Shattered Trust: When Replacement Smartphone Components Attack

Guilherme Suardi Calin

- Smartphones são comumente derrubados, rachando sua tela.
- De acordo com um estudo realizado em 2015, mais da metade dos usuários de smartphones já danificaram a tela do celular pelo menos uma vez.
- Uma tela rachada de um smartphone pode ser consertada em uma loja oficial da fabricante como a Apple Store, porém...

- …é mais barato consertar a tela danificada em uma assistência terceirizada.

- Reparos não-oficiais costumam utilizar componentes baratos que podem, intencionalmente ou não, introduzir *hardware* falsificado no telefone.

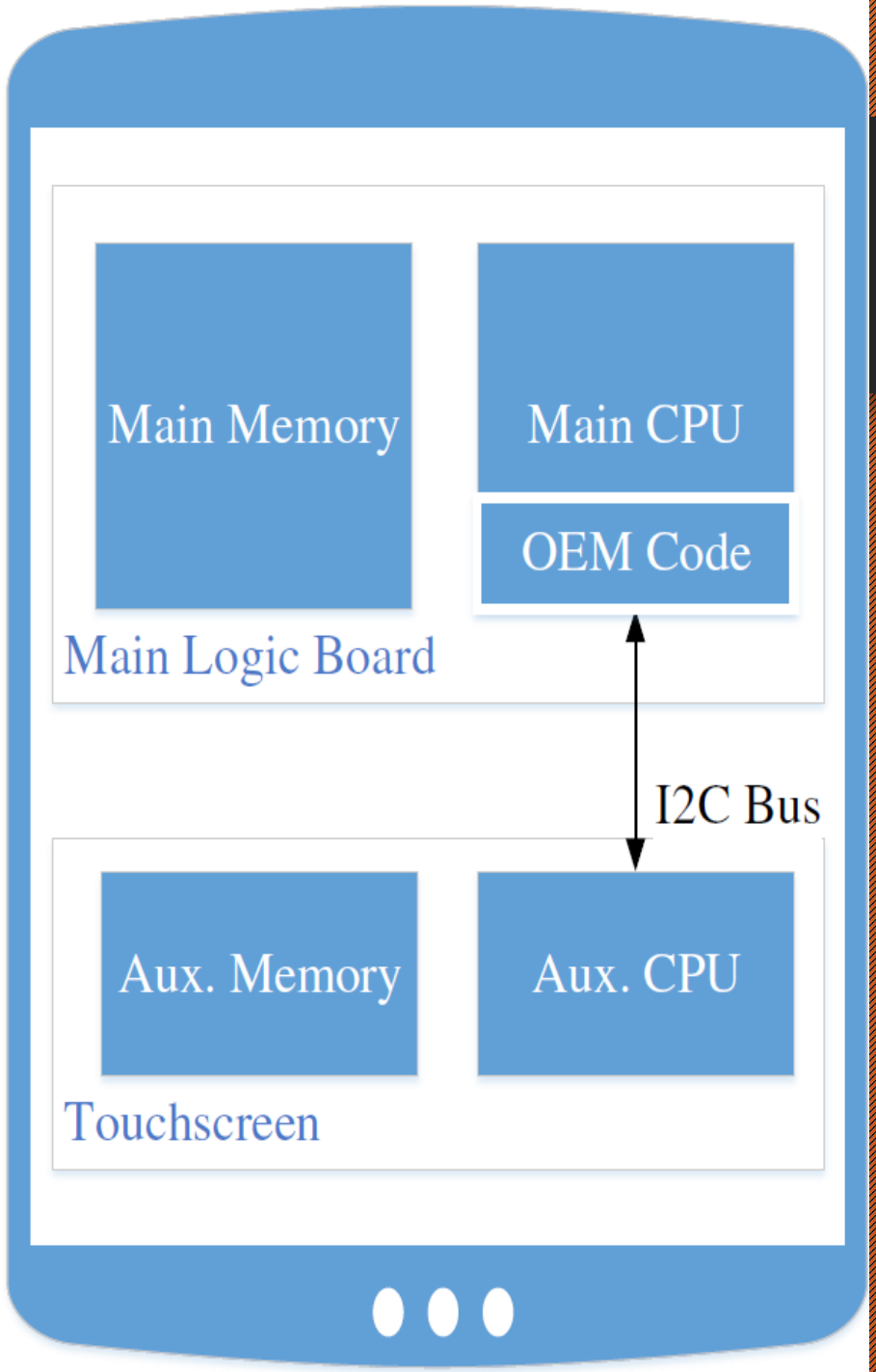
- Telas sensíveis ao toque, e outros componentes de *hardware* como sensores de orientação e leitores NFC raramente são produzidos pela fabricante do *smartphone*.

- Esses componentes e seus *drivers* são fornecidos por empresas especializadas, chamadas de *Original Equipment Manufacturer (OEM)*, como a Synaptics e a MediaTek.

- O código dos *drivers* dos componentes integrado no código da fabricante do *smartphone*, fazendo os ajustes necessários.

- A placa do *smartphone* executa algum *driver* da OEM integrado, e este por sua vez se comunica com o componente por meio de uma interface de comunicação I2C.

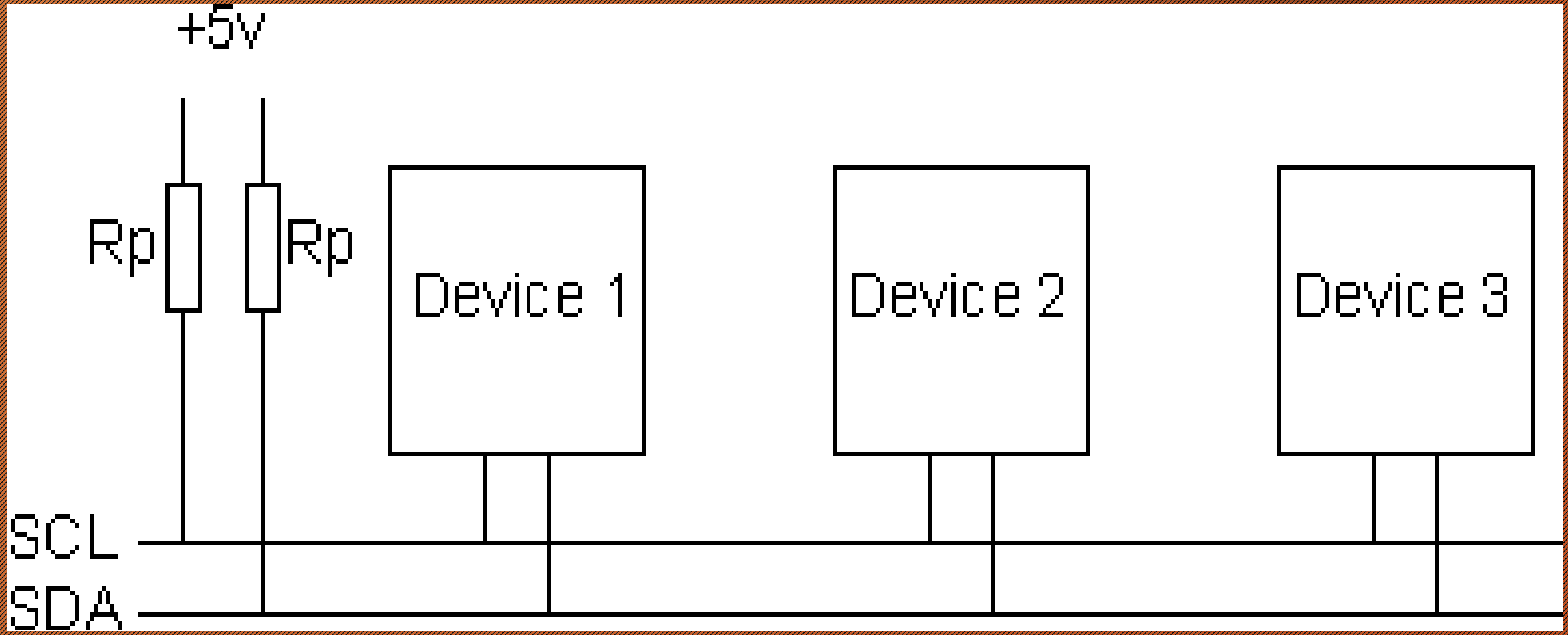
- O mesmo paradigma de arquitetura de *driver* é comumente utilizado até mesmo em *smartphones* utilizados por agências do governo.


Main Memory
Main CPU
OEM Code
Main Logic Board
I2C Bus
Aux. Memory
Aux. CPU
Touchscreen

Comunicação I2C

- Comunicação serial síncrona do tipo mestre-escravo desenvolvida pela Philips na década de 90.
- Possui uma linha para dados (SDA) e outra para *clock* (SCL).
- Cada escravo possui um endereço único que o identifica.

Comunicação I2C



Comunicação I2C

- Quando o mestre quer iniciar uma comunicação, ele traz o valor de SDA para nível lógico baixo, e logo em seguida, escreve o endereço do escravo com quem ele quer se comunicar.

- Se o escravo com esse endereço existir, ele envia um pulso na linha SCL.

- *Drivers* de dispositivo são assumidos como estando dentro de um limite de confiança do *smartphone*.

- Ao contrário de periféricos *plug-and-play* como USB, esses *drivers* assumem que os componentes do telefone com o qual eles se comunicam também estão dentro desse limite de confiança.

- Como um periférico substituído malicioso pode abusar dessa confiança?

- O artigo utilizou a seguinte restrição no modelo de ataque: apenas um componente específico com *hardware* limitado é malicioso.

- Os ataques permitiram gravar informação digitada, instalar aplicativos sem consentimento do usuário, direcioná-lo a sites de *phishing* e explorar vulnerabilidades no *kernel*.

- Interfaces de *hardware* são motivo de preocupação na parte de segurança no campo de PCs.
- Componentes com DMA como GPUs podem implantar *malwares* na memória do *kernel*.
- Devido a *smartphones* possuírem uma configuração mais estática, não há muito estudo no aspecto de segurança de *hardware*.

- Existem riscos de componentes falsificados na própria cadeia de fornecimento de *smartphones*.

- Em 2016, pesquisadores da Underwriters Laboratories obtiveram 400 carregadores de iPhone de múltiplas fontes em 8 países.

- Também em 2016, a Apple processou o fornecedor da Amazon Mobile Star LLC por venda de produtos falsificados, como cabos, carregadores e iPhones.

- Estimativas assumem que existem cerca de 2 bilhões de *smartphones* em circulação.
- Se 20% deles já tiveram a tela trocada pelo menos uma vez, então existem cerca de 400 milhões de *smartphones* com a tela substituída.
- Um ataque baseado em componente malicioso que comprometa mesmo que uma fração disso já terá uma escala comparavél a uma grande botnet.

- Que tipos de dano um periférico malicioso dentro do *smartphone* pode causar?

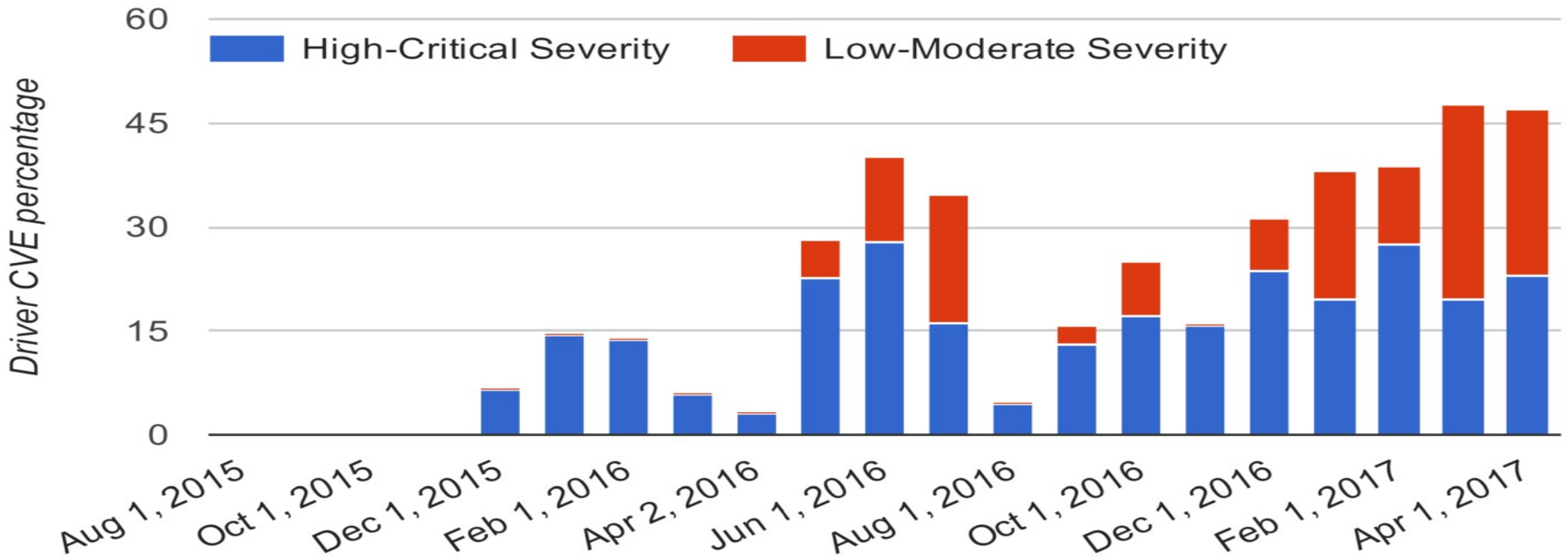
- Ataques de primeira ordem não dependem de vulnerabilidades de *software*, e utilizam os modos de interação padrão do componente sem o consentimento do usuário, como registrar atividade de toque na tela ou personificar o usuário para propósitos maliciosos.

- Ataques de segunda-ordem visam comprometer o aparelho ao causar um mal funcionamento no *driver* e por sua vez comprometer o *kernel*.

- Com o *kernel* comprometido, é possível desativar detecção e prevenção de atividade suspeita do Sistema e bisbilhotar sensores.

- Quase 30% das CVEs *(Common Vulnerabilities and Exposures) Android* corrigidas *entre 2015 e 2017 ocorreram no contexto de* driver.

CVEs Android em contexto de *driver*

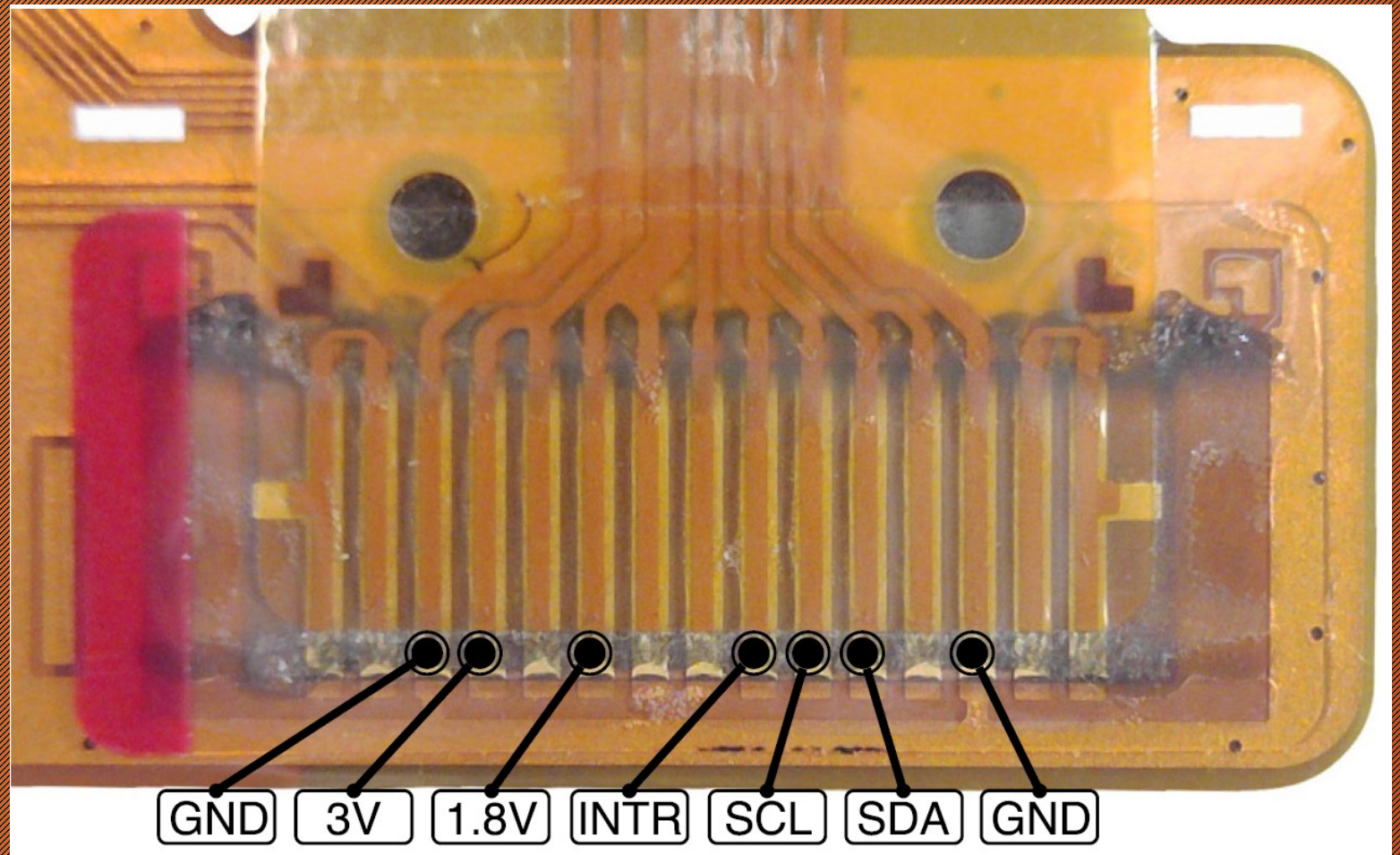


- Apesar de *touchscreens* possuirem capacidades e propriedades físicas diferentes de acordo o *smartphone* em que serão inseridas, elas possuem um aspecto geral em comum.

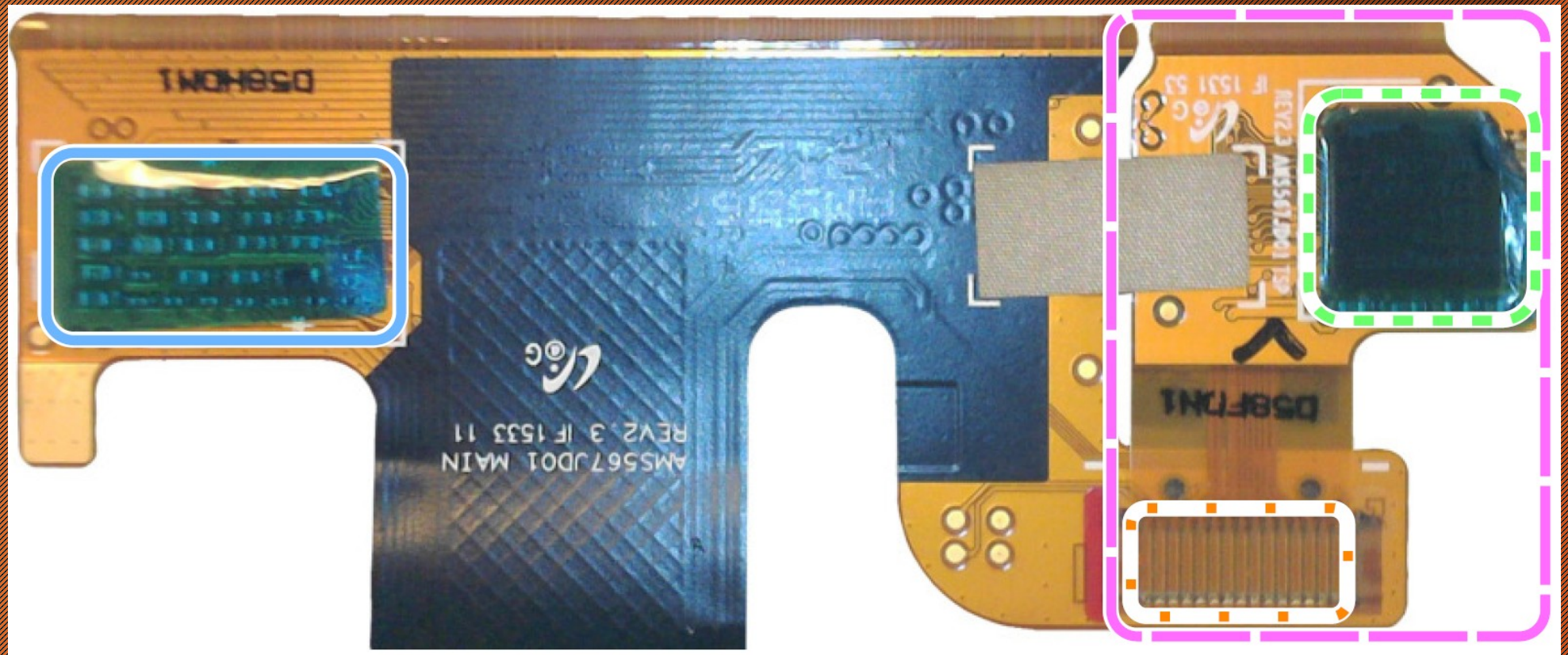
- Foi realizada uma análise do *hardware touchscreen do Huawei Nexus 6P que possui uma controladora touch* Synaptics S3718, bem como análise do *driver* disponível no repositório da Google MSM. O protocol de comunicação foi analisado utilizado um Analisador Lógico.

- Uma configuração *touchscreen* é basicamente composta de um *display*, superfície de sensoriamento e controladora *touch*.
- A controladora *touch* costuma estar em uma PCB auxiliar, chamada de *daughter board*, que também possui um conector para a placa principal do *smartphone*.
- No caso da tela do Nexus 6P, existem múltiplas *daughter boards*.
- A *daughter board* da controladora *touch* do Synaptics S3718 tem os sinais necessários para comunicação I2C, além de um sinal dedicado a notificação de interrupções.

DB da controladora *touch* do Synaptics S3718



Configuração completa da tela



- Durante o processo de *boot* , o *driver* do periférico solicita à memória da controladora *touch* sobre quais funções a controladora possui.

- Isso é reportado em um descritor de função de 6 *bytes* com os endereços necessários e quantidade de tipos de interrupções que podem ser geradas.

- O *driver* então verifica se não há atualizações disponíveis, e eventualmente realiza as configurações necessárias das funções.

- Dois ataques foram apresentados no artigo, em que o primeiro permite que o atacante grave, intercepte e injete eventos *touch*, enquanto o segundo se aproveita de vulnerabilidades descobertas no *kernel* para obter acesso privilegiado.

- Ataques são demonstrados no Huawei Nexus 6P executando o SO Android 6.0.1, com configuração de fábrica.

- A configuração completa da tela foi desacoplada do resto do *smartphone*, e ao identificar a *daughter board* da controladora, esta foi separada da placa principal da tela usando ar quente.

- Com isso, a controladora foi soldada a fios de cobre, que por sua vez foram conectados em uma placa protótipo, simulando um cenário *chip-in-the-middle*.

- O ataque realizado utilizou um Arduino, baseado em um microcontrolador ATmega328.

- A idéia do primeiro ataque é que o microcontrolador analisa os eventos provenientes da tela e os manipula, injetando eventos no barramento de comunicação.

- No *firmware* do microcontrolador, foi programado máquinas de estado para analisar o modo em execução do teclado e o que é digitado, além de um banco de dados para mapear regiões da tela no teclado virtual.

- No teclado básico do Huawei Nexus 6P, há quatro modos: alfabeto, símbolos, números e emojis.

- A máquina de estado responsável pelo que é digitado compara o que foi pressionado com eventos específicos (como digitar uma URL). Quando um evento é detectado, a injeção *touch* é ativada e eventos são inseridos na linha de comunicação.

- O *hardware* utilizado foi capaz de criar eventos *touch* em uma taxa de 60 toques por segundo.

- O outro ataque explora vulnerabilidades no código OEM do *driver* embarcado no *kernel* do SO, para execução de código arbitrário privilegiado.
- Ao forjar informações adicionais de funcionalidades, fez-se com que o *driver* descobrisse mais interrupções do que sua estrutura interna pode armazenar, causando um *heap overflow*.
- Com isso, foi possível aumentar a quantidade de interrupções disponíveis fazendo um *override* no inteiro que armazena esse valor.
- Uma interrupção então é provocada, fazendo com que o *driver* solicite uma quantidade anormal de dados, causando um *buffer overflow*, que por sua vez é explorado utilizando uma cadeia ROP (*Return Oriented Programming)*.

Return Oriented Programming

- Existem mecanismos de proteção do *kernel* para injeção e execução de código malicioso, como o W⊕X.

- W⊕X é uma técnica para evitar a utilização de vulnerabilidades em tempo de execução. Tabelas de paginação do espaço de memória virtual jamais devem ter permissões de escrita E execução ao mesmo tempo.

Return Oriented Programming

- Surgiu então o ataque *return-to-libc*, em que o atacante utiliza um *buffer overflow* para sobrescrever o endereço de retorno da pilha com o endereço de uma instrução legítima localizada dentro da biblioteca libc, colocando os argumentos da função em outra porção da pilha.

- A técnica de ROP generalizou o ataque *return-to-libc* ao realizar uma cadeia de pequenos fluxos de execução de instruções, que então retornam.

- Várias instruções podem então ser combinadas no que é chamado de *gadget*.

- O microcontrolador desativa a energia do controlador *touch* e começa a imitar seu comportamento.
- Durante o boot, quando o *driver* solicitar o descritor de função para a controladora em endereços de valor maior do que normalmente existe dentro da controladora, o microcontrolador responde com um conjunto de descritores forjados.
- Esses descritores forjados vão fazer com que o mapa de registradores de interrupção exceda seus limites.
- Dentro do *driver, um loop* itera sobre o mapa de registradores de interrupção, que irá escrever valores fora do mapa de ativação de interrupções, fazendo com que o inteiro que guarda o número de fontes de interrupção seja sobrescrito.

- Depois que o boot é completado, o microcontrolador provoca uma interrupção fazendo com que o *driver* leia uma quantidade anormal de bytes, causando um *buffer overflow*.

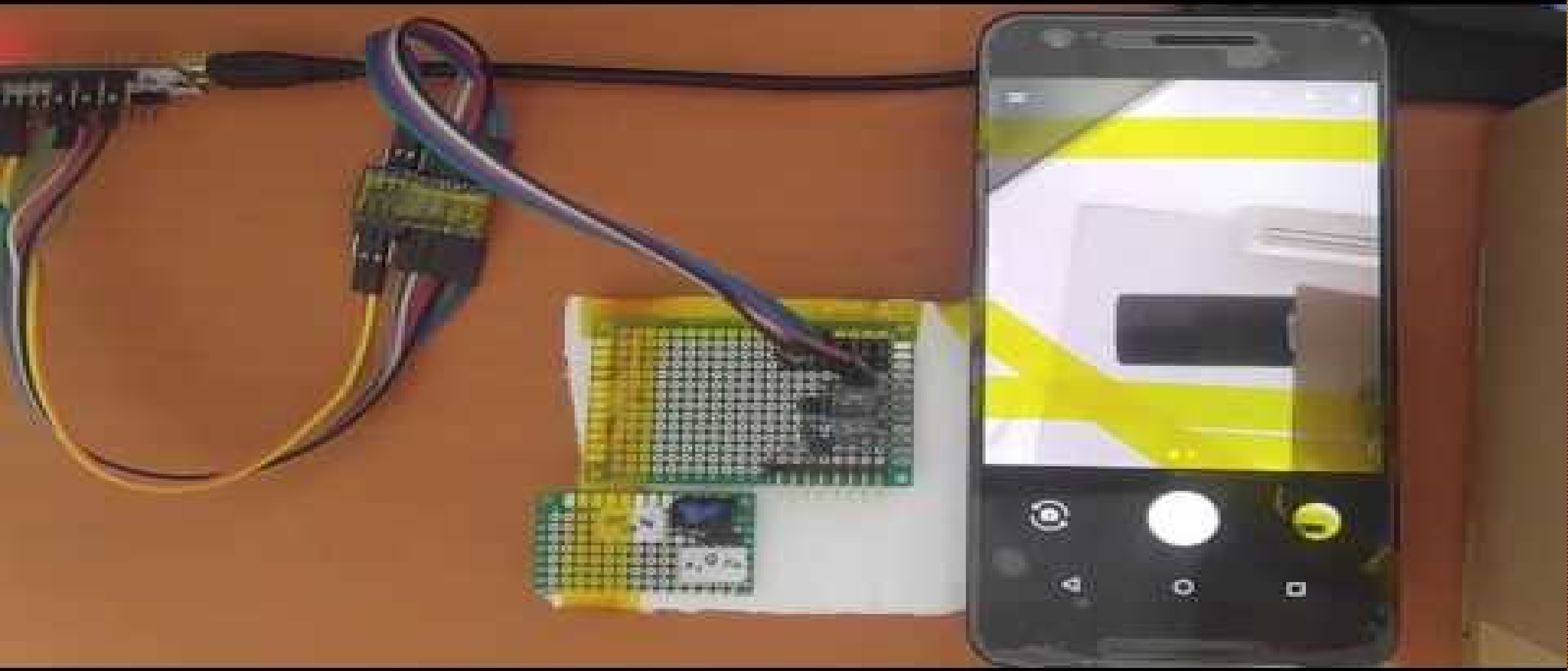
- Dentro dessa quantidade de bytes, existe uma cadeia ROP que chama funções do *kernel* para escrever determinadas informações em regiões de memória do *kernel* protegidas.

Gadget Order	Gadget Code	Relevant Pseudocode
1	ldp x19, x20, [sp, #0x10] ; ldp x29, x30, [sp], #0x20 ; ret;	Load arguments from stack to registers X19 and X20
2	mov x2, x19 ; mov x0, x2 ; ldp x19, x20, [sp, #0x10] ; ldp x29, x30, [sp], #0x30 ; ret;	Assign X2 := X19; Load arguments from stack to registers X19 and X20
3	mov x0, x19 ; mov x1, x20 ; blr x2 ; ldp x19, x20, [sp, #0x10] ; ldr x21, [sp, #0x20] ; ldp x29, x30, [sp], #0x30 ; ret;	Assign X0 := X19; Assign X1 := X20; Call X2(X0, X1)

- Com as informações escritas em regiões de memória do *kernel*, foi possivel:
  - Desativar checagens em chamadas de Sistema, permitindo que qualquer usuário e aplicativo consiga privilégios de *root*.
  - Incapacitar o modulo SELinux, não bloqueando atividades suspeitas.
  - Desabilitar checagem de *buffers* em todos os *buffers* de usuário em chamadas de sistema.
  - Abrir uma “porta dos fundos” para um atacante experiente.

- Após realizados os ataques individuais, foi realizada uma combinação de ataques para um efeito malicioso maior.
- É possível realizar ataques com a tela desligada, enquanto uma ação maliciosa é realizada.
- Nesse primeiro ataque combinado realizado, um aplicativo é instalado sem o consentimento do usuário, com permissões e direitos arbitrários.
- Em outro ataque combinado, o celular tira uma foto do usuário sem seu consentimento e envia por e-mail.


App



- Em um ataque combinado para comprometer todo o *smartphone*, foi utilizado uma combinação de *touch injection* e vulnerabilidades de *driver*.
- O atacante usa *touch injection* para instalar um aplicativo sem o consentimento do usuário. Da próxima vez que o telefone reiniciar, o microcontrolador cria uma vulnerabilidade durante o boot que é utilizada pelo aplicativo, tomando então o controle do sistema e realizando atividade maliciosa.
- Quando o aplicativo consegue executar comandos como *root*, ele desativa o modulo SELinux, exfiltra dados privados e *tokens* de autenticação e cria um *shell* de root remoto para o atacante.

Contramedidas sugeridas

- Sugere-se o conceito de “enfrentar *hardware* com *hardware*”, implementando um *firewall* I2C em *hardware*, acoplado na placa mãe do *smartphone*.

- Todas as vulnerabilidades encontradas pelos autores do artigo foram informadas à Google em fevereiro de 2017, e já foram corrigidas.

Obrigado pela atenção!