Practical Version Tracking

Mike J. Bell
July 24\textsuperscript{th}, 2019
Objectives

- Learn the basics of version tracking
- Feel comfortable experimenting on your own
- Talk about some “gotchas”
- See a real workflow in action
- Questions at the end
What is Forward Engineering?

• The act of taking human readable declarations, instructions, etc., and turning them into machine code for execution
• Ofttimes this is a lossy process; names, comments, other metadata is discarded or obfuscated by compilation

“Programs must be written for people to read, and only incidentally for machines to execute” — Abelson & Sussman, SICP

What is Reverse Engineering?

• Starting with machine code, recovering and constructing enough metadata about a program so that humans can once again understand the declarations and instructions originally created
• This is almost always an additive process, where analysts provide names and comments to help explain what is going on

“A good engineer thinks in reverse and asks himself about the stylistic consequences of the components and systems he proposes”
— Helmut Jahn

https://www.google.com/search?q=helmut+jahn&tbm=isch
How Ghidra supports Reverse Engineering

- **Functions** organize instructions into discrete units, similar to their source code counterparts.
- **Data types** applied to sections of the program greatly increase legibility of the raw bytes, the listing, and most of all the decompiled output.
- **Labels** provide navigable names to functions, branch entries, and data (or at any address for that matter, including parameters).
- **Comments** help explain difficult sections of code or data usage, including function parameters.
What does Version Tracking do?

• Version Tracking finds functions and data in a new program (the destination) that *match* ones in an old program (the source), and transfers *markup* from the source to the destination

• Markup means: function signatures, applied data, labels, and comments
Why Track Versions?

- New malware was discovered, and you need to find out what changed from the last version
- A hardware firm released a new driver, and you must determine if they indeed patched their bugs
- The software you’re analyzing was released for a new architecture, and you need to reuse whatever gains you can against the new program
What are Ghidra’s Version Tracking strengths and weaknesses?

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Find things that match</td>
<td>• Cannot track changes to data types</td>
</tr>
<tr>
<td>• Transfer metadata</td>
<td>• Cannot match a program to many others</td>
</tr>
<tr>
<td>• Flexible user interface</td>
<td>• Doesn’t show “what’s different” or “what’s missing” well</td>
</tr>
<tr>
<td>• Great filtering</td>
<td>• Non-standard measurement criteria</td>
</tr>
<tr>
<td>• Side-by-side views</td>
<td>• Need 3 monitors really</td>
</tr>
</tbody>
</table>
Program Correlators

• A *Program Correlator* is a Java class that implements the VTProgramCorrelator interface

• Its job is to make matches between old and new functions or data

• There are many different Program Correlators available in Ghidra
Address Correlators

• An *Address Correlator* is a Java class that implements the VTAddressCorrelator interface

• Its job is to map old addresses to new addresses for a function match

• There are currently only two Address Correlators, one for “exact” matching and another catch-all implementation\(^1\)

\(^1\) gotcha #1
A Whirlwind Tour
Open the Version Tracking Tool

• Click on the “footsteps” tool icon from the front-end
First Time opened...VT Workflow Help Screen

- Most of the information in Help is factual and helpful
- Some of it can be partially misleading
- Keep in mind that tracking versions is a difficult endeavor
Check your Preferences!

• Ensure that “Auto Create Implied Matches” is **OFF** in the Tool Options!¹

¹ gotcha #1
Create a new Session

• In the newly created Version Tracking window, click the “footprints” action
Navigate the Session Wizard

- Select Source and Destination programs
- Preconditions are casually interesting, but mama always told me…
It’s full of windows!
What are the full Source/Dest. Tools for?
Match Window: press the green plus to run
Run with a safe, innocuous Correlator

- Exact Data sounds good, and it’s first in the list
Now run with another (why not simultaneous?²)

- Let’s do Exact Function Bytes², gotcha #2, stay tuned
Match Window
Observe the coordinated navigation
Accept Markup Options
Apply Markup Options
Markup Items Window
**Gotcha #1: Implied Matches & Address Correlator**

What happens if the function calls become out of order?

**Version 1**

```c
int my_crazy_fn(int parm1, char* parm2) {
    int a = 23;
    call_1st_function(a, (int) *parm2);
    call_2nd_function(parm1, a);
}
```

**Version 2**

```c
int my_crazy_fn(int parm1, char* parm2) {
    int a = 23;
    call_2nd_function(parm1, a);
    call_1st_function(a, (int) *parm2);
    call_2nd_function(parm1, a);
}
```
Functions Window
Deep Dive
Correlators: Exact Data

- Data means that the correlator is only looking for data matches
- Exact means that the data must match once, and only once, in each the source and destination (in other words, the correspondence is 1:1)
- It’s wonderful to run first, because it finds 1:1 data matches even when the destination match is undefined or mistakenly made into code
- Safe to apply all matches in almost every case
Correlators: Exact Function.

- **Exact Function** *Bytes* means the actual bytes match.
- **Instructions** means that the opcodes match, but operands can be different.
- **Mnemonics** is a much looser correlator that concatenates mnemonic strings in order to determine matches.
- Each represents an increasingly loose interpretation of “Exact” similarity.
- Why might **Instructions** be best to start with?
Aside: Similarity vs. Identity

- Similarity == 1.0 means, ideally, that the two functions are identical in nature
- Identity means that the two functions actually represent the same concept or instance
- They are not equivalent! Consider: thunks, trampolines, C++ getters/setters, C++ template functions or other generated code
- Note that **EXACT** data doesn’t suffer this distinction as much at all
Correlators: Exact Symbol Name

• If you’re lucky enough to recover function and data names in both the source and destination programs, this correlator generates matches based on their symbols being identical.
• Can save a LOT of time if you’re sure the two programs are almost identical, and nothing crazy is going on.
Gotcha #2: Why not run multiple correlators?

- Most correlators benefit from restricting their matches based upon carefully curated results from *previous* correlator runs
- Best to run one, see how it does, accept or apply matches, and then continue where the last one left off
- Others simply don’t work *at all* unless other correlators have been run and matches accepted/applied
- Every situation usually requires a different approach
Correlators: .* Reference

Here’s where it gets interesting

- Functions that are different enough to be missed by the other correlators can be recovered by taking their references into account
- Data references—in particular strings—often yield very precise fingerprints for matching Functions
- If child function calls have been accepted/applied, those function references can be used just like data references
- Note that the possibility for false positives is much higher using child function references for matching
Correlators: Duplicate .*

• For those situations where you want to manually disambiguate very similar functions or data, the duplicate correlators make M:N matches where !(M == 1 && N == 1)

• Usually creates an inordinate number of matches; useful to make it look like you’re doing a lot of work
Correlators: Similar.*

• If the source and destination symbol names have changed subtly, the Similar Symbol correlator can create matches based on close matches.
• Likewise, if data have changed in small ways, the Similar Data correlator can match these.
• Note that the Similar correlator results can overlap Exact matches, so restricting address sets or excluding accepted matches is a good idea.
Example Workflow: New Version, Stripped

diff-2.8 → diff-2.8.1 on x86-64

• Exact Data (apply all)
• Exact Function Bytes...huh?
• Exact Function Instructions...OK!
• Note 13/12 of 110/109 functions left unmatched
Example Workflow: Same Version, Different Arch.

bash-4.4.12 on x86-64 $\rightarrow$ bash-4.4.12 on arm7-32

- Exact Data (size 9, apply all)
- Exact Symbol (apply all; note bug in correlator? auto-analysis?)
- Function and Data Reference (triage with conflicting 0, note 663/118 of 2665/2022 functions left unmatched)
Thank You!