Higiene Dental: Introdução à Segurança Embarcada

A enorme variedade de dispositivos embarcados torna seu estudo fascinante, mas essa mesma variedade também pode deixá-lo confuso diante de outro formato, pacote ou circuito integrado (CI) estranho e o que isso significa em relação à sua segurança. Este capítulo começa com uma análise de vários componentes de hardware e os tipos de software executados neles. Em seguida, discutimos invasores, vários ataques, ativos e objetivos de segurança, e contramedidas para fornecer uma visão geral de como as ameaças à segurança são modeladas. Descrevemos os fundamentos da criação de uma árvore de ataque que você pode usar tanto para fins defensivos (para encontrar oportunidades de contramedidas) quanto para fins ofensivos (para raciocinar sobre o ataque possível mais fácil). Por fim, concluímos com reflexões sobre a divulgação coordenada no mundo do hardware.

Componentes de Hardware

Vamos começar examinando as partes relevantes da implementação física de um dispositivo embarcado que você provavelmente encontrará. Abordaremos os principais bits que você observará ao abrir um dispositivo pela primeira vez.

Dentro de um dispositivo embarcado, há uma placa de circuito impresso (PCB) que geralmente inclui os seguintes componentes de hardware: processador, memória volátil, memória não volátil, componentes analógicos e interfaces externas (veja a Figura 1-1).

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

A magia da computação acontece em um processador (unidade central de processamento ou CPU). Na Figura 1-1, o processador está embutido A mágica da computação acontece dentro de um processador (unidade central de processamento ou CPU). Na Figura 1-1, o processador está embutido dentro do Sistema em Chip (SoC) centralizado[1]. Geralmente, o processador executa o software principal e o sistema operacional (SO), enquanto o SoC contém periféricos de hardware adicionais.

Normalmente implementada em chips de memória dinâmica de acesso aleatório (DRAM) em pacotes discretos, a memória volátil[2] é a memória que o processador utiliza durante sua operação; seu conteúdo é perdido quando o dispositivo é desligado. A memória DRAM opera em frequências próximas à frequência do processador e necessita de barramentos amplos para acompanhar o processador.

Na Figura 1-1, a memória não volátil[3] é onde o dispositivo embarcado armazena dados que precisam persistir após a remoção da energia do dispositivo. Este armazenamento de memória pode estar na forma de EEPROMs, memória flash ou até mesmo cartões SD e discos rígidos. A memória não volátil geralmente contém código para inicialização, bem como aplicativos armazenados e dados salvos.

Embora não sejam muito interessantes para a segurança por si só, os componentes analógicos, como resistores, capacitores e indutores, são o ponto de partida para a análise de canal lateral e ataques de injeção de falha, que discutiremos detalhadamente neste livro. Em uma placa de circuito impresso típica, os componentes analógicos são todas as pequenas peças pretas, marrons e azuis que não se parecem com um chip e podem ter rótulos começando com “C”, “R” ou “L”.

As interfaces externas fornecem ao SoC os meios para se conectar ao mundo exterior. As interfaces podem ser conectadas a outros chips comerciais prontos para uso (COTS) como parte da interconexão do sistema PCB. Isso inclui, por exemplo, uma interface de barramento de alta velocidade para DRAM ou chips flash, bem como interfaces de baixa velocidade, como I2C e SPI para um sensor. As interfaces externas também podem ser expostas como conectores e pinos na placa de circuito impresso; por exemplo, USB e PCI Express (PCIe) são exemplos de interfaces de alta velocidade que conectam dispositivos externamente. É aqui que toda a comunicação acontece; por exemplo, com a internet, interfaces de depuração local ou sensores e atuadores. (Veja o Capítulo 2 para obter mais detalhes sobre a interação com dispositivos.)

A miniaturização permite que um SoC tenha mais blocos de propriedade intelectual (IP). A Figura 1-2 mostra um exemplo de um SoC Intel Skylake.

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Este chip contém vários núcleos, incluindo os principais núcleos da unidade central de processamento (CPU), o Intel Converged Security and Management Engine (CSME), a unidade de processamento gráfico (GPU) e muito mais. Os barramentos internos em um SoC são mais difíceis de acessar do que os externos, tornando os SoCs um ponto de partida inconveniente para hacking. Os SoCs podem conter os seguintes blocos de IP:

Vários (micro)processadores e periféricos

Por exemplo, um processador de aplicativo, um mecanismo de criptografia, um acelerador de vídeo e o driver de interface I2C.

Memória volátil

Na forma de CIs DRAM empilhados em cima do SoC, SRAMs ou bancos de registradores.

Memória não volátil

Na forma de memória somente leitura (ROM) on-die, fusíveis programáveis uma vez (OTP), EEPROM e memória flash. Os fusíveis OTP normalmente codificam dados críticos de configuração do chip, como informações de identidade, estágio do ciclo de vida e informações de controle de versão anti-retrocesso.

Barramento interno

Embora tecnicamente sejam apenas um monte de fios microscópicos, a interconexão entre os diferentes componentes do SoC é, na verdade, uma consideração importante de segurança. Pense nessa interconexão como a rede entre dois nós em um SoC. Sendo uma rede, os barramentos internos podem ser suscetíveis a falsificação, espionagem, injeção e todas as outras formas de ataques man-in-the-middle. SoCs avançados incluem controle de acesso em vários níveis para garantir que os componentes do SoC sejam "isolados" uns dos outros.

Cada um desses componentes faz parte da superfície de ataque, o ponto de partida para um invasor, e, portanto, é de interesse. No Capítulo 2, estudaremos essas interfaces externas com mais profundidade e, no Capítulo 3, veremos maneiras de encontrar informações sobre os vários chips e componentes.

Componentes de Software

Software é uma coleção estruturada de instruções e dados da CPU que um processador executa. Para nossos propósitos, não importa se esse software está armazenado em ROM, flash ou em um cartão SD - embora possa ser uma decepção para nossos leitores mais velhos que não cobriremos cartões perfurados. Dispositivos embarcados podem conter alguns (ou nenhum) dos seguintes tipos de software:

Anotação

Embora este livro se concentre em ataques de hardware, muitas vezes um ataque de hardware é usado para comprometer o software. Através de vulnerabilidades de hardware, os invasores podem obter acesso a partes do software que normalmente são difíceis de acessar ou que não deveriam ser acessíveis de forma alguma.

Código de Inicialização Inicial

O código de inicialização inicial é o conjunto de instruções que um processador executa quando é ligado pela primeira vez. O código de inicialização inicial é gerado pelo fabricante do processador e armazenado na ROM. A principal função do código ROM de inicialização é preparar o processador principal para executar o código que se segue. Normalmente, ele permite que um bootloader seja executado no campo, incluindo rotinas para autenticar um bootloader ou para suportar fontes alternativas de bootloader (como por meio de USB). Também é usado para suporte durante a fabricação para personalização, análise de falhas, depuração e autotestes. Freqüentemente, os recursos disponíveis na ROM de inicialização são configurados por meio de fusíveis, que são bits programáveis uma vez integrados ao silício que fornecem a opção para desabilitar permanentemente algumas das funcionalidades da ROM de inicialização quando o processador sai da fábrica.

A ROM de inicialização possui propriedades que a diferenciam do código regular: é imutável, é o primeiro código a ser executado em um sistema e deve ter acesso completo à CPU/SoC para dar suporte à fabricação, depuração e análise de falhas do chip. O desenvolvimento de código ROM requer muito cuidado. Por ser imutável, geralmente não é possível corrigir uma vulnerabilidade na ROM detectada após a

fabricação (embora alguns chips suportem a correção de ROM por meio de fusíveis). A ROM de inicialização é executada antes de qualquer funcionalidade de rede estar ativa, portanto, o acesso físico é necessário para explorar qualquer vulnerabilidade. Uma vulnerabilidade explorada durante esta fase de inicialização provavelmente resultará em acesso direto a todo o sistema.

Considerando os altos riscos para os fabricantes em termos de confiabilidade e reputação, em geral, o código da ROM de inicialização é normalmente pequeno, limpo e bem verificado (pelo menos deveria ser).

Carregador de Boot

O bootloader inicializa o sistema após a execução da ROM de inicialização. Ele é normalmente armazenado em armazenamento não volátil, mas mutável, podendo ser atualizado em campo. O fabricante original do equipamento (OEM) da PCB gera o bootloader, permitindo que ele inicialize os componentes do nível da PCB. Opcionalmente, ele também pode bloquear alguns recursos de segurança, além de sua tarefa principal de carregar e autenticar um sistema operacional ou ambiente de execução confiável (TEE). Além disso, o bootloader pode fornecer funcionalidade para provisionamento de um dispositivo ou depuração. Sendo o primeiro código mutável a ser executado em um dispositivo, o bootloader é um alvo atraente para ataques. Dispositivos menos seguros podem ter uma ROM de inicialização que não autentica o bootloader, permitindo que invasores substituam facilmente o código do bootloader.

Os bootloaders são autenticados com assinaturas digitais, que normalmente são verificadas incorporando uma chave pública (ou o hash de uma chave pública) na ROM de inicialização ou fusíveis. Como essa chave pública é difícil de modificar, ela é considerada a raiz da confiança. O fabricante assina o bootloader usando a chave privada associada à chave pública, para que o código da ROM de inicialização possa verificar e confiar que o fabricante o produziu. Uma vez que o bootloader é confiável, ele pode, por sua vez, incorporar uma chave pública para o próximo estágio do código e garantir a autenticidade do próximo estágio. Essa cadeia de confiança pode se estender até os aplicativos executados em um sistema operacional (veja a Figura 1-3).

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Teoricamente, criar essa cadeia de confiança parece bastante seguro, mas o esquema é vulnerável a uma série de ataques, desde a exploração de fraquezas de verificação até injeção de falhas, ataques de temporização e muito mais. Veja a palestra de Jasper no Hardwear.io USA 2019 “Top 10 Secure Boot Mistakes” no YouTube (https://www.youtube.com/watch?v=B9]8qjuxysO/) para uma visão geral dos 10 principais erros.

Ambiente de Execução Confiável, SO e Aplicativos Confiáveis

No momento da escrita, o TEE é um recurso raro em dispositivos embarcados menores, mas é muito comum em telefones e tablets baseados em sistemas como o Android. A ideia é criar um SoC “virtual” seguro, dividindo todo um SoC em mundos “seguro” e “não seguro”. Isso significa que cada componente no SoC está exclusivamente ativo no mundo seguro, exclusivamente ativo no mundo não seguro ou é capaz de alternar entre

os dois dinamicamente. Por exemplo, um desenvolvedor de SoC pode optar por colocar um mecanismo de criptografia no mundo seguro, hardware de rede no mundo não seguro e permitir que o processador principal alterne entre os dois mundos. Isso poderia permitir que o sistema criptografasse pacotes de rede no mundo seguro e depois os transmitisse por meio do mundo não seguro – ou seja, o "mundo normal" – garantindo que a chave de criptografia nunca chegue ao sistema operacional principal ou a um aplicativo do usuário no processador.

Em telefones celulares e tablets, o TEE inclui seu próprio sistema operacional, com acesso a todos os componentes do mundo seguro. O ambiente de execução rico (REE) inclui o sistema operacional do "mundo normal", como um kernel Linux ou iOS e aplicativos do usuário.

O objetivo é manter todas as operações não seguras e complexas, como aplicativos do usuário, no mundo não seguro e todas as operações seguras, como processamento de criptografia e armazenamento seguro de dados, no mundo seguro. Isso ajuda a garantir que mesmo se o mundo não seguro for comprometido, o mundo seguro e seus dados críticos permaneçam intactos, como aplicativos bancários, no mundo seguro. Esses aplicativos seguros são chamados de aplicativos confiáveis (TAs). O kernel do TEE é um alvo de ataque que, uma vez comprometido, geralmente fornece acesso completo aos mundos seguro e não seguro.

Imagens de Firmware

O firmware é o software de baixo nível que roda em CPUs ou periféricos. Periféricos simples em um dispositivo geralmente são totalmente baseados em hardware, mas periféricos mais complexos podem conter um microcontrolador que executa o firmware. Por exemplo, a maioria dos chips Wi-Fi requer que um "blob" de firmware seja carregado após a inicialização. Para aqueles que executam Linux, uma olhada em /lib/firmware mostra quanto firmware está envolvido na execução de periféricos de PC. Como qualquer software, o firmware pode ser complexo e, portanto, sensível a ataques.

Kernel e Aplicativos do Sistema Operacional Principal

O sistema operacional principal em um sistema embarcado pode ser um sistema operacional de propósito geral, como o Linux, ou um sistema operacional em tempo real, como VxWorks ou FreeRTOS. Os cartões inteligentes podem conter sistemas operacionais proprietários que executam aplicativos escritos em Java Card. Esses sistemas operacionais podem oferecer funcionalidades de segurança (por exemplo, serviços criptográficos) e implementar isolamento de processo, o que significa que se um processo for comprometido, outro processo ainda poderá estar seguro.

Um sistema operacional facilita a vida dos desenvolvedores de software que podem contar com uma ampla gama de funcionalidades existentes, mas pode não ser uma opção viável para dispositivos menores. Dispositivos muito pequenos podem não ter kernel de sistema operacional, mas apenas executar um programa bare-metal para gerenciá-los. Isso geralmente implica nenhuma isolação de processo, portanto, comprometer uma função leva ao comprometimento de todo o dispositivo.

Modelagem de Ameaças de Hardware

A modelagem de ameaças é uma das necessidades mais importantes na defesa de qualquer sistema. Os recursos para defender um sistema não são ilimitados, portanto, analisar como esses recursos são melhor gastos para minimizar as oportunidades de ataque é essencial. Este é o caminho para uma segurança "boa o suficiente".

Ao realizar a modelagem de ameaças, fazemos aproximadamente o seguinte: assumimos uma visão defensiva para identificar os ativos importantes do sistema e nos perguntamos como esses ativos devem ser protegidos. Por outro lado, de um ponto de vista ofensivo, podemos identificar quem podem ser os atacantes, quais podem ser seus objetivos e quais ataques eles podem tentar realizar. Essas considerações fornecem insights sobre o que proteger e como proteger os ativos mais valiosos. O trabalho de referência padrão para modelagem de ameaças é o livro de Adam Shostack, Threat Modeling: Designing for Security (Wiley, 2014). O amplo campo da modelagem de ameaças é fascinante, pois inclui a segurança do ambiente de desenvolvimento até a fabricação, cadeia de suprimentos, remessa e vida útil operacional. Abordaremos os aspectos básicos da modelagem de ameaças aqui e os aplicaremos à segurança de dispositivos embarcados, focando no próprio dispositivo.

O Que é Segurança?

O dicionário de inglês de Oxford define segurança como "o estado de estar livre de perigo ou ameaça". Essa definição binária implica que o único sistema seguro é aquele que ninguém se importaria em atacar ou que pode defender qualquer ameaça. O primeiro, chamamos de tijolo, porque não pode mais inicializar; o último, chamamos de unicórnio, porque unicórnios não existem. Não existe segurança perfeita, então você poderia argumentar que qualquer defesa não vale o esforço. Essa atitude é conhecida como niilismo de segurança. No entanto, essa atitude desconsidera o fato importante de que um trade-off de custo-benefício está associado a cada ataque.

Todos nós entendemos custo e benefício em termos de dinheiro. Para um atacante, os custos geralmente estão relacionados à compra ou aluguel de equipamentos necessários para a realização de ataques. Os benefícios vêm na forma de compras fraudulentas, carros roubados, pagamentos de ransomware e saques de máquinas caça-níqueis, para citar apenas alguns.

No entanto, os custos e benefícios de realizar ataques não são exclusivamente monetários. Um custo não monetário óbvio é o tempo; um custo menos óbvio é a frustração do atacante. Por exemplo, um atacante que está hackeando por diversão pode simplesmente passar para outro alvo diante da frustração. Certamente há uma lição de defesa aqui. Veja a palestra de Chris Domas na DEF CON 23 para saber mais sobre essa ideia: "Repsych: Psychological Warfare in Reverse Engineering". Os benefícios não monetários incluem a coleta de informações pessoalmente identificáveis e a fama derivada de publicações em conferências ou sabotagem bem-sucedida (embora esses benefícios também possam ser monetizados).

Neste livro, consideramos um sistema "seguro o suficiente" se o custo de um ataque for maior que o benefício. Um design de sistema pode não ser impenetrável, mas deve ser difícil o suficiente para que ninguém consiga realizar um ataque completo com sucesso.

Em resumo, a modelagem de ameaças é o processo de determinar como alcançar um estado suficientemente seguro em um determinado dispositivo ou sistema. A seguir, vamos examinar vários aspectos que afetam os benefícios e custos de um ataque.

Ataques ao Longo do Tempo

A Agência de Segurança Nacional dos EUA (NSA) tem um ditado: "Os ataques sempre melhoram, nunca pioram". Em outras palavras, os ataques se tornam mais baratos e mais fortes com o tempo. Esse princípio se mantém particularmente em escalas de tempo maiores, devido ao aumento do conhecimento público sobre um alvo, à diminuição do custo do poder de computação e à fácil disponibilidade de hardware de hacking. O tempo desde o design inicial de um chip até a produção final pode abranger vários anos, seguido por pelo menos um ano para implementar o chip em um dispositivo, resultando em três a cinco anos antes que ele esteja operacional em um ambiente comercial. Este chip pode precisar permanecer operacional por alguns anos (no caso de produtos da Internet das Coisas [IoT]), 10 anos (para um passaporte eletrônico) ou até 20 anos (em ambientes automotivos e médicos). Portanto, os projetistas precisam levar em conta quaisquer ataques que possam acontecer daqui a 5 a 25 anos. Isso é claramente impossível, então muitas vezes as correções de software precisam ser implementadas para mitigar problemas de hardware não corrigíveis. Para colocar em perspectiva, 25 anos atrás, um cartão inteligente poderia ser muito difícil de quebrar, mas depois de ler este livro, um cartão inteligente de 25 anos deve oferecer pouca resistência para extrair suas chaves.

As diferenças de custo também aparecem em escalas de tempo menores ao passar de um ataque inicial para a repetição desse ataque. A fase de identificação envolve a identificação de vulnerabilidades. Segue-se a fase de exploração, que envolve o uso das vulnerabilidades identificadas para explorar um alvo. No caso de vulnerabilidades de software (escaláveis), o custo de identificação pode ser significativo, mas o custo de exploração é quase zero, pois o ataque pode ser automatizado. Para ataques de hardware, o custo de exploração ainda pode ser significativo.

Do lado dos benefícios, os ataques normalmente têm uma janela limitada dentro da qual têm valor. Quebrar a proteção contra cópia do Commodore 64 hoje oferece pouca vantagem monetária. Um stream de vídeo do seu jogo esportivo favorito só tem alto valor enquanto o jogo está em andamento e antes que o resultado seja conhecido. No dia seguinte, seu valor é significativamente menor.

Escalabilidade de Ataques

As fases de identificação e exploração de ataques de software e hardware diferem significativamente entre si em termos de custo e benefício. O custo da fase de exploração de hardware pode ser comparável ao da fase de identificação, o que é incomum para software. Por exemplo, um sistema de pagamento com cartão inteligente projetado com segurança usa chaves diversificadas, de forma que encontrar a chave em um cartão não significa que você aprenda nada sobre a chave de outro cartão. Se a segurança do cartão for suficientemente forte, os invasores precisam de semanas ou meses e equipamentos caros para fazer compras fraudulentas no valor de alguns milhares de dólares em cada cartão. Eles devem repetir o processo para cada novo cartão para ganhar os próximos milhares de dólares. Se os cartões forem tão fortes,

obviamente não há caso de negócio para atacantes motivados financeiramente; tal ataque escala mal.

Por outro lado, considere os modchips do Xbox 360. A Figura 1-4 mostra o modchip Xenium ICE como o PCB branco à esquerda.

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

O modchip Xenium ICE na Figura 1-4, à esquerda, é soldado à placa principal do Xbox para realizar seu ataque. A placa automatiza um ataque de injeção de falha para carregar firmware arbitrário. Este ataque de hardware é executado com tanta facilidade que a venda de modchips poderia se tornar um negócio; portanto, dizemos que ele "escala bem" (o Capítulo 13 fornece uma descrição mais detalhada desse ataque).

Ataques de hardware se beneficiam das economias de escala, mas apenas se o custo de exploração for muito baixo. Um exemplo disso são ataques de hardware para extrair segredos que podem então ser usados em larga escala, como a recuperação de uma chave de atualização de firmware mestre escondida no hardware, facilitando o acesso a uma multidão de firmwares. Outro exemplo é a operação única de extração de ROM de inicialização ou código de firmware, que pode expor vulnerabilidades do sistema que podem ser exploradas várias vezes. Finalmente, a escala não é importante para alguns ataques de hardware. Por exemplo, hackear uma vez seria suficiente para obter uma cópia não criptografada de um vídeo de um sistema de gerenciamento de direitos digitais (DRM) que é então pirateado, como é o caso do lançamento de um único míssil nuclear ou da descriptografia da declaração de imposto de renda de um presidente.

A Árvore de Ataque

Uma árvore de ataque visualiza as etapas que um invasor toma ao ir da superfície de ataque à capacidade de comprometer um ativo, permitindo-nos analisar uma estratégia de ataque sistematicamente. Os quatro ingredientes que consideramos em uma árvore de ataque são atacantes, ataques, ativos (objetivos de segurança) e contramedidas (veja a Figura 1-5).

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Perfil do Atacante

É importante traçar o perfil dos invasores porque eles possuem motivações, recursos e limitações. Pode-se argumentar que botnets ou worms são agentes não humanos sem motivação, mas um worm é inicialmente lançado por uma pessoa que pressiona o enter com alegria, raiva ou expectativa gananciosa.

Anotacão

Ao longo deste livro, usamos a palavra "dispositivo" para os alvos de um ataque e "equipamento" para as ferramentas que um invasor usa para realizar o ataque.

A criação de um perfil do atacante depende significativamente da natureza do ataque necessário para um determinado tipo de dispositivo. O próprio ataque determina o

equipamento e o gasto necessários, e ambos os fatores ajudam a traçar o perfil do atacante até certo ponto. O governo querendo desbloquear um celular é um exemplo de um ataque caro que possui um alto incentivo, como espionagem e segurança do estado.

A seguir estão alguns cenários de ataque comuns e os motivos associados, os personagens e as capacidades dos respectivos atacantes:

Empreendimento Criminoso

Ganho financeiro é o principal motivador de ataques de empreendimentos criminosos. Maximizar o lucro requer escala. Como discutido anteriormente, um ataque de hardware pode estar na raiz de um ataque escalável, o que necessita de um laboratório de ataque de hardware bem equipado. Como exemplo, considere ataques à indústria de TV paga, onde os piratas têm sólidos casos de negócios que justificam milhões de dólares em equipamentos.

Concorrência Industrial

A motivação de um atacante neste cenário de segurança varia de análise competitiva (um eufemismo inocente para engenharia reversa para ver o que a concorrência está fazendo) a investigar violações de propriedade intelectual, até coletar ideias e inspiração para melhorar seu próprio produto relacionado. A sabotagem indireta por meio do dano à imagem de marca de um concorrente é uma tática semelhante. Esse tipo de atacante não é necessariamente um indivíduo, mas pode fazer parte de uma equipe empregada (talvez clandestinamente) ou contratada externamente por uma empresa que possui todas as ferramentas de hardware necessárias.

Estados-nação

Sabotagem, espionagem e contraterrorismo são motivadores comuns. Estados-nação provavelmente têm todas as ferramentas, conhecimento e tempo à sua disposição. Nas infames palavras de James Mickens, se o Mossad (agência nacional de inteligência de Israel) te direcionar, o que quer que você faça em termos de contramedidas, "você ainda será Mossad'ed".

Hackers Éticos

Hackers éticos podem ser uma ameaça, mas com um risco diferente. Eles podem ter habilidades de hardware e acesso a ferramentas básicas em casa ou ferramentas caras em uma universidade local, tornando-os tão bem equipados quanto atacantes maliciosos. Hackers éticos são atraídos por problemas onde sentem que podem fazer a diferença. Eles podem ser amadores motivados a entender como as coisas funcionam, ou pessoas que se esforçam para ser os melhores ou bem conhecidos por suas habilidades. Eles também podem ser pesquisadores que trocam suas habilidades por uma renda primária ou secundária, ou patriotas ou manifestantes que apoiam ou se opõem fortemente a causas. Um hacker ético não necessariamente apresenta nenhum risco. Um fabricante de fechadura inteligente certa vez lamentou que uma grande preocupação da empresa era acabar no palco como exemplo em um evento de hacking ético; eles perceberam isso como um impacto na confiança em sua marca. Na realidade, a maioria dos criminosos usará um tijolo para "hackear" a fechadura, então os clientes

da fechadura têm pouco risco de hack, mas o slogan "Não se preocupe, eles vão usar um tijolo e não um computador" não funciona tão bem em uma campanha de relações públicas.

Ataques por Pessoas Comuns

Este último tipo de atacante é tipicamente um indivíduo ou pequeno grupo de pessoas com um motivo pessoal para prejudicar outro indivíduo, empresa ou infraestrutura. No entanto, eles podem nem sempre ter o conhecimento técnico necessário. Seu objetivo pode ser ganho financeiro por meio de chantagem ou venda de segredos comerciais, ou simplesmente ferir outra parte. Ataques de hardware bem-sucedidos por esses atacantes geralmente são improváveis devido ao conhecimento e orçamento limitados. (Para todos os leigos por aí, por favor, não nos enviem mensagens diretas sobre como hackear a conta do Facebook do seu ex.)

Identificar potenciais atacantes não é necessariamente simples e depende do dispositivo. Em geral, é mais fácil traçar o perfil dos atacantes quando se considera um produto concreto em vez de um componente do produto. Por exemplo, a ameaça de hackear uma marca de cafeteiras IoT pela internet para produzir um café fraco poderia estar ligada aos vários tipos de atacantes listados anteriormente. A criação de perfil se torna mais complexa à medida que se sobe na cadeia de suprimentos de um dispositivo. Um componente em dispositivos IoT pode ser um acelerador de padrão de criptografia avançada (AES) fornecido por um fornecedor de IP. Este acelerador é integrado em um SoC, que é integrado em uma PCB, a partir da qual um dispositivo final é feito. Como o fornecedor de IP do acelerador AES identificaria as ameaças nos 1.001 dispositivos diferentes que usam esse acelerador AES? O fornecedor precisaria se concentrar mais no tipo de ataque do que nos atacantes (por exemplo, implementando um grau de resistência contra ataques de canal lateral).

Ao projetar um dispositivo, recomendamos fortemente que você verifique com seus fornecedores de componentes quais tipos de ataque foram protegidos. A modelagem de ameaças sem esse conhecimento não pode ser completa e, talvez mais importante, se os fornecedores não forem questionados sobre isso, eles não serão motivados a melhorar suas medidas de segurança.

Tipos de Ataques

Ataques de hardware obviamente visam o hardware, como abrir uma porta de depuração Joint Test Action Group (JTAG), mas também podem visar o software, como contornar a verificação de senha. Este livro não aborda ataques de software em software, mas aborda o uso de software para atacar hardware.

Como mencionado anteriormente, a superfície de ataque é o ponto de partida para um invasor - os bits de hardware e software diretamente acessíveis. Ao considerar a superfície de ataque, geralmente assumimos acesso físico total ao dispositivo. No entanto, estar dentro do alcance do Wi-Fi (proximidade de alcance) ou estar conectado através de qualquer rede (remoto) também pode ser um ponto de partida para um ataque.

A superfície de ataque pode começar com a placa de circuito impresso (PCB), enquanto um atacante mais habilidoso pode estender a superfície de ataque para o chip usando técnicas de decapagem e microprovação, conforme descrito posteriormente neste capítulo.

Ataques de Software em Hardware

Ataques de software em hardware usam vários controles de software sobre o hardware ou o monitoramento do hardware. Existem duas subclasses de ataques de software em hardware: injeção de falha e ataques de canal lateral.

Injeção de Falha

A injeção de falha é a prática de levar o hardware a um ponto que induz erros de processamento. A injeção de falha em si não é um ataque; é o que você faz com o efeito da falha que o transforma em um ataque. Os invasores tentam explorar esses erros produzidos artificialmente. Por exemplo, eles podem obter acesso privilegiado contornando as verificações de segurança. A prática de injetar uma falha e então explorar o efeito dessa falha é chamada de ataque de falha.

O bombardeio de DRAM é uma técnica bem conhecida de injeção de falha na qual o chip de memória DRAM é bombardeado com um padrão de acesso anormal em três linhas adjacentes. Ao ativar repetidamente as duas linhas externas, ocorrem inversões de bits na linha central da vítima. O ataque Rowhammer explora as inversões de bits DRAM fazendo com que as linhas da vítima se tornem tabelas de página. As tabelas de página são estruturas mantidas por um sistema operacional que limitam o acesso à memória dos aplicativos. Alterando bits de controle de acesso ou endereços de memória física nessas tabelas de página, um aplicativo pode acessar memória que normalmente não poderia acessar, o que facilmente leva à escalada de privilégios. O truque é manipular o layout da memória de forma que a linha da vítima com as tabelas de página esteja entre as linhas controladas pelo atacante e então ativar essas linhas a partir de software de alto nível. Este método provou ser possível nos processadores x86 e ARM, desde software de baixo nível até JavaScript. Consulte o artigo "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms" de Victor van der Veen et al. para mais informações.

O overclocking da CPU é outra técnica de injeção de falha. O overclocking da CPU causa uma falha temporária chamada falha de temporização. Tal falha pode se manifestar como um erro de bit em um registro da CPU. CLKSCREW é um exemplo de ataque de overclocking de CPU. Como o software em telefones celulares pode controlar a frequência da CPU, bem como a voltagem do núcleo, diminuindo a voltagem e aumentando momentaneamente a frequência da CPU, um atacante pode induzir a CPU a cometer falhas.

Cronometrando isso corretamente, os atacantes podem gerar uma falha na verificação de assinatura RSA, o que lhes permite carregar código arbitrário assinado incorretamente. Para obter mais informações, consulte "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management" de Adrian Tang et al.

Você pode encontrar esses tipos de vulnerabilidades em qualquer lugar onde o software possa forçar o hardware a funcionar fora dos parâmetros normais de operação. Esperamos que novas variantes continuem a surgir.

Ataques de Canal Lateral

O tempo de software está relacionado à quantidade de tempo de clock da parede necessária para um processador concluir uma tarefa de software. Em geral, tarefas mais complexas precisam de mais tempo. Por exemplo, classificar uma lista de 1.000 números leva mais tempo do que classificar uma lista de 100 números. Não deve ser surpresa que um invasor possa usar o tempo de execução do software como um gatilho para um ataque. Em sistemas embarcados modernos, é fácil para um invasor medir o tempo de execução, muitas vezes com a resolução de um único ciclo de clock! Isso leva a ataques de temporização, nos quais um invasor tenta relacionar o tempo de execução do software ao valor de informações secretas internas.

Por exemplo, a função strcmp em C determina se duas strings são iguais. Ele compara caracteres um por um, começando pela frente, e quando encontra um caractere diferente, ele termina. Ao usar strcmp para comparar uma senha digitada com uma senha armazenada, a duração da execução de strcmp vaza informações sobre a senha, pois ela termina ao encontrar o primeiro caractere não correspondente entre a senha candidata do invasor e a senha que protege o dispositivo. O tempo de execução de strcmp, portanto, vaza o número de caracteres iniciais da senha que estão corretos. (Detalhamos esse ataque no Capítulo 8 e descrevemos a maneira correta de implementar essa comparação no Capítulo 14.)

RAMBleed é outro ataque de canal lateral que pode ser lançado a partir de software, conforme demonstrado por Kwong et al. em "RAMBleed: Reading Bits in Memory Without Accessing Them". Ele usa as fraquezas do estilo Rowhammer para ler bits da DRAM. Em um ataque RAMBleed, as inversões acontecem na linha do invasor com base nos dados nas linhas da vítima. Dessa forma, um invasor pode observar o conteúdo da memória de outro processo.

Ataques Microarquitetônicos

Agora que você entende o princípio dos ataques de temporização, considere o seguinte. As CPUs modernas são rápidas devido ao grande número de otimizações que foram identificadas e implementadas ao longo dos anos. Um cache, por exemplo, é construído com a premissa de que locais de memória acessados recentemente provavelmente serão acessados novamente de novo. Portanto, os dados nessas localizações de memória são armazenados fisicamente mais próximos da CPU para acesso mais rápido. Outro exemplo de otimização surgiu da percepção de que o resultado da multiplicação de um número N por 0 ou 1 é trivial, portanto, não é necessário realizar o cálculo de multiplicação completo, pois a resposta é sempre simplesmente 0 ou N. Essas otimizações fazem parte da microarquitetura, que é a implementação de hardware de um conjunto de instruções.

No entanto, é aqui que as otimizações para velocidade e segurança estão em desacordo. Se a otimização for ativada relacionada a algum valor secreto, essa otimização pode sugerir valores nos dados. Por exemplo, se a multiplicação de N vezes K para um K

desconhecido às vezes for mais rápida do que outras vezes, o valor de K poderia ser 0 ou 1 nos casos rápidos. Ou, se uma região de memória estiver em cache, ela poderá ser acessada mais rapidamente, portanto, um acesso rápido significa que uma região específica foi acessada recentemente.

O notório ataque Spectre de 2018 explora uma otimização interessante chamada execução especulativa. Calcular se um desvio condicional deve ser tomado ou não leva tempo. Em vez de esperar que a condição do desvio seja calculada, a execução especulativa adivinha a condição do desvio e executa as próximas instruções como se a adivinhação estivesse correta. Se a adivinhação estiver correta, a execução simplesmente continua, e se a adivinhação estiver incorreta, a execução será revertida. Essa execução especulativa, no entanto, ainda afeta o estado dos caches da CPU. O Spectre força uma CPU a realizar uma operação especulativa que afeta o cache de uma forma que depende de algum valor secreto, e então usa um ataque de temporização de cache para recuperar o segredo. Conforme mostrado em "Spectre Attacks: Exploiting Speculative Execution", de Paul Kocher et al., podemos usar esse truque em alguns programas existentes ou criados para despejar toda a memória do processo de um processo vítima. O problema maior é que os processadores foram otimizados para velocidade dessa forma por décadas, e existem muitas otimizações que podem ser exploradas de forma semelhante.

Ataques ao Nível de PCB

O PCB é frequentemente a superfície de ataque inicial para dispositivos, por isso é crucial para os invasores aprenderem o máximo possível do design do PCB. O design fornece pistas sobre onde exatamente se conectar ao PCB ou revela onde estão localizados melhores pontos de ataque. Por exemplo, para reprogramar o firmware de um dispositivo (potencialmente permitindo controle total sobre o dispositivo), o invasor primeiro precisa identificar a porta de programação do firmware no PCB.

Para ataques ao nível de PCB, tudo o que é necessário para acessar muitos dispositivos é uma chave de fenda. Alguns dispositivos implementam resistência física à violação e resposta à violação, como dispositivos validados FIPS (Federal Information Processing Standard) 140 nível 3 ou 4 ou terminais de pagamento.

Embora seja um esporte interessante em si, contornar a proteção contra violação e chegar aos componentes eletrônicos está além do escopo deste livro.

Um exemplo de ataque ao nível de PCB é aproveitar as opções do SoC que são configuradas puxando determinados pinos para cima ou para baixo usando straps. As straps são visíveis na PCB como resistores de 0 Q (zero ohm) (consulte a Figura 1-6). Essas opções do SoC podem muito bem incluir habilitação de depuração, inicialização sem verificação de assinatura ou outras configurações relacionadas à segurança.

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Adicionar ou remover os straps para alterar a configuração é trivial. Embora as PCBs multicamadas modernas e os dispositivos de montagem em superfície compliquem as modificações, tudo o que você precisa é de uma mão firme, um microscópio, pinças, uma pistola de calor e, acima de tudo, paciência para concluir a tarefa.

Outro ataque útil no nível do PCB é ler o chip flash em um PCB, que normalmente contém a maior parte do software que roda no dispositivo, revelando um tesouro de informações. Embora alguns dispositivos flash sejam somente leitura, a maioria permite que você grave alterações críticas de volta neles de uma forma que remove ou limita as funções de segurança. O chip flash provavelmente impõe permissões somente leitura por meio de algum mecanismo de controle de acesso, que pode ser suscetível a injeção de falha.

Para sistemas projetados com segurança em mente, as alterações no flash devem resultar em um sistema não inicializável porque a imagem do flash precisa incluir uma assinatura digital válida. Às vezes, a imagem do flash é embaralhada ou criptografada; a primeira pode ser revertida (vimos XORs simples), e a última requer a aquisição da chave.

Discutiremos a engenharia reversa de PCB com mais detalhes no Capítulo 3, e discutiremos o controle do clock e da energia quando examinarmos a interface com alvos reais.

Ataques Lógicos

Ataques lógicos funcionam no nível de interfaces lógicas (por exemplo, comunicando-se através de portas de I/O existentes). Ao contrário de um ataque ao nível de PCB, um ataque lógico não funciona no nível físico. Um ataque lógico é direcionado ao software ou firmware do dispositivo embarcado e tenta violar a segurança sem hackear fisicamente. Você poderia compará-lo a invadir uma casa (dispositivo) percebendo que o proprietário (software) tem o hábito de deixar a porta dos fundos (interface) destrancada; portanto, não é necessário arrombar a fechadura.

Ataques lógicos famosos giram em torno de corrupção de memória e injeção de código, mas os ataques lógicos têm um escopo muito mais amplo. Por exemplo, se o console de depuração ainda estiver disponível em uma porta serial oculta de uma fechadura eletrônica, enviar o comando "desbloquear" pode fazer com que a fechadura abra. Ou, se um dispositivo desliga algumas contramedidas em condições de bateria fraca, injetar sinais de bateria fraca pode desabilitar essas medidas de segurança. Ataques lógicos visam erros de design, erros de configuração, erros de implementação ou recursos que podem ser abusados para quebrar a segurança de um sistema.

Depuração e Rastreamento

Entre os mecanismos de controle mais poderosos embutidos em uma CPU durante o design e a fabricação estão as funções de depuração e rastreamento de hardware. Isso geralmente é implementado em cima de uma interface Joint Test Action Group (JTAG) ou Serial Wire Debug (SWD). A Figura 17 mostra um cabeçote JTAG exposto.

Esteja ciente de que em dispositivos seguros, fusíveis, uma cinta de PCB ou algum código secreto proprietário ou mecanismo de desafio/resposta podem desativar a depuração e o rastreamento. Talvez apenas o cabeçote JTAG seja removido em dispositivos menos seguros (mais sobre JTAG nos próximos capítulos).

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Fuzzing Dispositivos

Fuzzing é uma técnica emprestada da segurança de software que visa identificar especificamente problemas de segurança no código. O objetivo típico do fuzzing é encontrar falhas para explorar para injeção de código. O fuzzing simples consiste em enviar dados aleatórios para um alvo e observar seu comportamento. Alvos robustos e seguros permanecem estáveis sob tal ataque, mas alvos menos robustos ou menos seguros podem apresentar comportamento anormal ou travar. Despejos de falha ou uma inspeção do depurador podem identificar a fonte de uma falha e sua possibilidade de exploração. O fuzzing inteligente foca em protocolos, estruturas de dados, valores típicos que causam falhas ou estrutura de código e é mais eficaz na geração de casos extremos (situações que normalmente não deveriam ser esperadas) que farão um alvo travar. O fuzzing baseado em geração cria entradas do zero, enquanto o fuzzing baseado em mutação pega entradas existentes e as modifica. O fuzzing guiado por cobertura usa dados adicionais (por exemplo, informações de cobertura sobre quais partes do programa são exercidas com uma entrada específica) para permitir que você encontre bugs mais profundos.

Você também pode aplicar fuzzing a dispositivos, embora sob circunstâncias muito mais desafiadoras em comparação com o fuzzing de software. Com o fuzzing de dispositivos, geralmente é muito mais difícil obter informações de cobertura sobre o software em execução, porque você pode ter muito menos controle sobre esse software. O fuzzing em uma interface externa sem controle adicional sobre o dispositivo impede a obtenção de informações de cobertura e, em alguns casos, isso dificulta o estabelecimento de se ocorreu uma corrupção. Por fim, o fuzzing é eficaz quando pode ser feito em alta velocidade. No fuzzing de software, isso pode ser de milhares a milhões de casos por segundo. Alcançar essa velocidade o desempenho não é trivial em dispositivos embarcados. A re-hospedagem de firmware é uma técnica que pega o firmware de um dispositivo e o coloca em um ambiente de emulação que pode ser executado em PCs. Isso resolve a maioria dos problemas com o fuzzing no dispositivo, ao custo de ter que criar um ambiente de emulação funcional.

Análise de Imagem Flash

A maioria dos dispositivos inclui chips flash que são externos à CPU principal. Se um dispositivo for atualizável por software, muitas vezes você pode encontrar imagens de firmware na internet. Depois de obter uma imagem, você pode usar várias ferramentas de análise de imagem flash, como binwalk, para ajudar a identificar as várias partes da imagem, incluindo seções de código, seções de dados, sistema(s) de arquivos e assinaturas digitais.

Por fim, a desmontagem e descompilação das várias imagens de software é muito importante para determinar possíveis vulnerabilidades. Também há alguns trabalhos iniciais interessantes sobre análise estática (como execução concólica) do firmware do dispositivo. Consulte “BootStomp: On the Security of Bootloaders in Mobile Devices” de Nilo Redini et al.

Ataques Não Invasivos

Ataques não invasivos não modificam fisicamente um chip. Ataques de canal lateral usam algum comportamento mensurável de um sistema para revelar segredos (por exemplo, medindo o consumo de energia de um dispositivo para extrair uma chave AES). Um ataque de falha usa injeção de falha no hardware para contornar um mecanismo de segurança; por exemplo, um grande pulso eletromagnético (EM) pode desabilitar um teste de verificação de senha para que aceite qualquer senha. (Os Capítulos 4 e 5 deste livro são dedicados a esses tópicos.)

Ataques Invasivos a Chips

Esta classe de ataque visa o pacote ou o silício dentro de um pacote e, portanto, opera em uma escala minúscula - a dos fios e portas. Fazer isso requer técnicas e equipamentos muito mais sofisticados, avançados e caros do que discutimos até agora. Esses ataques estão além do escopo deste livro, mas aqui está uma breve olhada no que atacantes avançados podem fazer.

Descapsulação, Depackaging e Rebondenamento

Descapsulação é o processo de remover parte do material de encapsulamento do CI usando guerra química, geralmente pingando ácido nítrico ou sulfúrico fumegante no pacote do chip até que ele se dissolva. O resultado é um furo no pacote através do qual você pode examinar o próprio microchip e, se o fizer corretamente, o chip ainda funciona.

Anotação

Você pode fazer a decapsulação em casa, desde que haja um exaustor químico e outros recursos de segurança em vigor. Para os corajosos, o livro "PoC// GTFO" da No Starch Press contém detalhes sobre como realizar a decapsulação em casa.

Durante a despacapelização, você mergulha todo o pacote em ácido, depois do que todo o chip fica exposto. Você precisa reconectar o chip para restaurar sua funcionalidade, o que significa reconectar os minúsculos fios que normalmente conectam o chip aos pinos de um pacote (veja a Figura 1-8).

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Mesmo que possam morrer no processo, chips mortos são bons para imagens e engenharia reversa óptica. No entanto, para a maioria dos ataques, os chips precisam estar funcionando.

Imagem Microscópica e Engenharia Reversa

Uma vez que o chip é exposto, o primeiro passo é identificar os blocos funcionais maiores do chip e, especificamente, encontrar os blocos de interesse. A Figura 1-2 mostra algumas dessas estruturas. Os maiores blocos no chip serão a memória, como RAM estática (SRAM) para caches de CPU ou memória acoplada, e ROM para o código de inicialização. Qualquer grupo longo e principalmente reto de linhas são barramentos que interconectam CPUs e periféricos. Simplesmente conhecer os

tamanhos relativos e a aparência das várias estruturas permite que você comece a fazer a engenharia reversa dos chips.

Quando um chip é decapsulado, como na Figura 1-8, você só pode ver a camada superior de metal. Para fazer a engenharia reversa de todo o chip, você também precisa descascá-lo, o que significa polir as camadas individuais de metal do chip para expor a camada abaixo.

A Figura 1-9 mostra uma seção transversal de um chip semicondutor de óxido de metal complementar (CMOS), que é como a maioria dos chips modernos são construídos. Como você pode ver, várias camadas e vias de metais de cobre acabam conectando os transistores (policristalino/substrato). O metal de nível mais baixo é usado para criar células padrão, que são os elementos que criam portas lógicas (AND, XOR e assim por diante) a partir de um número de transistores. Metais de nível superior são geralmente usados para roteamento de energia e clock.

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Um bom imageamento de chip permite reconstruir uma netlist a partir das imagens ou de um dump binário da ROM de inicialização. Uma netlist é essencialmente uma descrição de como todas as portas estão conectadas, o que abrange toda a lógica digital em um projeto. Tanto uma netlist quanto um dump da ROM de inicialização permitem que os invasores encontrem pontos fracos no código ou no design do chip. "Bits from the Matrix: Optical ROM Extraction" de Chris Gerlinsky e "Integrated Circuit Offensive Security" de Olivier Thomas, apresentado na conferência Hardwear.io 2019, fornecem boas introduções ao tópico.

Imagem de Microscópio Eletrônico de Varredura

Um microscópio eletrônico de varredura (SEM) executa uma varredura raster de um alvo usando um feixe de elétrons e faz medições de um detector de elétrons para formar uma imagem do alvo digitalizado com uma resolução melhor do que 1 nm, permitindo a imagem de transistores e fios individuais. Tal como acontece com o imageamento microscópico, você pode criar netlists a partir das imagens.

Injeção de Falha Ótica e Análise de Emissão Ótica

Uma vez que a superfície do chip está visível, é possível "se divertir com fótons". Devido a um efeito chamado luminescência de portadora quente, os transistores de comutação ocasionalmente emitem fótons. Com um sensor CCD (dispositivo de carga acoplada) sensível a IR, como os usados na astronomia amadora, ou um fotodiodo de avalanche (APD) se você quiser ser mais sofisticado, você pode detectar áreas ativas de fótons, o que contribui para o processo de engenharia reversa (ou mais especificamente para a análise de canal lateral), como correlacionar chaves secretas com medições de fótons. Consulte "Simple Photonic Emission Analysis of AES: Photonic Side Channel Analysis for the Rest of Us" de Alexander Schlosser et al.

Além de usar fótons para observar processos, você também pode usá-los para injetar falhas alterando a condutividade dos gates, o que é chamada de injeção de falha óptica (veja o Capítulo 5 e o Apêndice A para mais detalhes).

Edição de Feixe Iônico Focalizado e Microprobing

Um feixe de íons focado (FIB), pronunciado como "fib", usa um feixe de íons para remover partes de um chip ou depositar material em um chip em escala nanométrica, permitindo que invasores cortem fios de chip, redirecionem fios de chip ou criem pontos de prova para microprobing. As edições FIB levam tempo e habilidade (e um FIB caro), mas como você pode imaginar, tais edições podem contornar muitos mecanismos de segurança de hardware se um invasor for capaz de localizá-los. Os números na Figura 1-11 mostram furos que um FIB criou para acessar camadas metálicas inferiores. As estruturas de "chapéu" ao redor dos furos são criadas para contornar uma contramedida de escudo ativo.

Microprobing é uma técnica usada para medir ou injetar corrente em um fio de chip, o que pode não exigir uma plataforma de prova FIB para tamanhos maiores de recursos. A habilidade é um pré-requisito para realizar qualquer um desses ataques, embora uma vez que um invasor tenha os recursos para realizar ataques nesse nível, seja extremamente difícil manter a segurança.

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Cobrimos vários ataques diferentes relacionados a sistemas embarcados aqui. Lembre-se que qualquer ataque individual é suficiente para comprometer um sistema. O custo e as habilidades variam drasticamente, portanto, certifique-se de entender qual tipo de objetivo de segurança você precisa. Resistir a um ataque de alguém com um orçamento de um milhão de dólares e resistir a um ataque de alguém com 25 dólares e uma cópia deste livro são empreendimentos muito diferentes.

Ativos e Objetivos de Segurança

A pergunta a se fazer ao considerar os ativos que estão sendo projetados no produto é: "Quais ativos realmente me importam?" Um invasor fará a mesma pergunta. O defensor dos ativos pode chegar a uma ampla gama de respostas a essa pergunta aparentemente simples. O CEO de uma empresa pode se concentrar na imagem da marca e na saúde financeira. O diretor de privacidade se preocupa com a confidencialidade das informações pessoais dos consumidores, e o criptógrafo residente fica paranóico com o material da chave secreta. Todas essas respostas à pergunta estão inter-relacionadas. Se as chaves forem expostas, a privacidade do cliente pode ser impactada, o que por sua vez afeta negativamente a imagem da marca e, consequentemente, ameaça a saúde financeira de toda a empresa. No entanto, em cada nível, os mecanismos de proteção diferem.

Um ativo também representa um valor para um invasor. O que exatamente é valioso depende da motivação do invasor. Pode ser uma vulnerabilidade que permite ao invasor vender um exploit de execução de código para outros invasores. O ativo desejado pode ser dados de cartão de crédito ou chaves de pagamento das vítimas. As intenções do mundo corporativo podem ser maliciosamente direcionadas à marca de um concorrente.

Ao fazer a modelagem de ameaças, analise as perspectivas tanto do atacante quanto do defensor. Para os propósitos deste livro, limitamo-nos aos ativos técnicos em um dispositivo, portanto, assumimos que nossos ativos são representados como uma

sequência de bits em um dispositivo alvo que deve permanecer confidencial e com integridade protegida. Confidencialidade é a propriedade de manter um ativo oculto dos invasores, e integridade é a propriedade de não permitir que um invasor o modifique.

Como entusiasta de segurança, você pode estar se perguntando por que não mencionamos a disponibilidade. A disponibilidade é a propriedade de manter um sistema responsivo e funcional, e é particularmente importante para data centers e sistemas que lidam com segurança, como sistemas de controle industrial e veículos autônomos, onde a interrupção na funcionalidade do sistema não pode acontecer.

Faz sentido defender a disponibilidade do ativo apenas em situações em que um dispositivo não pode ser acessado fisicamente, como quando o acesso é via rede e internet. Tornar esses serviços indisponíveis é o propósito dos ataques de negação de serviço que derrubam sites. Para dispositivos embarcados, comprometer a disponibilidade é trivial: basta desligá-lo, bater com um martelo ou explodi-lo. Um objetivo de segurança define o quão bem você deseja proteger os ativos que define, contra quais tipos de ataques e invasores e por quanto tempo. Definir objetivos de segurança ajuda a focar os argumentos de design nas estratégias para conter as ameaças esperadas. Inevitavelmente, ocorrerão trade-offs devido a muitos cenários possíveis, e embora reconheçamos que não existam soluções universais, damos alguns exemplos comuns a seguir.

Embora não seja muito comum, a especificação dos pontos fortes e fracos de um dispositivo é um sinal claro da maturidade de segurança de um fornecedor.

Confidencialidade e Integridade do Código Binário

Normalmente, para o código binário, o principal objetivo é a proteção da integridade ou garantir que o código executado no dispositivo seja o código pretendido pelo autor. A proteção de integridade restringe a modificação do código, mas apresenta uma faca de dois gumes. Uma forte proteção de integridade pode bloquear um dispositivo de seu proprietário, limitando o código disponível para execução nele. Toda uma comunidade de hackers tenta contornar esses mecanismos em consoles de jogos para executar seu próprio código. Por outro lado, a proteção de integridade certamente tem o benefício não intencional de proteger contra malware infectando a cadeia de inicialização, pirataria de jogos ou governos instalando um backdoor.

O objetivo da confidencialidade como objetivo de segurança é tornar mais difícil copiar propriedade intelectual, como conteúdo digital, ou encontrar vulnerabilidades no firmware. Este último também dificulta para pesquisadores de segurança genuínos encontrar e relatar vulnerabilidades, bem como para atacantes explorá-las. (Veja a seção "Divulgações de Problemas de Segurança" na página 33 para mais informações sobre este dilema complexo.)

Confidencialidade e Integridade de Chaves

A criptografia transforma problemas de proteção de dados em problemas de proteção de chave. Na prática, as chaves são normalmente mais fáceis de proteger do que blocos de dados completos. Para modelagem de ameaças, observe que agora existem dois ativos: os dados de texto simples e a própria chave. A confidencialidade das chaves como

objetivo, portanto, geralmente está ligada à confidencialidade dos dados que estão sendo protegidos.

Por exemplo, a integridade é importante quando as chaves públicas são armazenadas em um dispositivo para verificações de autenticidade: se os invasores puderem substituir as chaves públicas originais pelas suas próprias, eles poderão assinar dados arbitrários que passam na verificação de assinatura no dispositivo. No entanto, a integridade nem sempre é um objetivo para as chaves; por exemplo, se o objetivo de uma chave é descriptografar um bloco de dados armazenado, modificar a chave simplesmente resulta na impossibilidade de realizar a descriptografia.

Outro aspecto interessante é como as chaves são injetadas com segurança em um dispositivo ou geradas durante o estágio de fabricação. Uma opção é criptografar ou assinar as próprias chaves, mas isso envolve outra chave. É um ciclo sem fim. Em algum lugar do sistema existe uma raiz de confiança, uma chave ou mecanismo que simplesmente temos que confiar.

Uma solução típica é confiar no processo de fabricação durante a geração inicial da chave ou durante a injeção da chave. Por exemplo, a especificação Trusted Platform Module (TPM) v2.0 exige um EPS (endorsement primary seed). Este EPS é um identificador único para cada TPM e é usado para derivar algum material de chave primária. De acordo com a especificação, este EPS deve ser injetado no TPM ou criado no TPM durante a fabricação.

Essa prática limita a exposição do material da chave, mas cria um ponto central crítico de coleta de material da chave na instalação de fabricação. Os sistemas de injeção de chave, especialmente, devem ser bem protegidos para evitar o comprometimento das chaves injetadas para todas as peças configuradas por este sistema. As práticas recomendadas envolvem a geração de chaves no dispositivo, de forma que a instalação de fabricação não tenha acesso a todas as chaves, bem como a divisão secreta, garantindo que diferentes estágios da fabricação injetem ou gerem diferentes partes do material da chave.

Atestado de Inicialização Remota

Atestado de inicialização é a capacidade de verificar criptograficamente que um sistema realmente inicializou a partir de imagens de firmware autênticas. Atestado de inicialização remota é a capacidade de fazer isso remotamente. Duas partes estão envolvidas na atestação: o provador pretende provar ao verificador que algumas medições do sistema não foram adulteradas. Por exemplo, você pode usar atestado de inicialização remota para permitir ou negar o acesso de um dispositivo a uma rede corporativa ou para decidir fornecer um serviço online a um dispositivo. Neste último caso, o dispositivo é o provador, o serviço online é o verificador e as medições são hashes de dados de configuração e imagens (firmware) usadas durante a inicialização. Para provar que as medições não foram adulteradas, elas são assinadas digitalmente usando uma chave privada durante os estágios de inicialização. O verificador pode verificar as assinaturas em uma lista de permissão ou bloqueio e deve ter um meio de verificar a chave privada usada para criar as assinaturas. O verificador detecta adulteração e garante que o dispositivo remoto não esteja executando imagens de inicialização antigas e possivelmente vulneráveis.

Como sempre, isso apresenta alguns problemas práticos. Primeiro, o verificador deve de alguma forma ser capaz de confiar na chave de assinatura do provador - por exemplo, confiando em um certificado contendo a chave pública do provador, que é assinado por alguma autoridade confiável. Na melhor das hipóteses, essa autoridade conseguiu estabelecer confiança durante o processo de fabricação, conforme descrito anteriormente. Segundo, quanto mais abrangente for a cobertura das imagens de inicialização e dados, mais haverá para configurações diferentes em campo. Isso significa que se torna inviável permitir todas as configurações conhecidamente boas, então é preciso voltar a bloquear as configurações conhecidamente ruins. No entanto, determinar uma configuração conhecida como ruim não é um exercício trivial e geralmente só pode ser feito após uma modificação ter sido detectada e analisada.

Observe que a atestação de inicialização protege os componentes do tempo de inicialização que são hash para autenticidade. Não protege contra ataques em tempo de execução, como injeção de código.

Confidencialidade e Integridade de Informações Pessoalmente Identificáveis

Informações pessoalmente identificáveis (PII) são dados que podem identificar um indivíduo. Os dados óbvios incluem nomes, números de celular, endereços e números de cartão de crédito, mas os dados menos óbvios podem ser dados do acelerômetro registrados em um dispositivo vestível. A confidencialidade de PII se torna um problema quando os aplicativos instalados em um dispositivo exfiltram essas informações. Por exemplo, dados do acelerômetro que caracterizam o andar de uma pessoa podem ser usados para identificar essa pessoa: consulte "Gait Identification Using Accelerometer on Mobile Phone" de Hoang Minh Thang et al. Os dados de consumo de energia do telefone celular podem localizar a posição de uma pessoa a partir da forma como o rádio do telefone consome energia, dependendo da distância das torres de celular, conforme descrito em "PowerSpy: Location Tracking Using Mobile Device Power Analysis" de Yan Michalevsky et al.

O campo médico também tem regulamentação em torno de PII. A Lei de Portabilidade e Responsabilidade de Seguro Saúde (HIPAA) de 1996 é uma lei nos Estados Unidos com forte foco na privacidade de informações médicas e se aplica a qualquer sistema que processe PII de pacientes. O HIPAA tem requisitos bastante inespecíficos para segurança técnica.

A integridade dos dados PII é essencial para evitar a personificação. Em cartões inteligentes bancários, o material da chave está vinculado a uma conta e, portanto, a uma identidade. A EMVCo, um consórcio de cartões de crédito, possui requisitos técnicos muito explícitos em contraste com o HIPAA. Por exemplo, o material da chave deve ser protegido contra ataques lógicos, de canal lateral e de falha, e essa proteção precisa ser comprovada por ataques reais realizados por um laboratório credenciado.

Integridade e Confidencialidade dos Dados do Sensor

Você acabou de aprender como os dados do sensor estão relacionados a PII. A integridade precisa ser importante, porque o dispositivo precisa detectar e registrar seu ambiente com precisão. Isso é ainda mais crucial quando o sistema está usando a entrada do sensor para controlar atuadores. Um ótimo exemplo (embora contestado) é o

de um drone RQ-170 dos EUA sendo forçado a pousar no Irã, supostamente depois que seu sinal GPS foi falsificado para fazê-lo acreditar que estava pousando em uma base dos EUA no Afeganistão.

Quando um dispositivo usa alguma forma de inteligência artificial para tomada de decisão, a integridade das decisões é desafiada por um campo de pesquisa chamado aprendizado de máquina adversarial. Um exemplo é explorar fraquezas em classificadores de redes neurais modificando artificialmente imagens de um sinal de parada. Para os humanos, a modificação não é detectável, mas a imagem pode se tornar completamente irreconhecível usando algoritmos padrão de reconhecimento de imagem, quando na verdade deveria ser reconhecível. Embora o reconhecimento da rede neural possa ser frustrado, os carros autônomos modernos possuem um banco de dados com a localização dos sinais que podem ser usados como alternativa, portanto, neste caso específico, não deve ser um problema de segurança. "Practical Black-Box Attacks Against Machine Learning" de Nicolas Papernot et al. tem mais detalhes.

Proteção da Confidencialidade do Conteúdo

A proteção de conteúdo resume-se a tentar garantir que as pessoas paguem pelo conteúdo de mídia que consomem e que permaneçam dentro de algumas restrições de licença, como data e localização geográfica, usando o gerenciamento de direitos/restrições digitais (DRM). O DRM depende principalmente da criptografia do fluxo de dados para transporte de conteúdo dentro/fora de um dispositivo e na lógica de controle de acesso dentro de um dispositivo para negar o acesso do software ao conteúdo em texto simples. Para dispositivos móveis, a maioria dos requisitos de proteção visa ataques apenas de software, mas para decodificadores, os requisitos de proteção incluem ataques por canal lateral e falha. Assim, os decodificadores são considerados mais difíceis de violar e são usados para conteúdo de maior valor.

Segurança e Resiliência

Segurança é a propriedade de não causar danos (às pessoas, por exemplo), e resiliência é a capacidade de permanecer operacional em caso de falhas (não maliciosas). Por exemplo, um microcontrolador em um satélite estará sujeito a radiação intensa que causa chamadas perturbações de evento único (SEUs). SEUs invertem bits no estado do chip, o que pode levar a erros em sua tomada de decisão. A solução resiliente é detectar e corrigir o erro ou detectar e redefinir para um estado conhecido como bom. Essa resiliência pode não ser necessariamente segura; dá a alguém que tenta injeção de falha tentativas ilimitadas, pois o sistema continua aceitando abusos.

Da mesma forma, não é seguro desligar a unidade de controle de um veículo autônomo em alta velocidade assim que um sensor indica atividade maliciosa. Primeiro, qualquer detector pode gerar falsos positivos e, segundo, isso potencialmente permite que um invasor use o sensor para ferir todos os passageiros. Como todos os objetivos, este apresenta ao desenvolvedor do produto trade-offs entre segurança e segurança/resiliência. Resiliência e segurança não são o mesmo que segurança; às vezes, elas estão em desacordo com a segurança. Para um atacante, isso significa oportunidades quebrar um dispositivo por boas intenções para torná-lo seguro ou resiliente.

Contramedidas

Definimos contramedidas como qualquer meio (técnico) para reduzir a probabilidade de sucesso ou impacto de um ataque. As contramedidas têm três funções: proteger, detectar e responder. (Discutimos algumas dessas contramedidas posteriormente no Capítulo 14.)

Proteger

Esta categoria de contramedidas tenta evitar ou mitigar ataques. Um exemplo é criptografar o conteúdo da memória flash contra olhares indiscretos. Se a chave estiver bem escondida, oferece proteção quase impenetrável. Outras medidas de proteção oferecem apenas proteção parcial. Se uma única corrupção de instrução da CPU puder causar uma falha explorável, randomizar o tempo da instrução crítica em cinco ciclos de clock ainda dá ao invasor uma probabilidade de 20% de atingi-la. É possível contornar completamente algumas medidas de proteção porque elas protegem apenas contra uma classe específica de ataques (por exemplo, uma contramedida de canal lateral não protege contra injeção de código).

Detectar

Esta categoria de contramedida requer algum tipo de circuito de detecção de hardware ou lógica de detecção em software. Por exemplo, você pode monitorar a fonte de alimentação de um chip em busca de picos ou quedas de tensão que indiquem um ataque de falha de tensão. Você também pode usar software para detectar estados anômalos. Por exemplo, sistemas que analisam constantemente o tráfego de rede ou logs de aplicativos podem detectar ataques. Outras técnicas comuns de detecção de anomalias são a verificação dos chamados canários de pilha, a detecção de páginas de proteção que foram acessadas, a localização de instruções switch sem caso correspondente e erros de verificação de redundância cíclica (CRC) em variáveis internas, entre muitas outras.

Responder

A detecção tem pouca utilidade sem uma resposta. O tipo de resposta depende do caso de uso do dispositivo. Para dispositivos altamente seguros, como cartões inteligentes de pagamento, limpar todos os segredos do dispositivo (efetivamente infligindo a si mesmo um ataque de negação de serviço) seria sábio ao detectar um ataque. Fazer isso não seria uma boa ideia em sistemas críticos para a segurança que devem continuar operando. Nesses casos, telefonar para casa ou recorrer a um modo debilitado, mas seguro, são respostas mais apropriadas. Outra resposta subvalorizada, mas eficaz, para intervenção humana é simplesmente desligar o dispositivo para evitar danos maiores desistir da vontade de continuar o ataque (por exemplo, redefinindo o dispositivo e aumentando cada vez mais o tempo de inicialização).

As contramedidas são críticas para construir um sistema seguro. Especialmente em hardware, onde os ataques físicos podem ser impossíveis de proteger totalmente, adicionar detecção e resposta muitas vezes eleva o nível além do que um invasor está disposto a fazer ou mesmo é capaz de fazer.

Um Exemplo de Árvore de Ataque

Agora que descrevemos os quatro ingredientes necessários para uma modelagem de ameaças eficaz, vamos começar com um exemplo em que nós, como invasores, queremos hackear uma escova de dentes IoT com o propósito de extrair informações confidenciais e (apenas por diversão) aumentar a velocidade da escovação para algo que 9 em cada 10 dentistas desaprovariam (mas o último adora um bom desafio).

Em nossa árvore de ataque de amostra, mostrada na Figura 1-12, temos o seguinte:

- Caixas arredondadas indicam os estados em que um invasor está ou os ativos que um invasor comprometeu (“substantivos”).
- Caixas quadradas indicam ataques bem-sucedidos que o invasor realizou (“verbos”).
- Uma seta sólida mostra o fluxo consequente entre os estados e ataques anteriores.
- Uma seta pontilhada indica ataques que são mitigados por alguma contramedida.
- Várias setas recebidas indicam que “qualquer uma das setas pode levar a isso”.
- O triângulo “e” significa que todas as setas recebidas devem ser satisfeitas.

Os números na árvore de ataque marcam os estágios do ataque à escova de dentes. Como invasores, temos acesso físico a uma escova de dentes IoT (1). Nossa missão é instalar um backdoor telnet na escova de dentes para determinar qual PII está presente no dispositivo (8) e também para operar a escova de dentes em velocidade absurda (11).

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Letras minúsculas indicam os ataques e algarismos romanos indicam as mitigações. Uma das primeiras coisas que fazemos é dessoldar a memória flash e ler o conteúdo - todos os 16 MB (a). No entanto, vemos que a imagem não possui strings legíveis. Após alguma análise de entropia, o conteúdo parece estar criptografado ou compactado, mas como não há cabeçalho identificando o formato de compactação, assumimos que esse conteúdo está criptografado, conforme mostrado no ataque (2) e mitigação de ataque (i). Para descriptografá-lo, precisamos da chave de criptografia. Não parece estar armazenada na flash, uma mitigação mostrada em (ii), então provavelmente está armazenada em algum lugar da ROM ou fusíveis. Sem acesso a um microscópio eletrônico de varredura, não podemos "lê-los" do silício.

Em vez disso, decidimos investigar usando análise de energia. Conectamos uma sonda de energia e um osciloscópio e obtemos um traço de energia do sistema durante a inicialização. O traço mostra cerca de um milhão de pequenos picos. Sabendo pela leitura da flash que a imagem tem 16 MB, deduzimos que cada pico corresponde a 16 bytes de dados criptografados. Vamos supor que se trata de uma criptografia AES-128 em um dos modos comuns de bloco de código eletrônico (ECB) ou encadeamento de blocos de cifras (CBC). O ECB é um modo onde cada bloco é descriptografado independentemente dos outros blocos, e o CBC é um modo onde a descriptografia dos últimos blocos depende dos blocos anteriores. Como conhecemos o texto cifrado da imagem do firmware, podemos tentar um ataque de análise de energia com base nos picos que medimos. Após muito pré-processamento dos traços e um ataque de análise diferencial de energia (DPA) (b), conseguimos identificar um provável candidato a

chave. (Não se preocupe; você aprenderá o que é DPA à medida que avançar neste livro.) A descriptografia com ECB produz lixo, mas o CBC nos dá várias strings legíveis no ataque (c); parece que encontramos a chave certa no estágio (3) e descriptografamos a imagem com sucesso no estágio (4)!

A partir da imagem descriptografada, podemos usar técnicas tradicionais de engenharia reversa de software (g) para identificar quais blocos de código fazem o quê, onde os dados são armazenados, como os atuadores são acionados e, importante do ponto de vista da segurança, podemos agora procurar vulnerabilidades no código (9). Além disso, modificamos a imagem descriptografada no estágio (d) para incluir um backdoor que nos permitirá conectar remotamente à escova de dentes via telnet (5).

Re-criptografamos a imagem e a gravamos na flash no ataque (d), apenas para descobrir que a escova de dentes não inicializa. Esbarramos no que provavelmente é a verificação de assinatura do firmware. Sem a chave privada usada para assinar as imagens, não podemos executar uma imagem modificada devido à mitigação (iii). Um ataque comum a essa contramedida é a injeção de falha de tensão. Com a injeção de falha, visamos corromper a única instrução responsável por decidir se aceita ou rejeita uma imagem de firmware. Normalmente, é uma comparação que usa o resultado booleano retornado de uma função rsa _signature _verify(). Como este código está implementado na ROM, não podemos realmente obter informações sobre a implementação a partir da engenharia reversa. Portanto, tentamos um truque antigo - obtemos um traço de canal lateral quando a imagem não modificada inicializa e o comparamos com um traço de canal lateral da inicialização de uma imagem modificada no ataque (e). O ponto onde os traços diferem provavelmente será o momento em que o código de inicialização decide se aceita a imagem do firmware no estágio (6). Geramos uma falha naquele instante para tentar modificar a decisão.

Carregamos a imagem maliciosa e reduzimos a tensão por algumas centenas de nanossegundos em um ponto aleatório em uma janela de 5 microssegundos no ataque (f), aproximadamente no momento em que determinamos que a decisão é tomada. Depois de algumas horas repetindo esse ataque, temos sorte; a escova de dentes inicializa nossa imagem maliciosa no estágio (7). Agora com o código modificado nos permitindo conectar via telnet, alcançamos o estágio (8), onde podemos controlar remotamente a escovação e espionar qualquer uso da escova. E agora, no divertido estágio final (11), aumentamos a velocidade para um nível absurdo!

É evidente que este é um exemplo bobo, já que as informações e o acesso obtidos provavelmente não são valiosos o suficiente para um atacante sério. O acesso físico é necessário para realizar os ataques de canal lateral e falha, e uma simples reinicialização do dispositivo pelo dono legítimo causaria uma negação de serviço. No entanto, é um exercício esclarecedor e sempre vale a pena brincar com esses cenários hipotéticos.

Ao desenhar árvores de ataque, é fácil se empolgar e criar uma árvore gigantesca. Lembre-se que os invasores provavelmente tentarão apenas alguns dos ataques mais fáceis (esta ferramenta ajuda a identificar quais são). Concentre-se nos ataques relevantes, que você pode determinar perfilando o atacante e as capacidades de ataque no início da modelagem de ameaças.

Identificação vs. Exploração

O caminho de ataque da escova de dentes concentra-se na fase de identificação de um ataque, encontrando uma chave, fazendo engenharia reversa do firmware, modificando a imagem e descobrindo o instante de injeção de falha. Lembre-se que a exploração é o esforço para escalar o hack acessando vários dispositivos. Ao repetir o ataque em outro dispositivo, você pode reutilizar muitas das informações obtidas durante a identificação. Os resultados subsequentes do ataque requerem apenas a atualização de uma imagem no ataque (d) no estágio (5), sabendo do ponto de injeção de falha no estágio (6) e gerando a falha pelo ataque (f). O esforço de exploração é sempre menor que o esforço de identificação. Em alguns formalismos de criação de árvores de ataque, cada seta é anotada com o custo e o esforço do ataque, mas aqui evitamos entrar muito em modelagem quantitativa de risco.

Escalabilidade

O ataque da escova de dentes não é escalável, porque a fase de exploração requer acesso físico. Para PII ou atuação remota, geralmente é do interesse de um invasor apenas se isso puder ser feito em escala.

No entanto, digamos que em nosso ataque de engenharia reversa (g), o estágio (9) consegue identificar uma vulnerabilidade para a qual criamos um exploit no estágio (10). Descobrimos que a vulnerabilidade é acessível através de uma porta TCP aberta, então o ataque (j) pode explorá-la remotamente. Isso muda instantaneamente toda a escala do ataque. Tendo usado ataques de hardware na fase de identificação, podemos confiar apenas em ataques de software remotos na fase de exploração (12). Agora, podemos atacar qualquer escova de dentes, ter acesso aos hábitos de escovação de qualquer pessoa e irritar gengivas em escala global. Que época para se estar vivo!

Analisando a Árvore de Ataque

A árvore de ataque ajuda a visualizar os caminhos de ataque para discuti-los em equipe, identificar pontos onde contramedidas adicionais podem ser construídas e analisar a eficácia das contramedidas existentes. Por exemplo, é fácil ver que a mitigação por criptografia de imagem de firmware (i) forçou o invasor a usar um ataque de canal lateral (b), o que é mais difícil do que simplesmente ler uma memória. Da mesma forma, a mitigação pela assinatura de imagem de firmware (iii) forçou o invasor a um ataque de injeção de falha (f).

No entanto, o principal risco ainda é o caminho de ataque escalável através da exploração (j), que atualmente não possui mitigação. Obviamente, a vulnerabilidade deve ser corrigida, contramedidas anti-exploração devem ser introduzidas e restrições de rede devem ser implementadas para impedir que qualquer pessoa se conecte diretamente à escova de dentes remotamente.

Pontuando Caminhos de Ataque de Hardware

Além de visualizar os caminhos de ataque para análise, também podemos adicionar alguma quantificação para descobrir quais ataques são mais fáceis ou mais baratos para um invasor. Nesta seção, apresentamos vários sistemas de classificação padrão da indústria.

O Sistema de Pontuação Comum de Vulnerabilidade (CVSS) tenta pontuar vulnerabilidades de acordo com a gravidade, geralmente no contexto de computadores em rede em uma organização. Ele assume que a vulnerabilidade é conhecida e tenta pontuar o quão ruim seria se fosse explorada. O Sistema de Pontuação de Fraquezas Comuns (CWSS) quantifica as fraquezas nos sistemas, mas essas fraquezas não são necessariamente vulnerabilidades e nem necessariamente no contexto de computadores em rede. Finalmente, a Biblioteca de Interpretação Conjunta (JIL) é usada para pontuar caminhos de ataque (hardware) no esquema de certificação Common Criteria (CC).

Todos esses métodos de pontuação possuem vários parâmetros e pontuações para cada parâmetro, que juntos criam uma contagem final para ajudar a comparar várias vulnerabilidades ou caminhos de ataque. Esses métodos de pontuação também compartilham a vantagem de substituir argumentos indefinidos sobre parâmetros por pontuações que só fazem sentido no contexto específico do método de pontuação. A Tabela 1-1 fornece uma visão geral das três classificações e onde elas são aplicáveis.

(no livro virtual que eu encontrei as imagens não aparecem então só inclui o texto)

Em um contexto defensivo, você pode usar classificações para avaliar o impacto de um ataque após sua ocorrência, como forma de decidir como responder a ele. Por exemplo, se uma vulnerabilidade for detectada em um software, a pontuação CVSS pode ajudar a decidir se é necessário lançar um patch de emergência (com todos os custos associados) ou adiar a correção para a próxima versão principal, caso a vulnerabilidade seja menor.

Em um contexto defensivo, você também pode usar classificações para julgar quais contramedidas são necessárias. No contexto da certificação de Smart Card do Common Criteria, a pontuação JIL na verdade se torna uma parte crítica do objetivo de segurança - o chip deve resistir a ataques classificados em até 30 pontos para ser considerado resistente a atacantes com alto potencial de ataque. O documento SOG-IS “Aplicação de Potencial de Ataque a Cartões Inteligentes” explica a pontuação e aborda vários ataques de hardware. Para dar uma ideia da classificação, se leva algumas semanas para extrair uma chave secreta usando um sistema de dois feixes de laser para injeção de falha a laser, esse ataque é classificado como 30 ou menos. Se levar seis meses para extrair uma chave usando um ataque de canal lateral, não é necessário implementar uma contramedida, pois esse ataque é classificado como 31 ou superior.

O CWSS visa classificar as fraquezas nos sistemas antes que sejam exploradas. É um método de pontuação útil durante o desenvolvimento, pois ajuda a priorizar as correções das fraquezas. Todos sabem que cada correção tem um custo e que tentar corrigir todos os bugs não é prático, portanto, classificar as fraquezas permite que os desenvolvedores se concentrem nas mais significativas.

Na realidade, a maioria dos atacantes também faz algum tipo de pontuação para minimizar o custo e maximizar o impacto do ataque. Embora os invasores não publiquem muito sobre esses tópicos, Dino Dai Zovi deu uma palestra interessante chamada "Attacker Math 101" na SOURCE Boston 2011 que tentou definir alguns limites para o custo do ataque para os invasores.

Essas classificações são limitadas, ambíguas, imprecisas, subjetivas e não específicas do mercado, mas formam um bom ponto de partida para discutir um ataque ou

vulnerabilidade. Se você estiver fazendo modelagem de ameaças para sistemas embarcados, recomendamos começar com JIL, que se concentra principalmente em ataques de hardware. Quando estiver preocupado com ataques de software, use CWSS, pois esses são os contextos para os métodos de pontuação. Com o CWSS, você pode descartar aspectos irrelevantes e ajustar outros, como impactos nos negócios, para avaliar o valor dos ativos ou a escalabilidade. Além disso, certifique-se de pontuar todo o caminho do ataque, desde o ponto de partida do atacante até o impacto no ativo, para que você tenha uma comparação consistente entre as pontuações. Nenhuma das três classificações lida bem com a escalabilidade: um ataque a um milhão de sistemas pode produzir apenas uma pontuação marginalmente pior do que em um único sistema. Sem dúvida, existem outras limitações, mas atualmente não existem padrões da indústria mais conhecidos.

Em vários esquemas de certificação de segurança, existe um objetivo de segurança implícito ou explícito. Por exemplo, como mencionado anteriormente, para cartões inteligentes, apenas ataques de 30 pontos JIL ou menos são considerados relevantes. Um ataque como o apresentado por Tarnovsky em 2010 na palestra "Deconstructing a ‘Secure’ Processor” da Black Hat DC ultrapassa 30 pontos e, portanto, não é considerado parte do objetivo de segurança. Para o FIPS 140-2, nenhum ataque fora da lista específica de ataques é considerado relevante. Por exemplo, um ataque de canal lateral pode comprometer um mecanismo de criptografia validado pelo FIPS 140-2 em um dia, e o objetivo de segurança do FIPS 140-2 ainda o considerará seguro. Sempre que você usar um dispositivo que possua um certificado de segurança, verifique se os objetivos de segurança do certificado estão alinhados com os seus.

Divulgando Problemas de Segurança

A divulgação de problemas de segurança é um tópico altamente debatido, e não pretendemos resolvê-lo em alguns parágrafos. No entanto, queremos contribuir para o debate quando se trata de questões de segurança de hardware. Hardware e software sempre terão problemas de segurança. Com o software, você pode distribuir novas versões ou patches. Corrigir o hardware é caro por vários motivos.

Acreditamos que o objetivo da divulgação é a segurança e proteção pública, não os interesses comerciais do fabricante ou a fama e fortuna do pesquisador. Isso significa que a divulgação deve servir ao público a longo prazo. A divulgação é uma ferramenta para forçar os fabricantes a corrigir uma vulnerabilidade e também para informar o público sobre os riscos de um determinado produto.

Um efeito colateral indesejável da divulgação completa é que um grande grupo de atacantes poderá explorar a vulnerabilidade até que uma correção esteja amplamente disponível.

Para vulnerabilidades de hardware, o bug geralmente não pode ser corrigido após a fabricação, embora a emissão de um patch de software possa mitigá-lo. Nesse caso, uma convenção semelhante à divulgação de software de 90 dias até a divulgação pública pode funcionar bem. Para correções puramente de hardware, não temos conhecimento de tais convenções (embora tenhamos visto a aplicação de convenções de software).

No hardware, é comum que uma atualização de software não consiga contornar um bug, e os patches sejam praticamente impossíveis de distribuir e instalar. Um fabricante bem-intencionado pode corrigir um bug na próxima versão, mas os produtos em campo permanecerão vulneráveis. Nessa situação, a única vantagem da divulgação é um público informado; a desvantagem é um longo período até que os produtos vulneráveis sejam substituídos ou descontinuados.

Uma alternativa é a divulgação parcial. Por exemplo, um fabricante pode nomear o risco e o produto, mas não divulgar os detalhes de como explorar a vulnerabilidade. (Essa estratégia não funcionou bem no mundo do software, onde as vulnerabilidades são frequentemente encontradas rapidamente, mesmo após uma divulgação não específica.)

As complicações aumentam quando a vulnerabilidade não pode ser corrigida e pode afetar diretamente a saúde e a segurança. Considere um ataque que possa desligar remotamente todos os marcapassos cardíacos. A divulgação desta última situação certamente afastará os pacientes da instalação de marcapassos, causando mais mortes por ataques cardíacos. Por outro lado, incentivaria o fornecedor a aumentar a segurança na próxima versão, reduzindo o risco de um ataque com consequências letais. Dilemas únicos ocorrerão para carros autônomos, escovas de dente IoT, sistemas SCADA e todos os outros aplicativos e dispositivos. Ainda mais desafios surgem quando as vulnerabilidades existem em um tipo de chip usado em uma variedade de produtos.

Não afirmamos ter a resposta mágica para todas as situações aqui, mas incentivamos todos a considerarem cuidadosamente o tipo de divulgação a ser buscada. Os fabricantes devem projetar sistemas com a premissa de que serão violados e planejar cenários seguros em torno dessa premissa. Infelizmente, essa prática não é predominante, especialmente em situações onde o tempo de lançamento no mercado e o baixo custo imperam.

Resumo

Este capítulo delineou alguns fundamentos de segurança embarcada. Descrevemos os componentes de software e hardware que você sem dúvida encontrará ao analisar um dispositivo e discutimos o que "segurança" significa filosoficamente. Para analisar a segurança adequadamente, introduzimos os quatro componentes de um modelo de ameaças: os atacantes, vários ataques (hardware), os ativos e objetivos de segurança de um sistema e, por fim, os tipos de contramedidas que você pode implementar. Também descrevemos ferramentas para criar, analisar e classificar ataques usando uma árvore de ataque e sistemas de classificação padrão da indústria. Finalmente, exploramos o tópico complicado da divulgação no contexto de vulnerabilidades de hardware.

Equipados com todo esse conhecimento, nosso próximo passo será começar a investigar dispositivos, o que faremos no próximo capítulo.