: 0</td>
</tr>
<tr>
<td>Debugger/pin/PIN_TOOL_OPTIONS: 1</td>
<td>Debugger/pin/PIN_TOOL_OPTIONS: 0</td>
</tr>
<tr>
<td>Debugger/pin/PIN_PRE_ARGS:</td>
<td>Debugger/pin/PIN_PRE_ARGS:</td>
</tr>
<tr>
<td>Debugger/pin/PIN_POST_ARGS:</td>
<td>Debugger/pin/PIN_POST_ARGS:</td>
</tr>
<tr>
<td>Debugger/pin/PIN_CLIENT_TIMEOUT: 0</td>
<td>Debugger/pin/PIN_CLIENT_TIMEOUT: 0</td>
</tr>
<tr>
<td>Debugger/pin/TracingBufferSize: 0xF4240</td>
<td>Debugger/pin/TracingBufferSize: 0</td>
</tr>
<tr>
<td>Debugger/pin/TracingStepOpt: 3</td>
<td>Debugger/pin/TracingStepOpt: 0</td>
</tr>
<tr>
<td>Debugger/pin/TracingFuncOpt: 1</td>
<td>Debugger/pin/TracingFuncOpt: 0</td>
</tr>
</tbody>
</table>
3.1.16. Other plugins merge phases

There are a number of IDA plugins which need to merge their data.

For example:

- Plugins/PDB
- Plugins/golang
- Plugins/EHPARSE
- Plugins/Callgraph
- Plugins/swift

Any third party plugin may add merge phases using the IDASDK.
4. Resolving conflicts in a file

When a user needs to commit changes made to a file, but that same file has received other modifications (likely from other users) in the meantime, it is necessary to first "merge" the two sets of modifications together.

When the two sets of modifications do not overlap, merging is trivial - at least conceptually. But when they do overlap, they produce conflict(s).

Since IDA Teams focuses on collaboration over IDA database files, the rest of this section will focus on the different strategies that are available for resolving conflicts among those.

IDA Teams comes with multiple strategies to help in conflict resolution of IDA database files:

- Auto-resolve (if no conflicts)
- Auto-resolve, prefer local
- Auto-resolve, prefer remote
- Interactive merge mode
- Use local, discard remote
- Use remote, discard local

4.1. Auto-resolve (if no conflicts)

Launch IDA in a non-interactive batch mode, attempting to perform all merging automatically.

If any conflict is discovered, bail out of the merge process, and don't modify the local database.

4.2. Auto-resolve, prefer local

Launch IDA in a non-interactive batch mode, attempting to perform all merging automatically.

If a conflict is discovered, assume that the "local" change (i.e., the current user's change) is the correct one, and apply that.

Once all merging is done and conflicts are resolved, write those to the local database and exit IDA.

4.3. Auto-resolve, prefer remote

Launch IDA in a non-interactive batch mode, attempting to perform all merging automatically.

If a conflict is discovered, assume that the "remote" change (i.e., the change made by another user) is the correct one, and apply that.

Once all merging is done and conflicts are resolved, write those to the local database and exit IDA.

4.4. Interactive merge mode

Manual merge mode.

This will launch IDA in an interactive, 3-pane mode, allowing the user to decide how to resolve each conflict.

Once all merging is done and conflicts are resolved, exit IDA and write the changes to the local database.

4.5. Use local, discard remote

Select the local database, ignoring all changes in the remote database.

No IDA process is run.
4.6. Use remote, discard local

Select the remote database, ignoring all changes in the local database.

No IDA process is run.