==Phrack Inc.==

Volume 0x10, Issue 0x46, Phile #0x07 of 0x0f

|=-----------------------------------------------------------------------=|
|=--------------------=[ Viewer Discretion Advised: ]=--------------------=|
|=--------------=[ (De)coding an iOS Kernel Vulnerability ]=--------------=|
|=-----------------------------------------------------------------------=|
|=--------------------------=[ Adam Donenfeld ]=--------------------------=|
|=-----------------------------=[ @doadam ]=-----------------------------=|
|=-----------------------------------------------------------------------=|

--[ Table of contents



--[ 0 - Introduction

The goal of this article is to demonstrate a (relatively) hard-to-reach attack surface on iOS, and showing the entire process from the beginning of the research till the point where a vulnerability is being found. While exploitation is out of the scope in this article, understanding the process of defining the attack surface, researching and while making your life easier (see sections 4 and 9), can provide beginners and expert hackers alike, a different approach for sandbox-accessible vulnerability research.

The bug in question is CVE-2018-4109 [1], which was found by yours truly, that is Adam Donenfeld (@doadam). A PoC of the vulnerability is also available with this paper, and you're free to use it for educational purposes only.

While an exploit can (IMO) be written for this vulnerability, I had too many things to do (writing this paper for instance) but if you feel like working on an exploit, feel free to write me if you want my help with it. Without further ado - let's start.

--[ 1 - Sandbox concepts

On all modern operating systems, most of the processes are by default restricted by sandbox technologies. A sandbox is an extra layer of protection, which prevents certain processes from accessing certain mechanisms. A sandbox is mandatory for many reasons, for instance:

* Preventing leakage of sensitive information. For example, let's take the case where an attacker has breached your phone using a WebKit exploit. While WebKit can steal information related to your browsing, it will not be able to read your contacts, because the sandbox checks who tries to access your contacts, and denies permission unless there's a legitimate reason (If the attacker has gained code execution using a vulnerability in the Contacts app, he will probably be able to access your Contacts).

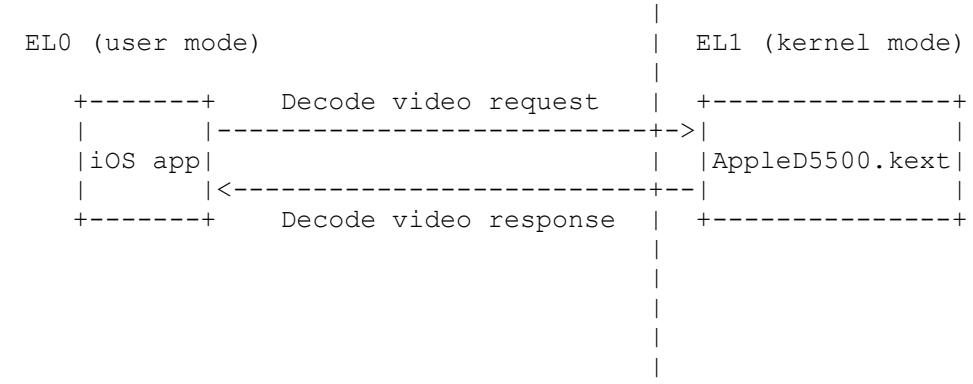
* Narrowing attack surface. Most of the vulnerabilities out there

can be found in different, unrelated components of the system (as will be shown soon). In the world of iOS, an interesting example is CVE-2015-7006 [2]. CVE-2015-7006 is a directory traversal which could be triggered via an Airdrop connection on iOS. The daemon in question was "sharingd". The directory traversal ultimately gave the attacker a write file primitive, meaning an attacker could overwrite any file on the system. Because sharingd was running unsandboxed as root back then, this vulnerability alone was enough to do various powerful operations (an installation of an arbitrary app, for example). Apple has since sandboxed sharingd. If sharingd would have been sandboxed before the publication of CVE-2015-7006, the vulnerability alone wouldn't be so powerful, because the primitives and privileges which could be gained are substantially limited (after being sandboxed, sharingd couldn't write any file on the system, thus couldn't manipulate installd to install arbitrary applications).

While fixing vulnerabilities like the one in sharingd does solve the specific issue in CVE-2015-7006, that alone doesn't approach the main issue: any exploit in sharingd results in compromise of the entire device. As a result, a lot of vendors (Apple among them) designed their system so that almost everything is sandboxed, and nowadays, almost every operation that requires hardware interaction is sandboxed and is only given upon permission from the user\Apple.

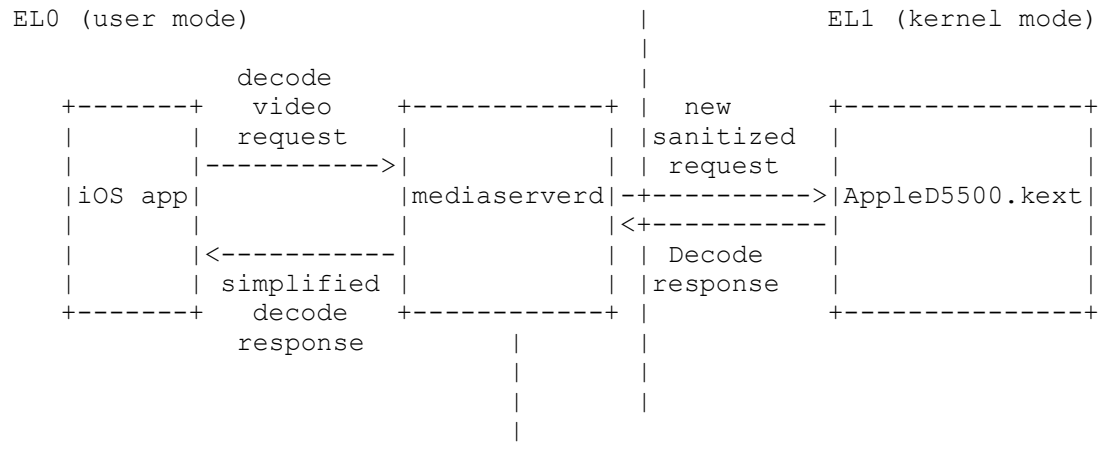
Because the vulnerability discussed in this paper (CVE-2018-4109) is in the accelerated hardware decoding driver, let's see the approach Apple took in sandboxing video operations, namely, video encoding\decoding:

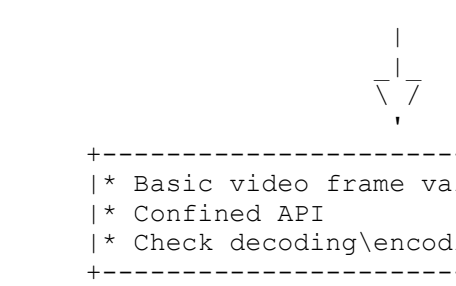
The following graph demonstrates how the app would interact with the video-decoding driver if there would be no sandbox:




Fortunately for Apple, communication with every hardware accelerated encoding\decoding driver is sandboxed, meaning each request goes through a "broker" (mediaserverd). This can be extremely time-consuming for an attacker, because it means the communication with AppleD5500.kext is defined by mediaserverd and that an unprivileged attacker can't access "exotic features" without having prior access to the driver (or code execution in a privileged context like mediaserverd).

This is how unprivileged apps communicate with AppleD5500.kext:






As we can see from the diagram, not only mediaserverd sanitizes our request, it never forwards requests: in fact, it recreates the request\response accordingly, which limits the attacker's power, both in causing memory corruptions and performing infoleaks.

--[ 2 - A bug - how it all got started (IOSurface)

The bug was hidden deeply within the AppleD5500.kext file and while I had no intention to reverse engineer AppleD5500.kext, I found myself doing so in pursuit of a candidate for a different bug I found (which Apple silently fixed without issuing a CVE for).

While the other bug is not in the scope of this article, in order to understand how CVE-2018-4109 was originally found, it is important to have some background on the other bug.

That other bug was in a driver called IOSurface.kext. IOSurface are objects which primarily store framebuffers and pixels and information about these. IOSurface allows transferring a lot of information between processes about framebuffers without causing a lot of overhead. Each IOSurface object has an ID, which can be used as a request to an IOSurfaceRootUserClient object. IOSurfaces map the information between different processes, and thus save the overhead of sending a lot of information between processes. In iOS, a lot of drivers use IOSurface when it comes to graphics. The user doesn't store anything on the IOSurface object except for its ID. This means that in order to use an IOSurface object (for example, for video decoding), the user just needs to supply the ID to the appropriate driver and the original video is extracted from the IOSurface. The video itself is never being sent to the driver as a part of the request.

IOSurface objects store a lot of properties about the graphics; one of them is called "plane". For brevity, there was a sign mismatch in the "offset" of the plane. It means that each driver which used the plane's offset (or base), would have had a negative int, while the kernel "IOSurface->getPlaneSize()" function regarded the plane's offset as a uint32_t. So this vulnerability resulted in a buffer overflow.

Because surface objects only store that information without really "using" it (e.g performing memory manipulations based on the plane offset), it was necessary to find a different driver that used the plane's offset to actually perform a buffer overflow (or anything else which would give us more primitives).

--[ 3 - A bug - finding a primitive for the IOSurface bug

Fortunately, if a driver wants to use IOSurface objects, it has to find the "IOSurfaceRoot" service, which is public and is named in the kernel's registryas "IOCoreSurfaceRoot". This means that each driver who actually needs IOSurface will have the string "IOCoreSurfaceRoot".

* Please note that IORegistry isn't within the scope of this paper.

* You can however read about it in the following Apple's document:

https://developer.apple.com/library/archive/documentation/DeviceDrivers/ \
Conceptual/IOKitFundamentals/TheRegistry/TheRegistry.html

Looking up the string in IDA yields the following results:

```
__PRELINK_TEXT:__PRELINK_TEXT_hidden:                  IOCoreSurfaceRoot
com.apple.iokit.IOSurface:__cstring:                   IOCoreSurfaceRoot
com.apple.driver.AppleM2ScalerCSC:__cstring:           IOCoreSurfaceRoot
com.apple.iokit.IOMobileGraphicsFamily:__cstring:      IOCoreSurfaceRoot
com.apple.driver.AppleD5500:__cstring:                 IOCoreSurfaceRoot
com.apple.driver.AppleAVE:__cstring:                   IOCoreSurfaceRoot
com.apple.drivers.AppleS7002SPUSphere:__cstring:       IOCoreSurfaceRoot
com.apple.driver.AppleAVD:__cstring:                   IOCoreSurfaceRoot
com.apple.driver.AppleH10CameraInterface:__cstring:    IOCoreSurfaceRoot
com.apple.iokit.IOAcceleratorFamily:__cstring:         IOCoreSurfaceRoot
com.apple.iokit.IOAcceleratorFamily:__cstring:         IOCoreSurfaceRoot
```

Because Apple's drivers are mostly closed-source, it takes a lot of effort to understand how each driver uses the IOSurface objects. Therefore it was necessary (and just easy) to look for the string "plane" in each one of these drivers. While this doesn't guarantee we actually find anything useful, it's easy and it doesn't consume a lot of time.

Fortunately, the following string came up (newlines added for readability):

```
Assertion "outWidth > pIOSurfaceDst->getPlaneWidth(0) ||
outHeight > pIOSurfaceDst->getPlaneHeight(0) ||
outWidth < 16 || outHeight < 16 || inWidth < 16 || inHeight < 16"
failed in "/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleD5500/
AppleD5500-165.5/AppleD5500.cpp" at line 3461 goto bail1
```

Around the usage of that string, there was the following assembly code:

```
SXTW            X2, W25
MOV             W1, #0x80
MOV             X0, X21
BL              memset
```

As AppleD5500.kext is closed-source, one needs to guess a lot, and try to infer from the code what is the context in each function. Because we search for the usage of an IOSurface object, which has a vtable, one useful thing would be to add a comment around every virtual call with the function name of the corresponding IOSurface object. Having this and our "plane" string in mind, we expect virtual calls which contain "plane" in their name. To find the vtable of IOSurface (or any vtable of any object in a kext), it is possible to reverse engineer the kext on a macOS. Kexts are still symbolicated on macOS and therefore it is possible to obtain meaningful names for the vtable entries.

For the sake of this example, we'll reverse IOSurface.kext here. Opening up the IOSurface kext binary (on macOS it is located in the following path: /System/Library/Extensions/IOSurface.kext/Contents/MacOS/IOSurface), we get a symbolicated kext. To get the actual IOSurface's vtable, we can simply open the "Names" view (Shift+F4 for the keyboard shortcut lovers) and search for the string "vtable for'IOSurface". This will give us the offset-0x10 of the vtable, along with all the entries, symbolicated. Although sometimes the vtable entries are in a different order, they are virtually the same (minus the diff between ARM and Intel CPUs), so it is necessary to make sure you look at the same function and not just blindly picking up the name from the macOS version.

This indeed works here:

```
LDR             X8, [X19]               ; X8=IOSurface.vtable
LDR             X8, [X8,#0x110]         ; X8=&IOSurface->getPlaneSize
MOV             W1, #0
MOV             X0, X19
BLR             X8                      ; IOSurface->getPlaneSize(0)
MOV             X23, X0
LDR             X8, [X19]               ; X8=IOSurface.vtable
LDR             X8, [X8,#0x110]         ; X8=&IOSurface->getPlaneSize
MOV             W1, #1
MOV             X0, X19
```

```
BLR             X8                      ; IOSurface->getPlaneSize(1)
MOV             X25, X0
SXTW            X2, W23
MOV             W1, #0x80
LDR             X0, [SP,#0x120+var_E0]
BL              memset                  ; memset(unk, 0x80, planeSize0)
SXTW            X2, W25
MOV             W1, #0x80
MOV             X0, X21
BL              memset                  ; memset(unk, 0x80, planeSize1)
```

So it looks as if we have a new primitive! We can arbitrarily overwrite something with 0x80, while we control the length of the overwrite. We do not control "unk" (which is later revealed that it is the mapping of the IOSurface object; keep reading). The length is taken from the plane member of something we assume is an IOSurface object, which we can arbitrarily control using the vulnerability in IOSurface.kext. Obviously this is a far fetched assumption. Except for the string we found, there's nothing else that hints this is indeed an IOSurface object. To verify that, it is necessary first to understand what AppleD5500 is.

AppleD5500 is a video-decoding driver, which is not accessible from the default sandbox. Communication with this device is done solely via mediaserverd, as described in the infographic above (section 1). So the next objective is to see how to trigger the function with the IOSurface usage. The function is approximately 20 functions from the entry point to the driver's communication (AppleD5500::externalMethod) [3]. Apple does not provide us with the right tools to debug the iOS kernel (in fact, it constantly makes it more and more complicated), and macOS doesn't have this driver. While guessing can get you started, getting a deterministic code-flow is something that we want to assure, and not assume, as the direction of our research might be oriented based on such an assumption.

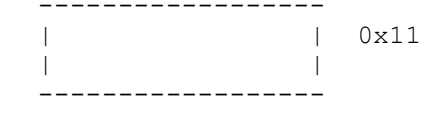
--[ 4 - Tracing the iOS kernel

I took Yalu102 [4] (thanks to @qwertyoruiopz and @macrograss for that) and utilized its KPP bypass. KPP [5] is a (not so new) mechanism that was introduced in iOS 9, checking the integrity of the text section of the kernel, meaning you can't modify the kernel's text section. I didn't care about setting breakpoints in the kernel, but I just wanted to get a dump of all the registers given a specific address. This would be enough to understand how to control the code-flow, or at least how to progress steadily towards the function which uses the plane from our IOSurface object (and understand whether this was actually an IOSurface object in the first place).

What I did was as follows; assuming we want to see the registers' state at address 0x10C:

Kernel code with no KPP

```
  ------------------  ADDRESS
  |                |  0x100
  |                |
  ------------------
  |                |  0x104
  |                |
  ------------------
  |                |  0x108
  |                |
  ------------------
  |                |  0x10C
  |                |
  ------------------
  |                |  0x110
  |                |
  ------------------
  |                |  0x114
  |                |
```




We overwrite 0x10 bytes with the following assembly code:

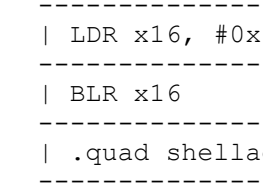
```
LDR x16, #0x8
BLR x16
.quad shellcode_address
```

shellcode_address contains code which prints the registers' state, a snippet from the shellcode:

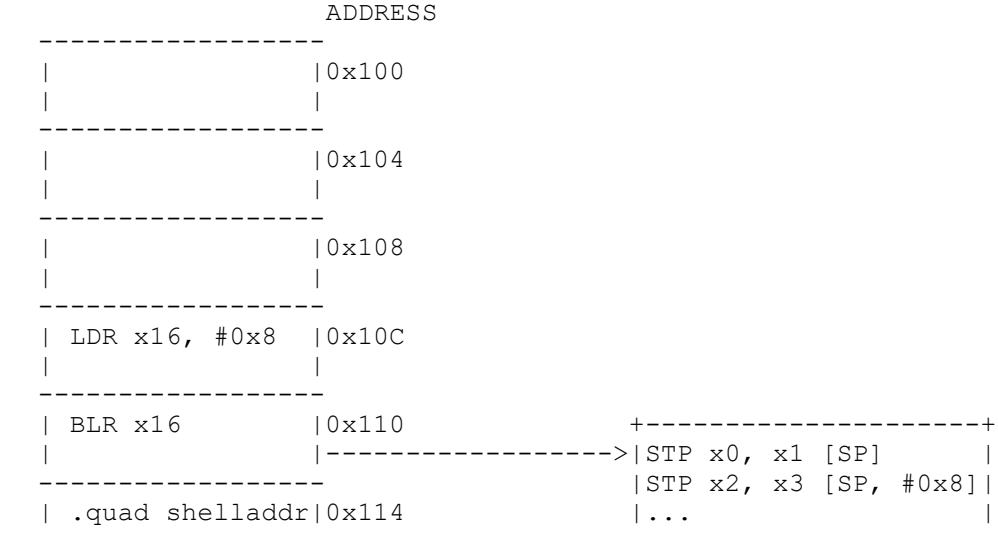
```
STP x0, x1 [SP]
STP x2, x3 [SP, 0x10]
...

LDR x0, debug_str
LDR x16, kprintf
BLR x16
MOV x0, x0
MOV x0, x0
MOV x0, x0
MOV x0, x0
RET
```

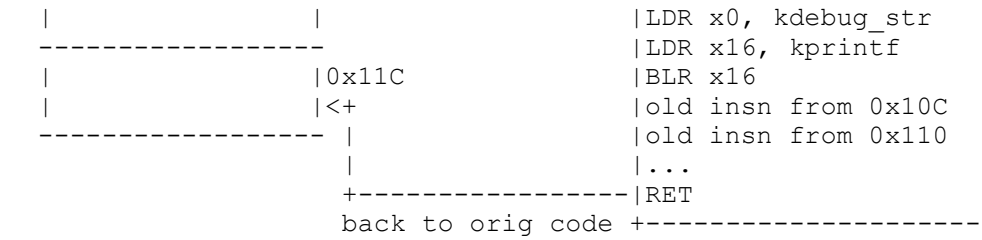
Before overwriting 0x10C, the last 4 NOP instructions (MOV x0, x0) are replaced with the original instructions at 0x10C-0x11C. This way, the code executes seamlessly (as long as no branches are being replaced). x16 was chosen because according to the ABI, x16 is only used for jumping to stubs (so it is safe to overwrite it). This way we can see the registers' state at (almost) any address in the kernel without hurting performance or slowing the research. Generally speaking, I've found that this infrastructure work, as time consuming as it might be, will be insanely helpful later on, and always worths the invested time.

Ultimately, the state of the kernel text will look like the following:




```
 ; memcpy(0x10C, 0x1000, 0x10)
```






The shellcode advances X30 as well, so that we return to a valid instruction (0x10C-0x11C are not restored upon the shellcode's execution). At the time of this writing there are other (public) ways to achieve the same result, but that's what I did, and the most important point I'd like to show here is that infrastructure work is extremely important, and I think every decent researcher who has experience in the field, has written some tools\scripts to ease the research process. Besides, at the return\appearance of an AMCC bypass, this could sleep be handy ;)

--[ 5 - Reversing AppleD5500.kext

Continuing our research, we know that AppleD5500 has something to do with IOSurfaces. So the next step is to see where the driver actually looks up\fetches IOSurface objects based on their IDs. A quick string search reveals the following string:

"AppleVXD393::allocateKernelMemory kAllocMapTypeIOSurface - lookupSurface failed. %d\n"

Going to the place where this string is being used, I did the same thing - I added a comment near every virtual call to see where the driver probably uses IOSurface (you can probably guess by now that this is an automated script, another 'infrastructure' work :) ). This indeed looked like an IOSurface object, but to verify that for 100%, I used the same kernel tracing technique like before and checked the vtable of the object in use. This was indeed an IOSurface vtable! This means we know now where the IOSurface object is being looked up. IOSurface was stored exactly in the same offset used in our mysterious memset call. Using the kernel tracing technique we see that indeed this IOSurface object is used for the memset as well! So if we can control the IOSurface object we can do an arbitrary write.

Unfortunately, at this point Apple silently fixed the IOSurface plane bug, but I got involved in this research deep enough to continue researching this area of AppleD5500.

Now the next part is to make sure we control this IOSurface object. We can obviously do that assuming we magically have an AppleD5500 send right port, but perhaps we can influence mediaserverd to supply our own IOSurface object.

--[ 6 - Influencing AppleD5500.kext via mediaserverd

Reverse engineering mediaserverd and looking for calls to anything that looks like AppleD5500 yielded no results, but after further investigation (= symbols and strings search). I saw that VideoToolbox was responsible for video decoding, and thus I assumed it was responsible for AppleD5500 as well (though no mention of AppleD5500 was in VideoToolbox).

When looking for AppleD5500 strings in the entire dyld_shared_cache, I found out that a library named H264H8 contained several different references to AppleD5500. One of the interesting call flow was:

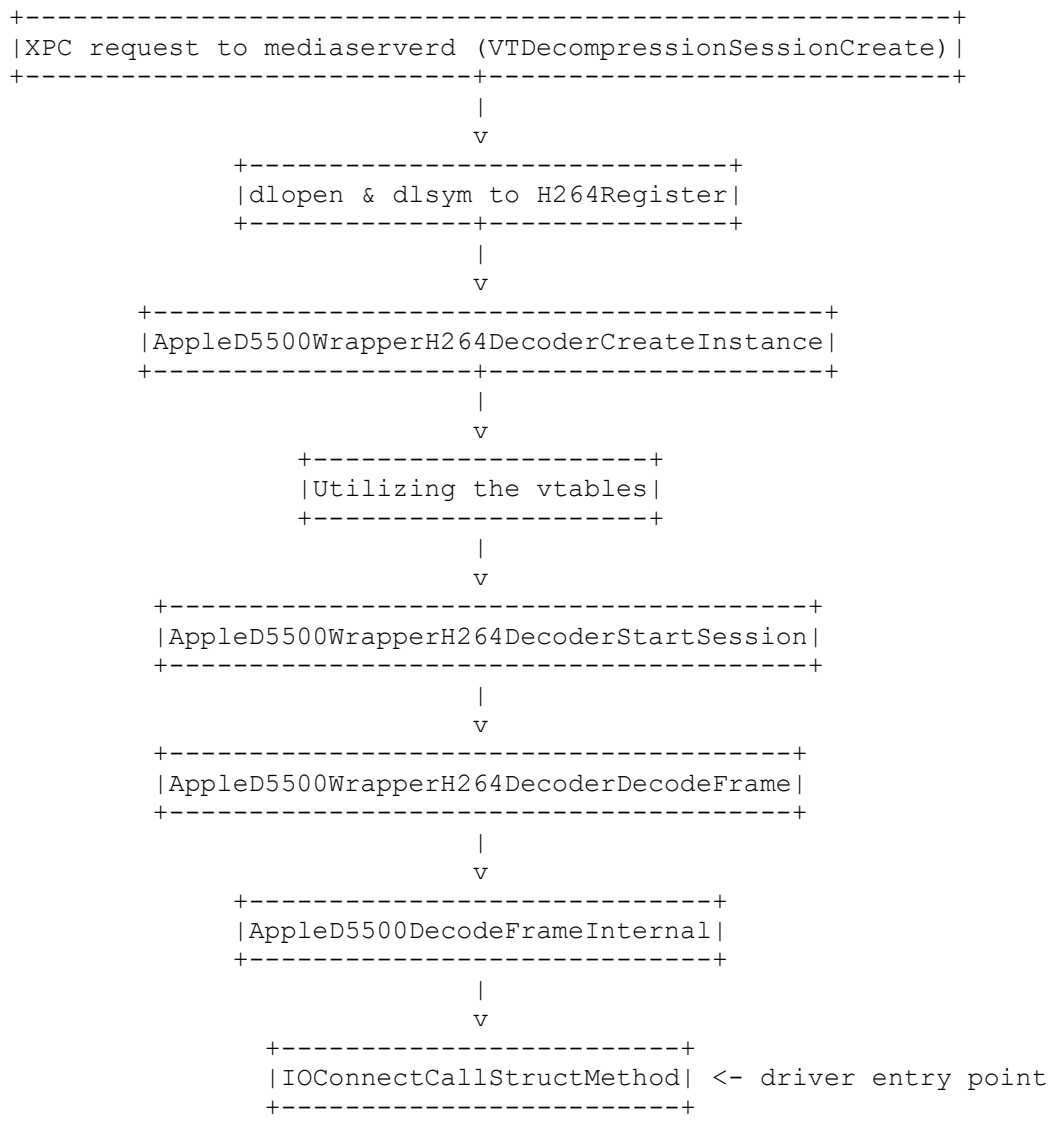
```
AppleD5500WrapperH264DecoderDecodeFrame
        --> AppleD5500DecodeFrameInternal
              --> IOConnectCallStructMethod ; Calling one of the driver's
                                            ; 'exposed usermode API' [3]
```

AppleD5500WrapperH264DecoderDecodeFrame had no xrefs unfortunately, but as (most) of the code isn't written not to be used (or it would be optimized

out in that case), I assumed this function might be inside a vtable.

Binary search in IDA for the address of AppleD5500WrapperH264DecoderDecodeFrame indeed resulted in something that looked like a vtable. The vtable used in an object's initialization code in a function called AppleD5500WrapperH264DecoderCreateInstance. H264Register was an exported function with no symbols and no xrefs, but the string "H264Register" did appear in VideoToolbox. It appears that VideoToolbox treated H264H8 as a dynamic library and H264Register as the "entry point" (found with dlsym).

So to actually trigger usage of the driver without having a send right to the driver, we needed to do the following:




VTDecompressionSessionDecodeFrame is a documented API that checks if we're a "server" (e.g, mediaserverd) and does a lot of logic assuming access to those drivers. Or it just sends a mach message to mediaserverd if we're not a server. Despite being 'documented', VTDecompressionSessionDecodeFrame had a secret undocumented feature which I discovered during reverse engineering AppleD5500WrapperH264DecoderDecodeFrame (so a much later stage of this API).

It is possible to embed some properties in the sampleBuffer, in a dictionary called "tileDecode":

```
tileDecodeDict = CMGetAttachment(sampleBuffer, CFSTR("tileDecode"), 0);

if (tileDecodeDict) {
    cfnum = CFDictionaryGetValue(tileDecodeDict,
            CFSTR("canvasSurfaceID"));
```

```
    uint32_t surfaceID;
    CFNumberGetValue(cfnum, surfaceID);
    ...
    x = ... CFDictionaryGetValue(..., CFSTR("offsetX"));
    y = ... CFDictionaryGetValue(..., CFSTR("offsetY"));
    lastTile = CFDictionaryGetValue(..., CFSTR("lastTile"));
}
```

The dictionary had 4 properties (or at least, I saw 4 properties): "canvasSurfaceID", "offsetX", "offsetY", "lastTile". I had no idea what these properties meant, but "canvasSurfaceID" sounded perfect for our case: What if we could supply a surface ID to canvasSurfaceID, and hope that, magically, this surface will be used in AppleD5500 in the behaviour we saw previously?

And so it appears - this could indeed influence the behaviour of mediaserverd and make sure it sends our requested surface object to AppleD5500!!

This could be verified both by reverse engineering mediaserverd and following the IOConnectCallStructMethod call, and the buffer given to AppleD5500, or simply using the kernel tracing technique to see whether the surfaceID of the object in AppleD5500 matches the surfaceID we sent (which requires prior reverse engineering of IOSurface.kext).

It could also be performed by calling the function IOSurfaceRoot has to lookup surface IDs, and see if we get back the value we expect given our specific surfaceID. Most important thing is - to make sure that this indeed influenced the given surfaceID. I personally did it by reverse engineering mediaserverd and following these calls, because I was interested in offsetX and offsetY as well, though this isn't necessary (but proved to be useful, as you'll see soon ;) ).

--[ 7 - _The_ bug

Back to our main objective, get to that memset with our arbitrary 0x80 write. Looking up the code, I noticed the following:

```
if ( context->tile_decode )
{
    dest_surf->tile_decode = 1;
    tile_offset_x = context->tile_offset_x;      // [0x1]
    dest_surf->tile_offset_x = tile_offset_x;
    tile_offset_y = context->tile_offset_y;      // [0x2]
    dest_surf->tile_offset_y = tile_offset_y;
    v73 = tile_offset_x +
        tile_offset_y *
            dest_surf->surf_props.plane_bytes_per_row[0];   // [0x3]
    v74 = tile_offset_x
        + ((dest_surf->surf_props.plane_bytes_per_row[1] *  // [0x4]
        tile_offset_y + 1) >> 1)
        + dest_surf->surf_props.plane_offset_again?[1];     // [0x5]
    dest_surf->surf_props.plane_offset[0] = v73 +
    dest_surf->surf_props.plane_offset_again?[0];
    dest_surf->surf_props.plane_offset[1] = v74;
}
...
if ( !context->field_4E0 &&
   !(context->some_unknown_data->unk & 0x30) ) // [0x6]
{
  surface_buffer_mapping = v85->surf_props.surface_buffer_mapping;
  if ( surface_buffer_mapping )
    memset_stub(
        (char *)surface_buffer_mapping +
        (unsigned int)*(_QWORD *)&v85->surf_props.plane_offset[1],
        0x80LL,
        ((dest_surf->surf_props.plane_height[0] >> 1) *
        (*(_QWORD *)&dest_surf->surf_props.plane_offset[1] >> 0x20)));
}
```

The data in [0x1] and [0x2] are completely controlled by the user. These are the offsetX and offsetY which we provided in the dictionary and they were forwarded exactly without any check.

It looks like, [0x1] and [0x2] are being used in a calculation that ultimately leads not only to a write of 0x80s with an arbitrary length, but also to control the offset from which the write is done! This makes our primitive much more powerful as we can make our overwrite more accurate.

The values mentioned in [0x3], [0x4] and [0x5] are attributes of the IOSurface in question, so they are usually somewhat controllable. In this particular case, the limitations on these attributes pose no restrictions on the impact of the memset's primitive. While these attributes aren't really within the scope of the paper, for the curious reader, you are welcomed to reverse IOSurface::parse_properties to see what IOSurface expects to receive for creation.

One problem I noticed with kernel tracing though, is that we never get to the memset because of the following condition:

```
context->some_unknown_data->unk & 0x30 // [0x6]
```

The obvious problem we face here is that there are no sources and these are actual offsets in a struct and unfortunately there's no easy deterministic way to know which object we look at. Looking at the assembly code for this line, it is decompiled from the following:

```
; X19 = context
LDR             X8, [X19,#0x448]    ; X8 = context->some_unknown_data
LDRB            W8, [X8,#6]         ; W8 = unk
AND             W8, W8, #0x30       ; unk & 0x30
CBNZ            W8, skip_memset     ; if (unk & 0x30) goto skip_memset;
```

Because the offsets weren't so common (0x448, 0x6), it is possible to actually grep the entire driver text section and start trying to find the right reference by grepping. Because this happens pretty often when reversing IOKit drivers (or reversing "large" binaries anyway), I highly recommend automating this process. Imagine how good life would be if you could just grep for "STR *, [*, #0x448]". It's not a oneliner in Python, but for the long run this worths it. For this case however, grepping would be enough:

```
$ cat d5500 | grep STR | grep 448 | grep -v SP
0xfffffff006c30448L STR             D1, [X19,#0xA90]
0xfffffff006c41448L STRB            W13, [X1]
0xfffffff006c44488L STRH            W17, [X13,X15,LSL#1]
0xfffffff006c4481cL STR             W8, [X19,#0x64C]
0xfffffff006c44890L STRB            W9, [X8,#6]
0xfffffff006c448e8L STR             W9, [X8,#4]
0xfffffff006c47448L STRB            W0, [X19,#0x2A0]
0xfffffff006c495ccL STR             X9, [X10,#0x448]   ; only option
0xfffffff006c50448L STR             W24, [X22,#0x17BC]
```

For brevity, I'll sum this xref looking process for you - it wasn't magical, and I made some tools to speed up the process. Sometimes the offsets are very common and then grepping won't work - for this case sometimes the best way is just manually following the code flow. Going further, I eventually got to this code:

```
LDR             X11, [X19,#0x1B0]
LDRH            W11, [X11,#0x24]
LDR             X12, [X19,#0x28]
LDRH            W13, [X12,#6]
MOV             W14, #0xFFCF
AND             W13, W13, W14
BFI             W13, W11, #4, #2
STRH            W13, [X12,#6] ; This is the "unk" we were looking for.
```

I then looked for 0x1B0, which was responsible for this entire calculation, and then I saw the following string:

"CH264Decoder::DecodeStream error h264fw_SetPpsAndSps"

In the same function, I found another interesting string:

"AVC_Decoder::ParseHeader unsupported naluLengthSize"

--[ 8 - H.264 in general and in iOS

I googled then "AVC nalu" and the first result I got was "Introduction to H.264: (1) NAL Unit" [6].

I figured, it might be easier to understand a little bit more about H.264 (as I had 0 experience with that before this research). The standard of H.264 can be found at [7].

The relevant page for NAL unit is section 7.3.1, "NAL unit syntax". As we can see from the copy, each NAL unit has a type and is being processed according to its type ("nal_unit_type"). From all of the different NAL unit types, there are 3 which are necessary to know:

*) SPS (sequence parameter set): General properties for a coded video sequence. An example of a property which is held by SPS is the "level_idc" which is a specified set of constraints that indicate a required decoder performance.

*) PPS (picture parameter set): General properties for a coded picture sequence. An example of a property that PPS contains is "deblocking_filter_control_present_flag" - flags related to the deblocking filter - a video filter which helps smoothing edges between macroblocks in the video. Macroblocks are like blocks of pixels (a very rough description, but good enough for our case).

*) IDR (Instanteous decoding refresh): This is a standalone frame, a complete picture which doesn't need other pictures to be displayed. IDR is always the first NAL in a video sequence (because it's standalone and other frames depend on it).

The question is - how to find the appropriate type in the kernel and the code that processes each NAL unit according to its type? I started searching for NAL unit type strings in the kernel (SPS, IDR, PPS, etc), and found the following piece of code:

```
LDP             W9, W8, [X19,#0x18]
CBNZ            W9, parse_nal_by_type   ; [0xA]
CMP             W8, #5
B.EQ            idr_type_and_no_idc_ref

parse_nal_by_type
SUB             W9, W8, #1              ; switch 12 cases
CMP             W9, #0xB
B.HI            def_FFFFFFF006C3A2DC
ADRP            X10, #jpt_FFFFFFF006C3A2DC@PAGE
ADD             X10, X10, #jpt_FFFFFFF006C3A2DC@PAGEOFF
LDRSW           X9, [X10,X9,LSL#2]
ADD             X9, X9, X10
BR              X9                      ; switch jump

idr_type_and_no_idc_ref
ADRP            X0, #aZeroNal_ref_id@PAGE ; "zero nal_ref_idc with IDR!"
ADD             X0, X0, #aZeroNal_ref_id@PAGEOFF
BL              kprintf
MOV             W0, #0x131
B               cleanup
```

As we can see here, "idr_type_and_no_idc_ref" happens "if [X19+0x18] == 0"

(at the [0xA] marker) and if [X19+0x1C]. Checking in the manual, we can see that for NAL type == 5, we get indeed an IDR NAL. Based on this findings, we can assume that [X19+0x18] is nal_ref_idc and that [X19+0x1C] is the type of the NALunit!

Back to our mysterious offset 0x1B0, I started thinking - perhaps it is either PPS or SPS? The string we found earlier is pretty clear that the function is doing something with them. I then decoded a video with the API and using the kernel tracing technique, I looked at the content of 0x1B0 to see if it looks like something which looks like SPS or PPS. Luckily for us - this was indeed the SPS object!

I figured that out because within 0x1B0, All of the values were in fact the values of the SPS object which is described in the standard (section 7.3.2.1.1 [7], I love you too).

By slightly changing the SPS of the video I was tracing, I saw that the changes in 0x1B0 were correlated. In fact, most of the SPS object was stored there in the same order as it appears on the manual :) so this was even easier once I found the function in the kext which filled up the object. This was sufficient to understand the mysterious unk & 0x30 check which means (wait for it):

If SPS->chroma_format_idc (section 7.3.2.1.1 [7]) == 0, we get to the memset we were waiting for! At this point, I already had some tools to create and manipulate videos. So creating a video with chroma_format_idc == 0 wasn't a big problem. To send a video for decoding, you first have to call the function CMVideoFormatDescriptionCreateFromH264ParameterSets which creates an object that holds information about the SPS and the PPS of the video. This object is given to mediaserverd to create the session. After the session is created, we get a handle representing the session, and give it to *DecodeFrame which ioctls the driver from mediaserverd (see graph above). I created such a video, sent it to decoding, was waiting for it to crash the device and... nothing happened!

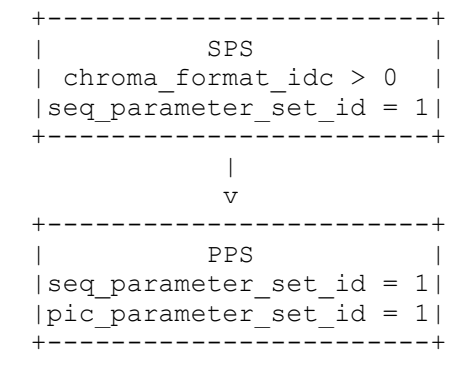
After a brief reversing of mediaserverd, it appears mediaserverd rejects chroma_format_idc == 0!

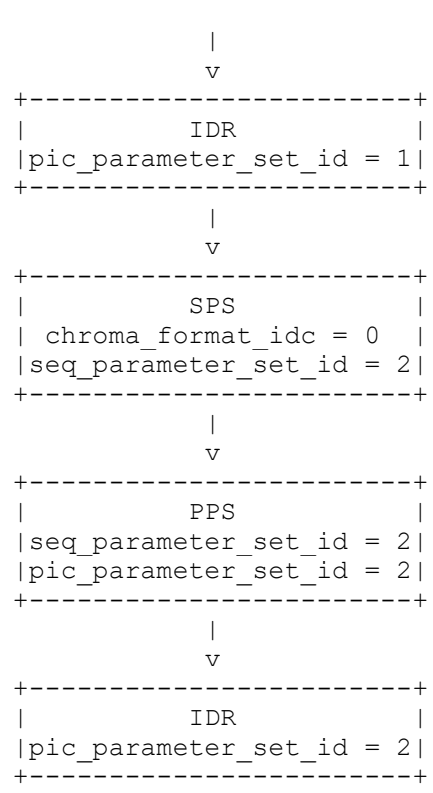
--[ 9 - mediaserverd didn't read the fucking manual

But...

mediaserverd only gets the SPS information at the beginning in the function CMVideoFormatDescriptionCreateFromH264ParameterSets which is only being called once. According to the manual (haven't seen a single case in practice though, and I've seen plenty of "Snow Monkey in Japan 5k"s during this research), there could be multiple SPS objects there (section 7.4.1.2.1 in [7]). Which is odd, because if mediaserverd only gets the SPS and PPS information once, and rejects them, then how it is supposed to be aware of the other SPS\PPS packets? (*DecodeFrame just passes the packets to the driver without doing any sanity check).

With this in mind, I decided I'd just try creating a video with a normal SPS\PPS properties, then in the middle of the video embed a new IDR, which points to a new PPS, which points to a new PPS with chroma_format_idc == 0, and see if that bypasses the check deployed in mediaserverd.






The moment the IDR packet with pic_parameter_set_id = 2 was sent, the kernel crashed with the panic we were expecting! Slightly afterwards, iOS 11 was released. And unfortunately - the same PoC code did not crash the kernel...

I diffed the driver code but there wasn't any change. What I did notice however, is that the string "canvasSurfaceID" did not appear in the binary of the driver anymore. I did notice that a bunch of undocumented APIs were introduced then, namely VTTileDecompression* (instead of VTDecompression). I was too lazy analyzing the function with IDA (to be fair, IDA and dyld_shared_cache aren't good friends yet), so I decided to go with a different approach: try attaching debugserver to mediaserverd, and change the given values to IOConnectCallStructMethod, hoping that the kernel crashes if I change it to the same values back in iOS 10.x. Attaching the debugger obviously doesn't work out of the box. I assumed both the driver and the process need the run-unsigned-code entitlement, so without even checking why things didn't work, I just injected the entitlement to mediaserverd and to debugserver and tried attaching debugserver again.

The entitlements' dictionary is stored in task->bsd_info->p_ucred->cr_label->l_ptr:

```
struct task {
    /* Synchronization/destruction information */
    decl_lck_mtx_data(,lock)         /* Task's lock */
    _Atomic uint32_t    ref_count;  /* Number of references to me */
    boolean_t   active;             /* Task has not been terminated */
    boolean_t   halting;            /* Task is being halted */
    ...
    /* Task security and audit tokens */
#ifdef  MACH_BSD
    void *bsd_info;                 // struct proc
    ...
};

struct  proc {
    LIST_ENTRY(proc) p_list;        /* List of all processes. */

    pid_t       p_pid;          /* Process identifier. (static)*/
    void *      task;           /* corresponding task (static)*/
    struct  proc *  p_pptr;         /* Pointer to parent process.(LL) */
    pid_t       p_ppid;         /* process's parent pid number */
```

```
    pid_t       p_pgrpid;       /* process group id of the process (LL)*/

    ...
    /* substructures: */
    kauth_cred_t    p_ucred;        /* Process owner's identity. (PUCL) */
    ...
};

struct ucred {
    TAILQ_ENTRY(ucred)  cr_link;
    u_long  cr_ref;         /* reference count */

struct posix_cred {
    /*
     * The credential hash depends on everything from this point on
     * (see kauth_cred_get_hashkey)
     */
    uid_t   cr_uid;         /* effective user id */
    uid_t   cr_ruid;        /* real user id */
    uid_t   cr_svuid;       /* saved user id */
    short   cr_ngroups;     /* number of groups in advisory list */
    gid_t   cr_groups[NGROUPS]; /* advisory group list */
    gid_t   cr_rgid;        /* real group id */
    gid_t   cr_svgid;       /* saved group id */
    uid_t   cr_gmuid;       /* UID for group membership purposes */
    int cr_flags;       /* flags on credential */
} cr_posix;
    struct label    *cr_label;  /* MAC label - contains the dictionary */
    /*
     * NOTE: If anything else (besides the flags)
     * added after the label, you must change
     * kauth_cred_find().
     */
    struct au_session cr_audit;     /* user auditing data */
};

struct label {
    int l_flags;
    union {
        void    *l_ptr;
        long     l_long;
    }   l_perpolicy[MAC_MAX_SLOTS];
};
```

This time it worked! First, I took an iOS 10.x device, triggered the problematic flow, and put a breakpoint just before the IOConnectCallStructMethod function (which is the actual ioctl to the driver). I knew that this works, so I just copied the entire input buffer to the IOConnectCallStructMethod.  I then called the corresponding functions (same API, but changed the VT prefix to VTTile) and set a breakpoint again. Once I reached IOConnectCallStructMethod, I simply overwrote the entire input buffer and replaced it with the input buffer I copied from the iOS 10.x device. The kernel crashed! From there, it was easy to reverse engineer backwards from IOConnectCallStructMethod and see that the 6th parameter given to VTTileDecompressionSessionDecodeTile is simply the X and Y offsets shifted so that they fit in a 64 bit integer (each one of the offsets is a 32 bit integer).

Apple eventually fixed the bug by checking in the kernel for out of bounds before performing the write. They re-verified the values once again in AppleD5500.kext. If you would like to find the actual code where Apple introduced the checks, you can search up the kernel for the following string as this is now printed when putting bad arguments:

"bad IOSurface* in tile offset check"

After this string there's a series of checks for the attributes of the IOSurface object.

--[ 0xA - Takeaways

*) I've just displayed one vulnerability in an attack vector accessible from within the sandbox. Parsing video and making sure there aren't mistakes isn't that easy, and it's all done from within the kernel! It's obvious that there are more vulnerabilities in this driver, and in other codecs in iOS as well. The attack surface is (sometimes) more important than the vulnerabilities, and I think this is a good example because nowadays it is not _that_ common to find simple buffer overflows.

*) Manuals are super important. Often when reversing drivers, it is easy to fall for looking for patterns (looking for integer overflows, races, appropriate refcounts, etc). Understanding what we actually reverse and not just looking blindly for patterns was the only reason I thought about putting two SPS in the same packet. I didn't try "bypassing" mediaserverd, I just understood that SPS has an ID, and hence it is likely that there can be more than one of them. Maybe it wasn't the reason I found the vulnerability this time, but that happens as well.

*) Infrastructure is super helpful. Sometimes people can get lazy writing tools, but these might be helpful eventually, even if it takes a lot of time writing them (the kernel patching technique was really easy to write, but I did find myself writing a single tool for a few days just to have things easier when researching). It's an investment for the long term, but without the kernel tracing technique I would have probably given up already. I had so many assumptions which were mandatory to verify, and it was very easy thanks to the kernel tracing technique.

--[ 0xB - Final words

I'm not sure how you readers feel about this paper, but from section 7 till the first crash, it took me about a week. I tried to put as much details as I could into that paper, but unfortunately sometimes you either forget or ignore details. While it was time consuming and some experience IS needed for that, I'm trying to show you that it is not impossible to actually find bugs (good, reachable from the sandbox bugs). I highly encourage you to stop mentally masturbating about iOS bugs and just throw a freakin' kernelcache into IDA and just start reversing. It's much easier than it looks! We're still in the era where someone can drag a kernelcache into IDA and have a good bug within 2 weeks. Remember my words, in 5 years we'll miss these days, where we can completely wrap up such a project within a month.

Additionally, I would like to thank Zimperium for letting me doing this research. It is not always easy for a company to simply let a single person to do his own research on the internals of a video decoder driver, hoping that when he says there's something coming up, something actually comes up.

P.S. - I did start working on an exploit, and then more important things had priority over it. Hence some of the attached code might be redundant.

Sincerely yours,
Adam Donenfeld, aka @doadam.

--[ 0xC - References

[1] https://nvd.nist.gov/vuln/detail/CVE-2018-4109
[2] https://nvd.nist.gov/vuln/detail/CVE-2015-7006
[3] https://developer.apple.com/documentation/iokit
[4] https://github.com/kpwn/yalu102
[5] https://xerub.github.io/ios/kpp/2017/04/13/tick-tock.html
[6] https://yumichan.net/video-processing/video-compression/
    introduction-to-h264-nal-unit/
[7] https://www.itu.int/rec/T-REC-H.264

--[ 0xD - Code

begin 664 src.tar.gz
M'XL("#8LOUL"`W-R8RYT87(`[%Q[=]LVLL^_TJ=`FG,:*W7T\"O)=;>M(LN-

```
M-K+D*\EI<WKVZ%`D9.&:(E4^+&MW^]WOS`"D0!*4W;1)S]UKM;$M8O##8&8P
M&`P`AH'=>/*9/TWXO'IUC+];KXZ;^N_D\Z1U>'C\ZN3@I-D"NE;S\.#P"3M^
M\@4^<1A9`6-/K.!ZM8ONOO+_HY\0]-\;OA=1XS/K__CWZ+]U"(\>]?]%]3\<
M7_`PM*[YP(_$7-A6)'ROOOB3]']R=%2B_U;SX.@@I__CDZ/C)ZSYJ/_/_FF\
MJ+(7K..O-H&X7D1LSZZQUILWKU\>@&I8>[5R.90N5W'$@WW6\^PZ:[LN(^*0
M!3SDP2UWZ@"".#^T+R_[W>EP/.J>=T?=0:<[[?<ZW<&X.WW7;9]U1]/QI#V:
M_("T^&^R$"&;"VC#]KW($E[(A@`M/,N%5AW.+,]I^`&[\)W4)$/FS[-4B&2%
MS.%SX7&'"0^KL6AA1<P*.`OCV?]P.V*1#\^XZM-E/'.%S<9^'-B<]87-O9"`
M/O`@A%;80;W)]I#^N2I\7JNSCW[,EM:&>7[$XI`#7L(_O[/Y*H*V$<,&@;G"
M\@!Y+:(%-:M0ZM!GGHCO.K"\"#@&SC:``''L.#W1BQ$J:FW%LD6CM@%L1WV<@
M&.Y9,Y=3'7J*G,/C@#LBC`(QB^63^3Y"Q9YKK>>QBQ3PMVS#`6Y7@I-4+4])
MQU_Q`,"\:Q9NPH@OJ:W(I[Z)P(Z7M]R+]MFM\%T3)RF)8D?2*4Z@E0T"@4$M
M=[?*0G\>K5&%BE=F70><+P&XGEC0I<LM*/!G:#W,PLYL$%23(K,BI%Q$T>J_
M&HWU>EV'AKR0-%^WL.4Z**QAK4*W098#@@0K0I'/_0#%CBREJJYOC9<7K959
M,#I2OE,M@)@+VD7C1!R=R$?;9<_;8]8;/V<S*Q3A/ONI-WDWO)JPG]JC47LP
M^<B&YZP]^,C>]P9G^ZP+I=T1`G5_OAQUQV,V'+'>Q66_UX72]N",T:!D0-1]
M^Y&=]<:=?KMW,6;M?I^-KSKO$MQ>=TQFTAMT^E=GO<&/:</]WD5OTI[TAH-]
M:GA;`5FYZ(XZ[^!K^VVOWYM\)(SSWF2`K)P#+VUV"6.^U[GJMT?L\FIT.1QW
M]]E_7_6Z$]8=_'WX\:([F"#3@^'@96]P/H*FN_BLKBDXY#PCN[DO91FNN(VN
M@;F6=QW#[,6N_5L>>*@R-<I`*PCDBJ6(E`<IZ")5ZOT>K#LX0__5J):XSC=F
MKVETFTSYS7>]\60X^BB_-:K59V(.',XK4V`$_K\`4;9_[`Z&D]YYKT.*F+ZK
M/I,.C^TD0BB@`R)[Y<8A_JOR.V#+8U]UOF+_JC[C'OA6I/-L-P8C_G9IV8L&
M_IA&FQ4/ZXOOM$*'WX+$&O*7B4`&$[WAB$=QX&%9E7OQ$EIB\+DY%T$8$8T>
M9DP`AU4J?V.M9G-?$O:&8Y`1-$*>.EP\E/S"BNP%=_+DDKJ5IYZ`$X+Q&QDJ
M(/U!2M\.-YZ-*G4Y4ACACXO,%".JE/HDH>Y;Y1)!PC=OJK^=9J4X'.NTJI7>
M&9(?'^XG-#K//$OT2A%=6'=$U0ZN0]G:2:$QU?F1,MH>6.8=`]+FOK&\X\<P
M-U2S@@.OZ,?1>>S9:?62>H9J(SZ'\(`J&LMS+9(%P/C/-WIOF[F*IF9S)%K]
MA*(',SJ,[^AWMYZK:&H]1Z*,;"<-M4$JK58;#?8R\\G9E&)N?G)$M8#CU_O%
MLK'X)R?]%2J`(]O;$UY48R'0^/,]X4\A9@FF`9\#3QXZC%H-F4&_@5[)0,"V
ML/+1R=$OQ:;^`1V"63.VD5X?"^]@[N;!R9'J%;FQ97@]18ZF405_G58JC1<4
M<$(@0:RBST5J<`1Q8+E`5T$.3^EAD9]*RNXI^=<D\MR;0GA[UIE^Z([&Z'VG
M-?;UUZSPE'U'@_I-L]6O40NQ%XIKC%SMA154%&._0!>?<1>#P%*2)M&0_T8-
M/UL%UO728BO+OMD[JB4"2KU1:AJR>BHDK=M+Z24F:>\-&DJ(P&?$&(O]T@(V
MH/U,\S6#M9&LGK[OC@;=_O1JW!T='E1+3+#<`!/S@[J:(:5=**CKESSJ/:;S
M^0SG/\1L"D93L)E;7SCLQ0/-1'"0+(9RV`4"*6\Y.;?+"])M(&!5>Q&IW^&
M=&4G($:_#G_Y!X1/5J2"=*#=VT/FN5.KZ<+6:S1W5]G*7@9ZF?`LL>E=1FIZ
M>)JO6:K!L@(#0HG<RPI.M7BR$'?^MBU,=;XS>B43>/+X^2+YO]YP(J/YSY+_
MW9'_.SQHM7+YO\-F\_`Q__=7Y_].'O-_C_F_Q_S?8_[O,?]GRO]MDW0P@?8F
M\'/R\;([SF3F\B5IP$<%*2%]8RTM+GHJ'\ED8#8S)\/J;-:-2FZ7VX3<[HS<
MPY."LHN#JWY?#ZHQ?DZKUI)N$%FEJ4+BI&_X$,)?&1W7FK4$7/VJJ#SJ6&`H
MR1PKLICLA2YD8/6BW9%2G.(JK`-V9]DJ8LD2:E37,0BK7J\C+=.(->F$D3/S
M?3<K3E?,;D`:C>%XLA5HJINT"6I7>[QED1I)H^DKB+,/#RJ5WG"X0@-\*Z)P
M&VN/T])S<<>=4T,U-2F9BGK@*&E=FR]D#$H[8!<<9D]<<N;J8O';3<2I\N%!
MMMOK)4;;\Y*@(T!M>+C8AS,INVW$@$@@-(`4:!"K%Z7/O.EKLA)$DQ$YJQ/W+
MDZ/I-*U#2W<8"Y9L$5:H($`11''*1+)J2VIDB(W4Z7#0EI6R65I.)@_E"K:6
M`F<D6-$%FF<AHXNL[+?M;XFS',HU<-^_U@1]^CO8U8`+ZJH4'YX:Z!.]5`K/
M3C4OEY0<'NRQA=AGKL]JE<K>GE1S#;]_P]*O>PM18]]^RPX/:K4<"&:!*B='
M527%\@Z@,']'![+D#^Z`ZQL9Q/Q6HKO<TKJ:9"YRK%6LA$59G%I!I>(J;G[3
M:HU@UM/'MEJWI^A9.WD@N*J48)>-LRPA6*QJ1=4CS=!4EM1$IVAFM#AD,YPF
MQ2H7I_.:;S.=/_5&U11S8:T@IHO6G'OL63*34>@:+Y]C'.S!6M"3$0.$IW<1
M#)30#@3Y;#D=Y=C_5[4BO(C=6FX,K5<(@7)M[(5G+;ED<0!_.1\D2<H*7_K!
M!H)%<>UA/,M>ODQ"&>`)%C,66_EKB%$PE%W[V<;3K!ZTC6)/0%!7J1%.<=Z=
M?KB83MKC]SC&,#JPPIMIA'-PPL=E#.$J!,XH8:R7AE-A%,_GLMEJHY$/)A3Y
M-F?PW3:R@4GPW51Z%CV_)2.AR^%H,IZ>=<][@^X9V%(^4LJ79WSZR@\B4+_P
MISZMZ*;1:6;Z-4+DION2_4M]1R(!QX9`HYYJJ80"C0>7+7ZP@TA$]Y($_!K7
M`1L`Q)_EA*%,W4TU=8,13(=O_][M3$CKH&RM3HTU,UE633]*NV`M'=E/&"'*
M,I?TB[2;VPP\XW,K=J.+E`"W`@V;AHJ.(I!*9KNPYRW$3&Q+MKNR/P4@J<DB
MB-.R[0XLYBAFEGV3%AUFJW7\Y0QD(8O_QHZ()>I<TB58Z^&RP*?QG.\6=+WM
M;=8+'DAN[]3)Q):^L;B2\10,(XWFE;8)G9",%V(>5;0-#"K*BR0G)48?F/#R
M0#I&7G@Y>3X((R?FOQ5%_P"0@C[R.GHP)SG-F11JQM&`KF!8R.`Z)/6DVIG/
MYQG"$2SNAYZ[V2JPU40%:B1C7";:BJ`ES6`_`Y'LZ$B2@R+)E2=^C9/R(UDN
M#?+%4_8#6=X8IF_.SBT;_$)(57]P1&C'(:6?KC`S`]:(.8T0*:=SHF0K*X#Y
M!/P)$YA_^S46@<P**4]JP?0@(IHWQ!+7M0B<3&WL9F!Y?LCANR/;ES\5-BZJ
MO90`TP\(#3!A`>="V,%NH.66XAXDUQ7W(*44.Y$FPKXI@:!4`5`Y5@"+5]#I
MNW_6`,6^*8",=W%B8()\J.Y*\A)F6P=7$!J=8&FFI3E!R%*8I34BG3WZ9(BR
MM%MYL(1V+P/00.J:LDO*6H+ASX0KHHU<@V>7L7K<P=!UX3PTB)<SW'MZ\([/
MTV):!)MYW(KYR_=_D@31%][_.3DZR9__/VR=O'K<__FK]W\.'O=_'O=_'O=_
M'O=_'O=_RL]_;X]N-ZHL/1,>\.VA),H.0:`XA/:"D`;T#*-*"&LWB=.J9\Y]
M:Q'2J#NY&@V,>TIIT:>>\>9!X`=Z]J."FQ[3@-B6F7#9!>V@$HS,J?!O1)0R
```

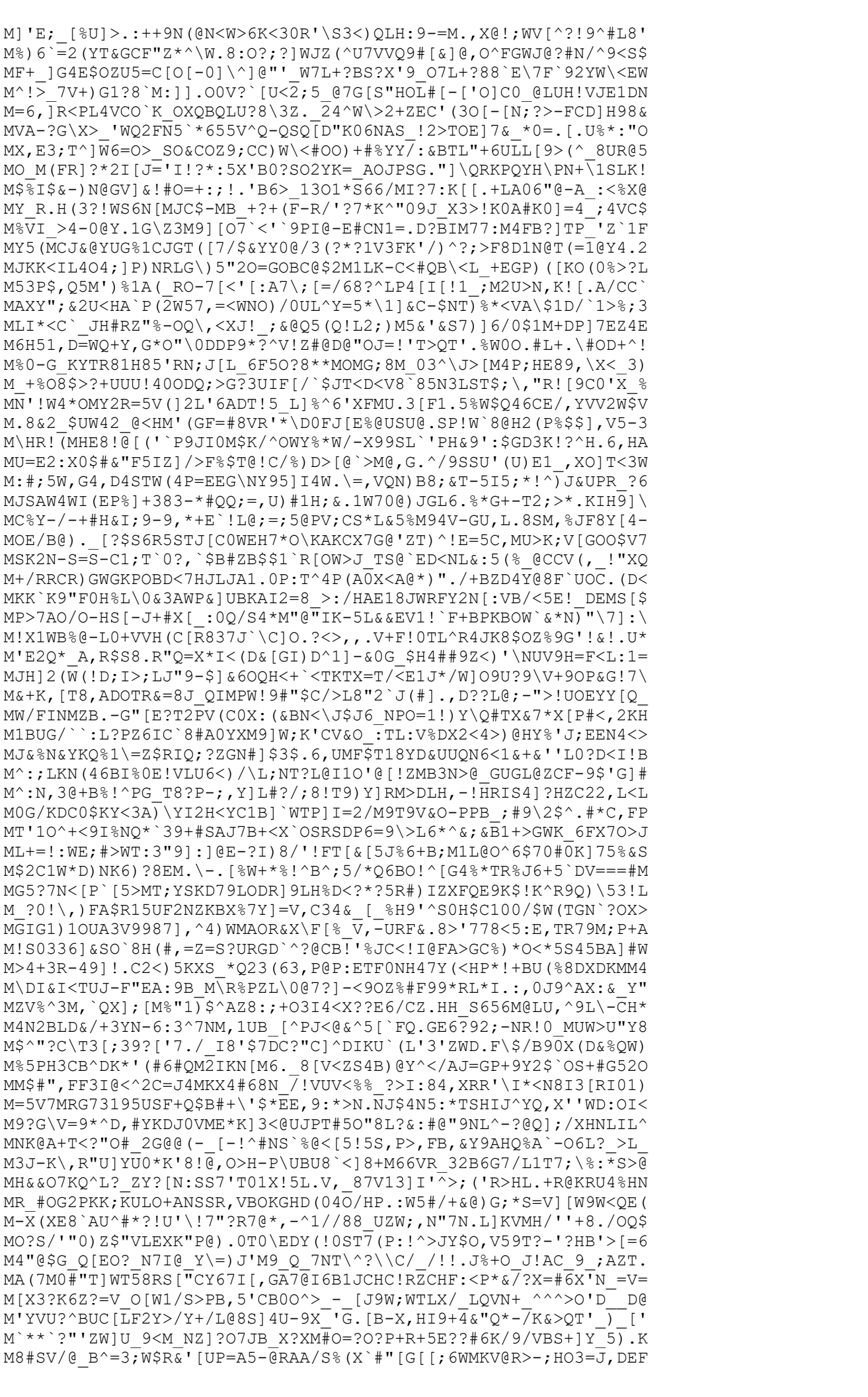
```
ME#YA91]H9RI']%[S[O!U30L&MY5Q*R5!C&?RX13&Y1*&A!DQGNTU:X9*<3@K
M9P,JM4R5YK",6L._LDH'IDHSU[=OIB&L#M#PBI6.3)7`Z:\6P@[+6CHV5?)X
MM/:#&[#J2F5+>F)F*N:1[\.T8\9_;:JT6L8[9?;&5,FR5V*GH(WJ"9>S.-Q5
MRZ@?:V'O;LNH(,JA+BT/U$-9UD*M0SRG6JRW$(Z:@HQM'9EKK6P;EY=E%M2J
M50TLPC3I^$"1^+%\M9?&CB6!5UEC+[7&*OI8F@+%GASAM6VUO70<_CL_^/ZM
MB'-HTGT1&M38SR!FT?(`S-CO#%]&=G(54M@4]R9Q5^/8MM6N&'TPV3F%F:Z#
M,Y+V:338\'VQ>A?=HD96$%_S[F`&83)4O^8>A"PN(T_**D6L@:^2C[NP',*R
M+>]YA.&##W$N5PE89H(<<1F\A.60G"`#1<C"A1]$Z*.*<+W+3J;#1K@YP<E>
M.G&`DRO4,S(GLP*[T.PFH7D^V+.]8#(5;\2*+@-Q"R'7-1FZ$:M%6*N$3@LU
MBX!O+2<YC%S.W`$!"@]F;>'@85Y)7T3K^WB>%].).[IZ2&BJBQ17NE2M%(\R
MG^5X1SK>FFA+`;MW,-F'XI:WY7`P`AY+S2:DS)*T%#-]Z@<`6<*CY6*O-QC;
MEZ@DO9A4VND3XC$$-30";G.!?D^=@@G_()<+T`B$(I3/C1B>]Q=.D<LK+XQ7
MN/ND'*Z1RU?$9:R1SF//+C'%#Q=9+V-$?$V(2Q'"HOO#!42,PHT#XQC&8]RP
M"DDPC6AOE&%+4C68#5C#!W!F$=:/ROOU&L,$;CLMIH#??_^]AFB$FQ$<$!KX
MZ5@>+']Q<.P`L'4/:E-.G,:%T:D,T11W=L_11QDNODNL%\!P_-/"MQ0L<<5@
M:0BU-0\C'H[_W7C2%\NQ_P!`VAWF.WR4(]WQ=BNZS#)HK#K"VB4Z1_IC2=<M
M`1I'PG4U+1B!#C0=[(4U&F*E,D6,^F?W3F*.],6!Z^P826<7[?N!I!,&TAU`
M;^-PP^X)(ASI?-6,236*0!.QY'X<[0:2'A+'84)>!!K.YRX^V`GT2K=\7]4H
M-?R=88WS6DWS$9-3@-D\HP@W],K]JO,F/QBMI(H)L+,`?\'=<`=?5A)^H`;M
MA-[E\\@$.%Y9]CU"FZ7Q#-%B:H*.,7Z:-W3L'=[P$D9Z]TZ$T:X.2O>%3B&=
M?CG5*?.N/Z4+7R,>U[PKK9$=?^VQ/SCSKM0))A7HFD1/TUH0K\JC-6>>B%ZD
MM(EYF!#/<>\TW*5,WM2M8R[IN?=KS&,3I`I>)K[?AU"1FR&E3\2T%!Y@<C#,
M8$D@\P>E","%MN>H<7,8C;?<(Q6]&%D\2,?J*J$U(5W2F:2=\CM,Y"?/+]'&
M-`Y=\]I(FT[,<-+7+HF0V,.5KQ>98C08=TL+.9>P1KCC6@8MK;(SYL/SLV:X
MD\Q*9HE[$08@!_7C[>RF\KOH-6)%;_#@MQD@,]+K+9)_6P(D9QSR0Q4SBG2Z
MF"!5CE>$)#"5A`(U8++>W3PU^N#T5F`)NO3`EMPG(4(6P*1%R8'MH88B='M&
M^BA!E6Y8;2)@(EU2FSA\"^N;M7"B10F4=,&8IYHEE-#U&$*'F=SJX8X9&*;%
M<.5[#B;JS-"%V#)(:QB"\=`'US)TR[HL/;,(?7L1^)X/_&(,@`+D820G(B'/
M-:RL,'I:UL"`KTL:F#^X@7D,>,91'IW[8-.[C';>W!KMV@K5T(1*IJ6.7)CO
M@&O5I'<,%Z0QC\,P0+V%G'N?=$HA2;3_)QY3T/?_X6=?S+[P_O_!0;/5*NS_
M-X\?[W_^Y?O_;SYU__]QSS^WY_]%=ZD1YG&C^O_)1O5#-Z>KVCT(\O1HRP&T
M-0NL8)/;?4XF/OC9[[W5-Y\+)<E\J@[\/Y/)H^T8%\EL'C#<7.8NMFO8D`XW
M8<,&H-S;Q/"QZ4[CSG>4;0N%)Z)<3=R;IX"$)-SHG+\%K67K%TC.!.5Q05#W
M$(YBK^_[JY(+F)/RMZ7!S_=\8[ZY6?)BU@QE^]82KB6/;EY8>,J4"*;3M]T?
M>X/I6;?3'ZO#QPOY!I%DNJ\F?S"Z@(D90%29]U*I"^^,)+L">/9([LA+$_*5
MDZ6CHB_IQ(.\Q-#%.PS[V_>,X)\]=0>"\<BF4U`X.%2&/"2G3;MI,$)>RL,1
M>"@)'2^L-6X%=+)]V9,.'FQJB2XO&5+U;V?!=]M^!%PM$L,,FS"'Q%&(0#B<
M5.]25RW[H%V;V&>&2QGT,+V`L<^^9OH=D3H[3S+^'N4-XC`&+XO'/<#W9MM"
M[Y^(;\$WU/_D`*ZKII"7C%_7H0]#HAMQZ6!L\+O)83`BL[R-SCC4\'+=W)?'
MLK[YAMFN!7I<"%!%8"\VM-VX#J6#BF=4*H^"<1@^NGY1]RIG(:FA@\DZ`(9Z
MX"^50X%HVX5F!,I_G\8]O[/H:B^(0&<I-0[4IY4VCXT7#&F]$+B^YA9:BI=.
MG3FZGW_^6;,HX:4VGIRYV<I,5^/S$".)-8?YT`HS6H<BLM2`ID#=6N5P<TDL
M`MKVT18C*>:D%YJ9*4'*;N3T!>,!U%#7S+9>S=JMW.D&UF>D,(T/M<29H2K(
M.E3W<C93K]*,P/ZT_QK5XIN"=.^$:4J::/(/1WPN+\&!()K>(]1?W:A>KH>W
M2.31?&L&35C01/)XNZ\&PI_1(EI>F_,U++:B8,]25_<0B&Y.@*#PO7KR%*;\
M<X5&KRY;H,X\C6=:C@J\&^"Z%.9N@9([7025J9.62'U[_IHM<%7+UXE*22'I
M[6-?.-6]%Z5BJ.U5*^J56I6*Y'F_FKU55JFD;=9.R\2KOQGP$\6+V?)K3F,+
M1!)Q=7PS;>+S2%J-5$+:*GB]\$.NV$#Q2MZR-;57DT%TIW_#&QO[Z?(!!E8R
M$U\DKU?`@ZOP/-(;!5<A#[Z86DG/%K2][;F!)/Q3--@DA/4.%OF>7H)<U1GK
MS?6',B4"_M'%[#(B@;35RQ71)%[4R+5[!1;2I17>(Z4,<;(4TI%/F8]G$M<B
M!,9*(?*UR'A+K3=O9:76F_K?2D7]"<]C@???<V\BW-<0\EQ*<_]3/1N-'GF5
MERYSA=`A]%WJJEYNV$PH1THE=$M5[KLD1[4Q`!5HH1@WQ5YJZ]))@\75"S>_
M?D)]RM!F@XL#"V"1!0D,F')R2"<!.;G36S92JQ,07N,IHI!])9>L*8]?U65P
M+N?<#01ZFR49*0)`>#/'D^C/875IH670@M>B:&9[/UKJ7QXPK4HY:==SC3)+
M?%+J6G2:G$!EGD^9W9\AT/-TNJ3HT?%Q'TL)&+T7[8W1Q5`Y2ZO\LQKY*.N<
M`L`I2D^F9)GV2J[O53M(G7$3,]^/L(\KFATO,4J@N[AX89GN[:;.0NDJZV12
M>9')Z2R);8B@JLCW((++RQSH53O[MN]P>6TM4UK5=;*7N7&=X1Q&HTR]ZDI_
M4='XJZ53O*;O?"30H4L,^3F('H8JOR'MP,I.$TI)<[KE@/$H'48C->ET%('A
MC"LS[2%Z<SE7J#FL:"=)XX69R4]S24D$G&V)HE\5LNF-PG@,Q*WE&AHO!@VJ
M$17=X]X3+!PLEJXI9=:*KEKH@I?FF(P.%RP9O`=%W;<RE'R8"4G+Q3-+LB^:
M)=3JTE;_M[UO76\;.1+-7^HI$,W&)C74U?)E[)GY0E'4F!M)U(KT3&9S\O&#
M2(C"FB08`+2LS,X#G=<X3W:ZJOJ.!@A*E"PG8C(R"?2UNJJZNKHN+KK&/J/H
MHS>?Z8BK-VUCI`R"*C8D!Q0(.YW2XUH>)K%M1D=9&Q\7H>,AV[+BZ,;"1_XT
M#PG)$4?*W.Z!&4N`K=PL!0;<8','S*;MA%.%]U1F[C\%J8%D.0PY@XSYR&?0
M1Y;2&DYHPB%M&(Q#N*XQ"8PUKG7.3Y=X&AH.2>O)1,SY%%!QYG&35_W0E!I<
MF@-<5'"M3PYUVKALP8.!7;+PW.7,UEE;O"(K6N431B`%.ZZO.,O#+BUN\R39
MP7+R]0TO236B=2ZL/H7L[.!=QEIOR5,-G/&3:!`J>K5727KF\($2Z@Q8?=IA
M#L-D!B<S$=B==<;/"M2>')NT4;T]8JEU6(A/&M-;*USTV^!0_IQM1LD+LND/
MQ#G2L4D:2QW<)_8(G1'7D,"@LFR`S/=3[K6X!*;@QHXBH\)#@X_,IT@ZIW1G
```

```
M-*RS,RS2U#^#.#+WY&3$9IUS1A.89!^K^<Z!VS))J8YUE:CCV$SRU[4J#9+P
MR,5.7!LTE[I\(4.CDR:9;;8;[%?=J`CUU$!=J$6;XSG-T,&4W**@.A`D<VXH
MZY($%9^XU"]?@)G!,3N#=VU:1K)AD$+_#1Y_L*X_)LLMT2Z,"H9!]N>@MH#;
M.-&;"#2!):24<'&C26["SP/$JP_\5A';DB#2&G55`UP)ILE<TQM3X`9$66Z-
M`0W&3!9-A$2L'Y$D`J)"I,>5T@+W:`,`?21A)2.=F5BE:EX"$I!.<Q./U$PL
M'GR2DJB`T&I/,QD,JWI"O2!F73</,6I4]Z)6H+LWTLRN*56#1A2&UC:S2^-3
MP#W!&-EA8L@H8`9+',T3=F#0I1YIT)%EL^.Q4[DL]>5Y>F63@U)X&GDUQ%!_
M0O=]3*Z:@KIJBG92@.!"2N,#]R\Q@$V*BKL+B,<F^P&RPS/.#;>.0A]:.*;#
M<@O]&>>]5]$U*SF]44R1&")>6D';G*'B;07OV[B&,$B!%^C)<[]^\..*]+L@
MI+&ZU34]D%F%=^7BE*(:2#E9R8VT?'=&B9\8C21R'V2+PYH$Z1KO/<?F<<6A
M,UFJVQ+0AFFM!MC0TK*P9E)3$VYQG'*R=LDS1;3B*G`UA2QT3371/.&@AIO=
MU,>N1,@#`SO)>C86=VZB.S4`C:1"Q`)4H$_1=K\,L(TRV"0$*T2A&5"`;S9L
M#>?@(F7?`>)XP"1D.EK%6@FH.U>+=,;0(WND1LK6L&(M8O.HBT,ZQ[L/:AKL
MF^ZVHI7\Y1Q&?#F1RZ17J+1-,LA49U!,I2B#)U""ED]':SEJZ&S!JE4$I'NR
M&>A5;X0&Z6#D_$Z#R'P++AQ\CVPJ%2:AI35=UH13?L6`&DI]<V8K65G3H;V6
M`;<>AY`W7UMK_-QH'S<.CEL0+QFR?_Q5IF'9W>GO]QNGA_WC1J]UGJ'/O*7M
MSF=!C`MX%,4+UCF190UZ&C&DGA(2/`3Y:J/0"5HM/?[$&CTGY1M(LVJ$2/'*
MF[9R-5+4/0I<T2"X!+[DHTMV":MZ60F.^T"?@SEL7FVXE(-#0KP8B2ZPAA?*
M*DXTJMPO#GV<1$-QS`TU#N0G[C$^$N2Z+PS*6\75XY'J%7U.DEYD8<I9$.,+
MV$8ZW<,;UF$X:/I)JGDGD`'E/8H-*::V#+@:39J@@!?ED$;$5B5)28?""=P*
M0=S<Y+L[')#QJA@<0(9)W6HTY%S,3X7U2ZQ9]+@$D<:TC!SBQ,ZZ-39#\F@;
MX^)2*1L>HER=*Q.D>9@2FRWXX3VA!B1YD.!B5SP/=,2LJWMT[](?)US,@;P"
M@:^+.`IC\H0<ND_-$75R\+"=M/XQ]\<9-&Q>!:#.2J\C$R`X&[!3%1I6"1+%
M1`S#)8?"Y-+5+!A).<^3QF6U<80$=$3]!5VX3FXYM-+(Y4\[HA`AM*.DPJ4+
M"*/CF*0+G8H/FK?'&;FZ!7*Q?"BG5WBX^0L.B<Y&E"#4?2'`AT[',;9,D.-S
M^>..;GS'FAF&_F@:L=/C(#$9%=+7/,9(#':765LZ/]4?HMEA>5PHQRTRG2;F
MN')6$W2\QK:%"RM43VM%ZY!S4%VP5[TJL5>QWB!$\N(UIX.)-DNA1A9+,XLC
MU/A\`4200^&7,MI0,FMU'\@@L``M75>+"M;BW"LBK(SP/9*VP(L;[9(*$+N&
MIU]E-SN+0PPIB9EY.EUO=V?KU1,CN2WV+,8<,LFY%X6V,.RO6S;<18IN4><T
M^%S$B*:0A$-MI[ZPK55&O;G($I=K!@^U!.A+..ZB8T$<3&!Q.+>1IK5@-D?N
MIVY#8(DB\@EA@A-7K#(6MN"(`W+'$(*$''-Y?`$!6K2RZ'!&ZOP0L?$RG(;)
ME<`X#9_6S*6KVG;(63-DQZ)#_O/LJK-GB;$T'MXB<\$;72BF2MXW95%_ZEHE
M7%M^4A%SQ/PN(&.F/I@.?$*O1G-D))3BA:T!]:+AE,4$_1+6Z/1VD&PG/\-$
MW7*_!10*H*.CKP;#;C11*)D0!"[`1`PB'W"7ZU#9?X,,C._$_0HZ`<Q)AB=G
M%@XWN"+X!"=_DSH'_&ABD9<V1A2N%<[B^G'3<1,'):7CW3J`7@Y8M,R-ZF)<
M7?(3U1<6S)5]"!5P%[+NH7R_B+8MF3_+W77YWUKE12AR+\Q=.G`5^.=(%D\`
MT?"4UT8K%')JX$\LC#WF-H2^1[XW:'VC3/VEHY0/U_)D9N+3-T"PH7+/<V\*
M7#LVPUNL&._QTZL(K]8O0[I,M#M+A`2!5UER3.)0"V["Z(E6E^7QP97_*<"@
M<JR_,)8U>=M,"J*>D45%D`Z<$4:$@W",0'@S@[WR39!JXZB38;9&X+YA9V`M
M$-V$"W0"DP/966,X%$NCW^@CR3`,C80E*A[;0QYC!:-K3(?"72-C'4E&L+(/
M;NXJUTMO1@ASJ5Y`+2C*?)_@BW'O:_A9"?>0BP#>@CDJ.#'/'HWIJQ@!GQR8
M#>HSY'X6\,H%H3I,D92\T330+F+APF]\[=\D:)@_G[A,':ML3359&\;9"T=X
M>"%G0M+T7Z-&+9G`6@*+AVO>*RX7ZVN!$0)X,'HI1(27>BO\GAZB,`63&?)_
MU.C$9$)C-8F8R'5$@'&H_4)_LBPN@#OH)5XCH@?J`+@L5;),.$J&,_8M#3/
M.`C(@[=PEE>5YG@(:C7]\4'WT'X$4>J.PG@"3O)G?GHEWEF\'VR'PCA))2X2
MEQKJ_!WP-J'(M$0/LNP\20MTYJPG+H1)CYFU`LYJV4$KQ*^O571/9K#\DT!V
MR1BY/20YS+N`=2<:[W[BW$^<^XES/W'N+\NY:6FE&5-CZA+:':R[;IU+BL[2
MF4.[/%;?R5HFGS'?AO?;KL1,ZI=00:U13O>,_>ALIZAK^TY+9SG<O[-@='JS
MU_['@#<*:K!9"L<@25C9J1A=>37/.VR=G;>:C5[KL-_H]<[;!Q]ZK?R]+X?%
MWL?^5R]BRR+>6H8]:\VN<@/EK=Z0G3N$)L*@!9A0$+FPB&!AC%)<'B?""%KZ
MN^F4@9L1N0CSNWX=XL\-2,FM11<4/&.WH[@=J'`0-_](BX!J!%,,;Z3"1&FQ
M5K[\YD:N%[BY-7*L^/6S+VPX<31.R.Y5V*(+90U>0DK&X\>PKYAQ"8`#Q:!I
M^,7I@1\FRO>%U)`F.<FK3F'-:RZ_Y(E4!8SMK^)H/KJ"?606#NB$#Z;'\Q1-
MY)7]:*(-R<DQG8-ELA%,<LOK!L$2[G<:Y[$=3A"OE3LGH<D:A)>I,$D"XX<E
M5P;^'4IG(;*&GIIQ311/V)+-',%)X4YMD2EA-&72701;_@SOL651;@.Z15H]
MG;;X#4H<:+Q*8$PH?%1#+9:"&/,)G6=*#Q=LKT&8AN56]`/,)$ZLB_%9'%T$
M7-TGU'`&J+#OA^GYX0';PSLKH):<&9*I?&:*:#2?RLI/`NN_A\`JG>L:ZJM2
MJFLZ\@R_!,Z?V(YD3?2Y4PLMT^#J-^WD%#=/\EGI_8G/-LLGKZ0A^3Z+&RZ[
M$&U]E@A.%PR(*,JS,Z%[:^,Z89-JBKM`(1/IMDUUY]CX9H1TX'J/L>O5T$AA
MQ(_^*SD1Y(BK"QP_<X3T,E*Z)J8+Y[_<*$82E4Q'01$6IID-:L1.(I;\7BBI
MB\`S!9+Z.6?.&4=FE-?T``W\FM@.;*3?.F:D9K>+*MD<A%8`",XBV6,>*LW[
MQ/B+$6OH[5OABM<X:Q/=L$IDM*9)T(P:TQ`'#*$`T0$6:8CW;#A#"W]E%',2
M0S+.A#]Z*#DU1T(M+=()4"POSO'T-0)O9#H;<Z&4-WCQ"JDM'1)YR$8M4<6H
MKY"-K?\%I/^@KF&98QHWW6GV-)]@WQF?"8Z%6@L#?YZXA/15[B5UP^T=P0\X
M:@=HTK*O2ZC>XS:4,<%Y@$UH1;S;Q<!*\6X5"]T=5,MB[5Y%IQD'Z[9#>)5A
MW=H8I,//1J4T_\;=X@R3"J0W/;`0LUCW":E.7'R7T+#TE8*!XHEYPE?*$4=/
MABLZ&H'073UWB2@1LDYY..HJF[L)[?\28JVXU.]EH+HZ=_!<3*NZ"4C2CWAK
M23]>5OB1QAL8#XNF5'RU!GM%-\T&PM*-X"Y$F9QHBQJ"GX":CM4//X5<*-"P
M#[9['SSS>2H'KG/AT4[Y"5!."JW?^/F2\SW6&CLRCV(TN37EH[K<P[KBJ,L$
```

```
MDEBR57:>W/*:I`QDS:B"UEY'-U=22+)B+_)(P/%SO2MJW8P12_=OEOYDXL<?
MT9>5;9-<;R`CR?H7$28_5)!C4X1[1*[9`[O/6%2`J7*QK72<2F/&A)5F<#CY
MOF<L><9`=%5W!Q+SJF8T1F@]&X^1(;0:4P%*_^*'Z7_-PXQY'SQ'WN.;REX#
M+!>!'A]%0^L#R.5G>*<SH(3CS`I/(T*M.KJ0.&=;9L%@K):L!,\@M9K7E4IN
M4.-_#B?S">K%8?Q0R%P>&5EIJJT,J#$PCHF@Q?"?1&*S&"*_HR,^)):K9Y+`
M(5'::6X+%UJNAV.5]76F4.JL%U#BX]%+SMBUVFS#N17S8@<`[6[YB7U]<?:5
M74M3PE%'"MY3'$72CEZ0SVSL3P,^`28/COD=DEW=N?J/Y9+_@?BO!>RB6]/2
MK)>UN9CM6M!_?E?.:[2&M&DS7VNJCVR]OSJ&KJ^RA3:>9R#.DKR<+R+(Z1;V
M-.3%,R1^0-4+YJ[!=`<\A%JQ/-I`K.#7+3RL'UR5B':-UA@20LH;EZ8%!E>M
MB6PE\I`BN"9YD1HX^?;M-+@&IZHF#R9H1/TA3_8YN',^8Q5GE#DCI(!AV3P&
M%/T?`W^K^-UU`\\T5BW2:/B&'XHKL>!21T9,?6N!ORY33$BRR<#.82<"6)=G
MR`&Z+,ME^QH.GST(J=03,9DQP!)?1:%H4B,SJJ>DF),Z'2VTM9X,PT2K"QD4
MK.Z1BF_&`(E:',BG42HL.V\1+R>T+!V+=$-:Y$L9_:_=:5)U2#&!`/+:ND>-
MU`6)4+!\8,TQD_T9W\K$P,5($L@[1`Q";OWG1M]"&0^)HTB(A\42O$(MIAYM
M/27]O@$FMN\("!:PC7/"@3.X2BUB'VSN:)B(214Q;V"@JT!OPSJ@)>3`J&P>
MXU4C=H(;WG`(ZPK/)A$$X^1QH&7H`AP')VD(X.//TP@T*@,D)]!;#0+2@<B8
MS4<JF4G=ZS:[;6Q$]9^PEO"6A]>VIHC48W$'H:/7TE]F^8SP6'DX7J.`3"!]
M&#XS(\43:'%G(B(KP]/EZ7\I0HLU#"ZD,QW52QV:*V(:]^^?H[%/M\.ESI!L
M=S%X5KBQ(P9R'J>S>M,1RB!>K3$C2G9F6]=]O&`@VIV#UH815M#FT.;LG'M`
M+S-HT^A:&Q"R=G%7I1B[;?!@7"+8A@KN)#PK4-S0AF)RZD(VS?<M1GP0:=IB
MT,-A@B1NA2)?N+K+U+O].MPM^*<^[UL`+"\&YP0/VLO#C&J6`9O,`3.&Z#R&
M$8[8R\!]`+)-#L+40GXH-:/H0Q;+_U(K8(2[7&H)V(;B]A;4]4L:C[*G!49"
M_$#A<A<_"J?#Q-!CB&L>G?&X&KDK0,T]N"-DS[HQ&-Z4Q0>-X5@QKN^7`646
M)6=%Z_;.N"%N4HK6NAND]BVGM>A=<-V5*5A$&$HN+,GDF@NBTZO4-R+7(+_Q
M`PM5C'RF1Y@.X+X/#=%HQQ&;#EO6XGZ]5>()'I!RC:ZY]`+G)-B$=%%)>CS!
M1B6E2C3W,#J8"36,`"YA%=K238=.D-J1R-$>TDC:R.5.A"IE,5*EL78@['"4
M;@EO]]TAUU>"NPXDRT=B^VRD*UQFN2X(13A^XL].&`./;S)7VS.&$/$0$_%"
MIACV`PWUH2RL:*3!"`SDF5"\'&I+W<W$G\WH_M@`.[9(7HR$)BC^2%Q2YW2U
M<D;`E6O05R>HO;M4?23<]%VB*&1C&(`&@2T4#Z$^]@<8\QFK:&:TO$;6%7)E
M9$7@1;.57WA246NXG.IA<,N2EPY+@QJN;A(,EY._W@(6MI%3)!4PJ'LQDD/@
M$\@P*@B7+SC9-F![/`"#2'$17B+"R@39V`*O?B7U<`5:'1]$*[Q*P8`+V^S\
MJF''ICAEHZIVUIC>7*/=+AV(A#%D$G`E&E<M\WUL)B(;^[P'JANF*M^H-C%#
M5:3"*T!T-W@+(<B)E6'"O7`ZFZ>T/*)U?IDAFA/(`X(4Q:-$$QC2$`5V-,/+
M+E@:HS<O?%FF+:VQDFSM&P;4/_;[QV>O]OM][W__5Z!;E;&8SNEIJ]GKGS3.
M3EHGG?-?,1Q4+9<52EY4@@$J2K%T20(K(3_?I,_A">D@)'+0&P`-/"9HU<&D
MJ8.(<!"F2<733\G?!&,F=)<9M:'3JVCC]O0<@MK(/7WH^M@I!T#^#.1[/@]M
M(IXQ$WLJD./:8^?^Y=8,EWKA'O)J_VD7>=I%GG:1IUUDV5W$?.OB+%\#=\WA
MCQ^FDQPI6ZIN!%@Q?A4_)]LP6(XQQG;3+L9X3UR'3,.1YNE\R`ZLX115]8H%
MA=/\.=Z.-?`I/PAK*"`SXJU<+?;%A"D-Z>XB3H'!@TN<4I!8+"&M?BA9\C7'
M<S<Y9S$99P2=I0CYU?X3*?_+DW+.CF8@D4T-2V]I&DEX"VDB7\O9/.*VYV$F
MCA)H.)M'PMZ?23(7J,J=R?)HM9@%\G+Z3:,'M@9:\^9]!/H6H%I0E<`X"Z;T
MFBH9%418\BT8ZX';67N7X6C.G?J9"`46(SP=#L<QO4UT[;U)LZZO=Z0T;1H-
M$PJ;:M2^8>6/0TPY3$'6#,#O8B!$<]6"R#+)I).0XG1(YQ2D$X#;?#ID'"2-
M(DWHWP3P)63;JJ)6O34&46>_&G'LXY<N"+LBH05\._13'_X]G4\N()9I\^B`
MG!!4\`N95D/W?>=)0@81ZS29112,H=/5YAZD@RT9GJ$D/5I9=P5F+-+*ZD21
MOVTUCX`PP?E)6\KB.P79\(WS-B&?VFY616NRO6+BNI%Q4;Y>RKIQYNR0$[22
MRCCJ9^GRB:YRZ$K/Z5J"KO*%0>7<))/D5/3U=-'=S;T8M&Q[S6AR@;9%($5Z
MS[SV\=F+/:_U.05#*'!15@9Q;]\&['',I*,3)+I<*(`/(Q6QC685)=8K%=9[
M>ZI)QHR8P(P]BM4[\J*$$DS$98=85`*PU]FZ^*8Y3>VZZ#2)'ZK<14<]54J<
M@#WM+3:C]4!]1_.4.F=O.O-4ZYR_TGK?Q@*4E)>_U3O&M^KLK;UG;=2T)HI3
M#[PT4@\4+`;FAEW-BAB5(308_C#K$KQ4B#!'P_*E8\GNNMR5?^.5IK9727QW
M@.CCPO]5@N8!J.!?`>Y==H3WXR^_%2Q'VK7*?6'@"L'Q:/GP`\%ZQ9)0SM+U
M8G^V4\TY&W$I3FDR0B8R?48YK:"YW>6:,U^0Y<ML=U$G>ZOHQ/ET;T'/+^ZO
M9^?3%PO&L__0XW$^W5\PRI>/8Y3.IR\7C/W58QZ[\^FK>_$-R+,!I\.3I0MI
M8S`1T^P5W'49IV=/AK=6@H2B7:[$6-@\CX&;>#$[7K,*8&RY4D4%=R,2J1]7
M;`9]&^677)-":UKM,<W@'AU*1/AYS\>[Y2A.W`XEHAQD'XNBE">"<25,]J4M
M<\2S"_(>P)<\!Z4X%G&/>?2W%>.1S6B>Z+))<<L!(DDXY=&*L237AW&E#2:]
ME;5DA$">`DMD+O+)M;W.]6\\WI:N,<=$L4$RB,,+ND+1,MQ!51YK2J6J,Z(,
M7T8\_^BC<8Z62&XNF965@"`JPJ'6Y=U-@?&Y:8.#?L],L'K?/^N<]_J8Z!G@
MP6`WCP.9=4PL4#^`7I%N7&B7ZT7OU`X;,SEB<(2H1(Z0WHF>D`CT=CAS-HT9
M*[\HMA1;2<C6#$43`7T9(IPW%@J?0"BEC/A'<#D'$'S^/056`&WJCV^?J\3L
MJ-6\\F&D<"=)V>H`7^=3[%<+&DN#E@%C913JNKS60-WJD$+-JM19-%Y`*9@`
MNC_/8V2:PVB`@=W$7:0!SJVKQ^7ECY!MR%S1^M*5Q'2V$N!@[H/3ZGRJ.;%!
M4U\$\PV\M5`_)^X^S1[$#8`'$L6VM*8A)B1:MQ..U(LSDF1>=P,_'EQIBN?:
M6C"=3[S?$.H?56EJ(3@/!O,X"3^Q_;[R@[?S>8<^N_6<\F>0506L@;3">VN_
M%]"Y.6#;]Y';-&KQ:X&W`3&DRVY6)#\DY#EI#AN"SV8SC$/"NR`%7$:\$0L"
MHA--EE%Y.!ZR"6?V.63`B!LR,FB,D`PL1V8T_.3<AE].W%BQ;-G@\%Z)-C\>
M/DX@LQ8EUYY2"ZZ6B.S)M5E,E?M1?0R"&>1*1V.U2SD(E6T'61(?]I!&.HN#
M3V$T3\9X8>:CQIP1_.RQA8LASJS?=0&S%7DSY#)A,=:MY.-)03J"NA>,MF!T
```

```
M7/;$6DX/Z4(R$I(.)K`L1@MP=:(=PD]XYD\IPW+X&:&1"1EXHH.,]&2Z/:GT
MB<J`T0P1G>&>[G2DL'G)-`"(DV%Z-W>@'%ZW1`83$@LM0W:^.HXPQH6Y/`NX
MZS+,BIW`V/)/+7GE"W*L&(0@C)Z"I[U_;XY%5"8/(_03T7^4R>[K/[&:Q:PF
M5S*PQB3PDD0VB8]J/'5@.X+=(_L1&*RRV.9FAGVTK,V5/`[YDBU85@@(COCO
M3AZ7R^2X:BO#Z^Y51Y&7>5SQSR+-198K9/07A,_*>88GE2=$#IW(F:\FXTQW
MXG_DL0&-YNI>R-94^OU3%[I&R220NM*!0*AZ#`!.,:L8!(@0[^PVFP^HY;)5
M9]KY'#KU1?`<5(G9>=0Q1W,4RU!3.N2*P<T.;5'>%J<F5-R!>?06F2%PF(DQ
M3B![QIK3;'9GW:LDQ531>2$%[7!PX`3C@@D>2F%VH*"2E4ZC0XHZ1.GL8\&L
MQ6A]'/AE-JE\21002_=%DE$;W,TF>CIT%Y,\5OPI2,&R*2>HA*^T!;C?@O_Z
M:)I1&>9@WKFEX!&IQZ&I(5$J)FP8S,=^++2+#&BC<70!NZ6=)UT/QHE=/D]X
M66R2Q!CM@=C)I#&OJ,2HP>>YR;!<>"DV((I=`QE/?#MFB":UF%HP$6$#0QMC
M>^K\9(;G%R:!TN_P8@[FY'H,AY2+-BO<^?@2+]CRM(CXP@P.Q[M02N?-MZ<H
M-ZT`D>J$&B*#WI!+<U\$O0@]B#M8`U)-%2,156=",Q=*'SF2?AD)^\N2!L?=
MY81"3PJ%=R&=XXCB;Y0FGS&O<'\D1$=G7_7$>RY#43Q:D]*T8_8YK:D@S2,G
MI!&[3Q%ZS"DY*$+*#%63)V!,;///*2EO`]!S5D:RS0HD7S=]R6F7H#$+1*LE
M-PO?5TAR<MAER,YQS2442G1/0G&B%^N/@OSB=*$T"XE$TRS@M0LO<6UJ[CY^
MFK*C^82'['41'P^4/D\PY)XD*:UE5*HDVE6-Q"@VZ/$XNJ8#%1@\C(7508HI
MV;2WS[>?,P(!5*(L6Z!9BB8S)FV#"@$FCSJ$2%$_,).+(+T.>/A")/R)V)T6
M6,'?W1^'/JDK,&H\$H2\P-8T939#P`.!N&:^!:GB7!X5J?)K.KC3%`3*T`:0
M.R=X`Z=,-Y(3P.!&3<0/D+B)*?@0PY3CAN+I,C(LCH`79Y@W@A9$(EMN!B#O
M`,OQ`'Y7MPSABX]&__(>3[O(6T#UQN_V8=YFRZH>2MRSD$=H?OEV`GLO@R\X
M1R9N]D#56(.F^F+!T(0L.^?V&3R?\TV6R?B#.`+_*CD*C/1J%.%*0'G?#2,`
MY4+`^X+\C\#\R,K#E>&97^5:@Q01;JNUG#C1X'TDFY/)E(@YL7X9&.F"'?77
M%$M?LU=AT[A@7".Y!56SQG-H6LR96@#"P:S?AZO=Y2Q`+4!V89D+.<+$^$I>
MCA1XT,K]S,R@)1WF?&5\9H+R>:(YZFEL-3]WF+HY(3,8]O\@FB=>,O5GR159
M)BWLI:XCH>%XZD_]<32:!Z;_N0C;`SE&O>!3$-_(F)72[,B\M4&ACL^>>P`:
M/G;O-/<XW4NMSGYQY[A.%YWCV#\WDXMHC-^XFQRK@6YRG:YPD^MTN9L<&:EP
M[3C=XH0Q=3U'ET2VKUN!$DON7@8$B2//;HH\?36P@G@BQB23N0GG==+-:V%5
M-,E1F.>(9,9(\<TC,[(F]W:GG8L1#U1G,U8/V&`QPB'(&G+E4RHEE\0=%/LT
MDE]-IT8SHN/JK@VR/KFYQ-P\.IGC:ICYT3:TA=`=#1L"'IA"5D(G[ZIA:;YP
M4\`5EF(%]\($9.L6\6M^KT_$OYCX'63_,2`/8AJ@G6="5M7T*:LAV)70JX$`
M]\6FD/:E.^\"NK]92/7*8[C"8'^O)&Z;E3V1^%="XB8X\626X%IR_8$"%8-N
MC*"BQ.B(NDIAME"22DW3;;CXXD8*Z56,A[<@1`-690F0,<?U1#AB,`%PO.7)
M.O@1OH,QW,'.'P<<#0C4%*H=K44AK1S-F=O_*FK&BNPH,,\$2H!<&X+-)'`(
M3X()P[MPD%`VUD*:U>U:0LR?\@`6+=;I+-4"2,.:\K#FL.B/0-?PJ':)%5G7
M&(?VJY`UQ&#-<.$7AO&1"&%#?7+#;WXY/@Q$5I1$M01L;NC'PT6XQL[`,(8[
MV^)HZBY(T0C!/,5C88OS"'?+C*'S+<UK5K^->L5J8('AFY!BB"$5QKS$3@&)
M%ZQY.677;4%BQATAA==\D/;#Z05;65:$5'-Z"#MA=H2!,TN($7<+P(4JZ5(6
MEG<)PN7+-/:6B=$6#!&Y2_&8A.Z2B!O8A$P#5>?&=VC!PA.!`Z/FH67SHC+Q
MO.$J5Y\9+(F''BH;+XQ'KRFUG1B"/P<0Z^XIM-=C#D%41'0+CQ4BVI"W*,Y7
M;B_+1_NZ%\)&:5P_LCXVPKZY!5GG12J[-4W?%%#T4TBQ1T[/"_9X5W2Q_/!B
MGAU@K/AJJPG"88ZKA':QI8S\/Y%/9.8H2<<]<3:B:V>W\4B.\X3F/RS3T?)C
MGK@D]DMUZ[9>*45-TFQ=$[1AQH_@R"7!@SNK;3&/RV(8WIMG`EMVEQ6%"8#R
MOA2)`]7Y-U0&]JN][C*0[[:ROVE!FV-!7X((\IWO$TU!0?CFTE<M1'E3@:-I
M,*A-0GOJ15ZO.VRTLAV51^W'9;%`0^IE89N]/\]B,!7.H&^!H<-*L%6SV[X5
MJMJ5$%EI+N7LCV#/7`6_YMJ[!V?8KG[OP+&%"N2K9=AN'<X#\NNRMC<ZXGU!
M7DT#*<>L.:ZMEEOS1E6*/P=JHDN:P<AOQ[157X^7:RL@EV';O/1#\FT-8U;-
MN/EL%N*MVTK[4&1M2H357/:@FH]`.0QZF&FTJ"TZ4-^0=;1Q9"_&SR^*A\)P
MT;89]\5ZF$:)MJ>8;@N><3$CE+J@,V46G6YE?LSQZ/Z<NDX<IG=TA\*0XRH8
ML^-@4IC17#20=R/LLNW#ZR^Q<R=FJG-R/\%:KKSF>"G+U2NRZ4$,^T#H\U@J
MHBE(!H8<`C8VMD]BTUO9H1.?'BPU8BU&&6T2V#B9L;*NA&N<"#%$LP+W?2FO
M,+9E."CE.*=(;YPZJ5L(043-+:^IX,7=$#C3O)R//7VAGJL+5.ZU%^,\Q%!-
M"J&X'CFX@=HNS8%<<6H$P#^#.#*"ZD!C0],*3&B%53PP.5#&=\7*\"<X5OF^
M,1SJ:61U&1*`,Y\8"<)UD@W3W%S-\MK)VD[X)93+P&HM@TA50;]H2;U1R77)
MX?5``[-J^L$^<\GGEI1#OH[.<=^&>+2>$/@B=@9U@\F,90EUN8AA"F!I("C@
M41A/KFD/P';2.`AH*#S]N@8(&`Z/9"#O5CM'Z$[`,`BM%XC`(*7A6%,!NTG2
MA+6;+)]HZ;:TI.-6:7HZZ![>!RUA>44)K!>!;AK#UNA+0UIL5F4*4543LOOP
MT9LN&H2(VO!:^!,[S9_RAXYK10/56;U!5[+YQQ7$:4E;/:ITD0SEI828ET&#
MA"M/%+@\!5I45!202=[_JW@D!J6*57(1J\Z^P2_GGJD67KIV#'<8R2;C_'YH
MD')5(F3=NQA'@X]X713`.DP#=A0%-S-.VS6GXPOW[[/&3!9[TH60'SSR]K:Z
M86=HAS[AJ\R(DJ)/]%@-.BQ)#_#5L!4M"*-@+@M@^U4S&URW7LZR4`3*)\ZS
M`LZ3QQ+NPH+*N>@IE[)[8C\9]:%T_%IL]%R*/$4'$$*6H2,K]TF74:VPJ:X@
M=3`DTW$PS[=X*8="1K[Y;G!6_]R;#P<K'!.>*&@)"LKSD]22<)A^?O<:\5Y$
M;S\"8NY%7+"P*.Q#&HY#,C\1C-P@)-P/X/19P'R5/3<4M`1TEZ/VO]M6/Y:1
MM'70<BNH4ONX#N4$P6P=7+)'HL>SYYO;RBVW\J4.'O6\&`PR!,.=;BC<E%7-
M#4+MY7UL1;,-JX*JFG::`X6QE`I^O'OA+)".F:TU0Q3,G01)E"[\P<<=BZ%(
MY<Z`%U!D@!L:U+V*HRGX#[6W.R*K,QV6L:TIA(%@C3'1<(11UB&BQ`UDR4`^
M/H#]'(D!UPY724<17H1V./PJ#<HX*HB<TGK_9@MRS^-?^>48E`>U%'*0-089
M`1-T#ZENV*"I5?'%!HV#PH0CGZ1FI5R4`]K=E8,V8AS*A&TYT`JCS^@R#2@E
```

```
M[%Q=&4[1E`]^7=R`Q,2:GR:,QI@H]$761?=7B4<[E98QX<7+MKM@V>I4;P,:
M7[2$>RM?PO3Z*R&/!<M@OMY=>I7VEE@E[?ONHA5;^8*-(QYM]^M;M*2"AKR.
MT1LZ^?FDP<IZIY(/V(47+F;)M80%3`KV1"O!E1@9ST;OM:>4+1=]?++[>J<+
M8>&THP3DESAAIQA_%(@S\"09]:\"?QC$/#C-AL<>F;XX(JU69CCL(VUF`!YF
M$<J%)=ZK`XM('TDO-M".FT%55?XTZ?,4D@X_H!7+`8M@V.XT_92\>KO!=`C6
MWE4Y4$T^,M;+)2KI0-,_EV-_I$II,I_^V1`N4HM:DX#*G44/C3&8F.V<AEO,
M*QY[19/<*/S73(8ORQT&P\Q[AZ4-18YP)6&YL1B6)]%P/@Z.(T8^P]O-)!L8
MR@J[F-OW.3O*I:L'WSTE`8\NDF@<I('7.&NOK7W#CM!_Y)K1:K\/B<'[_1H6
MA#WH++IF;/=DE*Y)+DLN@S(>L/SR3E8Z\0>=[I8_F^7;7QV&R2Q*@A;CHKK5
M8"!_HSI#_/!JWF'K[+S5;/1:A_U&KW?>/OC0:SG6Y,2?G023*,[/QZZXZ00+
M]I!95B`L@"`\V/AZ[#>PQTG?'P[93I%@*"4?\LG!+WI%K!$QM!)==MDOO7U8
MPR1_Z!Q8S9^ZKN1U=',=CM,;VOHL16EV.C`$7`>^51;UVXO]88C:I;&1])W$
M:SS`Q]$\!4.QO+QZE-B6DMRVZ9].U<L#NHWN]C:&H=<G39#\*PFUB;EFX4%1
MJ0ZEEC6+;6UML=D7YY?=4?EE^PPN?0U6[5-W1MK\Q)@F+)`\YG'0J=XW-`[8
MD0@?LK5/1+>$A8\,%NV'@\5C@X0$P:H10S:,><&-N>:@!E&+692$7TC7*M.#
MLY+9]]%<?W]/D&2LB3B??T&:8]1?!G[,N!/;5L(1^45>0@`#R:!@!V-`[YR>
MMIJ]_FFG_V+OH'_2ZKWO'&8XJP;VB3_K3YQ;A6NOR-\L\G:+2F7#N5M`+G:Q
M53B##)3<Z[0ASZ<KFPRH:YV3D7-9?GAD?-SGU,O_[9@;FIZD=@U`JK*@4YP"
MK"3C%%`J\KIV<`+II$_:(O$:,I4[*_-<Y;FUM8SE=YZL3O0/-E\SJ,,7F6_[
M"\Y7XV4+FJ!"=YJXG/#MU]HU_,>TUJB(N@49ZR^N_8\!?J=A4Y-Q<-E/#75$
MSJAE$3[OKX\_N&%8"F.^&C!^,50LQ72^5C#>*S<SX;DL+WM(D#XN)OE-,!TR
MN9,)K/FBIRBSX67T*ZB+Z?=;3$8^;#6/C<)_]/KMSE_:/?I[W#[HOX?B?WCZ
M?)6?)!YLH^_:MN;!MN(^($?SJ_U]^'?W]<L=_5_X^G+WQ<L_[+YX\?+UJ[U7
M[.<?=G9?[+W>_8.W\Q``F$.,0L_[@Q^/9D7E%KW_2C_DA=>,9C=Q.+I*O>J@
MYNU^]]V;S3VV-%YC-AL'>.B=IQ"RICT=;'D-<"J&PF#WE@3Q)S1:Q7;^W#@[
M8T?N3O>\==0Z;YTV6_WC=K-UVFWUW[<:A^RTW>TUSGM_AK+P']D*A6,9<"OQ
M.N(VKPE!:AA#W`:5;3245U-XHVB4@I;\1/?&I*P;D!4W#L#!##7!_+J0YG0V
MOQB'`Z\;S6-V6C\.!\$TP89^#BB[WM[6CE>%\L_YR^>U+>_7:(YF5Q"#:8[Y
M>,3X@\^#8`:,'=H`SYYQZ$^%?Q$TPULA$T`.OE'LHXT5&]D-:QG##NF%H2W1
MG989@2P/T<`PF%(4=.&GCQ%O8[8N0]!CAQ=S'@.W#DW-IV/_&MWR8OA.?0S)
M6"CAOJ\$'7XK"O$Q;Y(TF&!?:81S"^/!?/()KN&\3V$T=HU$%N'#H7)\).#T
M!PV!KV]QKUX27:9H)L7'ZOFC.`C@8G5+8-`9F2J201>:9\QNQ/WNL:@%60&]
MJS2=O=W>OKZ^W@);HP17'JX!QL$66[!M?Y:,M_E5M`_A#AC(+R&8E@C'RI=Z
M2R%OD,56S$HAQRU7(1AF5Q>0$]K1"^%EN/>\T?7:W>=@Q!LF=>^7-MNR/_2\
M7QKGYXW3WJ_@/-8X_=7[2_OTL.ZUV-O6.334^BN3!;I=KW/NM4_.CMLM]K9Q
M>N@A47JL4.O@5^^PW6T>-]HG7:]Q?.QU/S3?BW;;K2ZB2?NT>?SAL'WZD^SX
MN'W2[C5Z[<YI'3M6%6`H)ZWSYGOVLW'0/F[W?L4VCMJ]4QC*$1M+PSMC--]N
M?CANG'MG'\[/.MU6W?NO#^U6SVN=_F?GUY/6:0\&?=HYW6R?'IVSKEOP;$M;
MX"0(#-B)7"7<R''@C?WI:.Z/`F\$\2FFZ+Y-5,96!1H:AY,PY1PDLQ9R41=S
M,"85_1D]DG-8YW=NKNEDFQ[GFTURBTPP`C,QLI!&2GZ1;&>69=^WN[W.^:_T
MB\EIH&&<PLV7+I?]I?5KM_]^[1OBB:Y7H,V<Z?$=P4S;-O:$%$:RD8\XCH-Y
M.!YR)LFDA4IE/?-XW:IR&/JC:91`"&.MAO9T?:W"1N/KEM$?@QLR\U;)F_16
MQ44=6I[*5LVGQC#D;2B6A<+RR3K"PO3U)R=_O0'=F9\:X$^,;O!"4BN#OXT2
MEM4LEE+/C**-^3",M,;PMU'B*(R#7]A_JBGQQ"CVH7N@-<-^65-N'W)+-A>(
MN46W#6/^F%JRW>?MY>)OT5?<`AY5/P^('R<Y#<CWO(EU[9$]A&$<0H0:AMP7
M6D3%F^R28EL*'^"7N9BLA:`[8'N`F+QZDL'QX&(^DFV)!_;0I!]!_J!8%Y_"
M(5QYBL&M6P^-KJ7#G.Q;/K'G@AVJLNO60[.X<(*1S<HG1CF1V\YHUGAH-0O6
M[>88U*-UUXH;A8V'1G%\$@P%HL'=$E1:=[[AU9UP#(:B,^V1618?,O81C*+X
M1I0V'EH4CU;BF5KK[E<9=.9&YDSZBQEUX$4X#_8`F\,X\H<H>6GO#812]^<&
M2EF/J5<(CLX$3%9UB%;Q,PR_84P='G%NP.=-3TP0::U#KJ[6=,C$84?7ZN7Z
MPNI-Z2_/H;>@5$Z+W2N&<<,V9G,8!-DQF>_7`2Q#N56)')H\>:%,%RZCOOO)
MQ[R)!&`KXH(!OK"7?:H[]H#^Q^02<'Q)KG23/9VM\M?V7I&DKGJJFE[$15UY
MW4DJL;O#%T6=90E96IT-<^K)`C;$\/(U2-(%H/N)8N>V>6F<A_7,&-#!/+E1
MA:&T_L3<L&<SW/"[3,X,9)WUG!<6?PZC.$QOW/7SWYIDQW;Y%()/D$#!#3@9
MRN%=^7K.^UO"L7ET-IZ/VE.XIY5<Q7BXGBMD)D:8@I"=EO%4A\YP<(B:1=$8
M7"+U[NC5&7L#U]/0(;<!6,^\7%>6:FP`59&^I%9R/!@@09CRDZ%ES$Z7J<4)
M/X>3^>0`O+1P4SEGIT8!:D\-S5ENW2AECK&HCU\@7(K5R7I>N?6E^A`6&:YI
MZ'WHY<QIE._#-0U7']EI+.RC&XP`DW*FLIY;;OV6?62GLIY?;GWI/G*69;VP
MW/HM^S#GLEY<;GU1'^P(J^HVQN%H:C2"_:PO+K=>>A[<`@2$H8,PE9VLERBW
M?BL^<<VV21"71"9CC"S*V`6K31RU#>X*/WCPXS?X0Z&P#^:0T:Q-Z4W!<LAC
M9=89Z]L*/OO`"K?H!"-^KK\3=6'K@;I"##X"S2.K*^J%@VE"I7]_QP=Q?<5X
MN;MB*.)-C@,]NGK,RQ@;"\PDPQ4`M/#"8@,<ELI#N&9NI_9`^")E7XA5H=CA
MM35K5S[PAZ,@0]_BQ7K9`=%>:$]P77^Q7M"2ABJH>,2LSX@>9H)KGN'U>0)(
M%/ICX3N&#IPD5;IV\BX6)KSD8J_KU7K%`M72H_KPH7VX:#!0QAH$/')USL#7
M._WYO'&BAL$.%\8.BJ\/`S#V%@=1WCB^V3QL';=ZK<VS\\Y9Z[SWJWFFHL;/
M_&DX:$\O(Y(_?'\VKL_@V6;('LHSC4A?`D3RF;RHX7T\(<\?':W8.UIWFZ>K
M-^N5"IX)!JGWOYXOO(3@=^)N"H[..ELUWR#T()?0YC28,T%C+!.PN5OCTH^C
M-11]H#7!R*1CH@F(3E?><CC!T.EFE7RL#C[TI(XO]TZ8%(Q/E\)?S?TOJ):B
M_C``W[MX:[+R/HKO?U_LO'CQ*GO_^_KI_O<A/M^$T\%X/@R\[W\&+.BQ$]1%
```



```
MDS(EL!JSF(DGFY#KOK;&^\7"JFYM6\052;;#,^BJT]V:\4?;A\&G8`RW;]NL
MIBKP'U5K$K4M-M2USL%_=EGS5X'/)+%9[-]L19I]-.UYD0<^ZY$I>6U%:P2`
MX<N7.SO]631@,,&!$FAJWN;(VX2('9X?3U[M>YMA<I-@A!4Y%%:F<_$_36_S
M%X@'O=G>6EMC7]ZR`C`L=G"&QEBARVFT"?%6!NFF/PY]5#!M_L(>\OOV_Y/)
M^+LY$6NP^8E"<;`GTQ^^V]H1C3LJ3=AI>6OB;4:L#,[`T>Z1&OQV%^-O;Q^'
M%[$?WVSC#<AU%'],6,5*A8UX.`XO?F#_#;[]EAY=BB+>$2C+Z+K*>H-Q4>V'
M$`_@)!B&ONL%'@KL%_KA)-N%N)YPS-`>R>:8[5(#4B"`^H`!*+X4$*)_MH;=
M7T_DJJU]X[4Z1VO_:OR?L./^^EC`__?W=_9L_K^_]^*)_S](_@>,$L89!(\9
MYM$U#8;?;@S]"3V3<<[_W__U]G9VWWC_'4X8+P[GDYR4$-N8T%<J&#ZT(<L(
M_J6((OSY++V"^VRN9A"A0W!`Q=%%"B*1:$P?HH^X+P.64__+:_J)/YNA)O-2
M#T`?#!7OJ0/@S'=T0%_R3EYI`$K<HB.[<ROQR4\MHT-0]B&_&;QRRV5R0K=U
M1C&W,2#=;KZC\6A:7!KI6V_G\Y]V/J\+_7*-%Z3Q5HT$4+5"M<PSI4`I=U4;
MN,RH;%NQP@M;-=`\3<TSJ941&AE3$:.NX.@&NUAO9$VQS`T7?8+/LW$4IC9>
MRZ4#<Q5_D,[]L1=,,>8]6"%*#.?5#6PU=E+#8)[-%>ZQKL,$`R!R,(.93'42
M):DW#C\&8XC^IB\MQWCHE?=6E4ALE.0`_TOK_+3?_=!LMKI=@H:#),C:PG@9
M<CLTM72*-?0AR4)5PQ^]$WG%L0"?,%$#$]W^B8@$S2Z!1MI88!=*Z3MKZ?:C
M$BUXESX[T0U+#4:C4F!\4+O3.1!5V5)7"8P,FM\S&C\Y:)WW.T=L3.='#3:J
M?J_3[YZ=-WY]YWW[;5C3QF4LC3;94<"G&@S[JDPUU`:KC0DZ_].P;K:&-V>A
M15_2#9*!K5HM&&EMX\U.;7MW9\?[_@<OK,F*)B.D15+4K3/?G<_[N_0_K=_?
MO6"<!,NTHE<VAF_>L[G'E[%>N,ZN7O[BJSY_SV)!E`:B%8UK0;&%.,L6%VQ'
MQ4^M/&^GZN9GR`Q@VU^6$^@##\:#B-^(*MA+#J.>:AT_Z7](UOIB\C^H>S+R
M/_OG2?Y_6/WO2:-]:FA_Q0-SGX8++^M2B1=\TN=^G?0/)ZM[//Z7H/_,^7_O
MY=X3_3_(^9\G80S1E7;,!).K:`S>$>(P@,=N#!(.,20HVTDTF85@ZWX=39^G
M[$AUL_5$^E\Y_5]]0?I_O;.;I?^G_*\/\E&!>2$4_#;\,;1S2IN_K;Y:)D%,
M]#9OC?D#^01^@WR>0O29RTE:][:VMFK>:?<X&E6K?U[_VY\&?_?^E+QE__?6
MH0#$`H.R_?[E?#KH]^L8J+Z/1^X^Q:=';4_=^^:;?O_G1K]Q_E.WWZ_AJ&!0
MIW@HD`-01P39NWRG3E':.SC'%=664WK^[7,^)6LD!<VKRIMYE:5L11-!98E^
MA4X07D&,=:!_^U9L\J#TO[NWE[G_??%Z9_^)_A^4_M=M+"@;VEN/';7AC?J,
M1?C*(=JM>K?ZZ@NC$OXTJY#GQB7H12_U[+8ZW5`7<``VCJ4)8T&\UC/J#:#H-
M!FD?DE'A-U3G.K2-8!I]&0<!MY]?K*N0H1=-3994[M>R;2O#;LAQA<,!?_63
M(+V*AE5M@&L5\``C-5>_]=<>Z[AQW#]I]=YW#OOGK>-6H]LBUQA-^[0K/9>]
M'?=7Z3OC5$*Z76@(IJ@.Y5>Q[4/;4TS"SNU4DV_270Z=Q%5!'C;)&`\:,N5$
M_S&756;SI@MD</D$)U!TV&P,AZ%P^ZSS0ET%@>MP/(8DWM16,'2$A("@2:70
MUIQ?":S=T&'/NI3.F]Q'B'OVS-*X/#:#D@^\S?X&L9)W,&C);SL4W@K>&"W_
M32'G8;L)B:0;Y[]B@"E9K<+=(\T1H3?D#\);P7A';M)I/)C=5-F3NK?^/3C5
M_HA.!?#<3\7SC\'-CW(Q?@F'Z=7WV_#,412B,8R"^$<VI>^WQ8]%3;X/X.YS
MM6VV*"5>Z::7;KDL',HW##F[DK,@YAV4@<<2K6L.G`M;?O/ZY9L7KU[NOBG?
M?#OY:1Q=^./<MM-X'FPO;`8CSD(B'FI'CNC%R[V]EZ_*#Z?)Y-W@)!K:[60;
MV=XN:.8TFI[-XU$`@3EY2Y?@C[5=HNX9XS(PFS-_%"2\L@:$174OX1[`71?=
M!KTN7!_P=-LJY3FPOT(T@/+@;9V[4.@'2OU`-YAGE'>3;;L4R\`^-7I1&+P$
M^D(;.C6K)5VR$2>9W7I(!V$*C:%'[QV;RC*6NS;EA-?N+>%U'ET7M[1FM[4M
ML8,0-F";Z_"^4&EG%;BTU!;SA$TKQ::]%\NA4TOB4B'WVS;XV7:"5K27\-;[
M%KRJQL$4?M2R6Q'EN;1&N?/Y3^/Q9VVD=7YFRTY>'^_=3R'-\U:CU]*#)>WP
MF`;H]:\FPJ:E3B:BK"'Z*>=V34R\S5E%6@-)H.7Z^6^8$C0&2Q+R]4;-%$W9
M#!0<Q+_MPW[GZ*C;Z@E7VJSL3:.<!!.0:3.OZYX%!*=4S/WR"_UBRYVBP$I#
M+6[V%-6Y2/UP"O<&JA1XM^NGJG.PBBYSLKJ,HPD>FN!QP$Y5C<$@@*@[3%J)
M7`<EO*"`A*O:X0?/3L5')W-2YM$)H_I.874I4FN)0U#.T8O'?4&E)>A1^V#9
MD*2PCJ!5Y4$RHW[('D$8>AAH&JOG/,\L&NE14FXR8Q)T>!4E*4Z%E:6&J]@7
M/D^"\665L8-G6I<"YTK1QA&:#LF0+]2,!\TLB((A<XC_%%!F=U:=/TFJVFCJ
MGIDZ'M2YBF`HE62W=?YSN]F">3#0W&W\U(7,WIL[B^LKB,/>[K3YLK23GR'B
M;A4&X#U[!D%RQ6JH0J?!YQ1+U/#L;(.B,V-LC=?C:NS43S[*58(0+0IQ<&3Y
M$Z586*PXFQA$`>7AKLQ(/IR=$UBD/FZ"LGR5#NL;>]ZFM[LC>Y-5J`/&@9(@
MY4_)UJGNR:HU5?=W4UWC6*,?]#7:$/3%QJ/FK`4A#B435.3PNQ7<8^WWIUN[
MU=[_932_#WO_M_MR_W56__]T__=`]_]KE<99&W?3"S\)!W"[/YE/>;(3TG@:
MV_D'QLR:XY#)TQAX)_<"43S.O4%415";NHU_C\,+O%U4]XMJ8\CXISI>.=X9
M^XF>1W3=F-=ZS579+<7*)G9J)>IP_;NLM.OLR5*,:N/<^7S0?./J*"-:>BO\
M(+]G(UV-0/7NRZBT%_:[DAL@PQHNBY2/V3@&^+_F+7,O9F"+[G]?[-OYG_=>
M[3_%?WG@^U_39ZK@ZM=Q4;QX%Y"/R8@C[R6<#O+>I>$DR'T7^[/$\5*T)YDG
ML%9P`V@<'W>:BI-7(=$DP\6:O,\V&,%(NBG`,Q),>3G]C#?J3Y(1_DC^IKP/
M.@?_V6KVE/.!O/+B+4C.,M*<(?0&,NX+ZJXMYRBO&=VCV,]#-0##N@CZT:<@
M1G\$QZ&>9\SCF34"'@^1G:;8S#QV/AI<><E5-!\/O>`3$P'F&.3V(O"@S<MQ
M=#TM<RG/'=U,SEQVS%4M;T[=`/^&)\!?^A93/Z/_(TI8*?:79^+9$O-GOU$A
MM34.IOPRDN?%494!\*SVE&+5G32:[_MGG?->7\2%I\Y5>0'8_N5\/*YFCH6L
M-]7&>?NG]^QOJ]EJ_]PR7PGUV#,V;/P7!K"<)LQ8951/B""8#)^4%Y%;*28`
M#LZ&%)%OV;M[;=DOP_&X#[^S:'G$7B6H'X+W))42?L!3/Q5(R3$1,(6ARI8C
M/G.(+JJS:)J$%^,`Y5X<,B@(YC.=A+UPRBHE`7AZ`ASF<;`*Y):SK"YG.6+X
MLU5RO;)L;B.=LH1FHC2=V:RR[CVKZAPN_'NME+HBHW65/FHF@PD2C!%/GEW8
MLHURMKZA7-!/%^R-&*I)(;Y%8\8,KP/2%;-Q4CA#7I.0#VT[1N&GP)LGX#9%
MWEZWPCYM![AO_#-A<`=L)(,/,7+0&96P]\`K$+';:/",P-9\K(&>00%^,%SY
```

```
M%$;SA&TW(',/)?G#C<DM?!0%.2PXB)@[/Y&`ODLS(J@;4[\C20B;J054(%1T
M*@N#,0:,C;Q1G3,T&X&M$5LPJ\BW['2W+RX+[L"SAY*"I4<G7Y@L41W*LIX_
MO9$+.&&=I[#P%T`_`7G:(YN?,,*8S<>D,<A/R\8Q'L]_I88F</U.;#7\]EO"
M(U*@&GQ1*E+E?@]Y8N/H)KO3>U956G'C&:<RH>N]%;+3("W,%<-<="BVB2#3
M$!^T^=08=E&"IER7TBP*H<U((LEE>0O0`O?5\C)C9B/5]O1W2U]"0F4@?:B_
MX(*E#!N_PP!HJ]/WM_6%%IIF_I"2Y.=0ZY?B-KS;HH2.P%G8%HOI6HF=\&`<
MR%$DGYU$P_DX6)3KT>6\3&A2?J(FY0$!<5C)*Z+F.$IL(B.X.XZ=1$L+(:6[
MV.MB%C"6+/A^"E*2=E"6YY%-G-(9;]-C4@\[("8BODDX'0:?;6AJ9S,.5/VT
M5G*P*O8)]E'V9I;?)T$5[\<?"AAYEBA"RK-)_;'&_S%GO%L<@CC*6A=3!K^&
M:G]_5Y;CN6,A%"Q1^U"&GN&R@K$F*N:,<TTD-/W$:$/BO2RP.%[#[58F;V&R
M01I6LS+&IJ2OS<*E`9N<[$*T1<0/QBL53V$$PJ3@:9A,DCMM2A239-FM:(%2
M_EF6MRRY11AA3DQS#^L29;G]PLD<22A9S![=_-&5`>KIGK>D_O_JX?7_.[N[
M.SL9_?]3_(>'^12H[H5[Y_M6XXPXLNGE:3Z7+_*W6M*TO]#N3`NX/Q6NYK?F
M;7M[-?RS5E[(Y\$KBMDO+Y0G`?+7M]\HWZU"#GI'(NI&IC[OE@=G=D>$=]8$
ENS->"(*9RD`?AN.IL>A/7O]/GZ?/T^?I\]5^_C\`W);'`#`"````
`
end
```