CSc 466/566

Computer Security

9 : Man-At-The-End — Tigress

Version: 2014/09/23 15:24:19

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

Get Documentation

1. Download these slides from

   `http://tigress.cs.arizona.edu/main.pdf`

2. Also, get the `fib.c` program:

   `http://tigress.cs.arizona.edu/fib.c`

3. Consult the Tigress documentation at:

   `http://tigress.cs.arizona.edu`

Get Tigress I

1. Download from

   http://tigress.cs.arizona.edu/#download

2. Unzip the `.zip`-file. You should get a directory named `tigress-1.3`.
3. Set the `TIGRESS_HOME` environment variable to the directory in which the tigress binary resides. Also put this directory on your `PATH`.

Get Tigress II

1. In the C-shell, you can do

```
> setenv TIGRESS_HOME /PATH_TO/tigress -1.3
> setenv PATH /PATH_TO/tigress -1.3:$PATH
```

   You can put these in your `.cshrc` file.

2. In the Bourne shell, you can do

```
> export TIGRESS_HOME=/PATH_TO/tigress -1.3
> export PATH=$PATH:/PATH_TO/tigress -1.3
```

3. Now try

```
tigress --help
tigress --options
tigress --version
```

fib.c

```
#include<stdio.h>
#include<stdlib.h>
int fib(int n) {
   int a = 1; int b = 1; int i;
   for (i = 3; i <= n; i++) {
      int c = a + b; a = b; b = c;
   };
   return b;
}
int main(int argc, char** argv) {
  if (argc != 2) {
     printf("Give one argument!\n"); abort();
  };
  long n = strtol(argv[1],NULL,10);
  int f = fib(n);
  printf("fib(%li)=%i\n",n,f);
```

Virtualize I

1. Apply a simple interpreter transformation:

```
tigress --Transform=Virtualize \
    --Functions=fib \
    --VirtualizeDispatch=switch \
    --out=v1.c fib.c
```

2. Try a few different dispatchers: `direct`, `indirect`, `call`, `ifnest`, `linear`, `binary`, `interpolation`. Are some of them better obfuscators than others? Why?

Virtualize II

1. Try two levels of interpretation:

```
tigress --Transform=Virtualize
    --Functions=fib \
    --VirtualizeDispatch=switch \
    --Transform=Virtualize\
    --Functions=fib \
    --VirtualizeDispatch=indirect \
    --out=v2.c fib.c
```

2. Try combining different dipatchers. Does it make a difference?
3. Try three levels of interpretation! Do you notice a slowdown? What about the size of the program?

Virtualize III

1. Look at the output from one level of interpretation, with switch dispatch. Do you think the instrution handlers would be easy to reverse engineer?
2. Try superoperators:

```
tigress --Transform=Virtualize \
    --Functions=fib \
    --VirtualizeDispatch=switch \
    --VirtualizeSuperOpsRatio=2.0 \
    --VirtualizeMaxMergeLength=10 \
    --VirtualizeOptimizeBody=true \
    --out=v3.c fib.c
```

3. What differences do you notice?

Virtualize IV

1. Notice that the instruction handlers all use stack operations. Does that make them easy to analyze?
2. Try registers instead:

```
    --VirtualizeOperands=registers
```

3. Or, try mixing registers and stacks:

```
    --VirtualizeOperands=registers ,stack
```

4. What differences do you notice?

Virtualize V

1. Do you think the instruction handlers are still easy to identify? How about breaking them up with opaque predicates:

```
tigress --Transform=InitOpaque \
    --Functions=main ... \
    --VirtualizeMaxOpaque=4 ...
```

2. What differences do you notice?

Virtualize VI

1. An add instruction handler could still be identified by the fact that it uses a + operator!
2. Try adding a arithmetic transformer:

```
 ... --Transform=EncodeArithmetic \
--Functions=fib,main ...
```

3. What differences do you notice?

Virtualize VII

1. Variable values (such as program counter and stack pointer) are always in the clear. This may help a dynamic analysis.
2. Try adding a data transformer to the stack pointer (you may have to look at the source to figure out the actual name of sp):

```
 ... --Transform=EncodeData \
--LocalVariables=fib:_1_fib_\$sp \
--EncodeDataCodecs=poly1 ...
```

3. What differences do you notice? Is this transformation useful here?

Virtualize VIII

1. A virus that uses virtualization would want to hide the virtualized function as much as possible.
2. Use function splitting to break up the virtualized function:

```
 ... --Transform=Split \
--LocalVariables=fib:_1_fib_\$sp \
--EncodeDataCodecs=poly1 ..
```

3. You can play around with the type of splitting to get one that looks good:

```
--SplitKinds=top,block,deep,recursive
```

Strings

1. Let's get rid of the constant string in main!

```
 ... --Transform=EncodeLiterals \
--EncodeLiteralsKinds=string \
--EncodeLiteralsEncoderName=STRINGS\
--Functions=main
```

2. Look at the STRINGS function! Easy to analyze, right? Well, apply a virtualization to it!

```
 ...   --Transform=Virtualize \
  --Functions=STRINGS  ...
```

Flatten I

1. Try flattening the original `fib.c`:

```
tigress --Transform=InitOpaque \
   --Functions=main \
   --Transform=Flatten \
   --FlattenDispatch=switch \
   --FlattenOpaqueStructs=array \
   --FlattenObfuscateNext=false \
   --FlattenSplitBasicBlocks=false \
   --Functions=fib \
   fib.c --out=f1.c
```

Flatten II

1. Try different kinds of dispatch (`switch`, `goto`, `indirect`).
2. Turn opaque predicates on and off.
3. Split basic blocks or not.

Virtualize + Split + Flatten

1. Now virtualize `fib`, split out as many parts as possible, and flatten the resulting (smaller) function!
2. Is it still easy (for a virus scanner, say) to determine that `fib` has been virtualized?

Other transformations

1. Look at the documentation for Tigress on

   `tigress.cs.arizona.edu`

   and try out the remaining transformations!

Diversity

1. Setting `Seed` to zero will initialize tigress' random number generator with a different value each time it is run:

```
tigress --Seed=0 ...
```

2. How different are two variants of the same program, run with the same transformations, but different seeds?