An Observational Investigation of Reverse Engineers’ Process and Mental Models
Daniel Votipka, Seth Rabin, Kristopher Micinski, Leilani Battle, Jeffrey S. Foster, and Michelle L. Mazurek

Research Questions

R1. How do reverse engineers analyze programs?
- What do they focus on?
- What methods do they use?
R2. How do reverse engineers make decisions?
- Where to look?
- What tools to use?

Background

Program Comprehension
How developers process unfamiliar code for maintenance, debugging, and modification
- Likely similar items of interest
- More adversarial environment

Naturalistic Decision Making
How people perform complex cognitive tasks in real-world situations
- Provides methods for studying complex decision-making

Interview Study

Participants were asked to demonstrate their reverse engineering process.
The interviewer asked further questions about “items of interest”
16 reverse engineers recruited through bug bounty platforms, hacking teams, reverse engineering forums, and related conferences

Reverse Engineering Process

Question/Hypothesis

Questions that must be answered or conjectures about what the program does.
- Began with code demographic questions to find places to start
- Asked more specific questions about the data flow and control flow as they processed the code
- Hypotheses based on an even mix of prior experience and observed behavior

Simulation

Any process where a participant discusses reading or running the code to determine its function.
- Static analysis: Skimmed components (data/control flow paths, strings, APIs); rarely read line-by-line
- Dynamic analysis: Inputs driven by manual, static analysis
- Frequently switch between static and dynamic contexts
- Rarely used complex automation

Decision

Moments where an option (code to analyze, input to use) must be selected from many.
- Mostly even mix of prior experience, observed behaviors, and systematic approaches
- Used heuristics to bootstrap investigation

Beacon

Patterns or tells that a reverse engineer recognizes, allowing them to quickly understand the functionality of some code without stepping through it line-by-line.
- Wide variety of beacons used
- Most commonly named entities: APIs, strings, variable names
- Utilized beacons not seen in program comprehension: program flow, metadata

Example

```c
int ambiguous_func(path, g) {
    FILE *secret = fopen("pwd", "r");
    FILE *user_file = fopen(path, "w");
    wchar_t *k = unk_func1(secret);
    int vi = unk_func2(user_file, k);
    fclose(secret);
    fclose(user_file);
    return vi;
}
char * unk_func1(FILE *in_file) {
    char *p = touch(in_file)
    return p;
}
```