Blog

Writeup: AOC LC32D1320 — Stack OverFlow via Media Player + Port de DOOM

Aviso: Olá. Este writeup foi feito em Abril de 2026 e foi reescrito em julho de 2026 porque entreguei pra uma IA escrever e sempre odiei o resultado. Mas só agora, em julho, estou reescrevendo no meu jeito.

1. Sumário

Este writeup foi feito para a TV AOC LC32D1320, lançada em 2012. O sistema da LC32D1320 e de outras TVs irmãs que rodam a mesma base contém uma vulnerabilidade de Stack Buffer Overflow no sistema de legendas do Media Player do Sistema. A vulnerabilidade utiliza um arquivo de legenda .psb craftado, junto com um vídeo qualquer. O Media Player, quando abre o vídeo, ele percebe que existe um arquivo .psb com o mesmo nome do vídeo, e joga ele para o sistema de legenda do sistema, que passa para o parser de legenda PSB. O parser pega o texto da legenda e copia pra um buffer de tamanho fixo, só que sem checar o tamanho da legenda antes. Caso a legenda seja grande o suficiente, ele estoura o buffer e sobrescreve o que vem depois na stack.

2. A TV

A TV AOC LC32D1320 roda um SoC da família Trident com CPU MIPS32 big-endian. O sistema fala que o processador é um "DIGITAL TRIHIDTV SYSTEM". O firmware usa o sistema de arquivos MTD multi-partição. O boot é controlado por scripts shell, como `sbtvd.sh`, `common.sh`, `live.sh`, etc.

3. A Vulnerabilidade

3.1 Achando ela

A exploração da vulnerabilidade é influenciada bastante pelo `common.sh`, já que ele contém um trecho que configura o sistema para deixar coredumps em um dispositivo de armazenamento USB.

```
Linha 121 até 138 do common.sh:
#Coredump Config

if test -n "$(mount | grep "/opt type nfs")"
then
        echo "nfs"

#2 gen coredump to pwd
        echo "core.%e.%p.%s" > /proc/sys/kernel/core_pattern
else
        echo "image"

#1 gen coredump to USB Disk
        mkdir -p /etc/core
        mount -t vfat /dev/sda /etc/core/
        echo "/etc/core/core.%e.%p.%s" > /proc/sys/kernel/core_pattern
fi

ulimit -c unlimited
```

Caso essa parte não existisse, provavelmente, mesmo com a vulnerabilidade achada no parser de legenda, não seria possível criar o PSB craftado com os endereços necessários para funcionar, já que não seria possível encontrá-los sem o coredump ou sem abrir a TV fisicamente. Mas mesmo com o coredump, eu ainda precisava de uma vulnerabilidade para usar os endereços, e foi assim que fui encontrar a vulnerabilidade no parser.

A cadeia de chamada que vai para a vulnerabilidade começa no mmApp(Media Player), com a função `APP_MediaSUB_GetSubtitle()`, que chama `Tplf_TMP_GenSub_GetSubfileDesc()` em libtplftmplayerClient, que vai para `int_tapi_TMP_GenSub_GetSubfileDesc` em libtapi, que chama `gensub_splitter_init()` em libGenSub, e que finalmente chama a função vulnerável, `gensub_ParsePsb()`, na mesma libGenSub.

3.2 O `gensub_ParsePsb()`

Em `gensub_ParsePsb`, o parser copia o texto da legenda para um buffer de tamanho fixo sem verificação de comprimento:

```
Trecho do código descompilado do gensub_ParsePsb no Binary Ninja:
strcpy(arg2 + 0x11, texto_da_legenda)
```

Uma legenda longa o suficiente faz o buffer transbordar e sobrescrever a stack do `gensub_splitter_init`. Isso é importante, já que a stack guarda, além da legenda, registradores e o return address(`$ra`), que é o endereço pra onde a função deve voltar quando terminar de executar. Caso eu consiga sobrescrever esse endereço com outro endereço que eu quiser, eu controlo pra onde o programa tem que pular. Usando o coredump conseguido pelo `common.sh`, consegui confirmar onde cada valor cai dentro da legenda.

Tabelinha:

Offset no PSB	O que é sobrescrito
`0x000 - 0x0FF`	Texto da legenda (onde vai o payload)
`0x100 - 0x103`	`$s0` salvo em `gensub_ParsePsb`
`0x104 - 0x107`	`$s1` salvo em `gensub_ParsePsb`
`0x108 - 0x10B`	`$ra` salvo em `gensub_ParsePsb`

3.3 Montando o Exploit

No libc que o sistema usa, existe o gadget que usei. O gadget preserva `$a0`, que o parser aponta para o início da legenda.

```
Trecho do libc que contem o gadget:
move $t9, $s1
jalr $t9
nop
```

O gadget pega o valor `$s1` e salta pra ele. Como eu controlo o `$s1` através do PSB, eu posso botar qualquer endereço que eu quiser, e escolhi botar o endereço do `system()`. Assim, quando o gadget executa, ele chama o `system()`. E como `$a0` aponta para o texto da legenda, o texto vira o argumento para `system()`.

Com isso, fiz um PSB com um comando que escreve em `/etc/core`, onde é montado o dispositivo USB. Depois do comando, boto # para o shell ignorar o que vem depois, que são

os bytes dos registradores.

Trecho da legenda:

```
:>/etc/core/ace_psb.log #[padding até 0x104][s0][s1=system()][ra=gadget]
```

Usando os endereços obtidos pelo coredump, eu consegui executar o comando e foi escrito ace_psb.log no meu dispositivo USB.

Tabelinha dos endereços para caso você queira usá-los (Esses endereços nunca mudam, mas acho que são apenas da minha LC32D1320, e não funcionam em outras unidades.)

O quê	**Endereços**
libGenSub.so	0x2b273000
libc.so.0	0x2bbb8000
libc system()	0x2bc07240
gadget	0x2bbfaa44
caller_sp	0x7c6e5d40

4. Rodando meu código na TV

Como consegui executar um comando shell na TV, eu fui executar binários feitos por mim mesmo na TV.

4.1 Toolchain

Para compilar, é preciso do cross-compiler `mips-linux-gnu-gcc`, com as flags `-EB -mips32r2 -mabi=32`.

Todos os binários devem usar para conseguirem ser executados:

- `-nostartfiles` com custom `start.S` (Querido Leitor, já existe um `start.S` que eu fiz para o libaoc, mas você vai descobrir ele mais pra frente.)
- `-fno-stack-protector`
- `RUNPATH /usr/lib:/lib`
- Intérprete forçado: `/lib/ld-uClibc.so.0`

4.2 Framebuffer

Eu tentei usar o DirectFB, mas ele tem um problema de bootstrap pthread. O processo iniciado pelo `system()` herda um ambiente de runtime incompatível com a inicialização do manager de threads do LinuxThreads.

Eu decidi tentar usar o `/dev/hidtv2dge`, que é o framebuffer/OSD. Ele é diretamente mapeável via `mmap` sem nenhuma dependência do DirectFB ou pthread.

Com isso, peguei os valores do framebuffer.

```
Valores
id:             HiDTV SVP OSD
resolução:      960 x 540
bpp:            32 (ARGB8888)
stride:         3840 bytes/linha
framebuffer:    0x0e000000, 33554432 bytes (32 MiB)
page_len:       2073600 bytes (1 página = 960×540×4)
páginas mmap:   16
```

5. DOOM

Após eu ter conseguido fazer meu próprio código, eu tive que fazer o que todo mundo faz quando consegue a mesma coisa, rodar DOOM.

5.1 O DOOM em si

Bem antes de achar a vulnerabilidade, eu já conhecia o doomgeneric, que é uma modificação do código do DOOM, para ser possível portar mais facilmente para outras plataformas. Quando finalmente consegui achar a vulnerabilidade etc, como já documentado aqui nesse writeup, eu comecei a portar o doomgeneric para a TV.

Para ser possível ver o DOOM, ele escreve diretamente em `/dev/hidtv2dge`.

Como eu quero que essa seção seja pequena já que para mim não importa, porque o writeup é focado na vulnerabilidade em si e não no DOOM, eu não vou incluir detalhadamente como os gráficos são renderizados.

5.2 Input

Para andar, atirar, etc no DOOM, obviamente eu preciso de controles. Como eu não tinha o controle remoto da TV, eu tive que adaptar, e usar os botões que existem fisicamente na TV.

O `libdirectfb_trid_input.so` cria um socket em `/tmp/hp_dfb_handler` para escutar quais botões são pressionados na TV. Ele recebe um pacote de 8 bytes. O DOOM cria um socket no mesmo path, para capturar os pacotes de input que a TV envia.

Tabelinha do mapeamento dos botões físicos da TV

Botão	**Raw**	**DOOM**
Vol+	`0x0000003c`	Frente
Vol-	`0x0000003d`	Atrás
Menu	`0x00010319`	Atirar
CH+	`0x00010316`	Virar para esquerda
CH-	`0x00010317`	Virar para direita
Input	`0x00010318` / `0x3e` / `0x100017` (Não tenho certeza, mas funciona)	Usar

5.3 Resultado

O DOOM compilou, e consegui jogar o Shareware diretamente na TV, com os botões físicos, sem controle remoto.

Claro, teve alguns bugs e comportamentos estranhos nesse DOOM.

- Do jeito que o DOOM escreve no framebuffer, ele fica bem lento e injogável. Tive que fazer um jeito para ao invés dele escrever toda hora os frames, ele pular alguns, para aí sim ser possível jogar, mesmo em uma velocidade média.
- O DOOM não tem áudio, porque não fui mais a fundo para saber como funciona o sistema de áudio da TV.
- Quando você chega no elevador do final, ele crasha e fica travado na tela.
- Isso seria mais para outra seção, mas o Media Player continua rodando o vídeo em baixo do DOOM. Acho que apenas o parser de legendas trava, e não o media player,

então o vídeo continua rodando. Minha solução foi apenas criar um vídeo sem áudio.

- Sem inimigos no jogo.

6. libaoc

Após eu fazer o DOOM funcionar, pensei em fazer um mini-SDK, para quem tiver o mesmo modelo de TV, ou uma TV com o mesmo sistema, conseguir fazer seus próprios binários. Eu chamei ele de libaoc. O libaoc empacota o necessário para compilar binários e abrir esses binários pela vulnerabilidade que expliquei.

6.1 Funções

Ele é um "faz tudo" para a TV, acha os endereços com o coredump, dá suporte a teclado pra TV usar, suporte para os botões físicos, etc. Atualmente, ele está meio parado, mas estou tentando implementar suporte a Ethernet com o adaptador qf9700, e via tethering usb do celular, mas o qf9700 está mais próximo de concluir. O código do qf9700 ainda não está no libaoc porque ainda estou desenvolvendo e acho melhor não fazer push até concluir.

Lista do que ele mais faz

- Cria os PSB craftados para conseguir o coredump e executar os binários.
- No repositório já tem o código do port do DOOM que eu fiz.
- Suporte para usar o framebuffer.
- O que é necessário, como o start.S, etc para a TV rodar os binários.
- Configura o dispositivo USB para usar na TV.
- PSBs já feitos.

7. Extra

Essa seção era para eu falar coisas que vinham na minha mente sobre a TV. Mas, eu retirei essa seção para diminuir o writeup, já que o extra todo tem 9.089 caracteres.

Então, reformulei para botar 2 links aqui, um pro writeup completo, com o extra original, e o outro é o que foi escrito com IA, para caso alguém queira ler.

Links:

Original reescrito por mim: Reescrito por mim

Original com IA: Feito por IA

Obrigado para quem leu isso.

2026-07-13 | geral