Urna Eletrônica: Segura?

Relatos do Teste Público de Segurança do Sistema Eletrônico de Votação – 2017

Caio Lüders, Diego Aranha, Paulo Matias, Pedro Barbosa, Thiago Cardoso

A Urna Eletrônica Brasileira

- Projeto de sistema embarcado extremamente complexo
- Dezenas de milhões de linhas de código
  - Bootloader customizado pelo TSE
  - Kernel Linux customizado pelo TSE
  - Drivers da Diebold (MSD — *Master Secure Device*)
  - Userland do TSE (`initje`)
  - Software da urna em si (`scue`, `vota`)

Panorama da área

- → Segurança de software é um problema **difícil**
- → Evolução lenta: “ainda executamos processos como *root* sob um kernel monolítico escrito em C”
- → 168 CVEs de execução de código arbitrário no kernel Linux só em 2017

Acidente ou sabotagem?

→ É tão fácil cometer erros de codificação que muitas vezes é difícil distinguir um erro acidental de uma backdoor

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
```

Q: Do you think the bug was an accident?

As others have said "if I wanted to backdoor Apple's SSL this is how I'd do it". It is hard for me to believe that the second "goto fail;" was inserted accidently given that there were no other changes within a few lines of it. In my opinion, the bug is too easy to exploit for it to have been an NSA plant. My speculation is that someone put it in on purpose so they (or their buddy) could sell it.

Seria a urna imune?

→ A urna **não** é magicamente imune a esses problemas

→ No TPS, nós conseguimos executar código arbitrário na urna

→ Corrigir a falha que encontramos não garante que não existam outras

Como tornar a urna segura?

- Pergunta errada
- Caminho mais viável:
  princípio da independência do software
- Linha do tempo
  - Lei 12.034/09: comprovante impresso
  - Anulado pela ADI 4543 (PGR)
  - Lei 13.165/15: volta do comprovante

Resistência e preconceito

SISTEMA ELETRÔNICO DE VOTAÇÃO
PERGUNTAS MAIS FREQUENTES

Justiça Eleitoral

8. POR QUE O VOTO NÃO É IMPRESSO?

O voto não é impresso pela urna eletrônica devido ao princípio constitucional de sigilo do voto e às vulnerabilidades associadas à manipulação de papel, essencialmente as mesmas que já existiam quando o voto não era eletrônico.

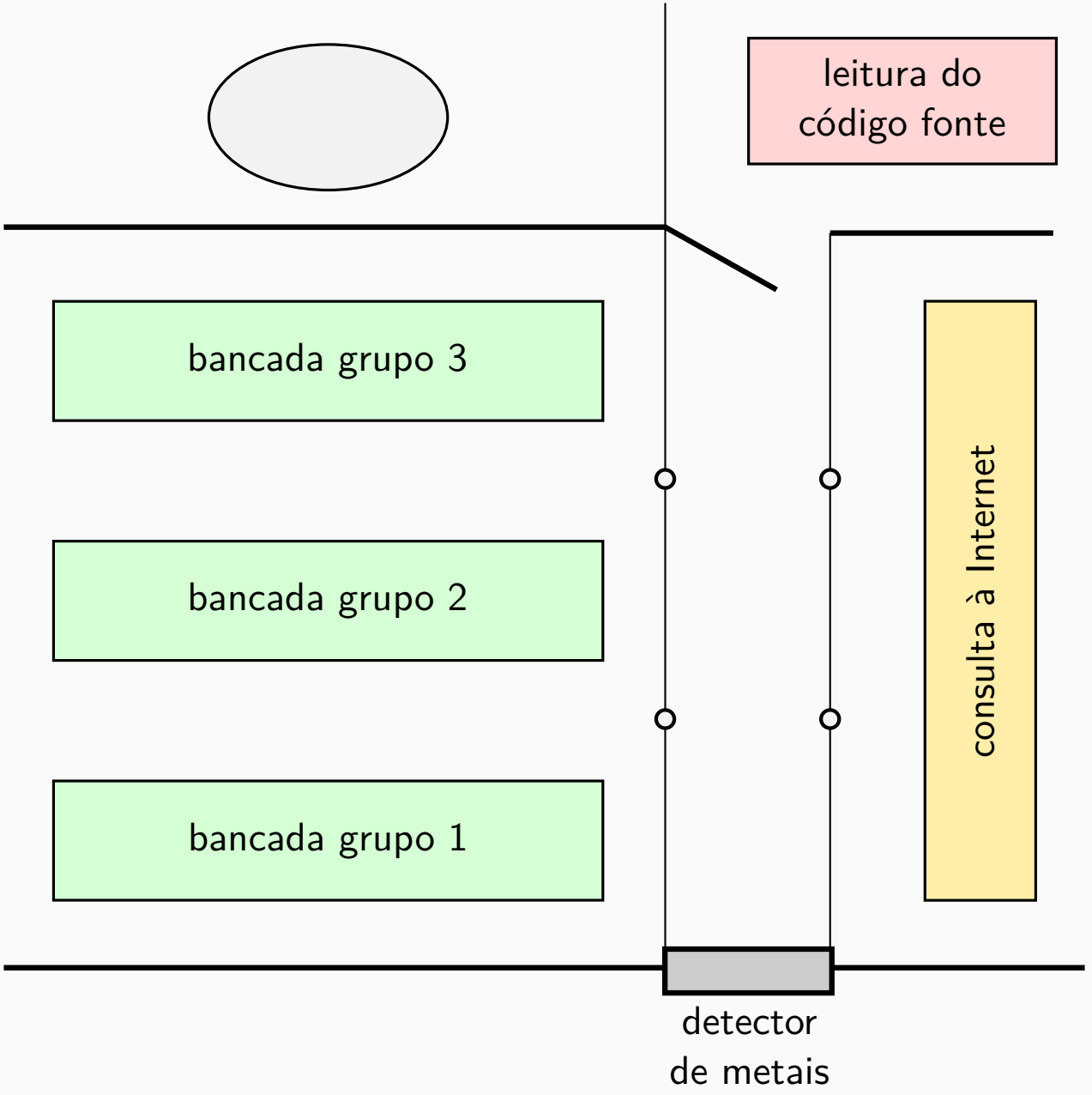
Nossos objetivos

- → Demonstrar a importância de avançar em direção ao *princípio da independência do software*
- → Inicialmente: explorar o maior número possível de vetores de ataque
- → Devido às dificuldades, mudamos para: explorar **em profundidade** o primeiro vetor de ataque que encontramos
- → Construir um ataque totalmente **sem premissas**

Como funcionam os testes?

- → Submetemos **planos de teste**
- → Os planos de teste são analisados e aprovados pelo TSE
- → Executamos os planos de teste em uma bancada com computador e urna eletrônica

Planta do ambiente



Dificuldades

- Entrada de software (em DVD-ROM) ou material impresso apenas após análise e aprovação de *solicitação de material*
- Regras aplicam-se mesmo para material discriminado nos planos de teste previamente aprovados
- Proibido transitar com anotações entre ambiente de leitura de código fonte e bancada de testes
- Problemas para habilitar virtualização nos computadores fornecidos

Segunda-feira e terça-feira

→ Burocracia

→ Preparação do ambiente

Carga da urna

- → Instalação do SO em uma urna virgem
- → Realizada por meio da inserção de um *cartão de carga*

Partições do cartão de carga

- Partição com o kernel
- Root do UENUX
  (GNU/Linux embarcado do TSE)
- Partição com executável do `vota`
  - Treinamento
  - Simulação
  - Votação

Ofuscação

→ Encriptação do kernel

→ Encriptação da partição do UENUX

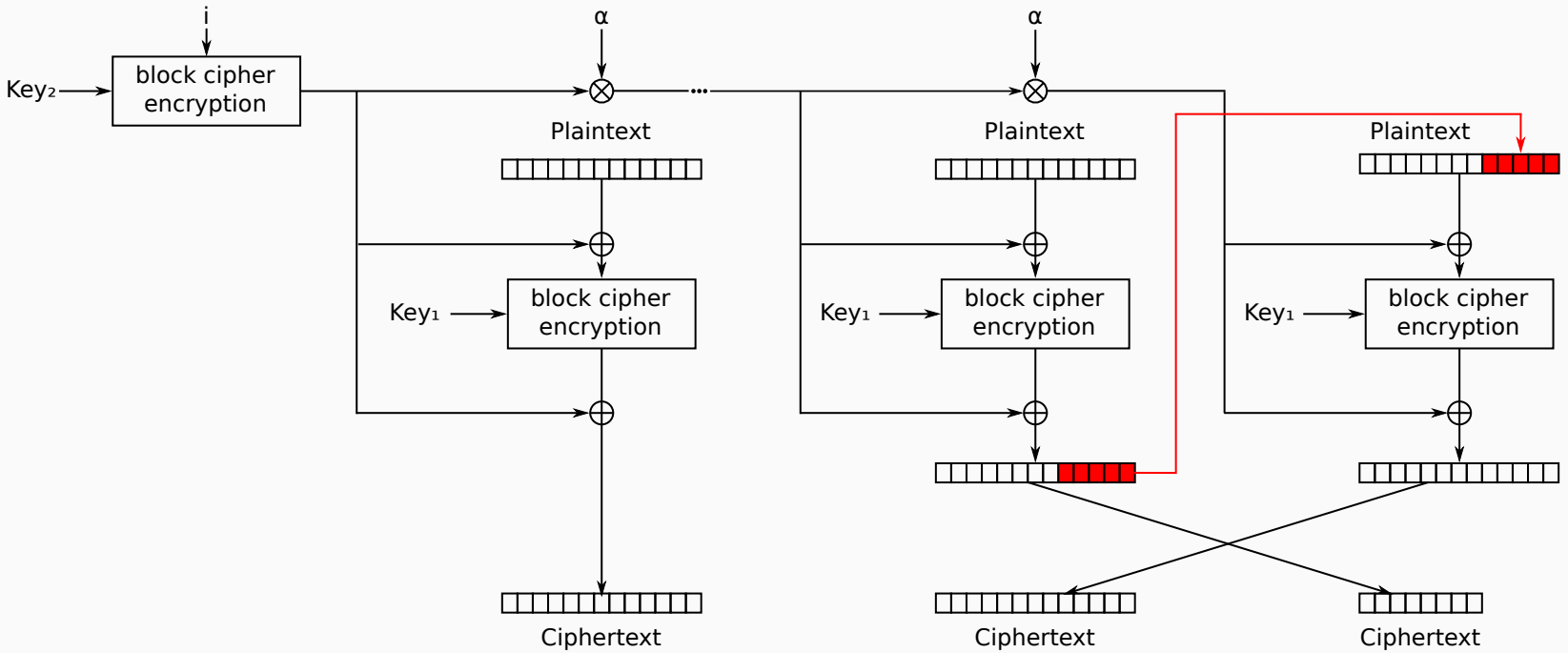
Kernel

- → Encriptado com AES'256-ECB
- → Decriptado no boot por `syslinux` modificado
  - → Acesso ao código fonte completo somente na sexta-feira
- → AES' $\neq$ AES (!!)
  - → Rounds pares são diferentes de rounds ímpares
  - → Aparentemente, intercala chaves de rodada de encriptação/decriptação do AES comum

Partição do UENUX

- Sistema de arquivos `ueminix`
  - Sistema de arquivos `minix` customizado pelo TSE
  - Estrutura de diretórios e nomes de arquivos intactos
  - Conteúdo dos arquivos encriptado individualmente com AES-XTS

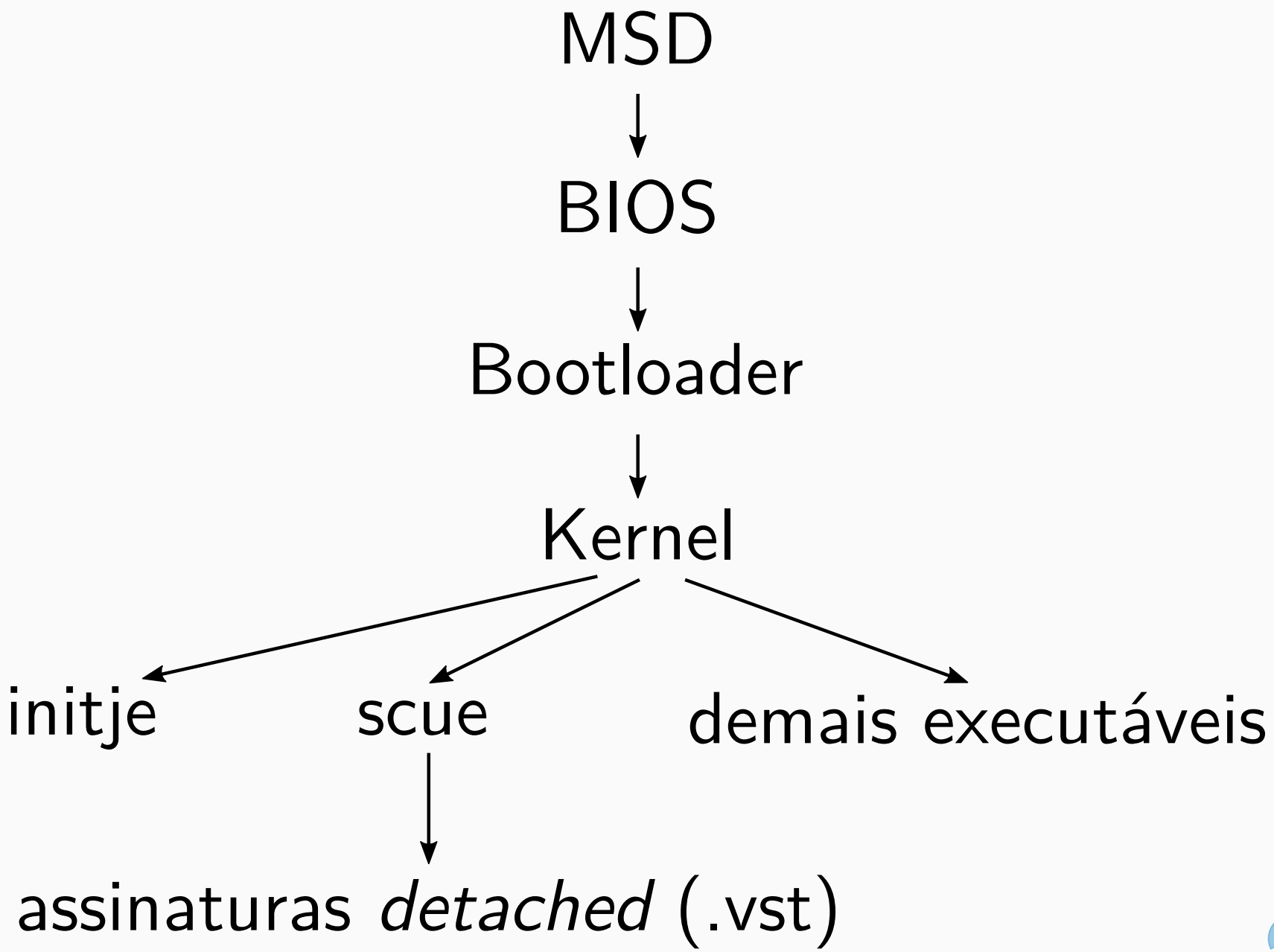
Modo XTS


i
Key₂
block cipher encryption
α
α
Plaintext
Plaintext
Plaintext
Key₁
block cipher encryption
Key₁
block cipher encryption
Key₁
block cipher encryption
Ciphertext
Ciphertext
Ciphertext

Encriptação do ueminix

- IV (*i*): inode do arquivo (consulte com `ls -i`)
- Chaves:
  - Key_1: *Hardcoded* no kernel
  - Key_2: Bytes no offset $512 + 128$ da partição, xor com o byte em $512 + 7$ (normalmente 0)
- Modificação com relação ao XTS padrão: reset do α de volta ao IV a cada 4096 bytes
  - Mais ofuscação
  - Enfraquece (!!) o algoritmo

Assinaturas (verificação)



Resumo da quarta-feira

- → Escrevemos um utilitário para ler (decriptando) e escrever (encriptando) partições `ueminix`
- → Usamos uma chave exfiltrada do código fonte do kernel para o AES-XTS
- → Encontramos bibliotecas compartilhadas que não estavam assinadas ⇒ **vetor de ataque**

Bibliotecas compartilhadas

- Execução de código arbitrário no espaço de memória de qualquer processo linkado à biblioteca!

- Bibliotecas vulneráveis
  - Log: usada por praticamente todos os executáveis
  - HKDF: usada pelo `scue` e pelo `vota`

Ataque simples ao sigilo do voto

→ Substituímos a função `hkdf` por:

```
xor eax, eax
ret
```

→ Essa função gera uma chave usada para encriptar o RDV com AES256-CBC

→ Assim, forçamos que o RDV seja encriptado com uma chave zerada

→ RDV só é lido em caso de “recontagem”: ataque difícil de detectar

Executando programas na urna

- → Kernel verifica assinatura de executáveis ⇒ não podemos trocar *e.g.* o `initje` por um programa nosso
- → Mas a função de log é chamada pelo `initje` antes do início da interface gráfica
- → Substituímos função de log por código em Assembly de um programa interativo escrito por nós, ou carregamos um ELF estático com `shellcraft.loader_append`

Resumo da quinta-feira

- → Violação do sigilo do voto
- → Violação da integridade do log
- → Execução de código Assembly escrito por nós na urna
- → Tentativa de analisar compactação UPX dos binários da urna ⇒ o código que levamos em DVD-ROM estava incompleto!

Modificando o software de votação

- Escrever um exploit sem ter um depurador
  - Necessário análise estática
  - Necessário descompactar o UPX
- Ausência de PIE (*position independent executable*) na urna
  - Incompatível com UPX?
  - Impede randomização de endereços do executável principal
  - Endereços fixos facilitam a escrita do exploit

Em qual executável estamos?

→ Gostaríamos de modificar somente o `vota`

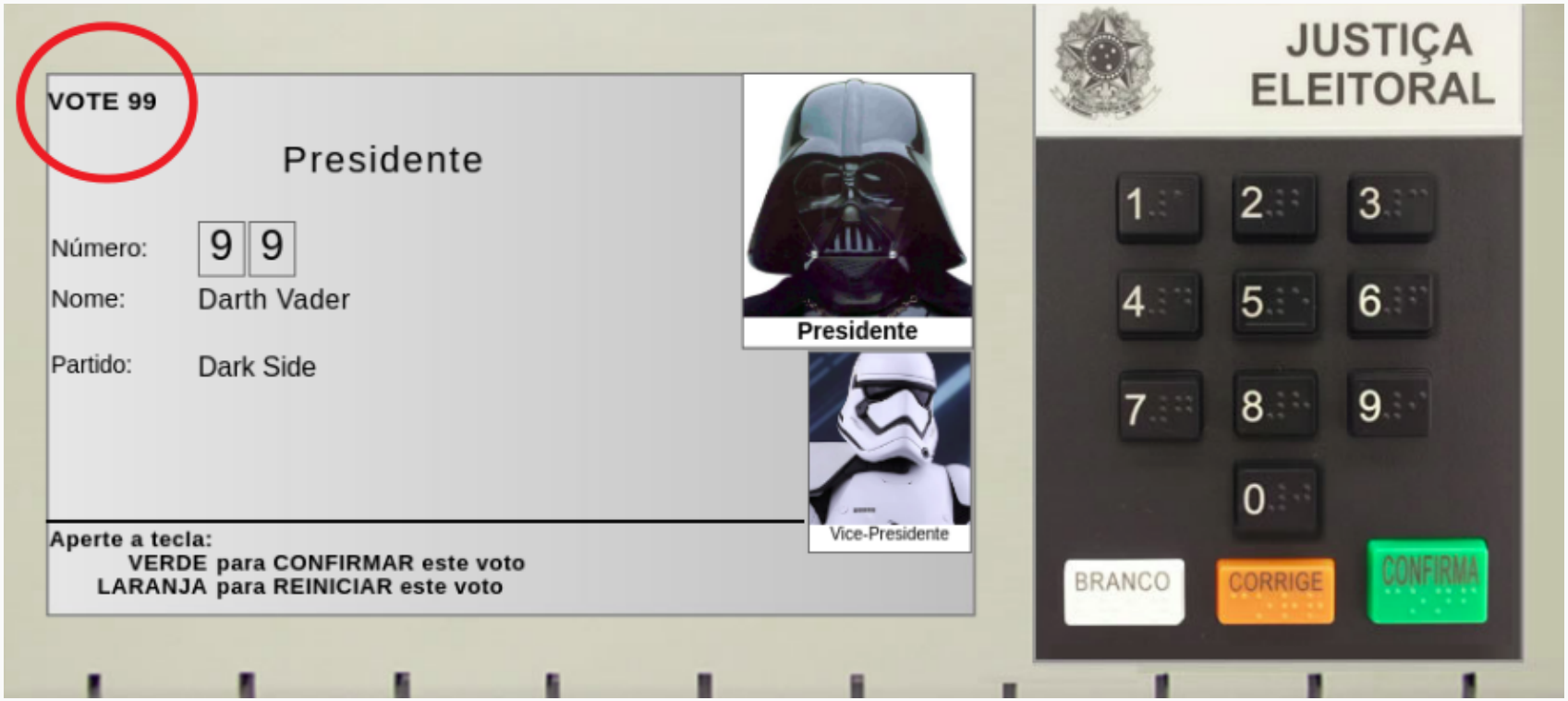
→ Procuramos um endereço que exista tanto no `vota` quanto no `scue`, mas com conteúdo diferente

```
mov eax, endereco
mov eax, [eax]
cmp eax, conteudo_esperado
jne nao_eh_o_vota
; altera memória
nao_eh_o_vota:
```

Mudança de strings

- → Strings constantes ficam na seção `.rodata`
- → Mudamos permissão de páginas de memória somente-leitura (syscall `mprotect`)
- → Sobrescrevemos endereços conhecidos (`stosd/w/b`)

Urna que faz boca de urna



Infecção furtiva da urna

→ Técnica de *infecção de ELF* na biblioteca HKDF

→ Desviamos o fluxo da função original para área inútil da biblioteca (ou usamos LIEF)

→ Adicionamos código novo mantendo a funcionalidade do original

Alteração do voto

→ Alterar somente o voto registrado pela urna

→ Não alterar o voto observado pelo eleitor na tela

Método que registra o voto

```
void InfoEleitor::AdicionaVoto(
    uint8_t cargo,
    int tipo,
    std::string &voto)
{
    cedula->AdicionaVoto(Voto(cargo,
                              tipo,
                              voto));
    LogaVoto(cargo, tipo);
}
```

Validação de hipótese

→ Substituímos o método `AdicionaVoto` por:

```
ret
```

→ Eleitor entra com votos para todos os cargos

→ Logo antes da tela de FIM, aparece a mensagem de erro: “cédula vazia”

Resumo da sexta-feira

- → Visita do ministro
- → Descompactação do UPX
- → Violação da integridade do software de votação
- → Exploits para troca de strings e alteração de cédulas de votação
- → Quebra do bootloader pela equipe do Peixinho, complementando nosso ataque e tornando-o completamente sem premissas

O que faríamos com mais tempo?

- Testar o exploit de alteração de voto na urna
  - Testamos código infectado do método `AdicionaVoto` no PC: funcionou
  - Mas cada carga demora $\approx$ 30 min
- Burlar votação paralela
  - Código para `AdicionaVoto` ligeiramente mais complexo

```
if (voto == ”99123”) {
  ativa_mutreta();
}
```

Conclusão

- Explorar a urna é conceitualmente o mesmo que fazer *jailbreak* em iPhone
- Falhas acontecem
- O *princípio da independência do software* é importante!