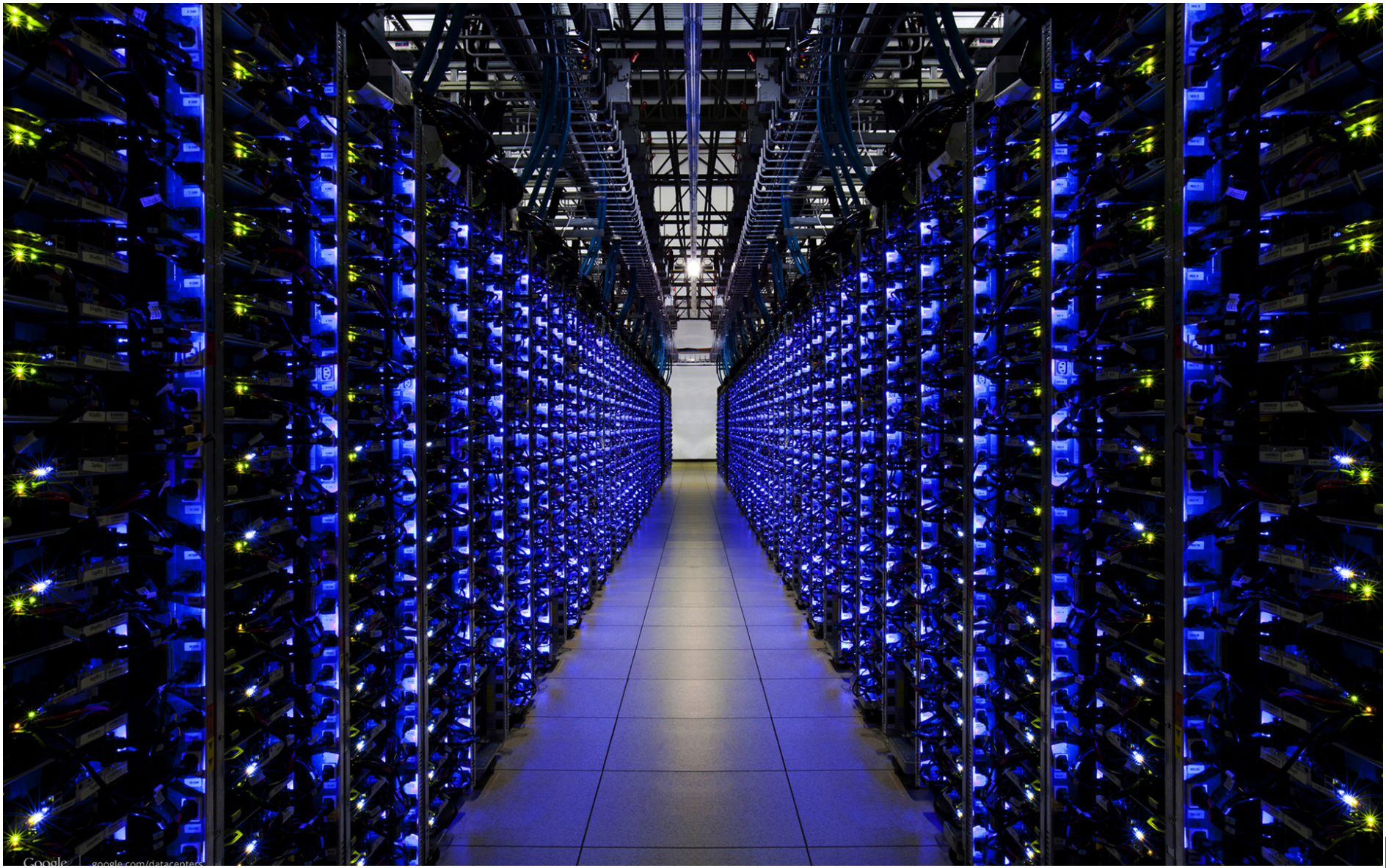
Side Channel Analysis

Drew Monroe

February 14, 2017

Outline

Traditionally, our cybersecurity talks focus on things like these...

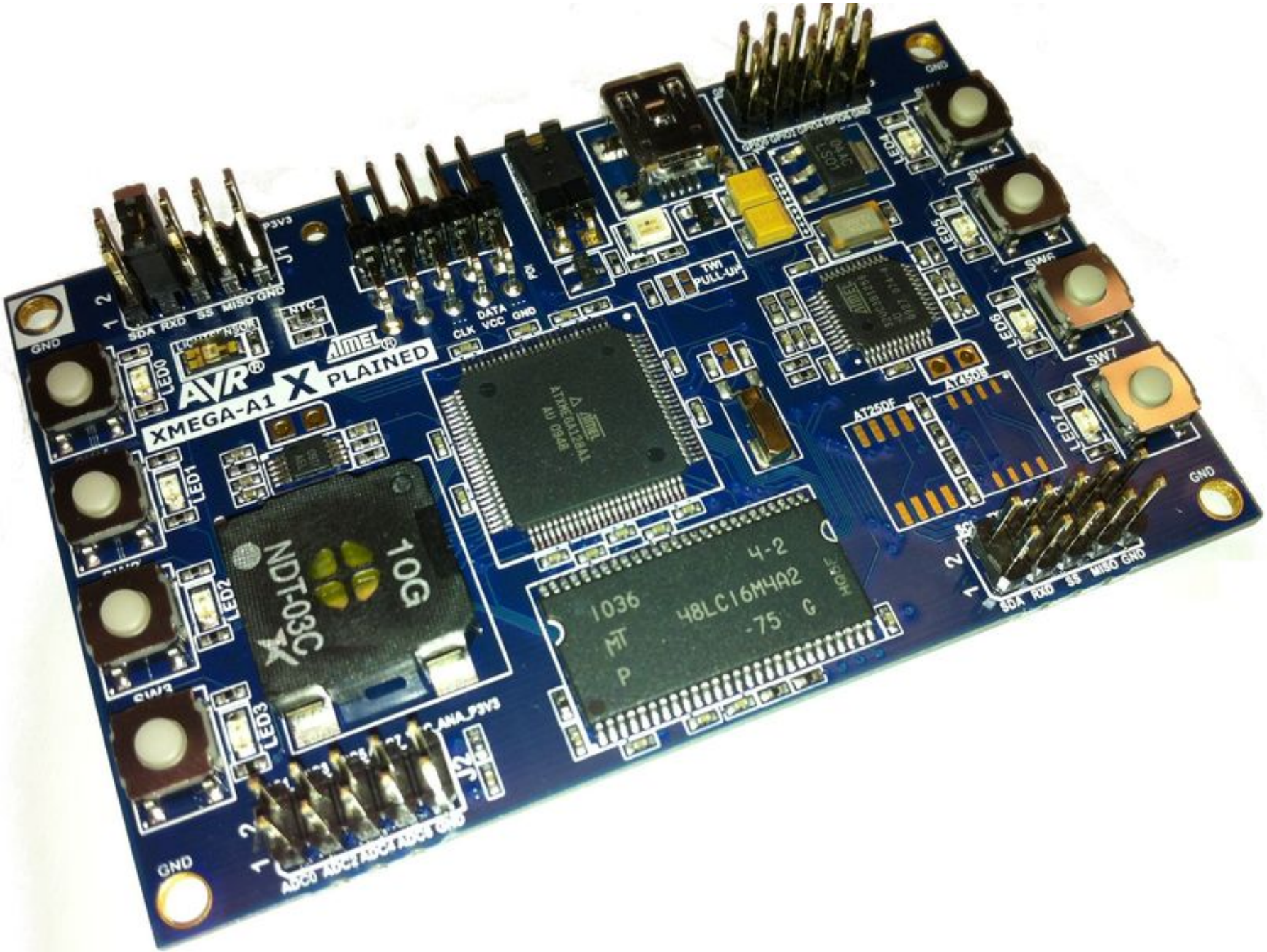


[1]


[2]

[3]

But now, let's think about things like these...


AVR
XMEGA-A1 XPLAINED
ATMEL
LED0
LED1
LED2
LED3
LED4
LED5
LED6
LED7
SW6
SW7
TWI PULL-UP
GND
SDA RXD SS MISO GND
DATA CLK VCC GND
NDT03C
10G
1036
MT
P
48LC16M4A2
-75
G
A1250F
A4450F

[4]

Your computer is made up of hardware after all!

In short, what if an attacker has physical access to your system?

Information Leakage

- You are leaking information about you behaviors all of the time!
  - Sound (fans)
  - Heat (GPS, intensive calculations, etc.)
  - **Power**
- You can think of your computer having two inputs, and two outputs

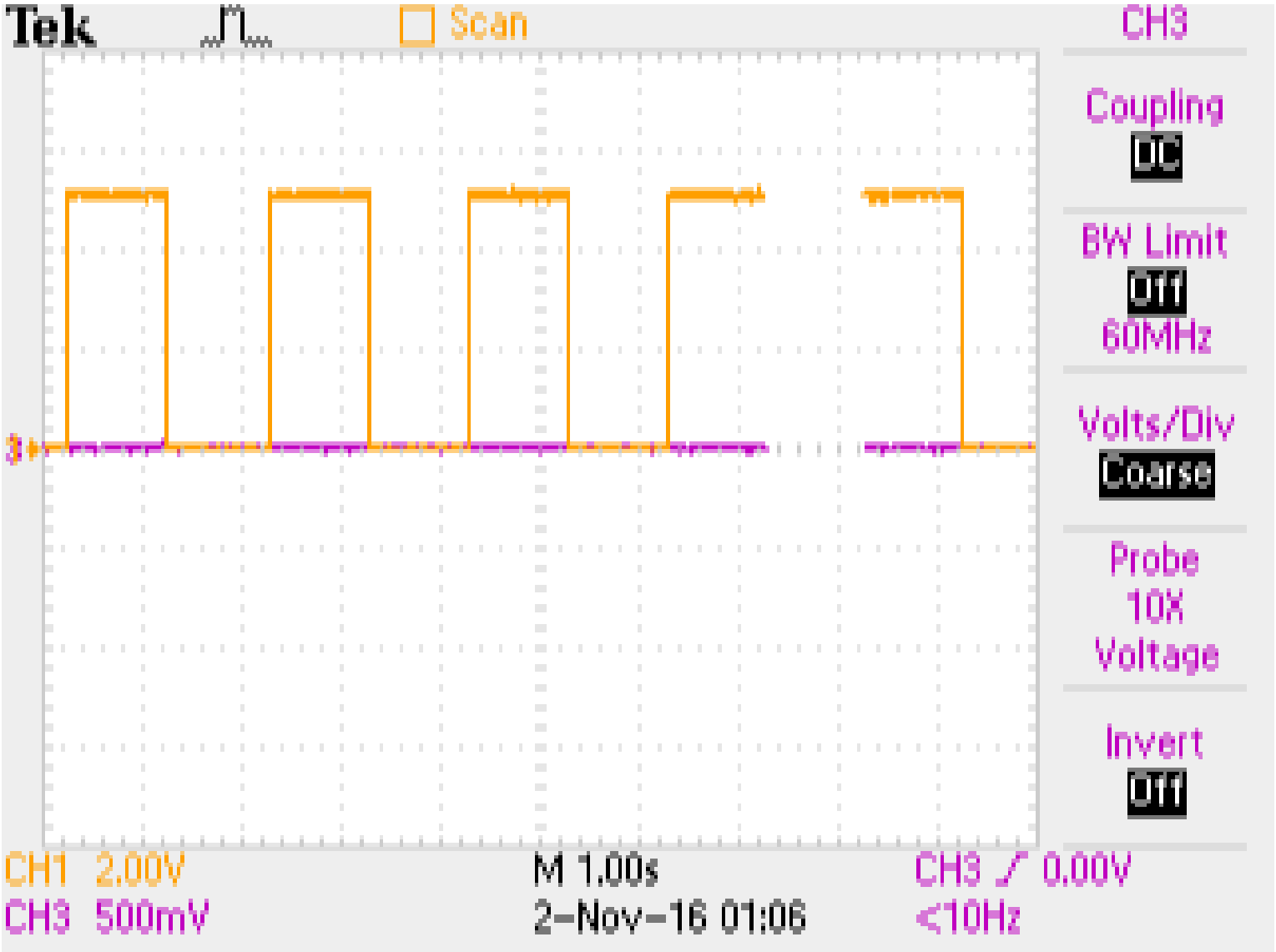
Okay, but so what?

- This information leakage can indicate what you are doing and when you are doing it.
  - Think outside of computing first: your home, the pool, ambient noise
  - Now think small scale: using a GPS, decrypting an encrypted message, hashing a password
- Remember, someone has gained physical access to your system! They can use this information to do malicious things

Okay Drew, that's great, this sounds like it could potentially be a problem. But how easy is this to do?

It's so easy, even **YOU** can do it!

LED Power Trace



Okay, cool, that was pretty easy! But that was just an LED. It doesn't matter if someone knows that my LED is blinking...

What does this show?

- Periodically, the chip is consuming more power, and we can see when this happens.
- What are some things this could indicate?
  - New email
  - Voicemail
  - Fire alarm

So how could someone exploit this?

Fault Injection

- Cause a malfunction in the computer chip
- Goal: small errors with a large impact
- What causes a fault?
  - Heat
  - Radiation
  - Electrical noise
  - Electromagnetic pulse
  - Changes in voltage
  - Electrical noise
- What can someone who has access to your code do? Lots of bad things...

Let's write some C!

Password Checks

C

```c
char* password = "12345";
char* hash = md5(password);
char* stored_hash = getStoredHash();

int checkPassword(char* hash,
    char* stored_hash)
{
    if (strcmp(hash, stored_hash)
        == 0)
    {
        return letUserIn();
    }
    else
    {
        return getWrecked();
    }
}
```

MIPS

```
# $a0 is the hashed password
# $a1 is the stored hashed password

checkPassword:
    bne $a0 $a1 getWrecked
    # Do stuff because password is
        correct
    j return

getWrecked:
    # Deny the user
    j return

return:
    # pop the stack and return
```

So what is wrong with this?

Password Checks

C

```c
char* password = "12345";
char* hash = md5(password);
char* stored_hash = getStoredHash();

int checkPassword(char* hash,
    char* stored_hash)
{
    if (strcmp(hash, stored_hash)
        == 0)
    {
        return letUserIn();
    }
    else
    {
        return getWrecked();
    }
}
```

MIPS

```
# $a0 is the hashed password
# $a1 is the stored hashed password

checkPassword:
    bne $a0 $a1 getWrecked
    # Do stuff because password is
        correct
    j return

getWrecked:
    # Deny the user
    j return

return:
    # pop the stack and return
```

Password Checks

- What if we just don't execute that **bne** instruction?
  - Yeah right! That's not how code works!

MIPS

```
1   # $a0 is the hashed password
2   # $a1 is the stored hashed password
3
4   checkPassword:
5       ~~bne~~    $a0    $a1    getWrecked
6       # Do stuff because password is
            correct
7       j       return
8
9   getWrecked:
10      # Deny the user
11      j       return
12
13  return:
14      # pop the stack and return
```

WRONG (kind of)

Let's see how...

Fault Injection Revisited

- The goal of fault injection was to cause a small disturbance with a large impact
  - Small disturbance: 1 instruction
  - Large impact: Authenticate any user

Fault Injection Revisited

- Tying it all together:
  - Power analysis allows us to determine when certain operations are happening
  - If we can determine when a certain operation happens, we can know where our “small” disturbance will take place
  - If we know where our disturbance will have a large impact, we can exploit this to have code perform actions that wouldn’t be expected to happen

Okay, you showed an example with MIPS. Surely that's just because it's MIPS right? This doesn't work with other assembly languages!

WRONG (this time for sure)

I have this C code

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4
5  int checkPassword(char* hash, char* stored_hash)
6  {
7      if (strcmp(hash, stored_hash) == 0) // if the strings match
8      {
9          return 0;
10     }
11     else
12     {
13         return 1;
14     }
15 }
16
17 int main(int argc, const char *argv[])
18 {
19     printf ("%i\n", checkPassword("you", "me"));
20     printf ("%i\n", checkPassword("me", "me"));
21 }
```

And this is the relevant disassembly from gdb

```
1  Dump of assembler code for function checkPassword:
2     0x0000000004005a6 <+0>: push \%rbp
3     0x0000000004005a7 <+1>: mov  \%rsp,\%rbp
4  => 0x0000000004005aa <+4>: sub  \$0x10,\%rsp
5     0x0000000004005ae <+8>: mov  \%rdi,-0x8(\%rbp)
6     0x0000000004005b2 <+12>: mov \%rsi,-0x10(\%rbp)
7     0x0000000004005b6 <+16>: mov -0x10(\%rbp),\%rdx
8     0x0000000004005ba <+20>: mov -0x8(\%rbp),\%rax
9     0x0000000004005be <+24>: mov \%rdx,\%rsi
10    0x0000000004005c1 <+27>: mov \%rax,\%rdi
11    0x0000000004005c4 <+30>: callq 0x400440 <strcmp@plt>
12    0x0000000004005c9 <+35>: test \%eax,\%eax
13    0x0000000004005cb <+37>:  jne    0x4005d4 <checkPassword+46>
14    0x0000000004005cd <+39>: mov \$0x0,\%eax
15    0x0000000004005d2 <+44>: jmp 0x4005d9 <checkPassword+51>
16    0x0000000004005d4 <+46>: mov \$0x1,\%eax
17    0x0000000004005d9 <+51>: leaveq
18    0x0000000004005da <+52>: retq
19 End of assembler dump.
```

Okay, having code just skip instructions sounds scary. But that's just academic theory right? That can't be done in pratice... right?

WRONG

Trigger

- When talking about power and voltage, a trigger is a spike that is reliably produced at a given time
- Used to determine when to start attacking the code

Hashing

- Takes a set amount of time
  - Trigger on the start of the hash/when the password is sent
- Wait for the amount of time that the hash takes
- Fluctuate power to obtain unexpected behavior after the hashing is completed

And now all we need is a way to send extra power to the board and a little python code

A quick google search yields

Well that was easy...

Pseudo-python code that doesn't actually work

```
# import the libraries to do the things
import powertrace
# set trigger
trigger.set(VOLTAGE)
trigger.wait()
# start power fluctuations at different times
for x in range(0, 1, .1):
    # fluctuate the power
    for y in range(3, 5, .1):
        time.sleep(TIME_HASH_TAKES - SMALL_AMOUNT)
        power.set(y)
        time.sleep(JUST_LONG_ENOUGH_TO_AFFECT_THINGS)
        power.set(NORMAL_POWER)
        # check to see if we broke things
        reset()
```

Real Life Examples

- Attacks against smart cards
- Can be used to break mathematically secure crypto!!
  - AES
  - DES
  - SSL
- I've attacked embedded devices (with permission!)
  - (Ask me about it if you want to know more)

This can be used to break mathematically secure crypto!

Okay, now I'm terrified. How do I protect against this??

- Perform multiplce calculations at one to cancel out traces
- Multiple checks
  - Faulting once can be tough, doing each one makes it more and more difficult
- Use a different algorithm that leaks less information
- Add noise
- Onboard brownout detection (which only kind of works)

Okay, now I'm terrified. How do I protect against this??

- Realisitcally, you don't need to worry about this in most cases (if you're not working with embedded devices)
- HOWEVER: When dealing with things that are sensitive, especially around embedded devices, choose good algorithms!
  - NSA Suite B
  - NEVER, EVER, EVER ROLL YOU OWN CRYPTO!!

Want to know more about the things that I referenced?

- Attacks on smart cards
- NSA Suite B
- Intro talk at Cal Poly Tech

Questions?

References I

- Server-Room
  http://www.lukecjdavis.com/wp-content/uploads/2014/01/server-room.jpg
- Hacker
  http://teamextenda.com/blog/wp-content/uploads/2014/11/phonesystemhackercreep.jpg
- Password
  http://now.avg.com/wp-content/uploads/2014/05/password1-618x336.jpg
- Embedded Device
  http://www.rcs.ei.tum.de/fileadmin/_processed_/csm_Foto_1_03_f24f1b0932.jpg