HOW TO GET MUMPS

THIRTY YEARS LATER



or hacking the government with FOIA'd code

Zachary 'seiranib' Minneker
defcon 30

AGENDA

* A Short History (of course)
* Breaking MUMPS
* The Future and Past

Some Definitions

* EMR: Electronic medical records
* VA: Department of Veterans Affairs
* FOIA: Freedom of Information Act

Who Am I?



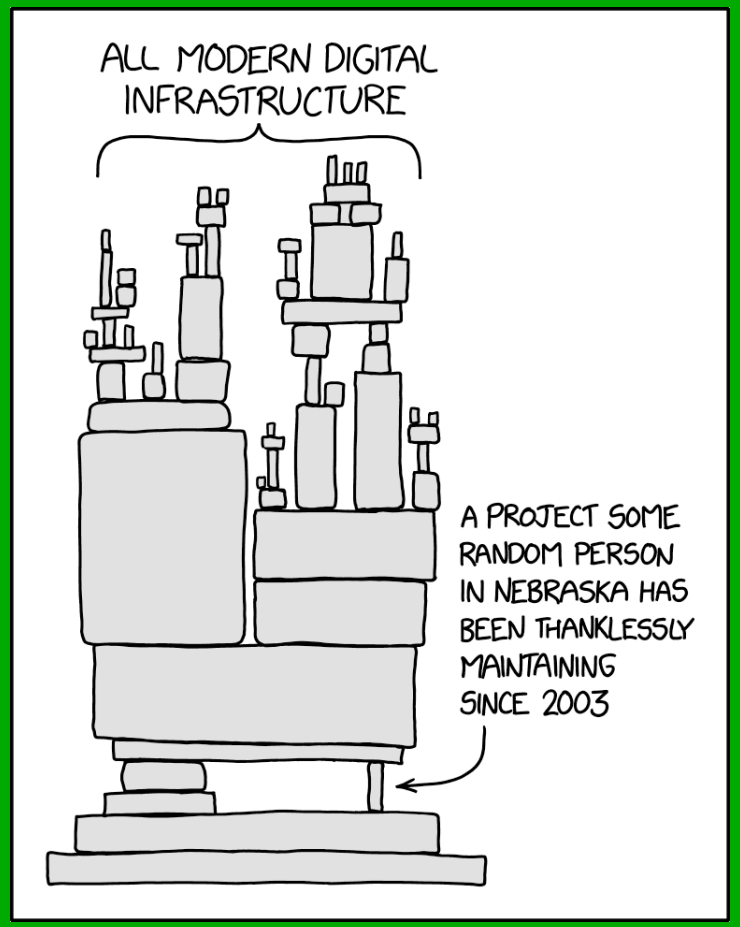
* My name is Zachary (seiranib) Minneker
* I work for Security Innovation
* I /don't/ work for the government
* I used to work in health care kinda, now I break health care stuff because I think it matters a lot
* Also I care a lot about history

Obligatory

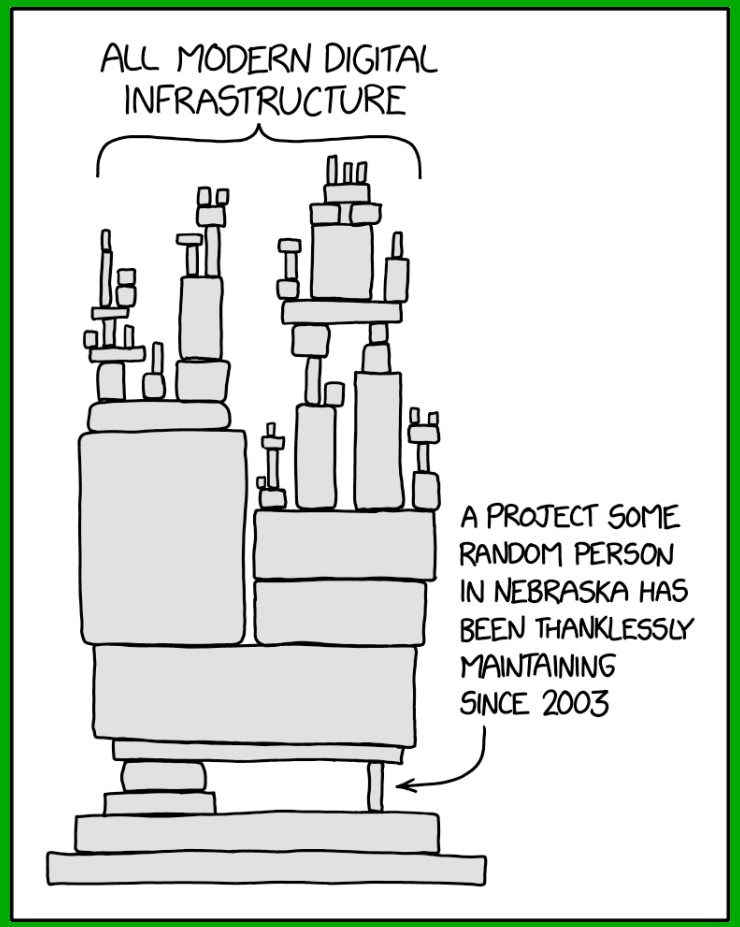
* I am speaking on behalf of myself and not my employer
* These slides were created in Moebius (except some images)
  * The slides are available as a text named <talk title>.nfo!

Anyway!

WHAT'S THIS TALK ABOUT?



WHAT'S THIS TALK ABOUT?



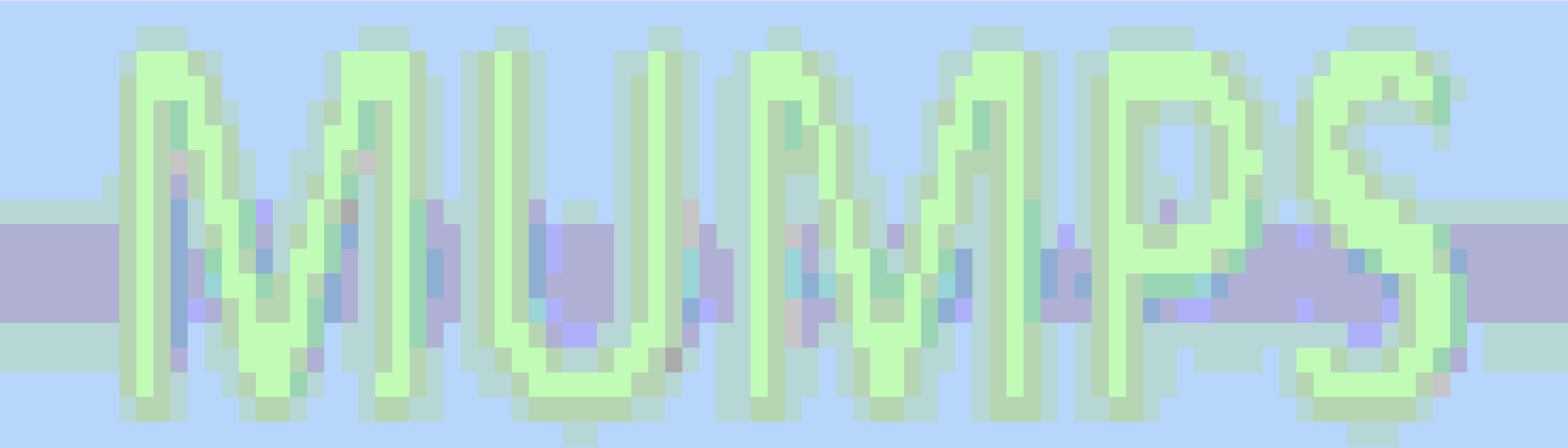
This talk is about hitting that little block with a hammer.


HISTORY!

Origin

* In 1966, a group of engineers under Octo Barnett, including Neil Pappalardo, began working on a new language

  Massachusetts General Hospital
  Utility Multi-Programming System


MAUMAPS

The Hardhats

* MUMPS was SPECIFICALLY created for healthcare environments

* It was originally written for PDP-9s, and then DEC rewrote it to make it into a stand-alone OS (!) called MUMPS-11 for PDP-11s
  * Some say the original was a PDP-7
  * Originally installed at a site in May 1971, in Denver

* Lots of influence from BBN TELECOMP

* (this information is being gathered from an email exchange on comp.lang.mumps from 1996, btw)

The Olde Days

* This was pre-Unix! There were no standards! The Unix epoch didn't exist yet!

* This isn't a language: it's a programming environment!

  * The current time is stored in $HOROLOG as two numbers separated by a comma, the first of which is the number of days since January 1st 1841

  * The maximum date is December 31, 9999.
    * YZK Compliant!

Making MUMPS Work

* MUMPS wasn't well standardized, but 2 engineers (Ted O'Neill and Marty Johnson) were hired to explore using it at the VA

* They started developing utilities to help computerize hospitals in the VA system. Eventually this coalesced into a single, total EMR: VistA
  * You get it? Capital V, capital A?

* The group of developers working on this, completely decentralized, were called The Hardhats




VistA related Links


Mailing List





Hardhats.Org

Welcome to a web site dedicated to fostering a virtual community for the worldwide users of the VISTA software!

VistA is the comprehensive suite of patient-care software developed by the Department of Veterans Affairs over the past 40 years, and now used by hundreds of enterprises worldwide, in and outside of the VA. At VistA's very core, represented by the shops on the left, you have all the tools and materials you need to build an economical, efficient, scaleable and integrated system. Add the VistA applications and you have a complete healthcare system.

Are you interested, but new to VistA? Read about Hardhats, VistA and DHCP. Do you already know a little about VistA and want to learn more? Read on.

News and Recent Events **(reverse chronological order)**

October 22-24, 2021: 39th VistA Community Meeting (Virtual due to the COVID-19 Pandemic).

May 20, 2021: Instructions on using systemd instead of initd with VistA on GT.M/YottaDB added by Kevin Toppenberg.

March 6, 2021: New version of MSC FileMan released.

February 14 2020: OSEHRA shuts down. After 9 years of good work, OSEHRA shuts down. OSEHRA

Vista and MUMPS Through The Ages

* VistA grew through the years, and became well loved and respected
  * On the one hand, this is an EMR made by people that are using it every day. On the other hand, its enterprise grade shadow IT

* MUMPS was an extremely fast, no-SQL (even before SQL hit the market) database, and found uses in things that needed a db for storing a lot of extremely active data: banks, science, hospitals

The Current Day

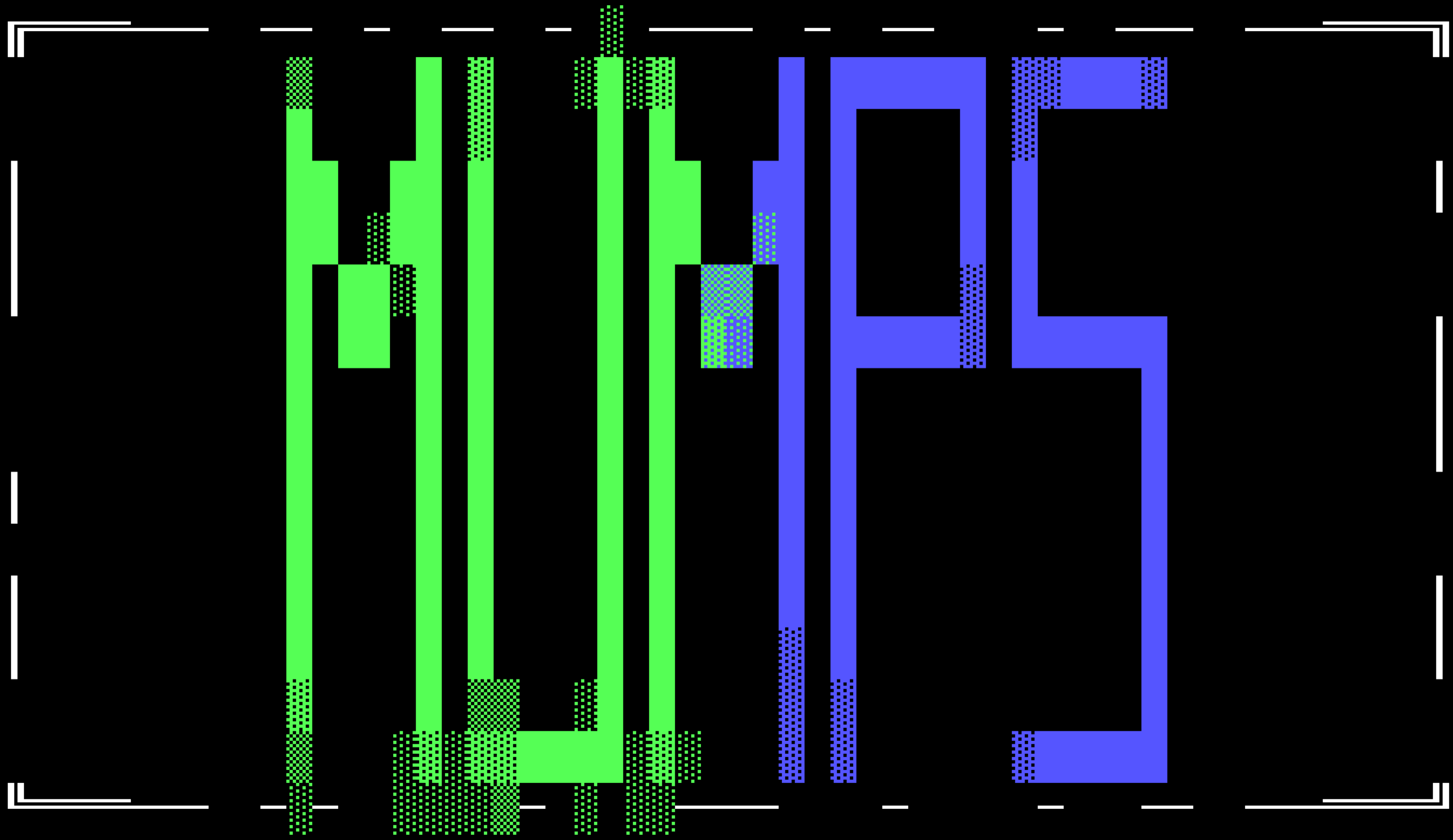
* VistA is used to this day by the VA in hospitals across the US, and hospitals all over the world use it. Doctors love it.
  * Efforts to "modernize" the VA have led the VA to start to move away from VistA, but its still deployed all over

* MUMPS is still widely used in other health products, from some of the biggest EMRs to core banking
  * "Allscripts, Epic, Coventry Healthcare, EMIS, Partners HealthCare (including Massachusetts General Hospital), MEDITECH, GE Healthcare ..., Sunquest Information Systems, DASA, Quest Diagnostics, and Dynacare, among others.*[Okane, 2017, Introduction to the Mumps Language]

Getting MUMPS, Getting VistA

* MUMPS can installed with `sudo apt install -y fis-gtm` to install FIS' GT.M, which is a MUMPS implementation

* VistA was at some point victim of a vicious Freedom of Information Act request, and so the VA literally uploads it to an FTP server every month
  * This appears to have started around September 2004



MUMPS is Weird!

* In MUMPS, 3 + 6 * 2 is 18! This is because it reads math strictly from left to right. It only gets weirder from here!

* In general, its a pretty readable language. But its from a time where size was at a premium
  * Most code is not commented!
  * Keywords in the language can be shortened to single letters in many situations

Normal MUMPS Code

```
Set a=1
If a=1 Do
. Set a=0
. Write a
. If a=0 Do
. . Set a=1
. . Write a
. Else  Do ; There's 2 spaces there!
. . Set a=2
Write a
```

Small MUMPS Code

```
S a=1
I a=1 D
. S a=0
. W a
. I a=0 D
. . S a=1
. . W a
. E  D
. . S a=2
W a
```


MORE

SMALLER MUMPS CODE

```
s a=1 i a=1 d  s a=0 w a i a=0 d  s a=1 w a e  d  s a=2 w a
```

THIS IS BASICALLY CODE GOLF

Oooooooooooof!

```
BLDPTYPE(DIEN,PROMPTS) ;
 N DSUB,DCIEN,PTIEN
 S DSUB=0 F  S DSUB=$O(^PXRMD(801.41,DIEN,10,DSUB)) Q:DSUB'>0  D
 .S DCIEN=$P($G(^PXRMD(801.41,DIEN,10,DSUB,0)),U,2) Q:'DCIEN
 .I $$ISDISAB^PXRMDLL(DCIEN)=1 Q
 .I "PF"'[$P($G(^PXRMD(801.41,DCIEN,0)),U,4) Q
 .S PTIEN=$P($G(^PXRMD(801.41,DCIEN,46)),U) I PTIEN'>0 Q
 .S PROMPTS(PTIEN)=""
 Q
```

Oooooooooooof!

```
PRAC(SCLOG,SCQREC,SCTYPE) ; -- validate practitioner selections
 N SCSEL,Y,SC0,VAUTT
 S SCSEL=""
 IF SCQREC("REPORTID")=3 D
 . S VAUTT=1
 ELSE  D
 . D BUILD^SCRPBK3(.SCQREC,"TEAM",.VAUTT)
 F  S SCSEL=$O(SCQREC("SELECTIONS",SCTYPE,SCSEL)) Q:SCSEL=""  D
 . S Y=+SCSEL,SC0=$G(^VA(200,Y,0))
 . IF $D(VAUTT),$$PRACS^SCRPU1() D
 . . Q
 . ELSE  D
 . . D SETSEL(SCLOG,SCTYPE,"TEAM",SC0)
 Q
 ;
```

Using MUMPS

* Modern MUMPS implementations, namely GT.M and YottaDB are both interpreted and effectively ahead of time compiled
  * You write code, and its stored as .m files. You then ask the MUMPS implementation to link and run the source and it checks if it can find a compiled version of the code. If not, it compiles the code into a shared object, and loads it into memory

* Applications written in MUMPS, like VistA, are often released as source. Users then compile and load sections from disk as needed

How Does VistA Fit In Here

* VistA is FTP'd up to a VA server as the "FOIA" version of VistA

* A couple different groups gather that and package it together

* On a deployed FOIA VistA server using GT.M, for example, VistA's MUMPS source is stored as routines (usually in a directory simply called r/)

Then I show



How Did I Get Wrapped Up In This?

* At SI, we have research time to work on interesting code and learn how to break new, interesting things

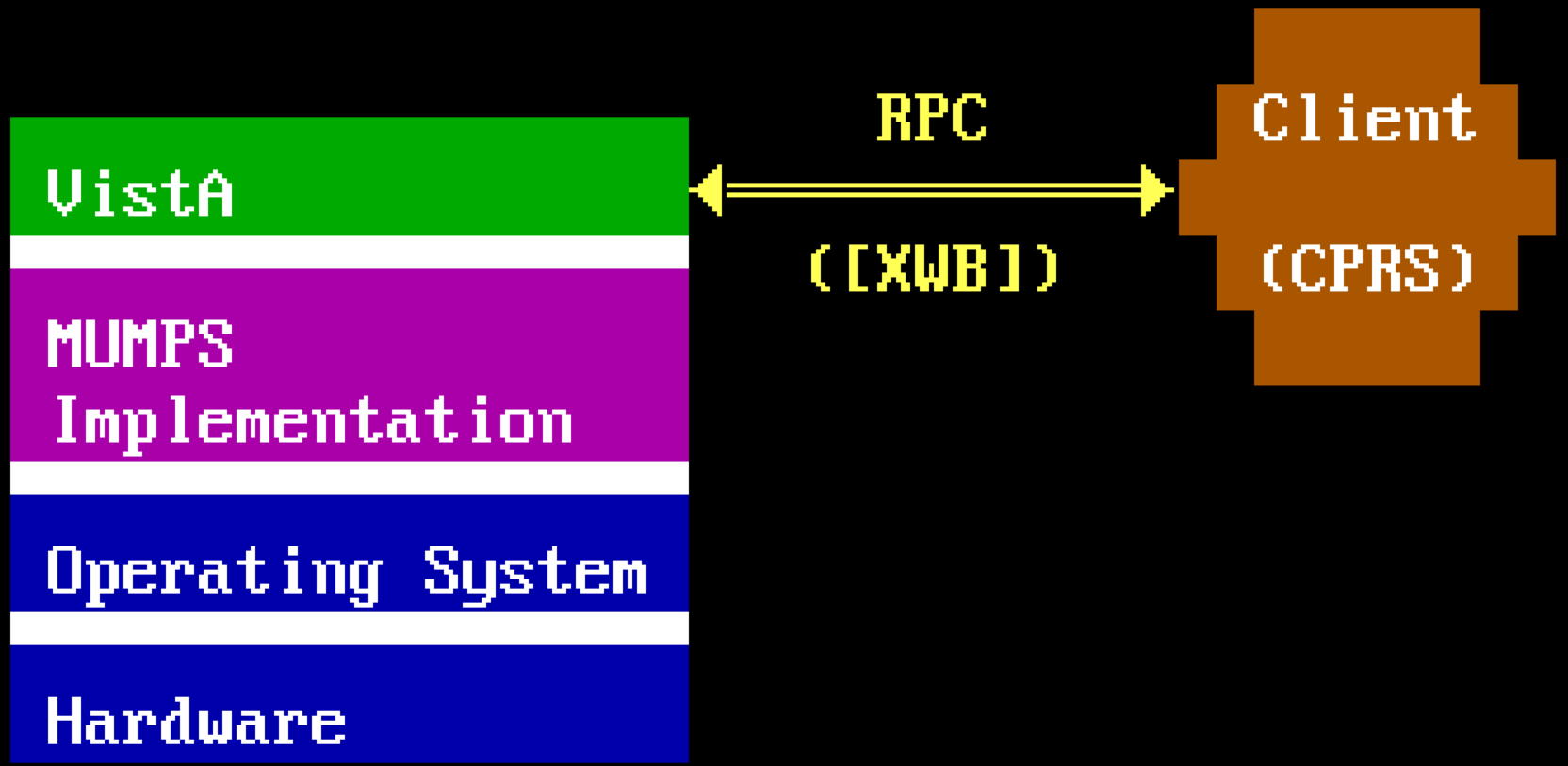
* I've been using mine to break different pieces of health care software like HL7 protocols or other open source EMRs

* I'd heard of VistA, but (to be frank) didn't realize it was MUMPS and how foundational MUMPS was to a lot of health care software
  * Places I've worked at have always used Java-based EMRs

The Everlasting Pursuit of Cool

I wanna be cool.
I wanna hack weird code.
(I think I've proven that VistA is weird code.)

The Stack


VistA
MUMPS
Implementation
Operating System
Hardware
RPC
([XWB])
Client
(CPRS)

Finding a Client

* My first thought is that I want to attack this as if I'm a normal client on the network, so I need to look at a normal client that interacts with VistA: CPRS
  * CPRS is widely available, written in Delphi

* So I install CPRS, run a version of VistA deployed without TLS (this is not hard), and start capturing packets

Packets

```
0000  02 00 00 00 45 00 00 61  ea f9 40 00 80 06 00 00   ····E··a ··@·····
0010  7f 00 00 01 7f 00 00 01  e3 a2 24 d6 6f 3c 62 70   ········ ··$·o<bp
0020  72 2a a8 4d 50 18 27 f9  01 cc 00 00 5b 58 57 42   r*·MP·'· ····[XWB
0030  5d 31 30 33 30 34 0a 54  43 50 43 6f 6e 6e 65 63   ]10304·T CPConnec
0040  74 35 30 30 30 39 31 32  37 2e 30 2e 30 2e 31 66   t5000912 7.0.0.1f
0050  30 30 30 31 30 66 30 30  30 39 31 32 37 2e 30 2e   00010f00 09127.0.
0060  30 2e 31 66 04                                     0.1f·
```


I don't know
MUMPS (yet)!

```
MAIN ; -- main message processing loop. debug at MAIN+1
 F  D  Q:XWBTBUF="#BYE#"
 . ;Setup
 . S XWBAPVER=0,XWBTBUF="",XWBTCMD="",XWBRBUF=""
 . K XWBR,XWBARY,XWBPRT
 . ; -- read client request
 . S XR=$$BREAD^XWBRW(1,XWBTIME,1)
 . I '$L(XR) D LOG("Timeout: "_XWBTIME) S XWBTBUF="#BYE#" Q
 . S XR=XR_$$BREAD^XWBRW(4)
 . I XR="#BYE#" D  Q  ;Check for exit
 . . D QSND^XWBRW("#BYE#"),LOG("BYE CMD") S XWBTBUF="#BYE#"
 . . Q
 . S TYPE=(XR="[XWB]")  ;check HDR
 . I 'TYPE D LOG("Bad Header: "_XR) Q
 . D CALLP^XWBPRS(.XWBR,$G(XWBDEBUG)) ;Read the NEW Msg parameters and call RPC
 . IF XWBTCMD="#BYE#" D  Q
 . . D QSND^XWBRW("#BYE#"),LOG("BYE CMD") S XWBTBUF=XWBTCMD
 . . Q
 . U XWBTDEV
 . S XWBPTYPE=$S('$D(XWBPTYPE):1,XWBPTYPE<1:1,XWBPTYPE>6:1,1:XWBPTYPE)
 . ;I $G(XWBPRT) D RETURN^XWBPRS2 Q  ;New msg return
 . I '$G(XWBPRT) D SND^XWBRW ;Return data,flush buffer
 Q  ;End Of Main
```

```
MAIN ; -- main message processing loop. debug at MAIN+1
 F  D  Q:XWBTBUF="#BYE#"
 . ;Setup
 . S XWBAPVER=0,XWBTBUF="",XWBTCMD="",XWBRBUF=""
 . K XWBR,XWBARY,XWBPRT
 . ; -- read client request
 . S XR=$$BREAD^XWBRW(1,XWBTIME,1)
 . I '$L(XR) D LOG("Timeout: "_XWBTIME) S XWBTBUF="#BYE#" Q
 . S XR=XR_$$BREAD^XWBRW(4)
 . I XR="#BYE#" D  Q  ;Check for exit
 . . D QSND^XWBRW("#BYE#"),LOG("BYE CMD") S XWBTBUF="#BYE#"
 . . Q
 . S TYPE=(XR="[XWB]")  ;check HDR
 . I 'TYPE D LOG("Bad Header: "_XR) Q
 . D CALLP^XWBPRS(.XWBR,$G(XWBDEBUG)) ;Read the NEW Msg parameters and call RPC
 . IF XWBTCMD="#BYE#" D  Q
 . . D QSND^XWBRW("#BYE#"),LOG("BYE CMD") S XWBTBUF=XWBTCMD
 . . Q
 . U XWBTDEV
 . S XWBPTYPE=$S('$D(XWBPTYPE):1,XWBPTYPE<1:1,XWBPTYPE>6:1,1:XWBPTYPE)
 . ;I $G(XWBPRT) D RETURN^XWBPRS2 Q  ;New msg return
 . I '$G(XWBPRT) D SND^XWBRW ;Return data,flush buffer
 Q  ;End Of Main
```

```
0    ····E··a ··@·····
0    ········ ··$·o<bp
2    r*·MP·'· ····[XWB
3    ]10304·T CPConnec
6    t5000912 7.0.0.1f
e    00010f00 09127.0.
     0.1f·
```

BooFuzz Time

* So I turn to ole reliable: BooFuzz

* To make a BooFuzz script, I need to capture a lot of traffic, and then convert that traffic into the script itself

* I did about 20 RPC calls before I was like "no, actually" and wrote a script to do the rest for me

* So, now I have a 18 KLOC BooFuzz script!

  ... that gets me nothing!

Switch Tack

* So, switch tack: I write a harness that takes input from stdin and feeds it to [XWB] RPC parsing code

* For this, I want to use AFL++ in dumb mode to feed input to VistA

* This doesn't appear to be working, so I'm like "well, I should get some visibility into the MUMPS runtime, so I'm going to instrument the underlying MUMPS implementation which handles the sockets/memory/etc"

Intrumenting MUMPS Implementations

* There's a couple MUMPS implementations that seemed worth attacking here: GT.M and YottaDB
  * YottaDB is based on GT.M because of historical reasons
  * The GT.M deployment I was working with was based on GT.M

* Both YottaDB and GT.M are made by C wizards who are way cooler than me
  * For instrumentation this means we have to make 3-ish patches to make AFL++ work

Attacking MUMPS Implementations

* Since I'm instrumenting the MUMPS implementation in order to get visibility on the RPC, I might as well fuzz the MUMPS source compilation logic anyway

* YottaDB has a lot of existing source code examples that they use for test-driven development, which make a perfect, no thinking required fuzzing corpus

Source Review

* At this point, I finally read MUMPS!

* I start tracing authentication, input handling, and system interactions for quick wins

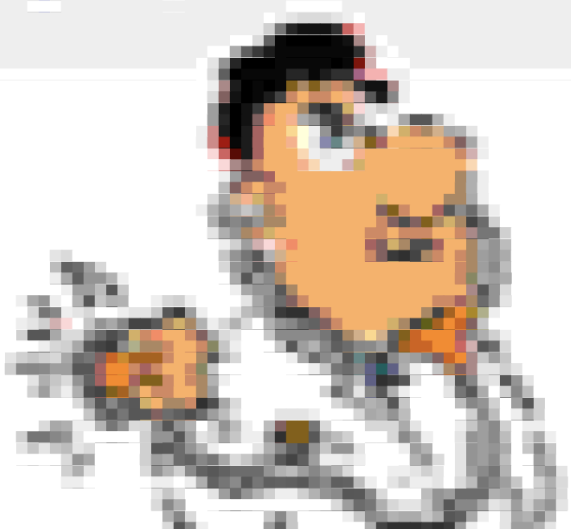
Recap

* First, we're fuzzing VistA's RPC using a MUMPS harness and AFL++

* Second, we're fuzzing MUMPS' source code handling in two implementations, GT.M and YottaDB, using YottaDB's TDD tests

* Third, we're reviewing VistA's source by hand

So... What'd We Find?



Fuzzing Findings

* RPC fuzzing got us nothing :(
  * (There's a boring technical reason for this I don't have time for)

* Fuzzing YottaDB and GT.M got us 30 CVEs, all from memory corruption bugs
  * Everything from buffer overflows to use after frees to null pointer dereferences to function pointer overwrites

* These bugs are in how the MUMPS compiler and runtime handle source code, so the attack surface is sorta specific

The CVEs

* The CVEs are CVE-2021-44481 to CVE-2021-44510

* These memory bugs are weird! This is a piece of software that has been developed by greybeards for 30 years at this point
  * They do every possible trick you can think of, and that leads to weird states!

Let's Look at CVE-2021-44486

* First, I'm going to show you the crashing input and the crash itself, then I'm going to explain why this happens, and then I'll show you the whole crash again



seiranib@ubuntu:~/YDB/build$

Why Is This Happening

* A specific order of strings being created and compile commands being performed is corrupting a data structure which contains a function pointer

* This function pointer is then used in a subsequent call to write. Because the function pointer is corrupted, the call jumps out to the middle of un-executable code
  * the location we're jumping to here is somewhere on the heap just by happenstance

What's Being Corrupted

* This is what we're trying to execute (and how it should work normally)

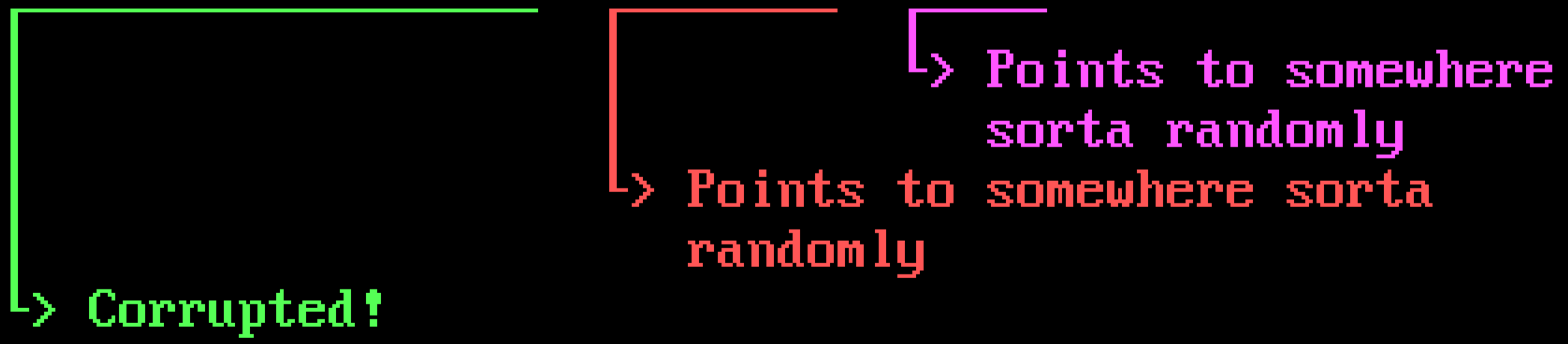
```
(io_curr_device.out->disp_ptr->write)(&v->str);
```



What's Being Corrupted

* And after the corruption occurs, this is what this line of code actually does

```
(io_curr_device.out->disp_ptr->write)(&v->str);
```



How Does That Corruption Happen

* Basically, we have two overlapped memory addresses. In other contexts, a similar attack is possible via a use after free or double free, where two pointers believe they own the same memory

* Lets take a look at the crash again. I'm going to run YottaDB with the same input again, but this time we're going to stop right before the crash in op_write.c.

The Video

```
Continuing.

YDB>
        '
        ^-----
                At column 0, line 1, source module longcon.m
%YDB-E-LSEXPECTED, A line separator is expected here
        Compiling with -dynamic_literals
                  ^-----
                At column 11, line 2, source module longcon.m
%YDB-E-INVCMD, Invalid command keyword encountered

Breakpoint 1, incr_link (file_desc=0x7ffff7ce9500 <object_file_des>,
    zro_entry=0x55555557f340, recent_zhist=0x0, fname_len=34,
    fname=0x7fffffff9140 "/home/seiranib/YDB/build/longcon.o")
    at /home/seiranib/YDB/sr_unix/incr_link.c:300
300             hdr = (rhdtyp *)malloc(SIZEOF(rhdtyp));
(gdb) s
gtm_malloc (size=312) at /home/seiranib/YDB/sr_port/gtm_malloc_src.h:654
654     {
(gdb) en 1
(gdb) c
Continuing.
%YDB-E-LABELMISSING, Label referenced but not defined: etr

Program received signal SIGSEGV, Segmentation fault.
0x00005555555c6950 in ?? ()
(gdb)
```

What Did We See

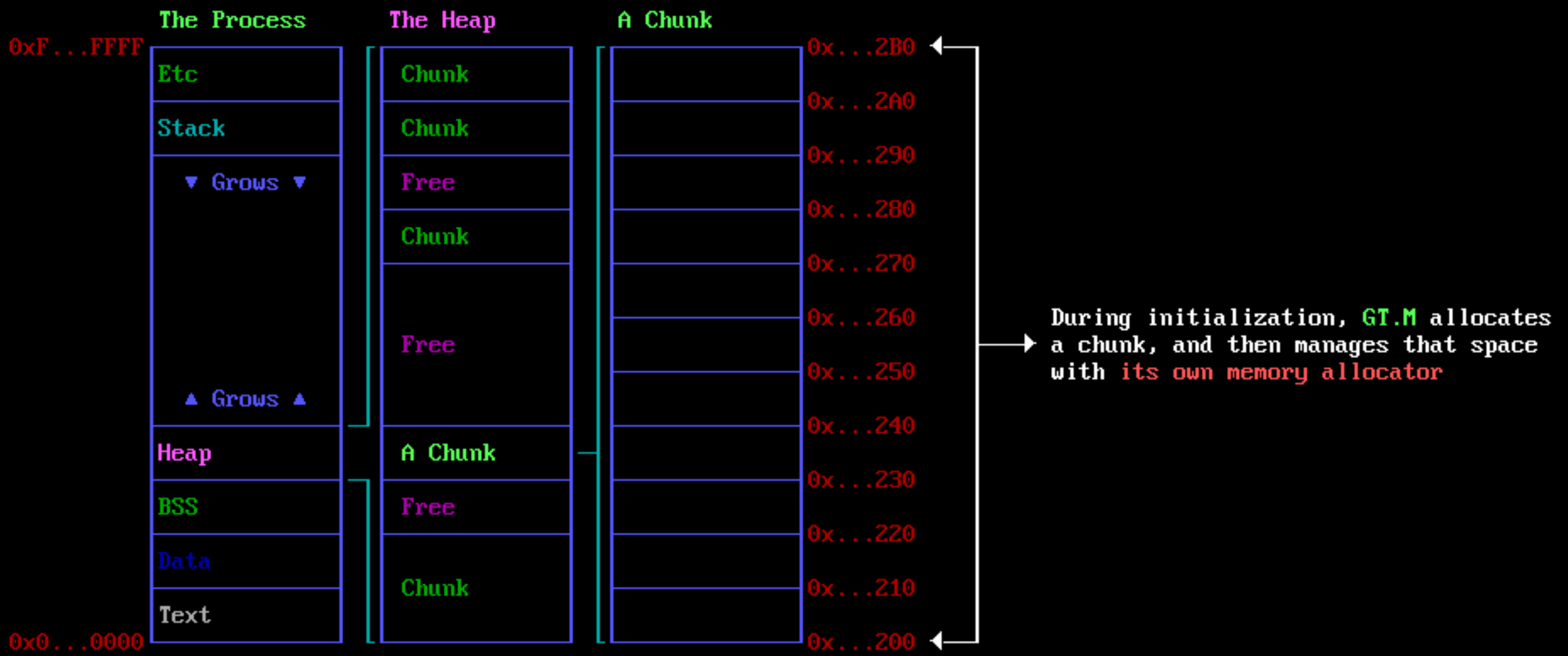
* Before the crashing call, the io_curr_device.out object is well formed, and at a memory address ending with 0x...e210

* A call to malloc does not go to GLibC malloc. Instead, it goes to a malloc created by the developers

* That malloc returns a pointer to memory starting at 0x...e200

* The devs have created their own memory allocator inside of the heap, and with careful manipulation it can be made to return addresses that overlap with critical structures
  * There's at least 2 memory allocators in use

Heap Diagram



Heap Diagram



But... why?

* Basically we have two mechanisms managing the same memory, malloc and gtm_malloc. Then, we have a type confusion-like bug in due to way gtm_malloc handles that memory
  * This is a heap exploitation bug inside a memory manager managing memory that's also managed by the system memory manager

* The address that gets placed isn't completely random, but its not super in our control

* I wanted to talk about this bug particularly because I think its really wild

What about the source review?



Source Review Wins

* Source code review focused on quick wins and looked at the auth mechanisms, input handling, and system interactions

* RPC access is gated by a mechanism that uses roll-your-own encryption from the 90s
  * If deployed without TLS, creds are poorly encrypted and transmitted in such a way that attackers can trivially decrypt them, or simply replay the packets

* There appears to be hardcoded creds in the source, but because of certain particulars, I'm not sure if they're usable

Im srry lmao

* I tried super hard to get this disclosed in time, but we had some issues there. And its not super sporting to kiss and tell, so...



Disclosure Timeline

* Jan 3rd 2022: first contact with VA following their VDP. Received an automated email. No further contact.
* Jan 10th 2022: second contact with VA following their VDP. Automated email, no further contact.
* Feb 10th 2022: third contact with VA following their VDP. No response.
* Mar 22nd 2022: reached out to DHS contact. No response.
* Apr 6th 2022: reached out to CISA. No response.
* May 23rd 2022: called CISA literally on the phone, got disconnected, no response.
* May 27th 2022: called CISA again. Was told that any information provided would not be provided to the VA to fix the issue.
* May 27th 2022: reached out to CMU CERT. Received follow up email with questions. We responded, then they never responded.

Disclosure Timeline

* YottaDB

* Nov 30th 2021: First contact with vendor
* Dec 1st 2021: Vendor confirms reports, begin working on fix immediately
* Dec 7th 2021: Spoke to the dev team explaining methodology
* Dec 8th 2021: Merge request for fuzzing changes, vendor takes over fuzzing
* Feb 25th 2022: New version released!

* GT.M

* Dec 1st 2021: First contact with vendor
* Dec 3rd 2021: Vendor confirms reports, begin working on fix immediately
* Mar 30th 2022: New version released!

So What Does This All Mean?

* When I started this I sorta figured that memory corruption bugs in big products were dead as hell
  * NOPE!

* MUMPS isn't going anywhere
  * When MUMPS first got started, it was faster than everything, it was inventive, it was cutting edge; a ton of industries jumped on it, from banks to health care, and are still there now
  * 54% of health care records touch MUMPS in some form

* There's still more weird machines out there to break!

What Should I Do?

* If you're working on a VistA or derived product, verify you're using TLS everywhere!

* If you deploy a MUMPS-based product and use YottaDB or GT.M, update!
  * If you use GT.M from apt, you're four versions behind and you should probably build from source

* If you're a hacker looking for research, I can't think of a better broad umbrella to be looking under than medicine

Everyone Who Worked On This Is A Hero To Me

* I'm not kidding

* MUMPS was this incredible idea from 1960something that is an ANSI standard, has been continually developed into something incredible across all the implementations, ideas, and years

* VistA is this well respected, easy to use, flexible, and by-doctors-for-doctors, that was then released into the world to help provide those same tools to as many people as possible
  * btw it was named vista in 1994, almost 30 years ago, cause it was originally called DHCP which led to some problems



For the inspiration and mentioning Moebius

KS Bhaskar and the crew at YottaDB

shoutz to the DC30 crew for being super flexible & helpful!



whoisjoe.com

for listening to me complain about mumps

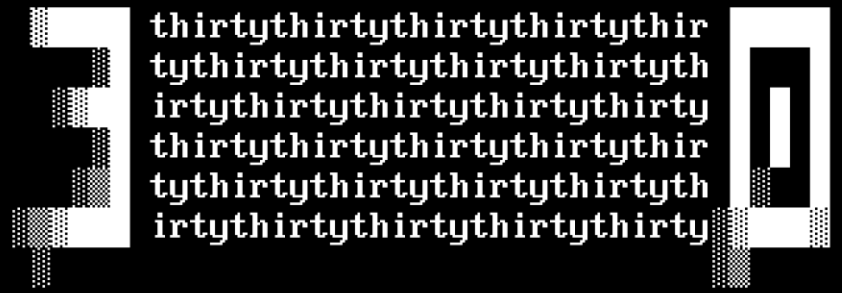
https://h3rmitthefr0g.dev/, LeJIT, @Zonr0, @Din3zh, and everyone else I'm forgetting at Security Innovation, The application security company

kdolan503

Queercon and queer hackers every where

seiranib presents... mumps at







hi mom hi dad
charlie buy louder drums

Max Dulin for teaching me how to pwn the heap

David Zeichick, Tyson Henry and the rest of the staff at csu chico

TWIABP&IANLATD for illusory walls

tysm david! ♥

Hey Martin!

Slides created in 2022 on an x86 macbook somewhere in the PNW in moebius by Andy Herbert

Thanks so much for your time!

questions, etc?

also, you can yell at me on
twitter @seiranib
(its "binaries", backwards)