Reversing P25 Radio Scanners

Let's beat a dead horse.

SUBATOMIC SECURITY

Super Quick Presentation

- Founder of an obscurely named infosec company (see footer)
  - founded at the end of 2011
  - infosec dev, pentests
  - president, lead coder, chief janitor
    - I promise it will change soon ;)
- President of northsec competition (nsec.io)
- Not particularly awesome in anything
- Twitter idling @etrangeca

What it's all about

- Uniden BC296D
- Complete process from firmware update file to code execution
- Toolset presentation
- AES over P25 still stand
  - for how long... ;)
- A nice adventure!

What is P25 anyway?

- Suite of standards for digital radio
- "Closed" open standards
  - You have to pay for the documentation
- Developed by a set of "trustworthy" organisations
  - NSA
  - DoD
- Large scale adoption in North America
  - Public services
  - Polices forces

P25, isn't it more like "molesting" a dead horse?

- Well... maybe.
- Osmocom OP25 did a very good job
- multiple p25 support in sdr radios
- some attacks are starting to appear.

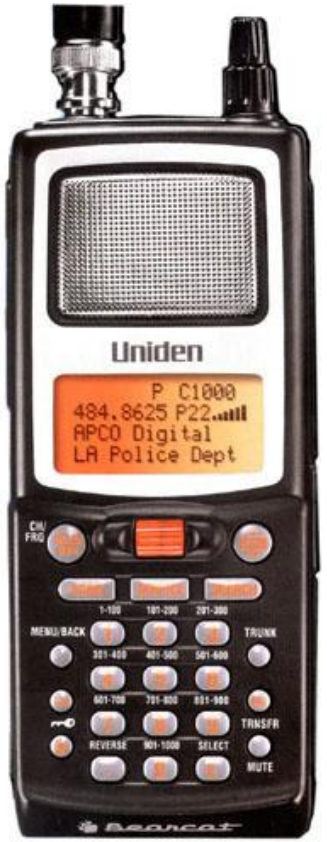
- Sorry, it was fun to be on stage for 5 minutes
- See you soon...

P25, isn't it more like "molesting" a dead horse?

- This talk is not about the protocol
- There's still some cool things to reverse
  - Trunking algorithms
  - Fast searching
  - Proprietary tweaks
- SDR's are not what we can call
  - Portable
  - User friendly
- Re-purposing devices is good for the planet...

The Beast

- Uniden BC296D
- Released in 2002
- 9600bps APCO P25 Compatible
- Trunk Tracking ~14 types
- 25 mhz - 1.3ghz (not continuous)
- Many more features
- Still an amazing device
- http://wiki.radioreference.com/index.php/BC296D



SUBATOMIC SECURITY

The Beast (2)

- Uniden BCi96D
- "Optional" P25 daughter board
- P25 Protocol implementation
- Audio Decoding -> C4FM/CQPSK DSP

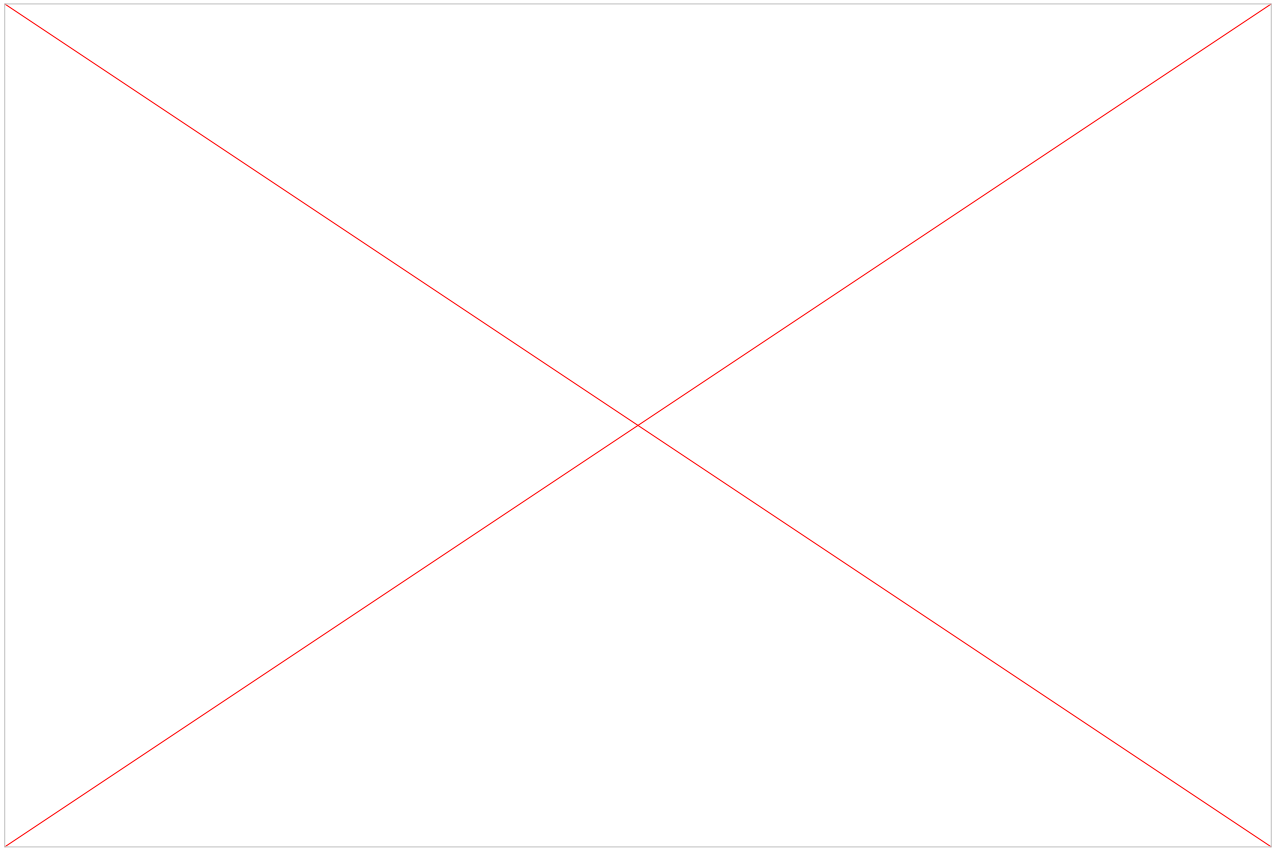
Why this model?

- Cheap P25 scanner (~350$ used)
- Firmware updates for both components
  - Radio
  - Daughter Board
- Old
  - ... so I don't care if I brick it.

Adventure time!

SUBATOMIC SECURITY

Hardware recon

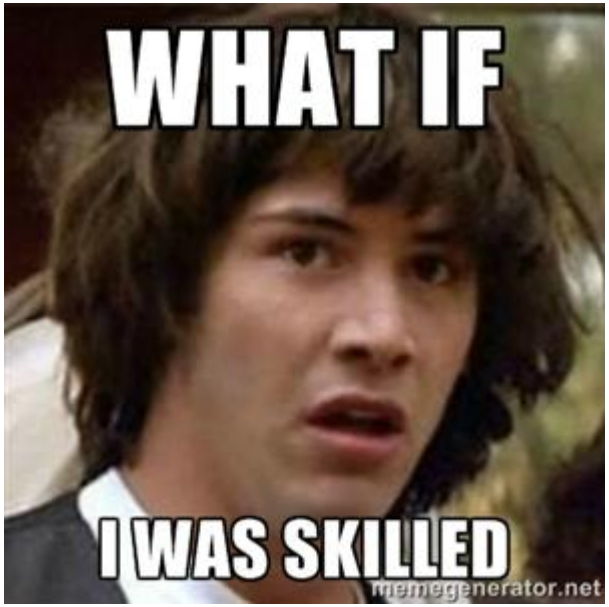


Interesting Hardware bits

- Main radio
  - Renesas m16c/62P (m16c/60 serie)
    - 256k ROM
    - 20k RAM
    - 15-32 mhz
    - 16 bits I/O
  - User config is stored in eeprom (as pictured)
- Daughter board
  - Renesas m16c/62N (cheaper 62P version)
  - Texas instruments TMS 160, 320VC5410APGE

Now that we know that

- What should we do?



Firmware file analysis

- Let's look at the firmware update file

```
S00C000000822300F28400822333
S2240A0000B8A9BDCCDBDB11202D2E242A29DBEEE9F1EBDBDBBBBBFDFEEDF4F1FFDBDBDBDB3C
S2240A0020DBDBDBDBDBDBBBBB10FDEEECF415DBDBDBDBDBDBDBDBDBDBBBBBBC9C46C5BBE126
S2240A004055C5BBFA47C5BB4047C5BB4047C5BB6F47C5BB6F47C5BB6F47C5BB8E47C5BBE115
S2240A006055C5BBE155C5BBE155C5BBE155C5BBE155C5BBE155C5BBE155C5BBE155C5BBE1C1
S2240A008055C5BBAD47C5BBF448C5BB1948C5BB5F48C5BB7E48C5BBE155C5BBB048C5BBB020
S2240A00A048C5BB684AC5BBE155C5BBE155C5BBE155C5BBE155C5BBD549C5BBE155C5BBE12A
S2240A00C055C5BBE155C5BB5A4AC5BBE155C5BBFF49C5BBE155C5BBE0BC0CC330BE0CC3BB4B
S2240A00E0BBBBBBE0BC0CC330BE0CC3E0C10CC330C30CC3BBBBBBBBE0BC4FC330BE4FC3E0EC
```

- Yay! Not binary.

Firmware file analysis

- Motorola S-Record
  - Nope IDA, intel s-record does not exists
- Stolen from Wikipedia
  - **Start code**, one character, an *S*.
  - **Record type**, one digit, *0* to *9*, defining the type of the data field.
  - **Byte count**, two hex digits, indicating the number of bytes (hex digit pairs) that follow in the rest of the record (in the address, data and checksum fields).
  - **Address**, four, six, or eight hex digits as determined by the record type for the memory location of the first data byte. The address bytes are arranged in big endian format.
  - **Data**, a sequence of 2*n* hex digits, for *n* bytes of the data.
  - **Checksum**, two hex digits - the least significant byte of ones' complement of the sum of the values represented by the two hex digit pairs for the byte count, address and data fields. For example:

```
S1137AF0 0A0A0D0000000000000000000000000061
```

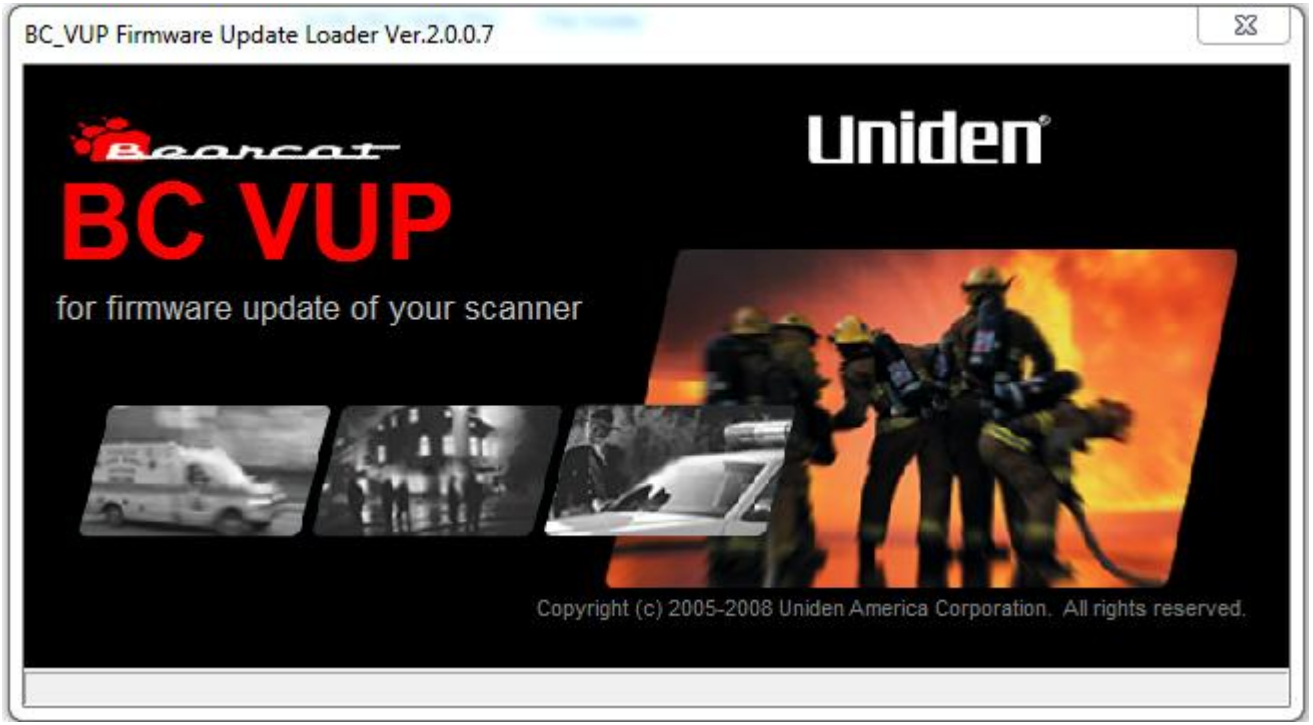
Tool #1: pysrec

- https://github.com/gabtremblay/pysrec
- Motorola s-record analysis tool
- Validate checksum, rebuild checksum
  - In fact, it will replace the checksum automatically if broken
- Show "ascii" representation
- Flip bytes (defeat the rot monster)
- Very bad python

Firmware file analysis

- Let's take a record in our file
- S2240A0120DC22C330DE22C330E321C330E322C3E0EC21C3E0EC22C3E0F121C3E0F122C33011
- checksum(24+0A+01+20+...) != 11
- Something smells fishy
  - Record correctly indexed and addressed (S20A0120)
  - Still, the checksum fails.
- Maybe the firmware update tool can explain some things.

Firmware Updater



SUBATOMIC SECURITY

Firmware Updater

- ~2 MB of pure Visual Basic 6 clusterfsck
- Supports about 10 different scanner protocol
  - in a "Copy-Paste" fashion.
- Not "hard" to reverse
  - Simply unpleasant 〰
- Turns out the firmware file is "scramblencrypted"
- It leaves us with some choices

Firmware Updater

- Plan A: Buy the Renesas hardware to dump the chip content
- (Edit) Plan A1: Buy a Die Datenkrake
- Plan B: Reverse the "scramblencryption" algorithm
- Plan C: ...

Scramblencryption

- Firmware file is partly scrambled and partly weakly encrypted
- Most data blocks uses a position rot(x) scrambling algorithm
- Code blocks uses a rot(x) + XOR cipher
- Some parts are not scrambled at all

- There must be a least depressing way to tackle this problem...

Plan C - As lazy as it gets

- The unscrambling is done at the updater level before the actual firmware update
- The update protocol *should* be much simpler to reverse
- In fact, it was!

Tool #2: BearMock

Tool #2: BearMock

- https://github.com/gabtremblay/Bearmock
- Fakes a BC296D (or a BCi96D)
- Use it with com0com or something similar
- Outputs a descrambled firmware in s-rec format

Next

- We now have a descrambled s-record file
- Epic +- 2 year pause
  - Waiting for IDA to support Renesas m16c
- IDA 6.2: To the IDA cave!

Inside IDA

```
seg001:000A0006                 .BYTE  56h ; V
seg001:000A0007                 .BYTE  65h ; e
seg001:000A0008                 .BYTE  72h ; r
seg001:000A0009                 .BYTE  73h ; s
seg001:000A000A                 .BYTE  69h ; i
seg001:000A000B                 .BYTE  6Fh ; o
seg001:000A000C                 .BYTE  6Eh ; n
seg001:000A000D                 .BYTE  20h
seg001:000A000E                 .BYTE  33h ; 3
seg001:000A000F                 .BYTE  2Eh ; .
seg001:000A0010                 .BYTE  36h ; 6
seg001:000A0011                 .BYTE  30h ; 0
seg001:000A0012                 .BYTE  20h
seg001:000A0013                 .BYTE  20h
seg001:000A0014                 .BYTE    0
seg001:000A0015                 .BYTE    0
seg001:000A0016                 .BYTE  42h ; B
seg001:000A0017                 .BYTE  43h ; C
seg001:000A0018                 .BYTE  32h ; 2
seg001:000A0019                 .BYTE  39h ; 9
seg001:000A001A                 .BYTE  36h ; 6
seg001:000A001B                 .BYTE  44h ; D
```

WUT!?

Inside IDA

- The cpu is supported but it's not common renesas code
- Code analysis is broken :(
- Multiples entry points
  - Triggered by boot or keypress
- There must be an easy way to clean up...

Tool #3: m16clean

- https://github.com/gabtremblay/idabearclean
- (Very) Simple helper IDA python script to help analysis

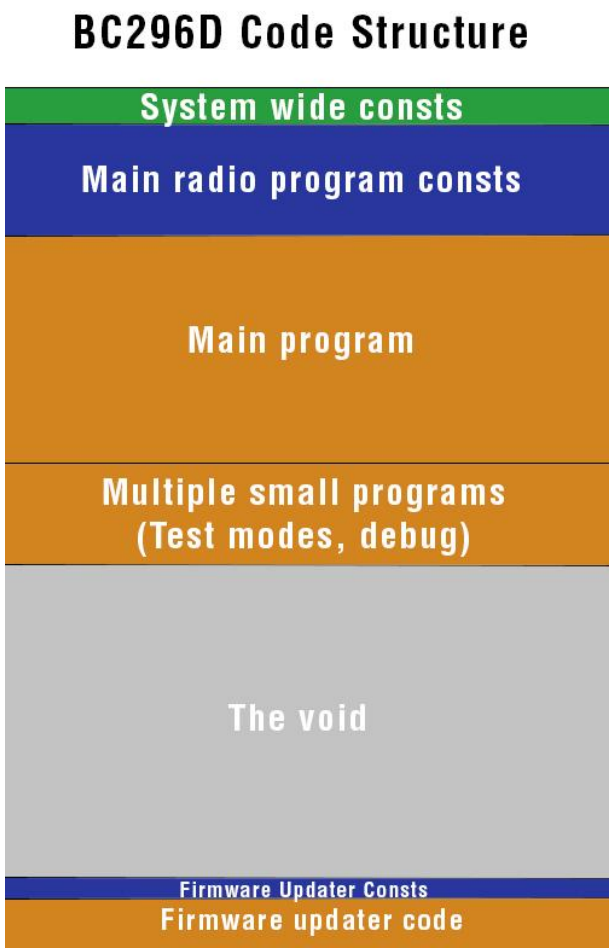
```
seg001:000A0005                 .BYTE  20h
seg001:000A0006 aVersion3_60    .BYTE 'Version 3.60  ',0 ; DATA XREF: sub_AD207+21↓r
seg001:000A0006                                          ; sub_B9480+4F↓r ...
seg001:000A0015                 .BYTE    0
seg001:000A0016 aBc296d         .BYTE 'BC296D        ',0
seg001:000A0027                 .BYTE    0
seg001:000A0028 aUb319z         .BYTE 'UB319Z        ',0
seg001:000A0039                 .BYTE    0
seg001:000A003A                 .BYTE    1
```

```
seg001:000AD260                 ENTER   #13h
seg001:000AD263                 JSR.A   sub_DAAD4
seg001:000AD267                 JSR.A   sub_DAB16
seg001:000AD26B                 PUSH.W  #0Fh
seg001:000AD26F                 PUSH.W  #0FFFFh
seg001:000AD273                 PUSH.W  #0Ah
seg001:000AD277                 PUSH.W  #0
seg001:000AD27B                 JSR.W   sub_AD703
seg001:000AD27E                 ADD.B   #8, SP
seg001:000AD281                 MOV.W   R0, var_13[FB]
seg001:000AD284                 MOV.B   #0, R0H
seg001:000AD285                 MOV.B   R0H, R0L
```

This is a blatant lie! Consts are still not supported, do them manually!

Code finding works well ;)

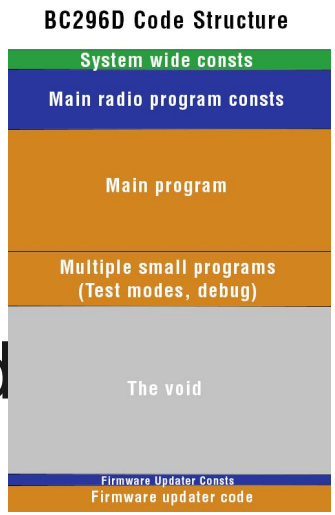
Firmware code layout



SUBATOMIC SECURITY

Firmware code structure

- System wide consts
  - Model number, version, regional tags
- Main radio program consts
  - Screen display, menus errors
- Smaller side programs are accessed at boot time (ex: hold l/o + 6)
- Note the updater aligned to the end of the file so it's hard to corrupt while updating



The code is "signed"

- Some kind of checksum signature at runtime
- However you control the part of the code which tests it.
  - Locate the corrupted firmware error message
  - find the caller
  - flip the jump.
- We can upload anything we want as long as we don't corrupt the updater code at the end

Tool #4-5: Bearflash/BciFlash

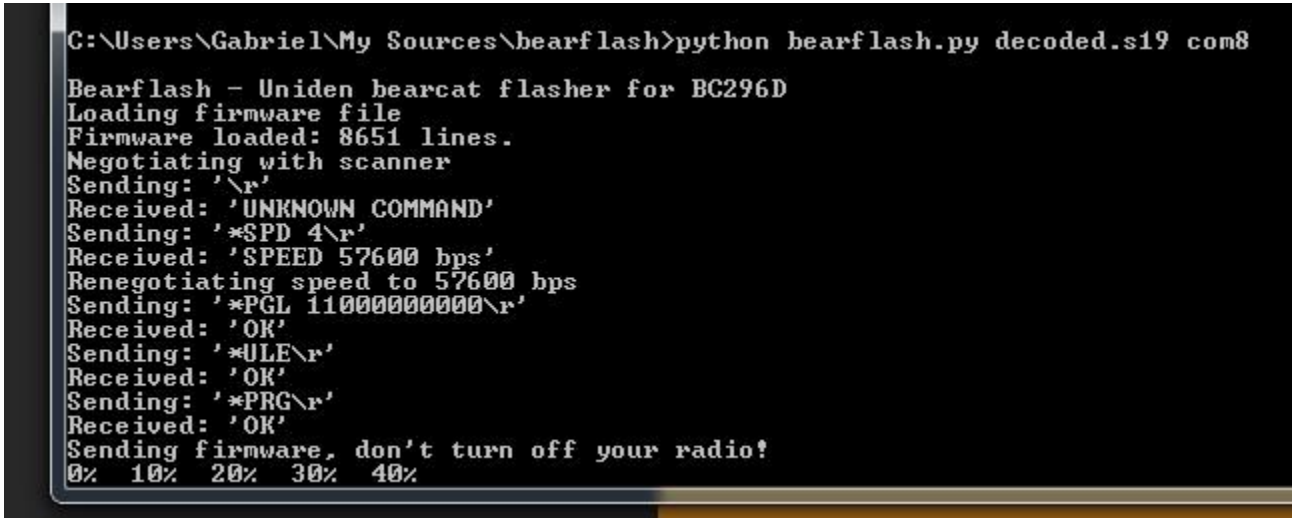
- https://github.com/gabtremblay/bearflash
- https://github.com/gabtremblay/bciflash
- Tools to flash your custom firmware to the radio and the daughter board
- Strongly inspired by the uniden updater (the two tools are almost identical ;))
- Could be merged in a single one.

Some differences

- Some protocol difference
- The daughter board has a fixed 9600 bps update speed
- The main radio updater uses a weird "speed dance"
  - Connects at 9600
  - Sends "*SPD X" where X is a speed (115200)
  - Radio agrees or not
  - The port is closed
  - Updater speed is changed to the selected speed
  - Update can proceed.

Proof of concept

- Just try to flash some modifications to the radio
- I am a kind of a science guy
- Small tribute to the internet famous "eight equals d minus" equation



Eight Equals D Minus Equation

- I am quite funny.

What about the newer models?

- BCD346T, BCD396XT, Home Patrol
- They still all uses s-record update files
- Files are UNSCRAMBLED
- Can't tell for the signature
- Firmware files are not distributed, they are fetched at flash time
- 396XT and Homepatrol have .net updaters
- I strongly suggest you "dotpeek" them
  - They had to put the ftp passwords somewhere ;)
  - Maybe you want to save 100$ on the extreme upgrade...

Questions

?