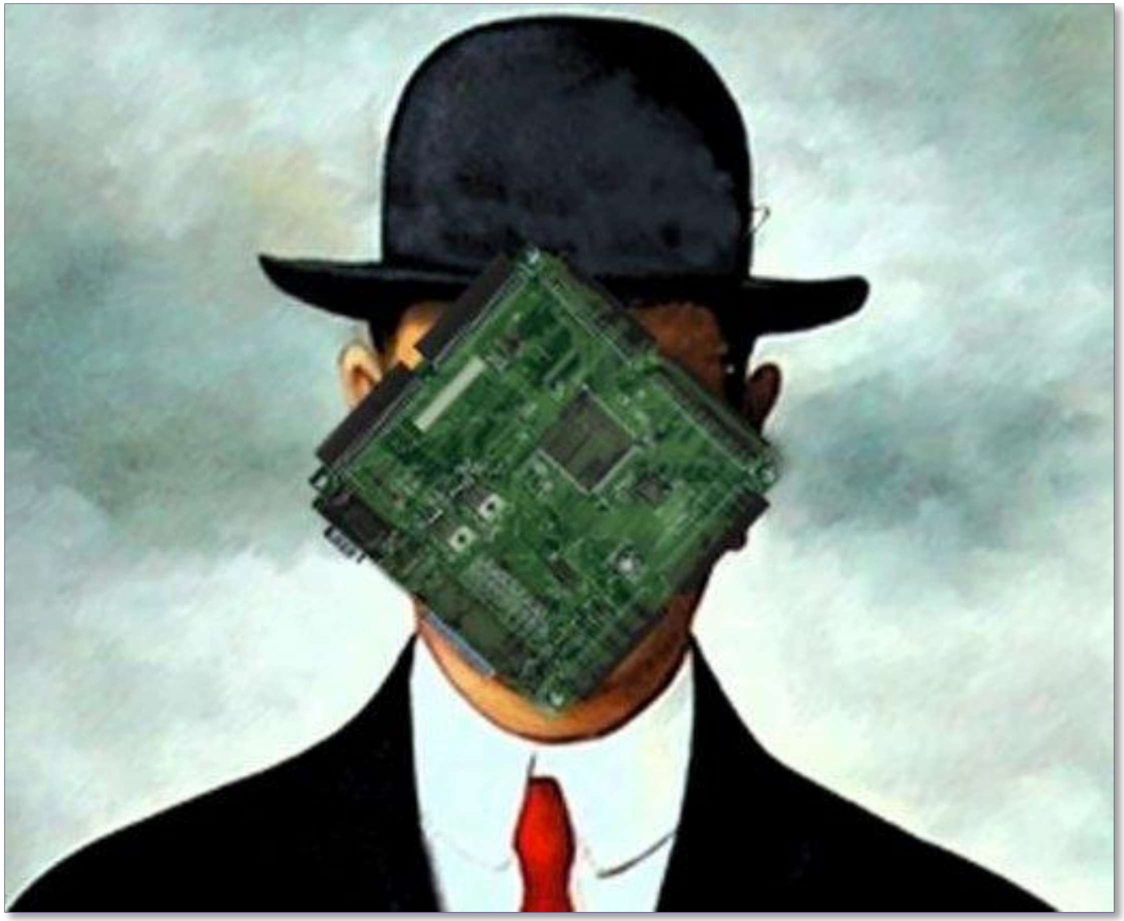
Micro Control
Attacking uC Applications

Don A. Bailey
(donb@isecpartners.com)

whois donb?



What's this uC thing all about?

- Single integrated computer
- Processor, volatile, non-volatile storage
  - All In One
- Can drive many peripherals
- Easily programmable
- Field update/upgrade capability
- Personalization (EEPROM)

No, really… Why do I care?

- Your car
- Implanted medical devices
  - WBAN (Wireless Body Area Network)
- Crops monitoring (hydro/aero/enviro-ponics)
- Infrastructure monitoring (SCADA, etc)
- “Smart Dust”
- Access controls (RFID, biometrics, etc)

Now with More Networking!

- Bluetooth
- USB
- 802.11
- 802.15.4
- RFID
- DECT
- GSM

Security?

- Some tamper resistance
- Hardware security
- From a software point of view?
  - Crypto support
  - …?

OODA Loop?

- Field upgrades are rare
  - But getting more common
  - ST M24LR64 Dual EEPROM (Leet!!)
- Most firmware is legacy code
- Spot updates for new functionality / peripherals
- Mostly written in C, C++, and/or ASM

Why wouldn’t you *PWN* an uC?

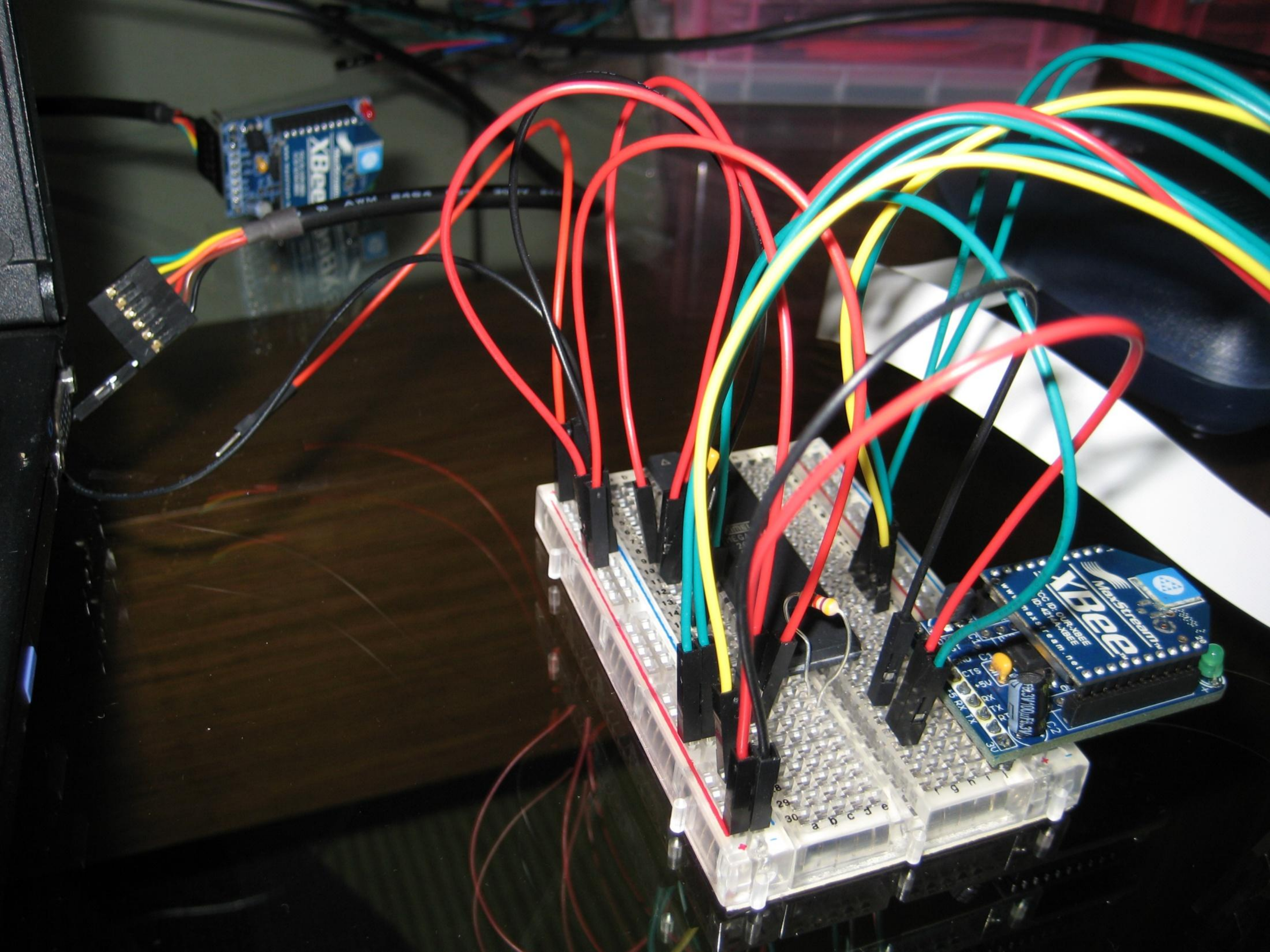
Prior work?

- Travis Goodspeed
  - GoodFET, neighbor!
- Josh Wright
  - Killerbee!

Picking on Atmel AVR8

Lots of uC out there, but…

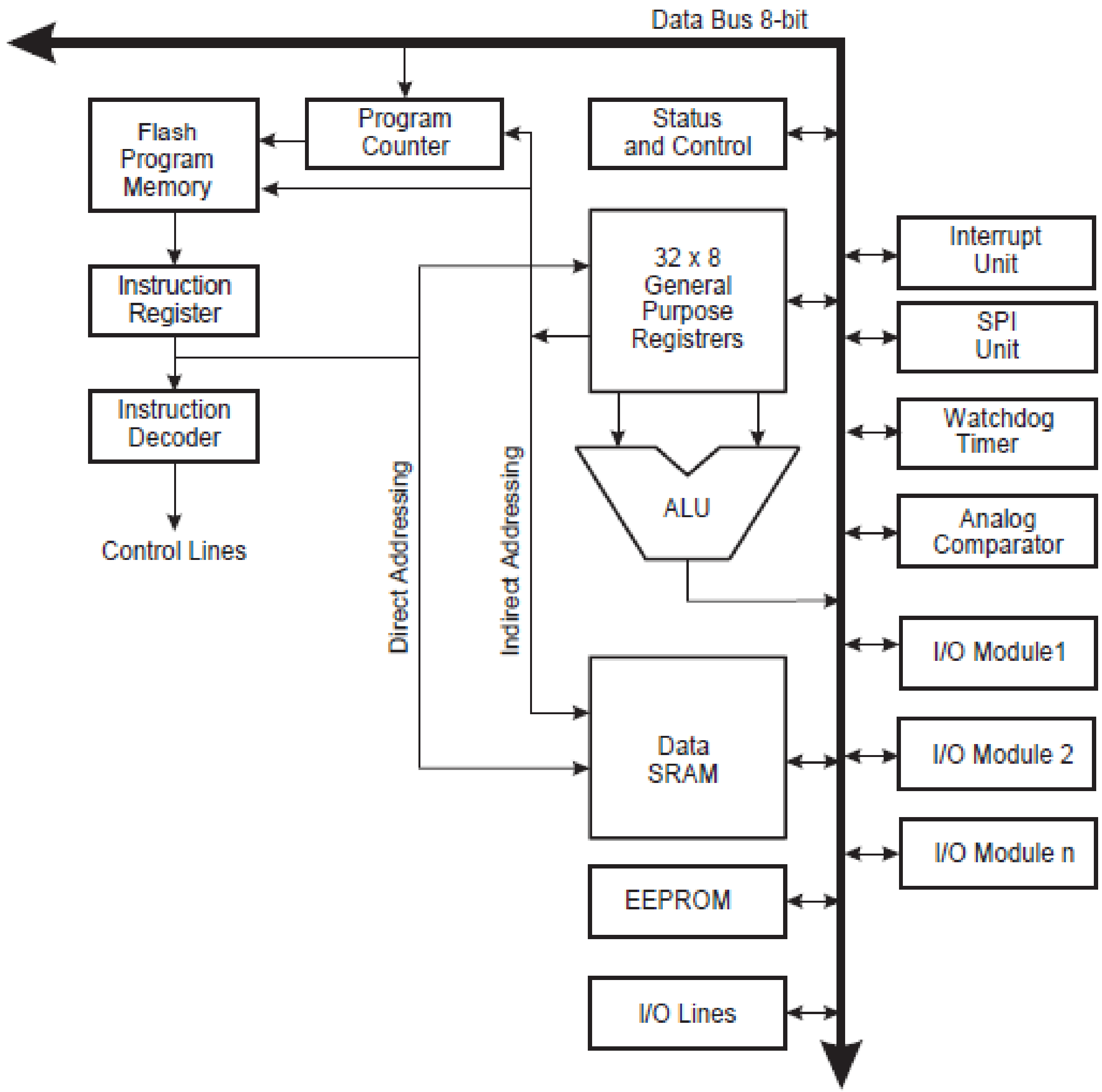
- Popular with hackers *and* engineers
- Free toolchain (gcc based)
- Free IDE (AVR Studio 4)
- No Soldering necessary
- Relatively cheap dev tools
  - AVRISP mkII (~30 USD)
  - AVR JTAGICE mkII
    (good deals from Arrow Electronics)


MaxStream
XBee

Let’s Talk Hardware

Typically included in AVR8

- ALU
- Flash
- SRAM
- EEPROM
- Peripheral support (USART, SPI, I2C, TWI, etc)


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
32 x 8 General Purpose Registrers
Interrupt Unit
SPI Unit
Instruction Decoder
Watchdog Timer
ALU
Control Lines
Direct Addressing
Indirect Addressing
Analog Comparator
I/O Module1
Data SRAM
I/O Module 2
I/O Module n
EEPROM
I/O Lines

That’s right, it’s Harvard

- Separate Data and Code lines
- Code always retrieved from Flash
- Data always retrieved from SRAM
- Flash can be written in software
  - Typically Boot Loader Support
  - Fuses determine this
  - Some AVR8 don’t support this

Point?

- Attack data, not instructions
- Return-to-whatever (ROP :-P)
- Easier! Less data to inject (typically)
- Takes longer
- That's what GoodFET is for
  - Snatch one Smart Dust sensor
  - GoodFET
  - Analyze code
  - Build ROP strategy
  - Own 100 more remotely

Let’s Talk Software

Typical AVR8 Stuff?

- Interrupts
- Atomic Execution (sort of ;-)
- Stack
- 32 8-bit registers
- LSB
- 8/16/32/64-bit integer support
- Access to I/O mem
- RISC

What doesn’t AVR8 have?

- Security boundaries
- Contexts (multiple stacks)
- Concurrency
- Segmentation/Paging
- No atomic instructions (cmpxchg?)
- Native 32/64-bit integer support
- Exceptions
  - Where’s the Page Fault, yo?!

Let’s Talk Program Flow

Typical programmatic flow

- Reset
- Init
- Main
- somefunc

On startup

- AVR sets PC to OxOO in Flash
- OxOO = Reset Vector
- JMP to init in crtO
- Init does stuff…
- Call main
- Do stuff…
- Call somefunc
- Do more stuff…

From RESET -> main()

```
0: File not found
+00000000:   940C003E    JMP     0x0000003E     Jump
+00000002:   940C0053    JMP     0x00000053     Jump
+00000004:   940C0053    JMP     0x00000053     Jump
+00000006:   940C0053    JMP     0x00000053     Jump
+00000008:   940C0053    JMP     0x00000053     Jump
+0000000A:   940C0053    JMP     0x00000053     Jump
+0000000C:   940C0053    JMP     0x00000053     Jump
+0000000E:   940C0053    JMP     0x00000053     Jump
+00000010:   940C0053    JMP     0x00000053     Jump
+00000012:   940C0053    JMP     0x00000053     Jump
+00000014:   940C0053    JMP     0x00000053     Jump
+00000016:   940C0053    JMP     0x00000053     Jump
+00000018:   940C0053    JMP     0x00000053     Jump
+0000001A:   940C0053    JMP     0x00000053     Jump
+0000001C:   940C0053    JMP     0x00000053     Jump
+0000001E:   940C0053    JMP     0x00000053     Jump
+00000020:   940C0053    JMP     0x00000053     Jump
+00000022:   940C0053    JMP     0x00000053     Jump
+00000024:   940C0053    JMP     0x00000053     Jump
+00000026:   940C0053    JMP     0x00000053     Jump
+00000028:   940C0053    JMP     0x00000053     Jump
+0000002A:   940C0053    JMP     0x00000053     Jump
+0000002C:   940C0053    JMP     0x00000053     Jump
+0000002E:   940C0053    JMP     0x00000053     Jump
+00000030:   940C0053    JMP     0x00000053     Jump
+00000032:   940C0053    JMP     0x00000053     Jump
+00000034:   940C0053    JMP     0x00000053     Jump
+00000036:   940C0053    JMP     0x00000053     Jump
+00000038:   940C0053    JMP     0x00000053     Jump
+0000003A:   940C0053    JMP     0x00000053     Jump
+0000003C:   940C0053    JMP     0x00000053     Jump
+0000003E:   2411        CLR     R1             Clear Register
+0000003F:   BE1F        OUT     0x3F,R1        Out to I/O location
+00000040:   EFCF        SER     R28            Set Register
+00000041:   E1D0        LDI     R29,0x10       Load immediate
+00000042:   BFDE        OUT     0x3E,R29       Out to I/O location
+00000043:   BFCD        OUT     0x3D,R28       Out to I/O location
+00000044:   E011        LDI     R17,0x01       Load immediate
+00000045:   E0A0        LDI     R26,0x00       Load immediate
+00000046:   E0B1        LDI     R27,0x01       Load immediate
+00000047:   EFE0        LDI     R30,0xF0       Load immediate
+00000048:   E0F8        LDI     R31,0x08       Load immediate
+00000049:   C002        RJMP    PC+0x0003      Relative jump
+0000004A:   9005        LPM     R0,Z+          Load program memory and postincrement
+0000004B:   920D        ST      X+,R0          Store indirect and postincrement
+0000004C:   38A6        CPI     R26,0x86       Compare with immediate
+0000004D:   07B1        CPC     R27,R17        Compare with carry
+0000004E:   F7D9        BRNE    PC-0x04        Branch if not equal
+0000004F:   940E0098    CALL    0x00000098     Call subroutine
+00000051:   940C0476    JMP     0x00000476     Jump
+00000053:   940C0000    JMP     0x00000000     Jump
```

crt0 Copy of .rodata

Memory

Data | 8/16 | abc. | Address: 0x100

000100	25	64	2E	25	64	2E	25	64	%d.%d.%d
000108	00	25	73	00	64	6F	6E	62	.%s.donb
000110	27	73	20	6D	65	6D	64	75	's memdu
000118	6D	70	20	73	74	61	72	74	mp start
000120	69	6E	67	20	75	70	2E	2E	ing up..
000128	2E	0D	00	30	31	32	33	34	...01234
000130	35	36	37	38	39	61	62	63	56789abc
000138	64	65	66	0D	00	2D	2D	2D	def..---
000140	2D	2D	2D	2D	2D	2D	2D	2D	--------
000148	2D	2D	2D	2D	2D	2D	2D	2D	--------
000150	2D	2D	2D	2D	2D	2D	2D	2D	--------
000158	2D	2D	0D	00	64	75	6D	70	--..dump
000160	69	6E	67	20	30	20	2D	3E	ing 0 ->
000168	20	66	66	66	66	20	77	68	ffff wh
000170	65	72	65	20	52	41	4D	45	ere RAME
000178	4E	44	3D	25	70	0D	00	25	ND=%p..%
000180	63	00	0D	25	2E	30	34	78	c..%.04x
000188	20	00	25	2E	30	32	78	20	.%.02x
000190	00	00	00	00	00	00	00	00	....
000198	00	00	00	00	00	00	00	00	....
0001A0	00	00	00	00	00	00	00	00	....
0001A8	00	00	00	00	00	00	00	00	....
0001B0	00	00	00	00	00	00	00	00	....

Stack Dump After Call to main()

Memory

Data | 8/16 | abc. | Address: 0x1000

```
001000 00 00 00 00 00 00 00 00   ........
001008 00 00 00 00 00 00 00 00   ........
001010 00 00 00 00 00 00 00 00   ........
001018 00 00 00 00 00 00 00 00   ........
001020 00 00 00 00 00 00 00 00   ........
001028 00 00 00 00 00 00 00 00   ........
001030 00 00 00 00 00 00 00 00   ........
001038 00 00 00 00 00 00 00 00   ........
001040 00 00 00 00 00 00 00 00   ........
001048 00 00 00 00 00 00 00 00   ........
001050 00 00 00 00 00 00 00 00   ........
001058 00 00 00 00 00 00 00 00   ........
001060 00 00 00 00 00 00 00 00   ........
001068 00 00 00 00 00 00 00 00   ........
001070 00 00 00 00 00 00 00 00   ........
001078 00 00 00 00 00 00 00 00   ........
001080 00 00 00 00 00 00 00 00   ........
001088 00 00 00 00 00 00 00 00   ........
001090 00 00 00 00 00 00 00 00   ........
001098 00 00 00 00 00 00 00 00   ........
0010A0 00 00 00 00 00 00 00 00   ........
0010A8 00 00 00 00 00 00 00 00   ........
0010B0 00 00 00 00 00 00 00 00   ........
0010B8 00 00 00 00 00 00 00 00   ........
0010C0 00 00 00 00 00 00 00 00   ........
0010C8 00 00 00 00 00 00 00 00   ........
0010D0 00 00 00 00 00 00 00 00   ........
0010D8 00 00 00 00 00 00 00 00   ........
0010E0 00 00 00 00 00 00 00 00   ........
0010E8 00 00 00 00 00 00 00 00   ........
0010F0 00 00 00 00 00 00 00 00   ........
0010F8 00 00 00 00 00 00 00 51   .......Q
```

Function Call

```
@000000E5: main
57:        {
+000000E5:   93DF        PUSH      R29              Push register on stack
+000000E6:   93CF        PUSH      R28              Push register on stack
+000000E7:   D000        RCALL     PC+0x0001        Relative call subroutine
+000000E8:   B7CD        IN        R28,0x3D         In from I/O location
+000000E9:   B7DE        IN        R29,0x3E         In from I/O location
58:            dummy(0x7f, 0x8f, 0x9f);
+000000EA:   E78F        LDI       R24,0x7F         Load immediate
+000000EB:   E090        LDI       R25,0x00         Load immediate
+000000EC:   E86F        LDI       R22,0x8F         Load immediate
+000000ED:   E070        LDI       R23,0x00         Load immediate
+000000EE:   E94F        LDI       R20,0x9F         Load immediate
+000000EF:   E050        LDI       R21,0x00         Load immediate
+000000F0:   940E0082    CALL      0x00000082       Call subroutine
60:            return run() ? True : False ;
```

Frame Setup/Teardown

```
+00000096:   93DF        PUSH    R29             Push register on stack
+00000097:   93CF        PUSH    R28             Push register on stack
+00000098:   B7CD        IN      R28,0x3D        In from I/O location
+00000099:   B7DE        IN      R29,0x3E        In from I/O location
+0000009A:   97A0        SBIW    R28,0x20        Subtract immediate from word
+0000009B:   B60F        IN      R0,0x3F         In from I/O location
+0000009C:   94F8        CLI                     Global Interrupt Disable
+0000009D:   BFDE        OUT     0x3E,R29        Out to I/O location
+0000009E:   BE0F        OUT     0x3F,R0         Out to I/O location
+0000009F:   BFCD        OUT     0x3D,R28        Out to I/O location
42:          strcpy(x, o);
+000000A0:   01CE        MOVW    R24,R28         Copy register pair
+000000A1:   9601        ADIW    R24,0x01        Add immediate to word
+000000A2:   940E0214    CALL    0x00000214      Call subroutine
+000000A4:   818B        LDD     R24,Y+3         Load indirect with displacement
+000000A5:   819C        LDD     R25,Y+4         Load indirect with displacement
+000000A6:   2789        EOR     R24,R25         Exclusive OR
+000000A7:   2F28        MOV     R18,R24         Copy register
+000000A8:   E030        LDI     R19,0x00        Load immediate
+000000A9:   E040        LDI     R20,0x00        Load immediate
+000000AA:   E050        LDI     R21,0x00        Load immediate
+000000AB:   8D8F        LDD     R24,Y+31        Load indirect with displacement
+000000AC:   2388        TST     R24             Test for Zero or Minus
+000000AD:   F411        BRNE    PC+0x03         Branch if not equal
+000000AE:   E041        LDI     R20,0x01        Load immediate
+000000AF:   E050        LDI     R21,0x00        Load immediate
+000000B0:   0F24        ADD     R18,R20         Add without carry
+000000B1:   1F35        ADC     R19,R21         Add with carry
45:        }
+000000B2:   01C9        MOVW    R24,R18         Copy register pair
+000000B3:   96A0        ADIW    R28,0x20        Add immediate to word
+000000B4:   B60F        IN      R0,0x3F         In from I/O location
+000000B5:   94F8        CLI                     Global Interrupt Disable
+000000B6:   BFDE        OUT     0x3E,R29        Out to I/O location
+000000B7:   BE0F        OUT     0x3F,R0         Out to I/O location
+000000B8:   BFCD        OUT     0x3D,R28        Out to I/O location
+000000B9:   91CF        POP     R28             Pop register from stack
+000000BA:   91DF        POP     R29             Pop register from stack
+000000BB:   9508        RET                     Subroutine return
@000000BC: dummy
49:        {
+000000BC:   92CF        PUSH    R12             Push register on stack
+000000BD:   92DF        PUSH    R13             Push register on stack
+000000BE:   92EF        PUSH    R14             Push register on stack
+000000BF:   92FF        PUSH    R15             Push register on stack
+000000C0:   930F        PUSH    R16             Push register on stack
+000000C1:   931F        PUSH    R17             Push register on stack
+000000C2:   93DF        PUSH    R29             Push register on stack
+000000C3:   93CF        PUSH    R28             Push register on stack
+000000C4:   B7CD        IN      R28,0x3D        In from I/O location
+000000C5:   B7DE        IN      R29,0x3E        In from I/O location
+000000C6:   97A0        SBIW    R28,0x20        Subtract immediate from word
+000000C7:   B60F        IN      R0,0x3F         In from I/O location
+000000C8:   94F8        CLI                     Global Interrupt Disable
+000000C9:   BFDE        OUT     0x3E,R29        Out to I/O location
+000000CA:   BE0F        OUT     0x3F,R0         Out to I/O location
+000000CB:   BFCD        OUT     0x3D,R28        Out to I/O location
+000000CC:   018C        MOVW    R16,R24         Copy register pair
```

Four Main Points Demonstrated…

- Function conventions are typical
  - Optimization may minimize this
- Code Layout
- Data Layout
- Atomic Code Sections

Code Layout in Flash

- Interrupt Vectors at OxOO
- RESET Vector at OxOO
- Main Application Code
- Data (???)
- Boot Loader Section
    - Can write to Flash (if Fuses allow) for field updates

Data Layout in SRAM

- Registers at 0x00
- I/O Memory at 0x20
- Extended I/O Memory
- Data (copied from Flash) at 0x100
- BSS
- Heap
- Stack
- ??? ;-)

Atomicity

- CLI used
- SREG can be accessed via SRAM (I/O memory)
- 1 CPU Cycle to write to SREG
- Flow:
  - Save a copy of SREG
  - Clear Interrupt Bit in SREG
  - Perform uninterrupted action
    - Write to low byte of SP
    - Write to SREG (old state with interrupt bit set)
    - Write to high byte of SP

Now, Let’s Have Some *Real* Fun

Entropy? What entropy?

- Randomness is very weak
- Crypto hurt as a result
- Pools can be accumulated
  - "True Random Number Generator On an Atmel uC" – IEEE Paper
- 8 Random Bits using RC oscillator
  - *Per second!!!*

Race Conditions

- No semblance of context switching
  - TinyOS/Contiki simulate it
- Critical Sections secured through CLI
- Attack these sections
  - Overwrite SREG; enable Interrupts
- Use Interrupts to cause unexpected behavior

Return Value Checks

- Snprintf returning <=O or >= sizeof buf?
- Logic Issue
- Always a problem

memcpy and Friends

- Latest avr-libc
- Don’t test for negative size values
- No option to “secure” with CLI
  - Interruptable
  - Oops…Where’d my SP go?! ;-)

Buffer Overflows

- Easy as pie
- Instruction address in mem is /2
- Return Oriented Programming
  - Get those Registers set up correctly!
- Force a jump to the Boot Loader
- Instant Flash update (simulate field update)
- Can be triggered remotely
- AVR doesn’t know the difference between you and developer

Frame Pointer Overwrite

- Standard FP overwrite
- Point stack to attacker controlled data
- Next frame has the RET
- FP saved LSB first

Setjmp

- Obvious target
- Often used
- Makes up for lack of exceptions
- Saves entire program state
- Overwrite all registers
- Overwrite PC

Integer Overflows

- Work as expected
- 8-bit registers
- 16-bit native instructions
- Easy to wrap OxFFFF

Integer Promotion

- Normal integer promotion
- Unsigned -> Signed = No Sign Extension
- Signed -> Signed = Sign Extension
- Stop using ‘char’ for everything ;-)
- Lots of 8-bit networking protocols
  - 8-bit size fields
  - Promoted to *int* during packet ingestion
  - Oops!!

Heap Overflows

- Heap Struct consists of { size, Next* }
- Next* points to the next free heap chunk
- Adjacent chunks are combined
- No function pointers ☹
- Easily mangle data
- Next* doesn't have to point to Heap ☺
- Heap data isn't zeroed on free()
- Easy way to create pseudo stack frames
- ROP Helper!

Double Free

- Latest avr-libc free() doesn't check
- Any address can be used (except NULL)
- Free() will happily overwrite first 2 bytes with
  - Next*
- Add it to the free list ;-)
- Can stealthily force malloc() to return (void*)0x00
- Write direct to Registers, I/O memory, etc
- ROP Helper!!

“Segment” Collision

- Heap is allocated slightly under stack
- Stack is dynamic!!!
- BSS is adjacent to Heap
- .rodata isn’t Read Only! Adjacent to BSS
- One big happy family!

Uninitialized Variables

- Allocate a large Heap chunk
- Spray with OxAABB
- Stack decends into Heap
- Bewm!
- Example code at:
  - http://pa-ri.sc/uC/dangle.tar.bz2

Format Strings

- Current avr-libc has no %n support
- No fun ☹
- But, kind of reasonable

NULL Pointer Dereferences

- There are no privilege rings, but still useful
- Functions like malloc() still return NULL
- (void*)OxOO points to Registers in SRAM
- NULL deref is a very good thing
- Like free() bug, instant access to Regs, I/O Mem
- On the flip side…
  - ??? ;-)

Beyond Memory

- Deref beyond physical memory addresses?
- Example: ATmega644P
  - 4096 bytes SRAM
  - Total 4196 addressable bytes
    - With registers, I/O memory
- Ox1OFF should be highest addressible address

```
1050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1060 00 00 00 00 00 00 00 00 00 00 00 10 70 02 40 01  .............P...
1070 ed 64 36 30 31 00 00 00 00 00 00 00 10 8f 10 a4  ..a01...........
1080 00 10 10 80 0a be 01 7e 01 73 01 76 00 00 01 ac  .......~.s.v....
1090 aa 10 c4 06 3f 00 03 00 1e 00 00 00 00 00 10 a3  ....?...........
10a0 10 a4 00 70 37 37 20 00 30 20 00 20 30 20 2d 3e  ...pp. .0 . 0 ->
10b0 20 66 66 66 66 20 77 68 65 72 65 20 52 41 4d 45   ffff where RAME
10c0 4e 44 3d 30 78 31 30 66 66 0d 00 00 00 00 00 00  ND=0x10ff.......
10d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
10e0 00 00 00 00 2e 00 10 e7 01 65 7e 01 01 54 10 ff  ..........3s.....
10f0 01 0d 00 10 16 00 0a be 01 7e 01 73 01 76 00 51  .........~.s.v.Q
1100 64 6f 6e 62 27 73 20 6d 65 6d 64 75 6d 70 20 73  donb's memdump s
1110 74 61 72 74 69 6e 67 20 75 70 2e 2e 2e 0d 00 30  tarting up.....0
1120 31 32 33 34 35 36 37 38 39 61 62 63 64 65 66 0d  123456789abcdef.
1130 00 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d  .---------------
1140 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 2d 0d 00  --------------..
1150 64 75 6d 70 69 6e 67 20 30 20 2d 3e 20 66 66 66  dumping 0 -> fff
1160 66 20 77 68 65 72 65 20 52 41 4d 45 4e 44 3d 25  f where RAMEND=%
1170 70 0d 00 25 63 00 0d 25 2e 30 34 78 20 00 25 2e  p..%c..%.04x .%.
1180 30 32 78 20 00 00 00 00 00 00 00 00 00 00 00 00  02x ............
1190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
11a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Memory

Data | 8/16 | abc. | Address: 0x1000

001000	00	00	00	00	00	00	00	00	....
001008	00	00	00	00	00	00	00	00	....
001010	00	00	00	00	00	00	00	00	....
001018	00	00	00	00	00	00	00	00	....
001020	00	00	00	00	00	00	00	00	....
001028	00	00	00	00	00	00	00	00	....
001030	00	00	00	00	00	00	00	00	....
001038	00	00	00	00	00	00	00	00	....
001040	00	00	00	00	00	00	00	00	....
001048	00	00	00	00	00	00	00	00	....
001050	00	00	00	00	00	00	00	00	....
001058	00	00	00	00	00	00	00	00	....
001060	00	00	00	00	00	00	00	00	....
001068	00	00	00	00	00	00	00	00	....
001070	00	00	00	00	00	00	00	00	....
001078	00	00	00	00	00	00	00	00	....
001080	00	00	00	00	00	00	00	00	....
001088	00	00	00	00	00	00	00	00	....
001090	00	00	00	00	00	00	00	00	....
001098	00	00	00	00	00	00	00	00	....
0010A0	00	00	00	00	00	00	00	00	....
0010A8	00	00	00	00	00	00	00	00	....
0010B0	00	00	00	00	00	00	00	00	....
0010B8	00	00	00	00	00	00	00	00	....
0010C0	00	00	00	00	00	00	00	00	....
0010C8	00	00	00	00	00	00	00	00	....
0010D0	00	00	00	00	00	00	00	00	....
0010D8	00	00	00	00	00	00	00	00	....
0010E0	00	00	00	00	00	00	00	00	....
0010E8	00	00	00	00	00	00	00	00	....
0010F0	00	00	00	00	00	00	00	00	....
0010F8	00	00	00	00	00	00	00	51	.......Q

Memory

Data | 8/16 | abc. | Address: 0x100

```
000100 25 64 2E 25 64 2E 25 64  %d.%d.%d
000108 00 25 73 00 64 6F 6E 62  .%s.donb
000110 27 73 20 6D 65 6D 64 75  's memdu
000118 6D 70 20 73 74 61 72 74  mp start
000120 69 6E 67 20 75 70 2E 2E  ing up..
000128 2E 0D 00 30 31 32 33 34  ...01234
000130 35 36 37 38 39 61 62 63  56789abc
000138 64 65 66 0D 00 2D 2D 2D  def..---
000140 2D 2D 2D 2D 2D 2D 2D 2D  --------
000148 2D 2D 2D 2D 2D 2D 2D 2D  --------
000150 2D 2D 2D 2D 2D 2D 2D 2D  --------
000158 2D 2D 0D 00 64 75 6D 70  --..dump
000160 69 6E 67 20 30 20 2D 3E  ing 0 ->
000168 20 66 66 66 66 20 77 68   ffff wh
000170 65 72 65 20 52 41 4D 45  ere RAME
000178 4E 44 3D 25 70 0D 00 25  ND=%p..%
000180 63 00 0D 25 2E 30 34 78  c..%.04x
000188 20 00 25 2E 30 32 78 20   .%.02x
000190 00 00 00 00 00 00 00 00  ........
000198 00 00 00 00 00 00 00 00  ........
0001A0 00 00 00 00 00 00 00 00  ........
0001A8 00 00 00 00 00 00 00 00  ........
0001B0 00 00 00 00 00 00 00 00  ........
```

There is no Page Fault on AVR8

- Memory faults cannot occur
- For program safety, don’t RESET
- Read AND Write support
- Just wrap addresses back to (void*)0x00
- Overwriting past end of PHYSMEM = start of PHYSMEM
- i.e. 0x1100 = 0x0100
- How convenient ;-)
- Overwrite EVERYTHING ANYWHERE

Example code?

- See the memdump application
  - Runs on any AVR8 with USART
  - http://pa-ri.sc/uC/memdump.tar.bz2

- Code tested on 10 different uCs in the AVR family
  - ATtiny
  - ATmega

We Pack and Deliver like UPS Trucks

Summary?

- Ripe environment for application vulnerabilities
- Little protection schemes
  - Except solid auditing and a tight SDLC
- Lots of legacy code in the field
- Lots of important devices

Thanks for your support…

- Dhillon Kannabhiran
- iSEC Partners
- Nick DePetrillo
- Mike Kershaw
- Travis Goodspeed
- Josh Wright
- Dennis Brown
- Dan Guido
- David Munson

Terima kasih!

@DonAndrewBailey
donb@isecpartners.com