A Day in the Life of a Cyber Tool Developer

by Jonathan Tomczak
jon@tzworks.net
Who is TZWorks?

Jonathan Tomczak ('Front Man')

Software Engineer w/ over 7 years experience working in software and web development

Dave Tomczak ('Mad Scientist')

Principal Developer w/ over 30 years of experience working with DoD, Federal Agencies and Industry
Security Detail

Lacey

- Rescued Belgian Shepherd
- Female, 65 lbs.
- Under 2 years experience
- Good at catching moving objects
What we do?

Develop Digital Forensics Tools

- Live Collection
  (ntfscopy, ntfswalk, nx/nxx, ...)

- Artifact Analysis
  (yaru, evtx, sbag, jmp, usp, ...)

Develop Custom Security Solutions

- System Monitoring
Getting Started...

1. Tool Development Philosophy
2. Problems facing Forensic Investigators
3. Tool Automation
4. Demo “Scripting an E-Discovery Solution”
5. Rendering NTFS
6. Automating Extraction of NTFS
7. Demo “Utilizing GENA in a Workflow”
8. Summary
How We Develop Tools
Tool Development Philosophy

- Code written in C/C++
- Source code portable across Linux, Mac and Windows
- Handles 32 or 64 bit Operating Systems
- Archived in source control system so any version can be retrieved
- Libraries are designed as re-usable modules and can be leveraged into new or existing tools.
  - Time to develop a new tool is much faster (1-2 months)
  - Bug fix turnaround time is quicker and affects all tools using that module.
- Finished Tool(s) are registered to the US Copyright Office
Tool Development Philosophy

Our in-house libraries are rather extensive and include...

- Time Conversion
- Unicode Conversion
- Socket wrapper library
- Registry internals
- NTFS internals
- VMWare internals
- Threading
- Event log internals
- NT Kernel internals
- NT memory internals
- PE internals
- Crypto
- SQLite internals
- Etc...

TZWorks uses very few 3rd party libraries

- FOX for GUI apps
- STL/C++ libraries
- C/C++ runtime libraries
Ensuring Tool Integrity

All binaries (built for Windows and Mac) are signed with an X509 TZWorks code signing certificate that was issued by the Certificate Authority GlobalSign.

As part of the post build process at TZWorks, all binaries are embedded with their own hash. This allows self validation to occur during runtime to ensure they have not been tampered with. Any mismatch of the computed hash with the embedded hash will report an error.
# DFIR Tools and User Interface

(From our perspective)

## GUI vs CLI

<table>
<thead>
<tr>
<th>Category</th>
<th>GUI</th>
<th>CLI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of use</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>View multiple things at once</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Speed</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Less System Resources</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>Scripting</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Remote access</td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>

**Needed for RE**

Both are needed for Artifact RE and Tool Development. Therefore, we build GUI Tools to help out in RE and CLI for the heavy lifting of processing the data.
Tool Development Cycle

(Two main cycles – RE & Creating modular library)

- Build a Viewer (Reconstruct data structures interleaved with raw hex dumps)
- Use API calls & read docs
- Analyze raw data in Hex Editor

Modular Library

CLI Tool1

CLI Tool2

CLI Tool3

Modular Library

Modular Library

Modular Library

Modular Library
Case Study - NTFS Analysis
(Looking Back... How we did it)

1. Started with the Windows Native API calls for NTFS reads/writes (Gary Nebbett’s book)

2. Examined the raw NTFS images using a Hex Editor.

3. Built an NTFS Viewer (ntfs_view) to look at the internals of raw cluster data. This allowed us to view the known NTFS structures with the unknown data (by intermingling structured output with hex dumps).

4. Generated re-usable modules for Windows raw cluster reads and NTFS traversal and ported them to Linux and Mac.

5. Leveraged re-usable modules, was able to build ntfsdir, ntfscopy, wisp, jp, and any tool requiring access to locked down files.
Finding Solutions...
Problem:

How do we simplify the job for the analyst?

- Rendering Intelligence out of data
- Ease of Use
- Automation
- Live Extraction
E-Discovery Workflow Issues?
Currently massive amount of time just for data collection
- Automatic processing of entire drives can be unrealistic
- As hard disk drives get larger, less feasible to image and then process entire drive
- Processing a single hard disk drive image can easily take up to a few days.

- Fully processing and analyzing every byte of data from numerous drives can take months.

- While it may be feasible to process all the information on a person of interest, it is not timely for an entire company in order to find accomplices, etc.
Subject Matter Experts (SME)s are expensive:

- Training
- Experience
- Need to maintain a current state of knowledge on the ever changing state of technology

Due to sufficient lack of expert analysis, incriminating evidence can often be overlooked.
Problem:

How do we simplify the job for the analyst?
Tool Automation

Embedding manual operations into a tool (Use-Case 1)

usp will read the live registry hives and setup API log, parse out the relevant USB data, and combine the results into a report for quick analysis.
Windows Event Log Parser (evtwalk):

Parsing USB events
Timeline Analysis

```
usp -> evtwalk
  
jmp
lp
sbag
ntfswalk
```

More Input!
Batch Script for Live Collection

- Read specified raw artifacts from a live system
- Parse the artifacts
- Archive post processed results to a designated directory
DEMO

Scripting an E-Discovery Solution
Rendering NTFS
Windows $MFT and NTFS Metadata Extractor Tool (ntfswalk)
$NEW* Windows $MFT and NTFS Metadata Extractor Tool (ntfswalk)

- Source
  - $MFT extracted file
    - Drive [dd image]
    - Volume [dd image]
  - Drive [live system]
    - Volume [live system]
    - VMWare monolithic disk

- Filter
  - Deleted
    - Time range
    - Inode range
  - Name(s)
  - Extension(s)
  - Binary signature
    - Binary pattern
  - Directories
  - Unalloc Clusters

- Extract
  - Normal data
    - Header Info
    - Data w/slack
    - Cluster info
  - Ads/indx data
  - All Data
  - Hash Data

- Results
  - Default
    - csv
    - Body file
  - Log2timeline
  - File vice stdout
  - Misc Options
Graphical Engine for NTFS Analysis

*gena*

- ntfs_view
- ntfscopy
- ntfsdir
- ntfswalk
- wisp
Viewing File Data
Viewing File Metadata
Viewing INDX Slack

Select Directory to Analyze
Generating File Hashes
Walking the File System w/ gena
Tool Automation via GUI (Use-Case3)

Filtering Options

Select Files to Add to Analysis

Current Selected Files and Options
Accessing Results

Items selected in *gena* and the results of extraction
Utilizing GENA in a Workflow
Summary and Conclusions

Simplifying the job of the analyst

- Intelligence from multiple sources (Data Correlation)
- Render data so that it is easy to understand
- Scriptability for automation
- Live extraction of target systems

Tool Development is unique to every shop

- Create tools that work across multiple operating systems
- Easy to maintain and straightforward to expand in capabilities
- Cumulative knowledge of the Windows OS, allows short development times for new or custom tools.
Development Case Studies... (backup charts)
Case Study - NTFS Analysis
(Looking Back... How we did it)

1. Started with the Windows Native API calls for NTFS reads/writes (Gary Nebbett’s book)

2. Examined the raw NTFS images using a Hex Editor.

3. Built an NTFS Viewer (ntfs_view) to look at the internals of raw cluster data. This allowed us to view the known NTFS structures with the unknown data (by intermingling structured output with hex dumps).

4. Generated re-usable modules for Windows raw cluster reads and NTFS traversal and ported them to Linux and Mac.

5. Leveraged re-usable modules, was able to build ntfsdir, ntfscopy, wisp, jp, and any tool requiring access to locked down files.
Case Study - Registry Analysis
(Looking Back... How we did it)

1. Started with the **Windows API calls** for Registry reads/writes.

2. Examined raw registry hives with a **Hex Editor** and reviewed available documentation.

3. Iteratively **built a Registry Viewer & companion cmdline version** that didn’t rely on Windows API calls (eg. **yaru**). Incorporated many capabilities in **yaru** to analyze and test internal structures in the Registry hive. Some options were stubbed out of the DFIR version of **yaru** (eg. the write capability).

4. Generated **re-usable modules** for Windows registry traversal and ported them to Linux and Mac.

5. **Leveraged** re-usable modules, was able to build **sbag, cafae, usp**, and other tools.
Case Study - Eventlog Analysis

(Looking Back... How we did it)

1. Skipped the Windows API calls for eventlog reads/writes and read the paper on Vista event log format by Andreas Schuster.

2. Examined the raw evtx files using a Hex Editor and compared them to the processed records from built-in Windows tools.

3. Built an EVTX Viewer (evtx_view) to look at the internals with just raw cluster data. Same theme as before...

4. Generated re-usable modules for Windows raw evtx file parsing and ported them to Linux and Mac.

5. Leveraged re-usable modules, was able to build evtwalk and any tool requiring access to parsed data from evtx (or evt) files.