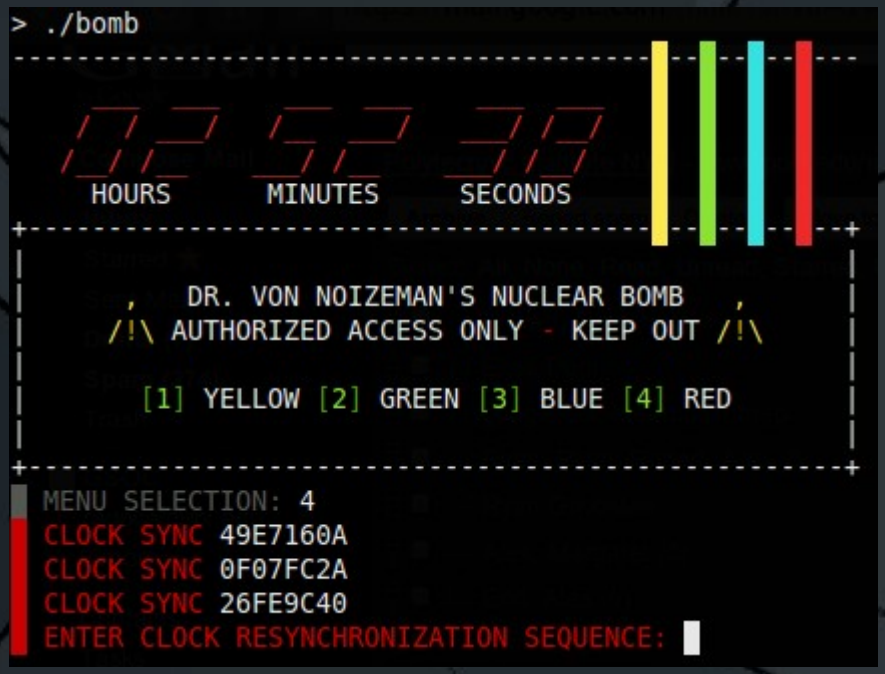
AR, RG, AZ, JS, AC

Reversing Tips

Good Challenge?



- Yellow Wire – easy
- Green Wire – hard according to Noizeman, but you guys did it in ~1hr
- Blue Wire – you guys did it after class
- Red Wire - ???

We Still Made It Too Easy

- Full Symbols
  - Function names
  - Global variables
- Dynamically linked binary
- Realistic?

Reversing in the Dark

- Whittle away, instruction by instruction
- 100k program ~= 20k instructions
  - 1MB ?

How do you reverse a large, symbol-less program?

- Invite these guys?

Start reversing from easy mode

- Use binary's data against it
  - Left over debug output
- Find and analyze interesting data references

Embedded System Trivia

```
35258:;
35259:Xe558:    mov     dptr,#X0369     ; e558   90 03 69   ..i
35260:  mov     a,r7            ; e55b   ef         o
35261:  movx    @dptr,a         ; e55c   f0         p
35262:  clr     c               ; e55d   c3         C
35263:  subb    a,#0x7b         ; e55e   94 7b      .{
35264:  jnc     Xe56f           ; e560   50 0d      P.
35265:  movx    a,@dptr         ; e562   e0         `
35266:  clr     c               ; e563   c3         C
35267:  subb    a,#0x61         ; e564   94 61      .a
35268:  jc      Xe56f           ; e566   40 07      @.
35269:  mov     dptr,#X0369     ; e568   90 03 69   ..i
35270:  movx    a,@dptr         ; e56b   e0         `
35271:  anl     a,#0xdf         ; e56c   54 df      T_
35272:  movx    @dptr,a         ; e56e   f0         p
35273:Xe56f:    mov     dptr,#X0369     ; e56f   90 03 69   ..i
35274:  movx    a,@dptr         ; e572   e0         `
35275:  mov     r7,a            ; e573   ff         .
35276:  mov     r6,#0x0         ; e574   7e 00      ~.
35277:  ret                     ; e576   22         "
35278:;
```

- 80c32 Tattoo from a Removal Machine
- Serial console returns uppercase input
- Which function did we find first?

Identify the magic bytes we used to find this function:

```
35258:;
35259:Xe558:     mov     dptr,#X0369       ; e558   90 03 69    ..i
35260:   mov     a,r7                      ; e55b   ef           o
35261:   movx    @dptr,a                   ; e55c   f0           p
35262:   clr     c                         ; e55d   c3           C
35263:   subb    a,#0x7b                   ; e55e   94 7b        .{
35264:   jnc     Xe56f                     ; e560   50 0d        P.
35265:   movx    a,@dptr                   ; e562   e0           `
35266:   clr     c                         ; e563   c3           C
35267:   subb    a,#0x61                   ; e564   94 61        .a
35268:   jc      Xe56f                     ; e566   40 07        @.
35269:   mov     dptr,#X0369               ; e568   90 03 69     ..i
35270:   movx    a,@dptr                   ; e56b   e0           `
35271:   anl     a,#0xdf                   ; e56c   54 df        T_
35272:   movx    @dptr,a                   ; e56e   f0           p
35273:Xe56f:     mov     dptr,#X0369       ; e56f   90 03 69     ..i
35274:   movx    a,@dptr                   ; e572   e0           `
35275:   mov     r7,a                      ; e573   ff           .
35276:   mov     r6,#0x0                   ; e574   7e 00        ~.
35277:   ret                               ; e576   22           "
35278:;
```

Answer

- ~0x20 == 0xdf
  - Common bit twiddling trick
  - Ironically, it was the first we tried
- 0x6a/0x7b would have been good choices too
  - 'a'/'z'+1 → range values
-

Dynamic Analysis

- Isolate code of interest
  - Flow-graph leading to point of crash
  - Easily Detect Attack Surface Entry Points
- Careful when analyzing malware

Work Backwards

- Set software/hardware breakpoints on interesting data
- Look at stack trace
  - Should reveal relevant functions

Be Creative

- Patch out uninteresting code
- Execute portions of your binary from testing harness
  - Ctypes.CDLL
  - Ruby DLL loading
- ???

TodoHT Demo

- Hit-tracing a static, stripped version of the bomb

Other Demos

- ...

Tips on finding super secret key algorithms

- Specialized math instructions
  - Extended instruction set SSE, MMX
- Magic bytes