Day 1 - Plan
Day 2 - Reverse
Day 3 - Own
day one the plan
ASSIGNMENT

1. FIND A TARGET

2. FIND AN ATTACK ANGLE

3. FORM A CHAIN OF ATTACK

Name
LESTER CREST

Expertise
PLANNING

Favorite porn star
STEFAN ESSER
This is where all the loaded people come to have fun.
Here’s a photo from the inside. They store their cash in that POS terminal.
Time to refresh my memory on how these things are protected
POS: Ingredients
POS: Ingredients

• **Terminal**

No direct access to cash
POS: Ingredients

• Card reader
  Heavily protected
POS: Ingredients

- Cashier
Expensive
to bribe
Is that it? Hit rewind, I’m sure we missed something.
What’s that steel box over there?
POS: Ingredients

- Cash drawer
  Just a dumb box
  ...or is it?
A Modern POS

...especially popular in bars and restaurants
APG NetPRO 488

• Most popular wireless model
• Connects over WiFi...
• To the INTERNET OF THINGS
Wait a minute... close up on that part.
Gentlemen... I believe we have a target
Let's get a device and crack it open
Give me a close up of those two chips
Atmel
Atmega 1284p

128k Flash
16k RAM

Wiznet
WizFi210

AT over serial

0008DC1B1058
WIZnet Co., Ltd. Made in Korea
Model: WizFi210 FCC Class B
FCC ID: XR2WIZFI210
KCC ID: KCC-CRM-WWW-WIZFI210

CE 1177
Get the firmware - options

• No firmware online
• Play with UART?
• Extract from MCU? (AKA Suicide)
• Ask the manufacturer... nah!
i can haz firmware?

Hi,

You can find the firmware here:
day two reverse the firmware
ASSIGNMENT

1. REVERSE THE BINARY

2. FIND A BUG

Name
PAIGE HARRIS

Expertise
Reverser

Favorite film
HOW I MET YOUR SKOCHINSKY
I heard that reversing Atmel code is a mindfuck because of these issues:

- Inconsistent register naming
- Creepy Harvard architecture
- Find xrefs to debug strings
Let’s deal with this sucker first:

- Inconsistent register naming
$ rasm2 -d fw.bin

ldi r30, 0x15
ldi r31, 0xE
st r20, Z

ldi - load immediate into register
st - store register into byte at address
Dafuq did I just see?

What does Z stand for?

Zebra?  Zorg?  Zimbabwe?
loc_4D7A:
  mov    r18, r16
  or     r18, r17
  breq   loc_4D89

loc_4D89:
  ldi    r18, 0
  st     X, r18

loc_4D8B:
  ldi    r30, 6
  jmp    FuncTerm_3

; End of function na_StrnCpySRAMToSRAM
AVR Programmer Manual:

In order to enable 16-bit addressing, the last six registers are paired to form registers X, Y and Z:

\[
\begin{align*}
    r26:27 & - X \\
r28:29 & - Y \\
r30:31 & - Z
\end{align*}
\]
$ rasm2 -d fw.bin

ldi r30, 0x15
ldi r31, 0xE
// Z is now 0xE15
st r20, Z

ldi - load immediate into register
st - store register into byte at address
Nailed it! But where the hell are the strings?

• Inconsistent register naming
• Creepy Harvard architecture
• Find xrefs to debug strings
I have a hunch that solving the next challenge will help:

• Creepy Harvard architecture
AVR Programmer Manual:

- 128K Flash
- Program Address Space
- 4K EEPROM
- CPU
- 16K RAM
- Data
- 8-Bit
Got it! Now I know how to find string refs.

- Inconsistent register naming
- Creepy Harvard architecture
- Find xrefs to debug strings
StrLen_PM:

<..>

adiw r30, 1
lpm r20, Z
tst r20
breq Return
<..>

adiw - add immediate to register pair
lpm - load byte from program memory

StrLen_RAM:

<..>

adiw r30, 1
ld r20, Z
tst r20
breq Return
<..>
r16:17 == 0xdae
Word addressing: 0x6d7

ROM: 06D7 6F65 6570 646E 6172+aEopendrawer: .db "eopendrawer", 0
<table>
<thead>
<tr>
<th>Function name</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub_FA5</td>
</tr>
<tr>
<td>sub_F69</td>
</tr>
<tr>
<td>sub_F2</td>
</tr>
<tr>
<td>sub_EA2</td>
</tr>
<tr>
<td>sub_E4D</td>
</tr>
<tr>
<td>sub_DF5</td>
</tr>
<tr>
<td>sub_D68</td>
</tr>
<tr>
<td>sub_AF</td>
</tr>
<tr>
<td>sub_9B</td>
</tr>
<tr>
<td>sub_85</td>
</tr>
<tr>
<td>sub_73</td>
</tr>
<tr>
<td>sub_6CEE</td>
</tr>
<tr>
<td>sub_6CEA</td>
</tr>
<tr>
<td>sub_6CD3</td>
</tr>
<tr>
<td>sub_6CA3</td>
</tr>
<tr>
<td>sub_6CA0</td>
</tr>
<tr>
<td>sub_6B</td>
</tr>
<tr>
<td>sub_693A</td>
</tr>
<tr>
<td>sub_68E2</td>
</tr>
<tr>
<td>sub_65EC</td>
</tr>
<tr>
<td>sub_65B7</td>
</tr>
<tr>
<td>sub_646F</td>
</tr>
<tr>
<td>sub_6436</td>
</tr>
<tr>
<td>sub_64</td>
</tr>
<tr>
<td>sub_63F7</td>
</tr>
<tr>
<td>sub_63E6</td>
</tr>
<tr>
<td>sub_63CF</td>
</tr>
<tr>
<td>na_HandleUSART_2</td>
</tr>
<tr>
<td>na_HandleUSART</td>
</tr>
<tr>
<td>na_GetDataOffset</td>
</tr>
<tr>
<td>naGenerateClosingXMLTag</td>
</tr>
<tr>
<td>naGenerateASCIIPortNumber</td>
</tr>
<tr>
<td>na_FreeLan</td>
</tr>
<tr>
<td>na_ForceFreeLAN</td>
</tr>
<tr>
<td>na_EventIP</td>
</tr>
<tr>
<td>na_EnableDHCP</td>
</tr>
<tr>
<td>na_DrawerOpenText</td>
</tr>
<tr>
<td>na_DrawerCloseText</td>
</tr>
<tr>
<td>na_DoRecoverAPIPA</td>
</tr>
<tr>
<td>na_DoNothing</td>
</tr>
<tr>
<td>na_DoFactoryDefault</td>
</tr>
<tr>
<td>na_DoConnectWrapper</td>
</tr>
<tr>
<td>na_DoConnect</td>
</tr>
<tr>
<td>na_DoCfgWPS</td>
</tr>
<tr>
<td>na_DoCfgDHCP</td>
</tr>
<tr>
<td>na_DoCfgBasic</td>
</tr>
<tr>
<td>na_DoCfgAssoc</td>
</tr>
<tr>
<td>na_DispatchMessageOrCallDisassociationEvent</td>
</tr>
<tr>
<td>na_Disconnect</td>
</tr>
<tr>
<td>na_DisassociationEvent</td>
</tr>
<tr>
<td>na_DisableDHCP</td>
</tr>
<tr>
<td>na_DecrementCounter</td>
</tr>
<tr>
<td>na_DecantRStack</td>
</tr>
<tr>
<td>na_CopyWizFiHeaderToResponseBuffer</td>
</tr>
</tbody>
</table>
Now that I got the debug strings, let’s look at the attack surface

• Inconsistent register naming
• Creepy Harvard architecture
• Find xrefs to debug strings
function register() {
    if (!$empty($POST)) {
        $msg = ''; 
        if (!$_POST['user_name']) {
            if (!$_POST['user_password_new']) {
                if ($_POST['user_password_new'] == $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Password must be at least 6 characters';
                    } else $msg = 'Password do not match';
                } else $msg = 'Empty Password';
            } else $msg = 'Empty Username';
        }
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
day
three
pwn all the things
Name: TREVOR PHILIPS
Expertise: MAYHEM
Favorite tool: DIE

ASSIGNMENT
1. Find a bug
2. Exploit it
3. Get gold
Mystery: Who wrote their libc, and when?
strlen walks until a NULL is reached
strcpy doesn’t add a NULL byte to the end of the string
Using these two primitives we can get code execution
Where to write into?
Stack return address is stored at beginning of RAM
Three stage pwn

1. Log header buffer
   - Flags (set to non-0)
   - strlcn returns wrong val

2. memcpy with value of strlen
   - Overwrite pointer
   - Trigger write to stack

3. Overwrite return address
   - Build ROP Chain
   - Trigger drawer open
Um, I think you’ve missed something
The Money Function

```assembly
939A  st    -Y, r25
938A  st    -Y, r24
B79F  in    r25, SREG
94F8  cli
2F80  mov   r24, r16
9380 0E5A  sts  unk_100E5A, r24
9828  cbi   ACH, 0
E302  ldi   r16, 0x32 ; '2'
E010  ldi   r17, 0
9300 0E58  sts  unk_100E58, r16
9310 0E59  sts  unk_100E59, r17
E200  ldi   r16, 0x20 ; '+'
EB1F  ldi   r17, 0xBF ; '+'
E022  ldi   r18, 2
E030  ldi   r19, 0
E5EF  ldi   r30, 0x5F ; '_'
E0FE  ldi   r31, 0xE
8300  st    Z, r16
8311  std   Z+1, r17
8322  std   Z+2, r18
8333  std   Z+3, r19
DFDA  rcall na_SendSignalToEngine
E500  ldi   r16, 0x50 ; 'P'
E010  ldi   r17, 0
9300 0E5D  sts  unk_100E5D, r16
9310 0E5E  sts  unk_100E5E, r17
E001  ldi   r16, 1
9300 0E64  sts  unk_100E64, r16
BF9F  out   SREG, r25 ; Status Register
9189  ld    r24, Y+  
9199  ld    r25, Y+  
9508  ret

; End of function na_OpenCashDrawer
```
They forgot to check credentials!
the heist
Ready for the job of a lifetime? Here’s the target.
We have one gun on the spot to trigger the open cafe.
And another gun to grab the cash when it’s open
...THIS IS IT! Go for it
questions?