Reversing Raw Binary Firmware Files in Ghidra

Nicholas Starke https://twitter.com/nstarke https://github.com/nstarke

This brief tutorial will show you how to go about analyzing a raw binary firmware image in Ghidra.

Prep work in Binwalk

I was recently interested in reversing some older Cisco IOS images. Those images come in the form of a single binary blob, without any sort of ELF, Mach-o, or PE header to describe the binary.

While I am using Cisco IOS Images in this example, the same process should apply to other Raw Binary Firmware Images.

That makes importing this type of firmware file difficult, as Ghidra doesn't have any idea what type of ISA it needs to disassemble and decompile for.

The following are a few things I learned while trying to get the Cisco IOS image in a reversible state within Ghidra.

First I had to extract the image. I pulled the firmware image off a switch I recently bought over TFTP. It turns out the first 112 bytes are some sort of Cisco proprietary header that is not useful for our purpose. We need to extract the bzip2 archive that we are interested in. The easist way to do that is binwalk:

```
binwalk -eM c3750-ipservicesk9-mz.122-50.SE3.bin
```

This will create a `_c3750-ipservicesk9-mz.122-55.SE.bin.extracted` directory which will have a file named `70` inside it.

Now we need to figure out the CPU ISA. For this we use binwalk again:

```
binwalk -m /usr/local/lib/python2.7/dist-packages/binwalk/magic/binarch
_c3750-ipservicesk9-mz.122-55.SE.bin.extracted/70
```

This will output a lot of things, so lets take a look at the output:

```
DECIMAL       HEXADECIMAL     DESCRIPTION
---------------------------------------------------------------------------
-----
24            0x18            PowerPC big endian instructions, function
prologue
1360          0x550           PowerPC big endian instructions, function
epilogue
1364          0x554           PowerPC big endian instructions, function
epilogue
```

```
1372          0x55C          PowerPC big endian instructions, function
epilogue
1380          0x564          PowerPC big endian instructions, function
epilogue
1388          0x56C          PowerPC big endian instructions, function
prologue
1612          0x64C          PowerPC big endian instructions, function
epilogue
1648          0x670          PowerPC big endian instructions, function
epilogue
1656          0x678          PowerPC big endian instructions, function
prologue
3224          0xC98          PowerPC big endian instructions, function
epilogue
3232          0xCA0          PowerPC big endian instructions, function
prologue
6772          0x1A74         PowerPC big endian instructions, function
epilogue
6780          0x1A7C         PowerPC big endian instructions, function
prologue
[...]
```

We can see that the binary has Big Endian PowerPC function prologues followed by epilogues. This is a good indicator that the firmware image ISA is PowerPC Big Endian.

Now that we know the ISA, we need to know the text-base offset and the data-base offset within the firmware image. The best way to figure this out is to load the firmware on an actual device and boot up the device.

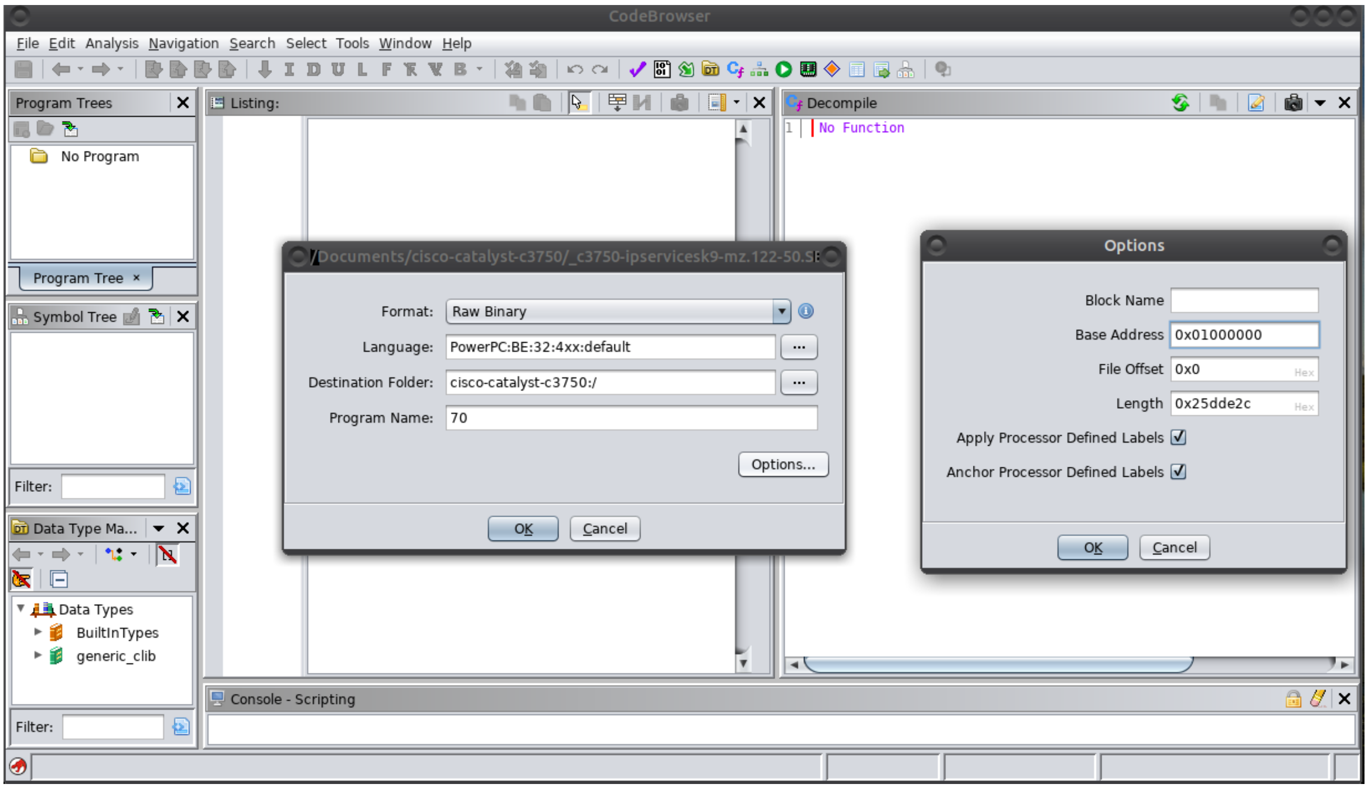
To retrieve the base address (fileOffset), run the `show version` command on the Cisco Switch:

```
Switchy>
Switchy>enable
Password:
Switchy#show version
Cisco IOS Software, C3750 Software (C3750-IPSERVICESK9-M), Version
12.2(55)SE, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2010 by Cisco Systems, Inc.
Compiled Sat 07-Aug-10 22:45 by prod_rel_team
Image text-base: 0x01000000, data-base: 0x02F00000

ROM: Bootstrap program is C3750 boot loader
BOOTLDR: C3750 Boot Loader (C3750-HBOOT-M) Version 12.2(44)SE5, RELEASE
SOFTWARE (fc1)
[...]
```

This dumps out a line: `Image text-base: 0x01000000, data-base: 0x02F00000`.
Both of those addresses are important, so note them and save them for later.

Ghidra Time

Now open the `70` binary blob in ghidra. Again, since there is no standardized binary format for the binary, you will have to import the file as `Raw Binary`, and then set the Code Architecture to `PowerPC Big Endian 4xx`. Also, click the options button and set the image offset base to the value we retrieve from the `show version` command: 0x01000000. Then import.



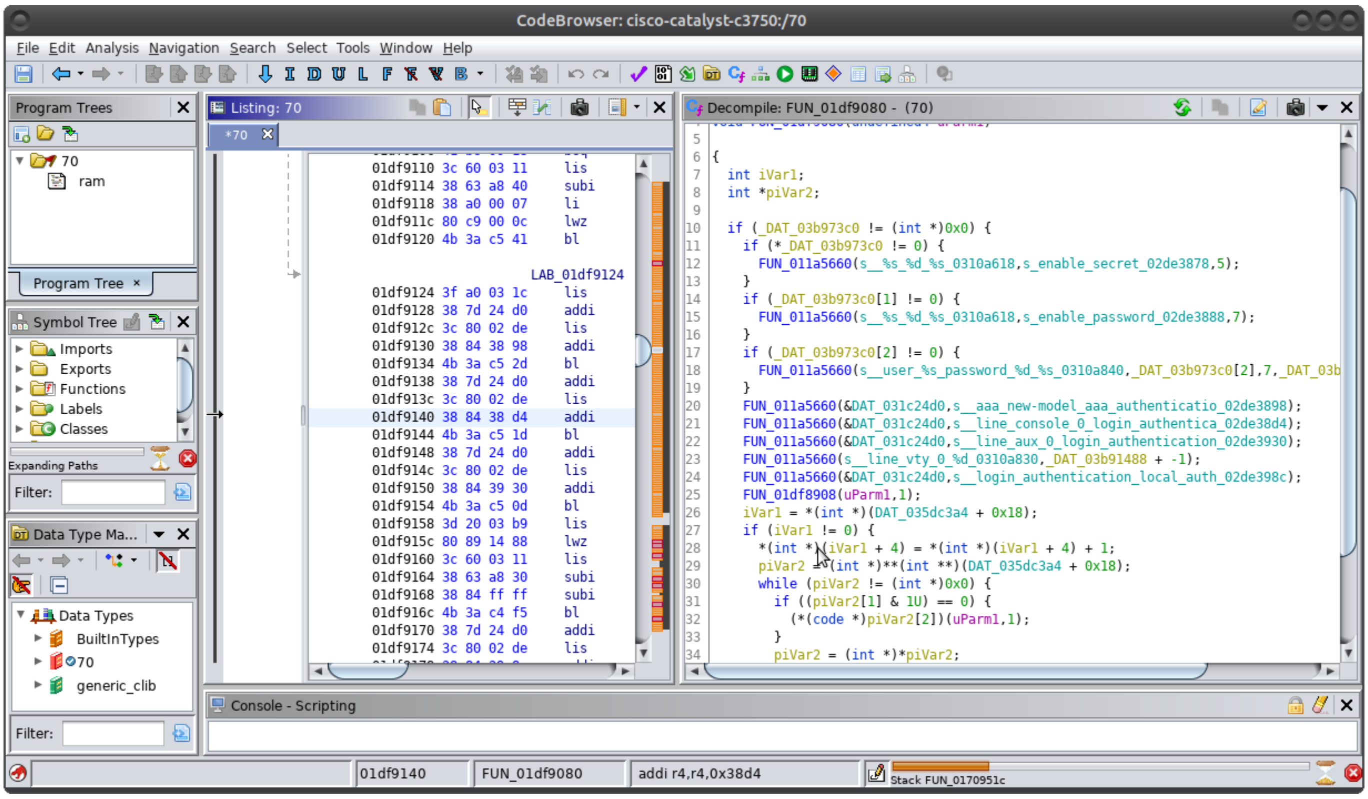
/Documents/cisco-catalyst-c3750/_c3750-ipservicesk9-mz.122-50.SE

Format: Raw Binary
Language: PowerPC:BE:32:4xx:default
Destination Folder: cisco-catalyst-c3750:/
Program Name: 70

Options

Block Name:
Base Address: 0x01000000
File Offset: 0x0
Length: 0x25dde2c
Apply Processor Defined Labels ☑
Anchor Processor Defined Labels ☑

Ghidra will then churn on the binary for a while, and when it is done the strings should be resolved to labels within the decompiler. This is because the label regions in memory are marked as Read/Write within ghidra. We want to resolve those labels to strings for ease of use.

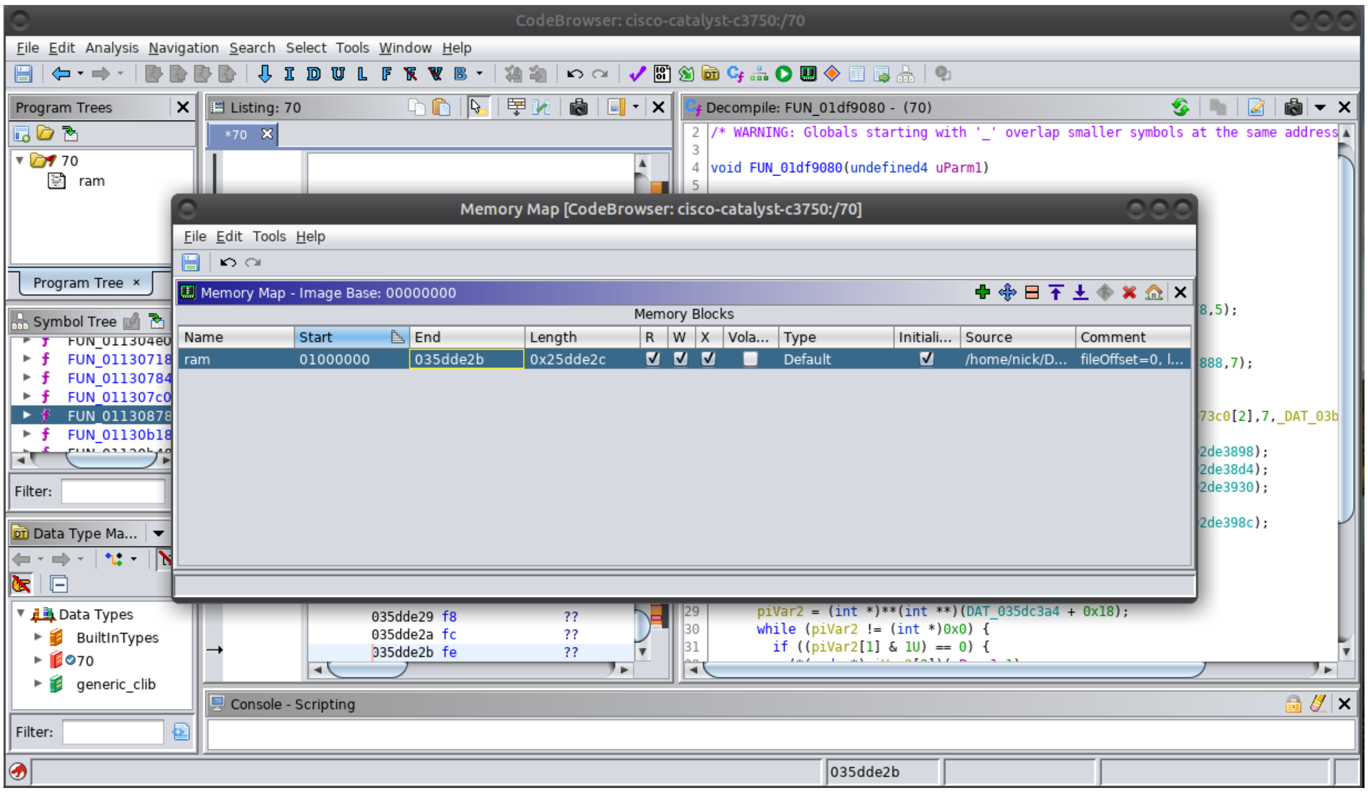


```
01df9110 3c 60 03 11     lis
01df9114 38 63 a8 40     subi
01df9118 38 a0 00 07     li
01df911c 80 c9 00 0c     lwz
01df9120 4b 3a c5 41     bl

                     LAB_01df9124
01df9124 3f a0 03 1c     lis
01df9128 38 7d 24 d0     addi
01df912c 3c 80 02 de     lis
01df9130 38 84 38 98     addi
01df9134 4b 3a c5 2d     bl
01df9138 38 7d 24 d0     addi
01df913c 3c 80 02 de     lis
01df9140 38 84 38 d4     addi
01df9144 4b 3a c5 1d     bl
01df9148 38 7d 24 d0     addi
01df914c 3c 80 02 de     lis
01df9150 38 84 39 30     addi
01df9154 4b 3a c5 0d     bl
01df9158 3d 20 03 b9     lis
01df915c 80 89 14 88     lwz
01df9160 3c 60 03 11     lis
01df9164 38 63 a8 30     subi
01df9168 38 84 ff ff     subi
01df916c 4b 3a c4 f5     bl
01df9170 38 7d 24 d0     addi
01df9174 3c 80 02 de     lis
```

```c
{
  int iVar1;
  int *piVar2;

  if (_DAT_03b973c0 != (int *)0x0) {
    if (*_DAT_03b973c0 != 0) {
      FUN_011a5660(s__%s_%d_%s_0310a618,s_enable_secret_02de3878,5);
    }
    if (_DAT_03b973c0[1] != 0) {
      FUN_011a5660(s__%s_%d_%s_0310a618,s_enable_password_02de3888,7);
    }
    if (_DAT_03b973c0[2] != 0) {
      FUN_011a5660(s__user_%s_password_%d_%s_0310a840,_DAT_03b973c0[2],7,_DAT_03b
    }
    FUN_011a5660(&DAT_031c24d0,s__aaa_new-model_aaa_authenticatio_02de3898);
    FUN_011a5660(&DAT_031c24d0,s__line_console_0_login_authentica_02de38d4);
    FUN_011a5660(&DAT_031c24d0,s__line_aux_0_login_authentication_02de3930);
    FUN_011a5660(s__line_vty_0_%d_0310a830,_DAT_03b91488 + -1);
    FUN_011a5660(&DAT_031c24d0,s__login_authentication_local_auth_02de398c);
    FUN_01df8908(uParm1,1);
    iVar1 = *(int *)(DAT_035dc3a4 + 0x18);
    if (iVar1 != 0) {
      *(int *)(iVar1 + 4) = *(int *)(iVar1 + 4) + 1;
      piVar2 = *(int *)**(int **)(DAT_035dc3a4 + 0x18);
      while (piVar2 != (int *)0x0) {
        if ((piVar2[1] & 1U) == 0) {
          (*(code *)piVar2[2])(uParm1,1);
        }
        piVar2 = (int *)*piVar2;
```

01df9140 | FUN_01df9080 | addi r4,r4,0x38d4

Navigate to the data-base address of `0x02f00000 - 0x01000000` in Ghidra. This last portion of the blob is where all the strings that are referenced in code live. The actual location where these strings start is a little lower than `0x02f00000 - 0x01000000`, we will need to manually inspect the binary to see just where that location is. This is most likely due to some sort of offset within text-base that I am simply not aware of. I came up with the location `DAT_02de33d8` (`0x02de33d8`). We arent necessarily looking for an exact location.



Memory Map [CodeBrowser: cisco-catalyst-c3750:/70]

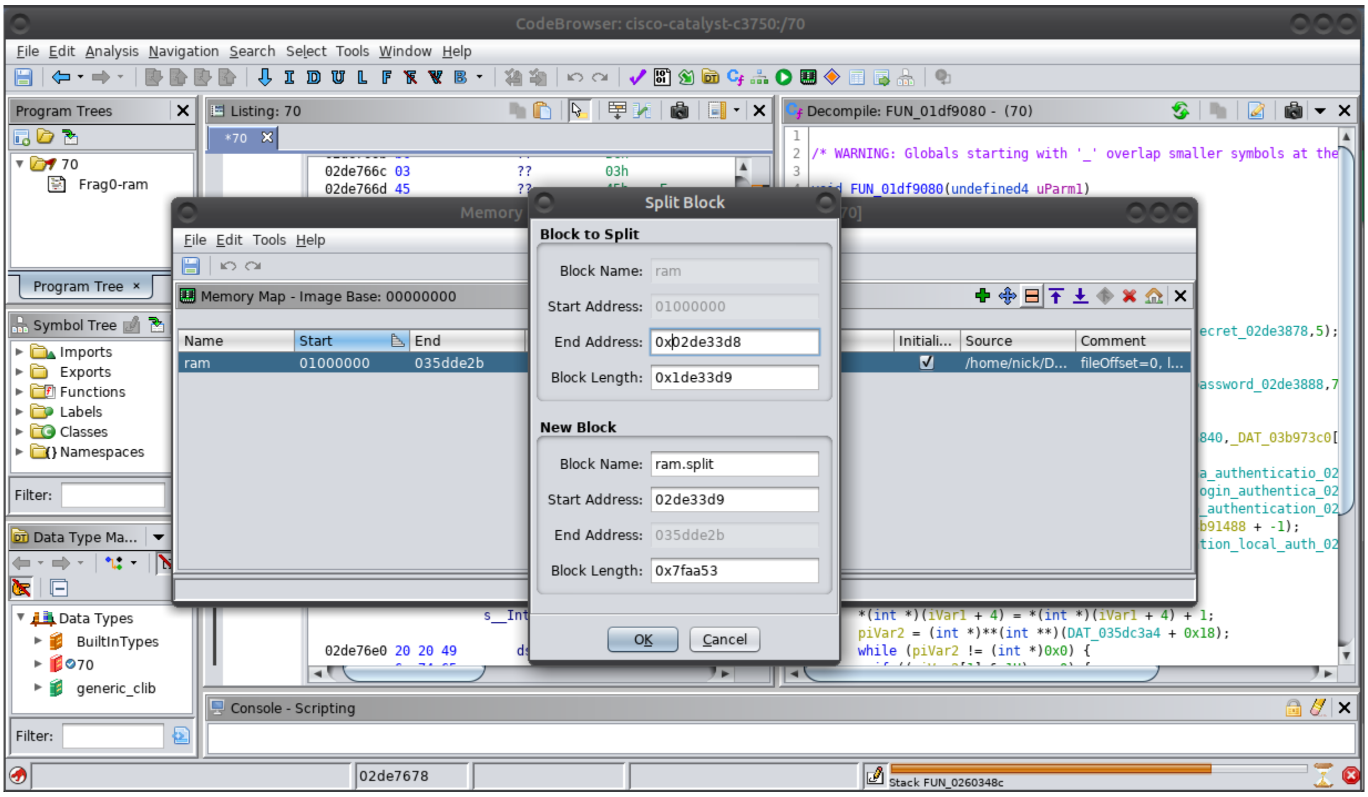
Memory Map - Image Base: 00000000

Memory Blocks

Name	Start	End	Length	R	W	X	Vola...	Type	Initiali...	Source	Comment
ram	01000000	035dde2b	0x25dde2c	✓	✓	✓		Default	✓	/home/nick/D...	fileOffset=0, l...

```
035dde29 f8     ??
035dde2a fc     ??
035dde2b fe     ??
```

Open up `Window->Memory Map` and click the `Split` button up in the right hand corner of that screen. You will need to split at the data-base address `0x02f00000`, and then mark the data-base memory region as Read Only.


CodeBrowser: cisco-catalyst-c3750:/70
File Edit Analysis Navigation Search Select Tools Window Help
Program Trees
Listing: 70
Decompile: FUN_01df9080 - (70)
Symbol Tree
Data Type Ma...
Console - Scripting

Split Block

Block to Split

Block Name:	ram
Start Address:	01000000
End Address:	0x02de33d8
Block Length:	0x1de33d9

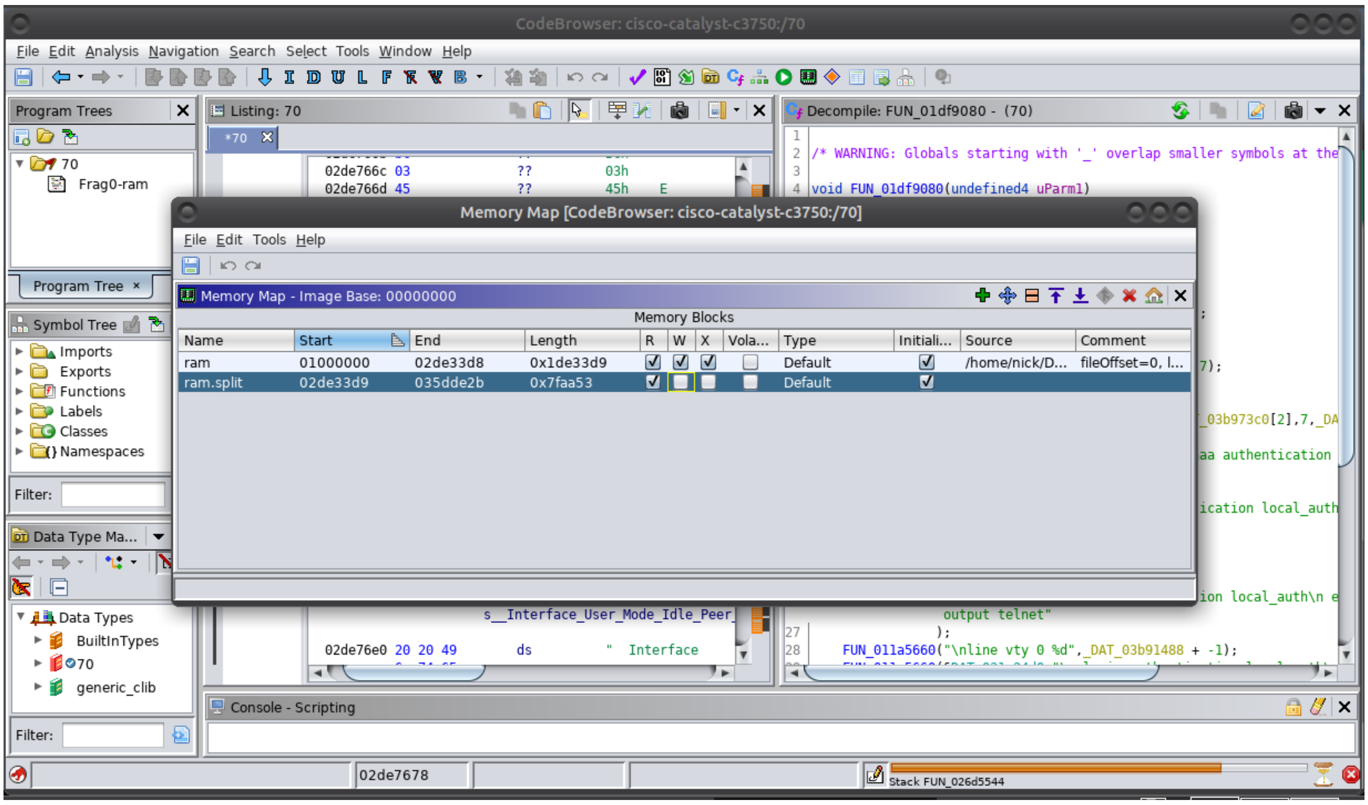
New Block

Block Name:	ram.split
Start Address:	02de33d9
End Address:	035dde2b
Block Length:	0x7faa53

OK Cancel

Memory Map - Image Base: 00000000

Name	Start	End	Initiali...	Source	Comment
ram	01000000	035dde2b	✓	/home/nick/D...	fileOffset=0, l...

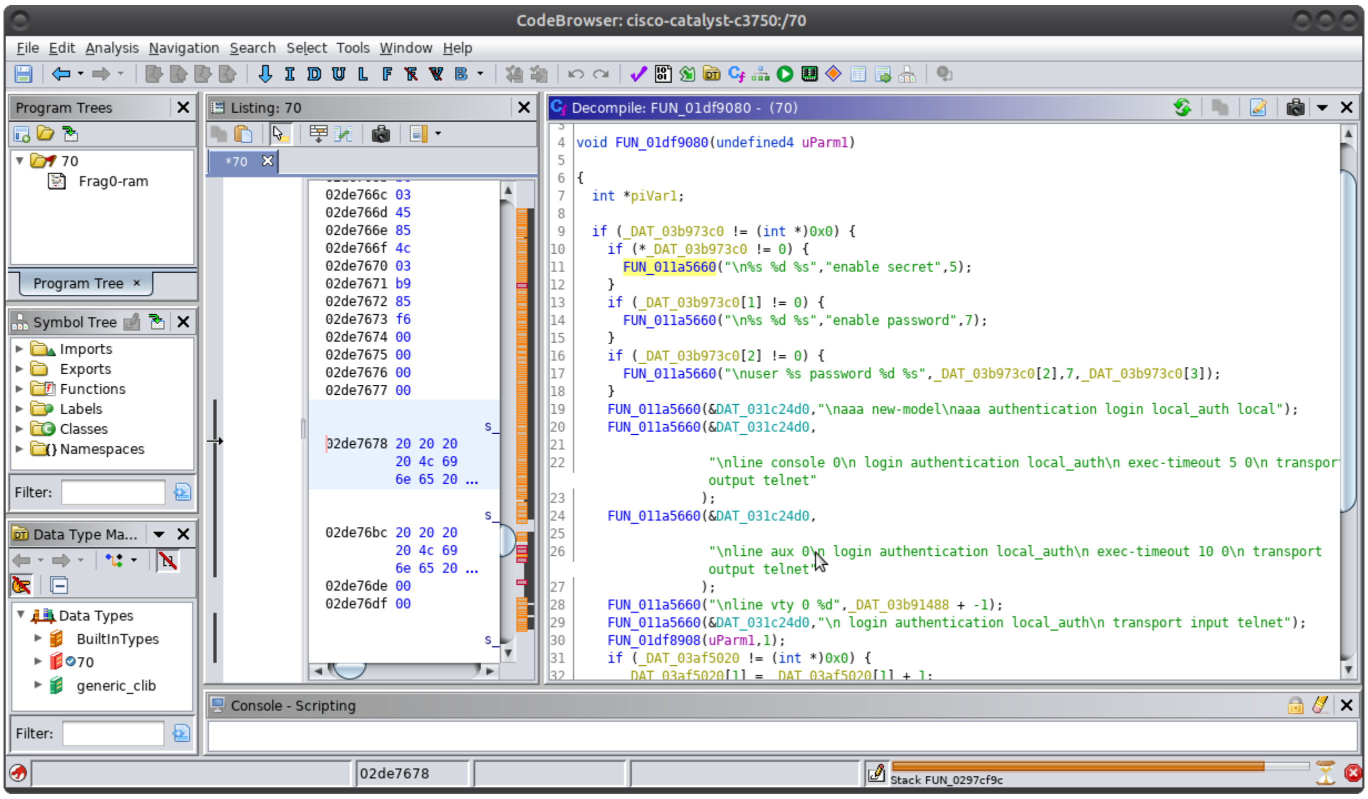

CodeBrowser: cisco-catalyst-c3750:/70
File Edit Analysis Navigation Search Select Tools Window Help
Memory Map [CodeBrowser: cisco-catalyst-c3750:/70]
File Edit Tools Help

Memory Map - Image Base: 00000000

Memory Blocks

Name	Start	End	Length	R	W	X	Vola...	Type	Initiali...	Source	Comment
ram	01000000	02de33d8	0x1de33d9	✓	✓	✓		Default	✓	/home/nick/D...	fileOffset=0, l...
ram.split	02de33d9	035dde2b	0x7faa53	✓				Default	✓		

After you have completed these steps, the labels should resolve to strings and you should be able to start reversing the image quite easily.



```
02de766c 03
02de766d 45
02de766e 85
02de766f 4c
02de7670 03
02de7671 b9
02de7672 85
02de7673 f6
02de7674 00
02de7675 00
02de7676 00
02de7677 00
                  s_
02de7678 20 20 20
         20 4c 69
         6e 65 20 ...
                  s_
02de76bc 20 20 20
         20 4c 69
         6e 65 20 ...
02de76de 00
02de76df 00
                  s_
```

```
void FUN_01df9080(undefined4 uParm1)

{
  int *piVar1;

  if (_DAT_03b973c0 != (int *)0x0) {
    if (*_DAT_03b973c0 != 0) {
      FUN_011a5660("\n%s %d %s","enable secret",5);
    }
    if (_DAT_03b973c0[1] != 0) {
      FUN_011a5660("\n%s %d %s","enable password",7);
    }
    if (_DAT_03b973c0[2] != 0) {
      FUN_011a5660("\nuser %s password %d %s",_DAT_03b973c0[2],7,_DAT_03b973c0[3]);
    }
    FUN_011a5660(&DAT_031c24d0,"\naaa new-model\naaa authentication login local_auth local");
    FUN_011a5660(&DAT_031c24d0,

                 "\nline console 0\n login authentication local_auth\n exec-timeout 5 0\n transpor
                 output telnet"
                );
    FUN_011a5660(&DAT_031c24d0,

                 "\nline aux 0\n login authentication local_auth\n exec-timeout 10 0\n transport
                 output telnet"
                );
    FUN_011a5660("\nline vty 0 %d",_DAT_03b91488 + -1);
    FUN_011a5660(&DAT_031c24d0,"\n login authentication local_auth\n transport input telnet");
    FUN_01df8908(uParm1,1);
    if (_DAT_03af5020 != (int *)0x0) {
      _DAT_03af5020[1] = _DAT_03af5020[1] + 1;
```