A Tool for Teaching Reverse Engineering

Clark Taylor and Christian Collberg
Department of Computer Science,
University of Arizona
Why Teach Reverse Engineering?

• Maintaining old code
  – Not related to security or obfuscation
Why Teach Reverse Engineering?

• Dissecting malicious code

Malicious Program → Tools → Behavior → Attribution

Identifiers
Why Teach Reverse Engineering?

• Analyzing reverse engineering vulnerabilities
Why Teach Reverse Engineering?

- Understanding obfuscation methods

Flowchart:
- Obfuscated Program
  - Tools
  - Understanding obfuscation patterns
Why Teach Reverse Engineering?

• Analyze and counter malware threats
• Protect software assets from man-at-the-end (MATE) attacks
• Contribute to the field
• Malicious uses?
The Problem

• Generating and administering unique reverse engineering exercises is difficult

For each student:
  1. Generate problem
  2. Obfuscate problem
  3. Send problems
  4. Grade problems

Alice
Generation

- Alice generates a problem for each student
Distribution

- Alice sends the problems to the students
Grading

• Students submit answers to Alice
The Problem

- Generating and administering unique reverse engineering exercises is difficult
Student Environment Setup

- Students have problems getting started

1. Download OS
2. Configure VM
3. Install tools and dependencies
4. Get P’ onto VM
5. Solve P’ → P
6. Turn in P
Our Solution

• Automate exercise generation, with randomization
• Automate exercise administration
• Automate environment setup
• Automation, *automation*, *automation*
Desired Functionality

1. Administrative functions
2. Challenge generation
   - Automated, random code generation
   - Automated, random code obfuscation
3. Grading system
   - Manual
   - Automated
4. Environment distribution
   - Static
   - Dynamic
5. Data collection
Implementation Strategy

• Web application
  – Easy for students to use
  – Few dependencies; no client setup
  – Accessible on the internet

• Student terminals
  – Preconfigured environment
  – Virtual or physical/device
System Usage

Create Challenge
Tigress commands

Server
System Usage

Server

Download Environment
Configured virtual machine
System Usage

Server

Download Challenge
Obfuscated file (P')
System Usage

Upload Answer
Reverse engineered file ($P_a$)
System Usage

Server

Download Answers
Original files (P)
Answer files ($P_a$)
System Usage

- Server
- Upload Grades
- Grades
Obfuscation

• Tigress
  – Operates on C language
  – Source-to-source obfuscator
  – Numerous transforms
  – Randomization built in
  – Includes code generation components

• Gcc compiler
Tigress Obfuscation Examples

1. **Tigress**
2. **P**
3. **Tigress**
4. **P’**

- Build program with assets
- Obfuscate program
Tigress Obfuscation Examples

```c
#include <stdio.h>
#include <stdlib.h>

void SECRET(unsigned long input[1], unsigned long output[1]) {
    ... }

int main(int argc, char** argv) {
    unsigned long input[1];
    unsigned long output[1];
    int i5;
    unsigned long value6;
    int i7;
    i5 = 0;
    while (i5 < 1) {
        value6 = strtoul(argv[i5 + 1], 0, 10);
        input[i5] = value6;
        i5 ++;
    }
    SECRET(input, output);
    i7 = 0;
    while (i7 < 1) {
        printf("%lu\n", output[i7]);
        i7 ++;
    }
}
```
Tigress Obfuscation Examples

```c
void SECRET(unsigned long i[1] , unsigned long o[1] ) {
    unsigned long s[4] ;

    s[0UL] = i[0UL] + 762537946UL;
    s[1UL] = i[0UL] | ((16601096UL << (s[0UL] % 16UL | 1UL)) |
                        (16601096UL >> (64 - (s[0UL] % 16UL | 1UL)))));
    s[2UL] = (i[0UL] ^ 643136481UL) ^ (s[0UL] + 292656718UL);
    s[3UL] = (i[0UL] << (((s[1UL] >> 4UL) & 15UL) | 1UL)) |
              (i[0UL] >> (64 - (((s[1UL] >> 4UL) & 15UL) | 1UL))));

    unsigned long l = 0UL;
    while (l < 3UL) {
        s[1UL] |= (s[2UL] & 15UL) << 3UL;
        s[l + 1UL] = s[l]; l += 2UL;
    }
    if ((s[0UL] | s[1UL]) > (s[2UL] | s[3UL])) {
        s[3UL] |= (s[1UL] & 31UL) << 3UL;
    } else {
        s[2UL] = s[0UL]; s[3UL] |= (s[2UL] & 15UL) << 3UL;
    }
    s[0UL] = s[2UL];

    o[0UL] = (s[0UL] << (s[1UL] % 8UL | 1UL)) <<
              (((s[2UL] << (s[3UL] % 8UL | 1UL)) >> 1UL) & 7UL) | 1UL);
}
```
Deployment

• System used for a ~35 student course
• Configured for two binary challenges
• Students answered several additional questions:
  – What was the level of difficulty?
  – How long did it take to solve the problem?
Results

- Students were able to use the system and solve the easier problem.
Future Work

• Dynamic environments
  – Docker
  – Provisioner
• Automated grading
  – Simple token grading
  – Input/output cases
  – Natural language processing
  – Code entropy
• Data collection
  – Syslog ng
  – Splunk
  – Custom built solutions
• Visualization
Conclusion

• Reverse engineering is a valuable skill
• Teaching that skill typically involves a lot of overhead
• Integrating Tigress with a webapp allowed us to easily generate and administer randomized exercises
Questions?