Reverse Engineering HID iClass Master Keys

12 JUNE 2016 on rfid, hacking, HID iClass, reverse engineering

The HID iClass line of proximity cards and readers is a widely deployed RFID system that's been poked full of holes by security researchers. The system boasts a higher level of security through encryption and mutual authentication.

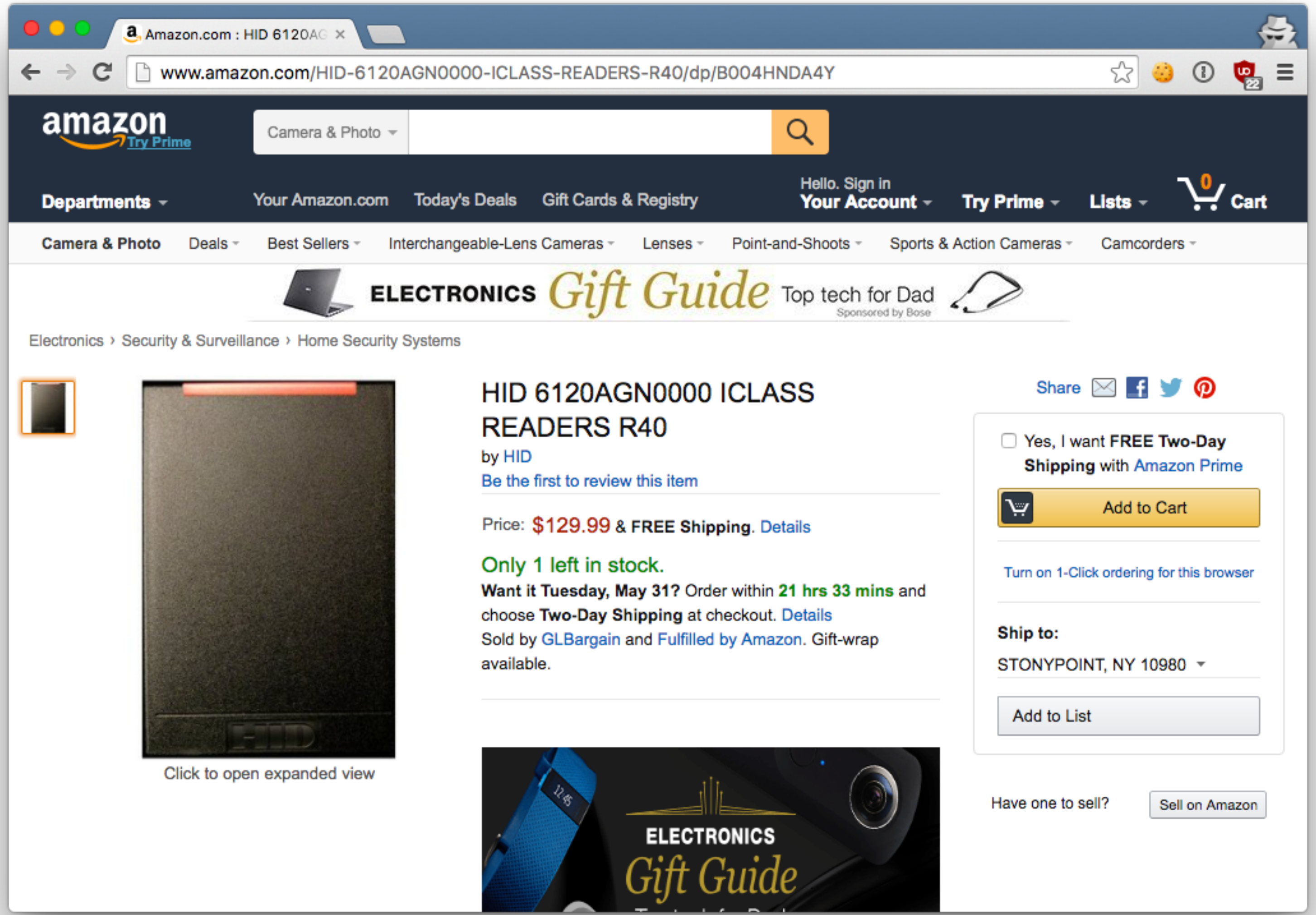
But neither of these defenses mean much when the master authentication key used by every standard iClass reader is retrievable by a moderately technical individual.

The authentication key is highly sensitive as it allows one to read decrypted card content and also overwrite card content. This effectively means that an attacker with possession of the authentication key is capable of cloning HID iClass cards and changing configuration settings on the physical reader itself.

Getting the Keys

The original approach for gaining the HID master key was disclosed in a paper entitled Heart of Darkness - exploring the uncharted backwaters of HID iCLASS™ security. This method takes advantage of a vulnerability in a specific line of readers released by HID which expose 6 debug pins on the rear of the reader. The Heart of Darkness approach entails leveraging those debug pins to modify the on-board firmware of two vulnerable readers. By modifying the firmwares, the readers each dump one half of the complete firmware image. The two halves can be stitched together to create a full firmware image which can be used to re-flash the two sacrificial readers.

The most commonly exploited reader is the HID RW300 Rev A, but you can use an RW300, RW400, RWK400, R30, R40, or RK40. The only caveat is that it must be Revision A. Revision B or C *will not work*. These are fairly hard to come by, but if you monitor Ebay or keep a watchful eye on Google, you could get lucky.



HID 6120AGN0000 ICLASS READERS R40
by HID
Be the first to review this item

Price: $129.99 & FREE Shipping. Details

Only 1 left in stock.
Want it Tuesday, May 31? Order within 21 hrs 33 mins and choose Two-Day Shipping at checkout. Details
Sold by GLBargain and Fulfilled by Amazon. Gift-wrap available.

Yes, I want FREE Two-Day Shipping with Amazon Prime

Turn on 1-Click ordering for this browser

Ship to:
STONYPOINT, NY 10980

Have one to sell?

It took me 20 mins while writing this blog post to find this

If you want to replicate the Heart of Darkness method, you will be looking for two of these model numbers:

- 6110A (R30)
- 6111A (RW300)
- 6120A (R40)
- 6121A (RW400)
- 6130A (RK40)
- 6131A (RWK400)

In addition, there exists an alternative technique pioneered by proxclone.com and Brad Antoniewicz which dumps the memory of only one reader. While this technique seems much easier and less expensive, it's been very difficult to replicate by myself and others.

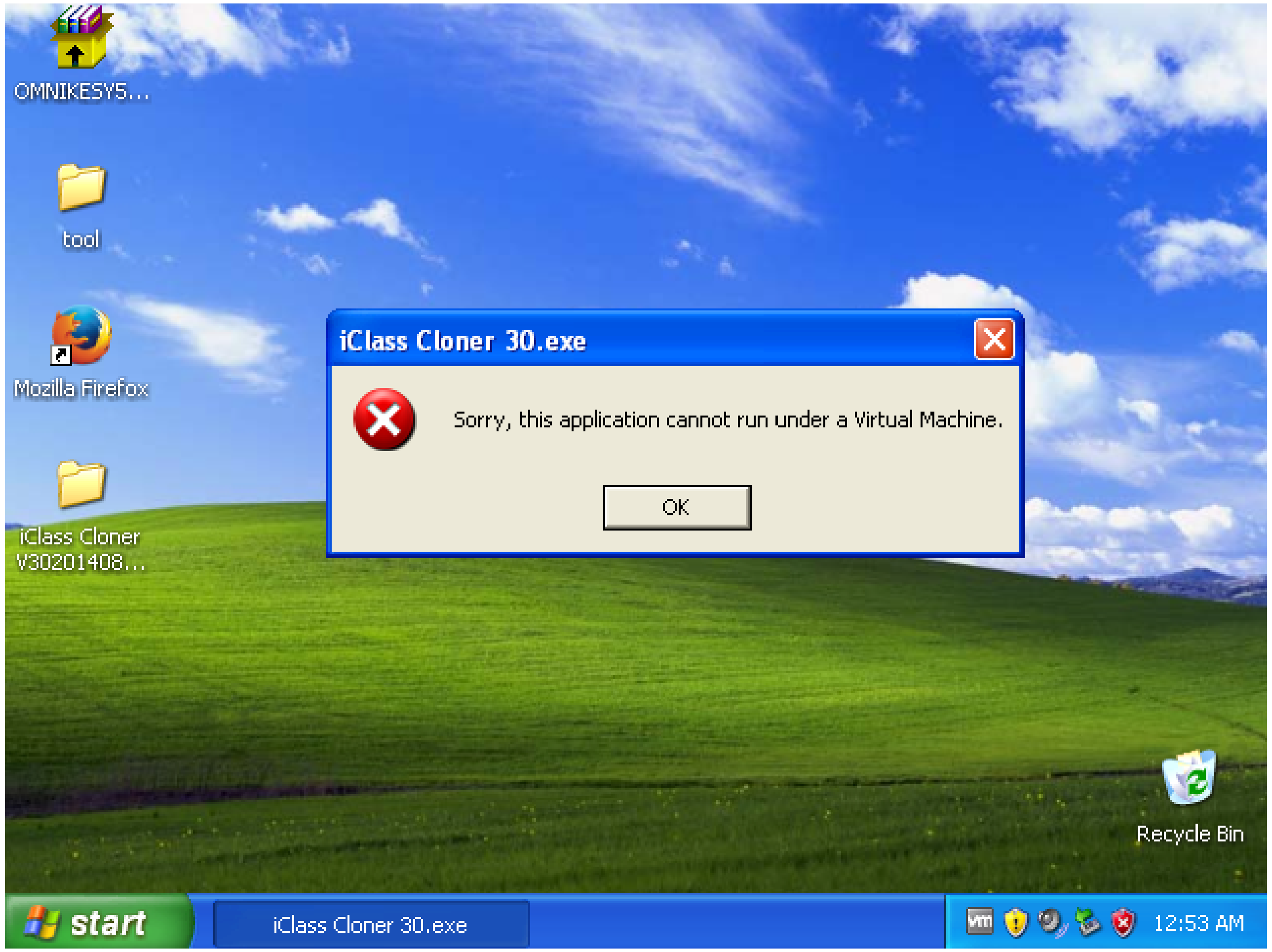
But if you want to avoid buying a vulnerable reader altogether, I'll be outlining a technique for reverse engineering the master keys from released software, and also reading and writing HID iClass cards without needing the master key.

At some point, I received a copy of chinese software used to clone iClass cards after gaining the master key in a more conventional way. Despite already having the master key, this application presented an interesting challenge.

For one thing, the application could only be run if the manufacturer provided USB dongle was attached to the computer. Not only is this annoying, but it also adds to the suspiciousness of the software.

Unfortunately, I don't have a picture of the dongle and no longer have it in my possession, but it's a rather suspicious looking PCB encased in blue translucent plastic. It emulated an HID device of some kind which also added to its suspiciousness.

Obviously, it would be prudent to run the software in a Virtual Machine (VM) in order to limit the impact it could have on your system. But you'll soon discover that the first hurdle to bypass is virtual machine detection being used in the application.



iClass Cloner 30.exe

Sorry, this application cannot run under a Virtual Machine.

OK

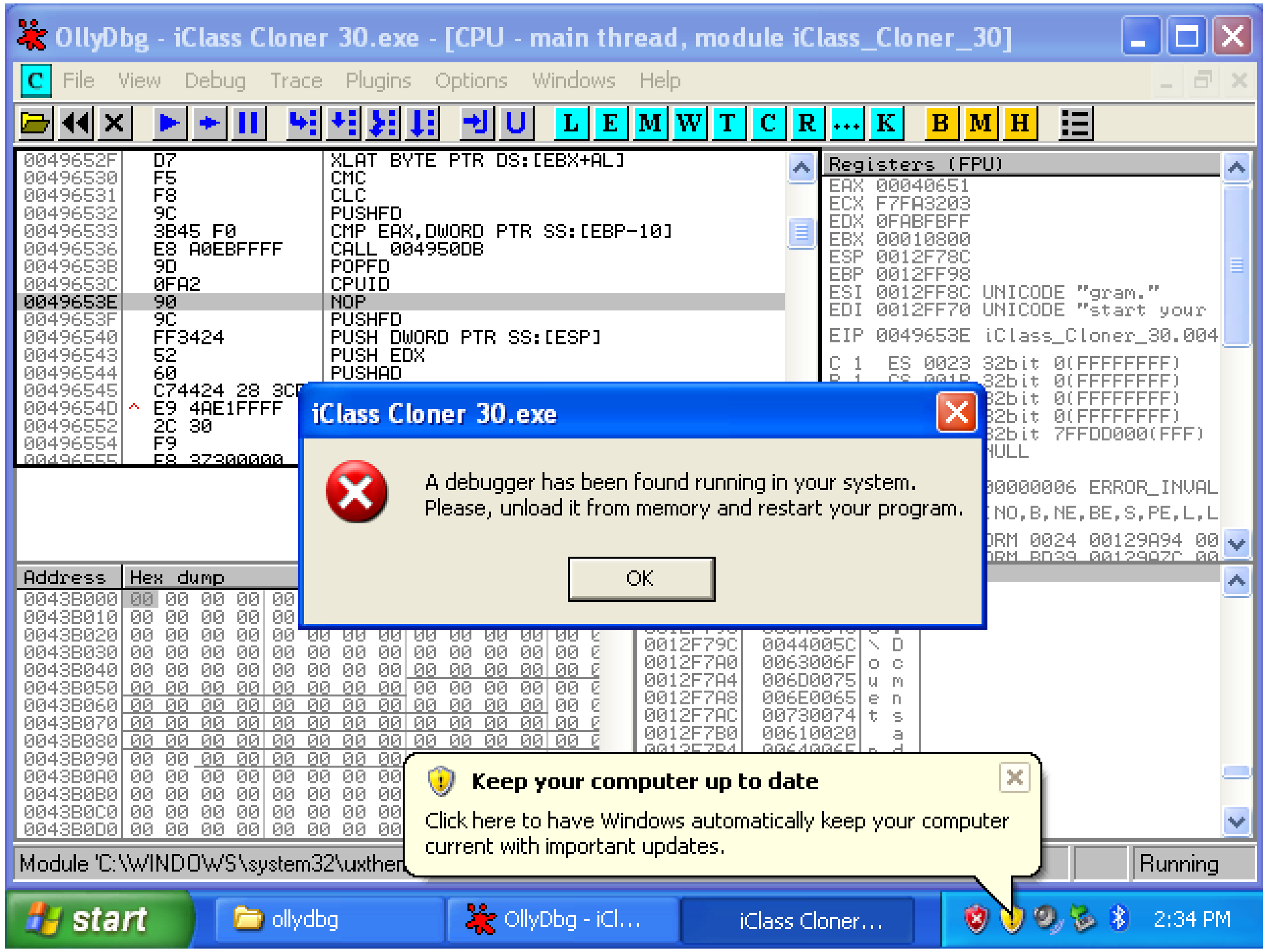
Virtual machine detection is super easy

Many applications that wish to resist the efforts of a reverse engineer will attempt to detect if they are currently in a virtual machine. Often times this is done by detecting features of a VM. In this case, one can disable querying the VMWare Tools version by adding the following line to the virtual machine's vmx file:

```
isolation.tools.getVersion.disable = "TRUE"
```

This breaks most if not all of the functionality of VMWare Tools so there's likely not much difference between uninstalling it and adding the config value. It's also likely that the application did not detect VirtualBox’s Guest Additions but this is untested.

Once the application ran in the VM, we could use USB passthrough to connect the USB dongle to the VM and utilize the program's functionality. But, attempting to run the application in a debugger led to the discovery that the application also detected debuggers.


OllyDbg - iClass Cloner 30.exe - [CPU - main thread, module iClass_Cloner_30]
File View Debug Trace Plugins Options Windows Help

```
0049652F   D7            XLAT BYTE PTR DS:[EBX+AL]
00496530   F5            CMC
00496531   F8            CLC
00496532   9C            PUSHFD
00496533   3B45 F0       CMP EAX,DWORD PTR SS:[EBP-10]
00496536   E8 A0EBFFFF   CALL 004950DB
0049653B   9D            POPFD
0049653C   0FA2          CPUID
0049653E   90            NOP
0049653F   9C            PUSHFD
00496540   FF3424        PUSH DWORD PTR SS:[ESP]
00496543   52            PUSH EDX
00496544   60            PUSHAD
00496545   C74424 28 3C
0049654D ^ E9 4AE1FFFF
00496552   2C 30
00496554   F9
```

```
Registers (FPU)
EAX 00040651
ECX F7FA3203
EDX 0FABFBFF
EBX 00010800
ESP 0012F78C
EBP 0012FF98
ESI 0012FF8C UNICODE "gram."
EDI 0012FF70 UNICODE "start your

EIP 0049653E iClass_Cloner_30.004

C 1  ES 0023 32bit 0(FFFFFFFF)
          32bit 0(FFFFFFFF)
          32bit 0(FFFFFFFF)
          32bit 0(FFFFFFFF)
          32bit 7FFDD000(FFF)
          NULL

          00000006 ERROR_INVAL
          NO,B,NE,BE,S,PE,L,L
          RM 0024 00129A94 00
```

```
Address   Hex dump
0043B000  00 00 00 00 00
0043B010  00 00 00 00 00
0043B020  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B030  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B040  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B050  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B060  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B070  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B080  00 00 00 00 00 00 00 00 00 00 00 00 00
0043B090  00 00 00 00 00 00 00 00
0043B0A0  00 00 00 00 00 00 00 00
0043B0B0  00 00 00 00 00 00 00 00
0043B0C0  00 00 00 00 00 00 00 00
0043B0D0  00 00 00 00 00 00 00 00
```

```
0012F79C  0044005C  \ D
0012F7A0  0063006F  o c
0012F7A4  006D0075  u m
0012F7A8  006E0065  e n
0012F7AC  00730074  t s
0012F7B0  00610020    a
```

iClass Cloner 30.exe

A debugger has been found running in your system.
Please, unload it from memory and restart your program.

OK

Keep your computer up to date
Click here to have Windows automatically keep your computer current with important updates.

Module 'C:\WINDOWS\system32\uxthen

Debugger detection is a little harder

Being a fool, I initially misconfigured ScyllaHide which led me to believe that the application resisted debugging in a very novel way.

While you can debug the application using ScyllaHide's Obsidian profile, making much progress with the application is difficult, as the binary is packed with some unknown packer. Instead of suffering with an unruly application, I opted for a less aggressive method than hiding the debugger.

Released by Brendan Dolan-Gavitt, the PANDA tool allows for the recording and replaying of an entire virtual machine. This allows for the moderately easy dynamic analysis of an operating system or application.

Building PANDA is reasonably easy and instructions can be found on its wiki. I won't be covering the setup for PANDA, as other locations cover it in far more detail than I could.

I chose to create a Windows XP image, but there shouldn't be too much difference between other operating systems. So long as the application will run, the image is suitable.

By using PANDA's memsavep plugin, one can dump the memory of the PANDA virtual machine at a specified point in time. By feeding this memory dump to Volatility, we can then extract a copy of the binary that's been loaded into memory. This binary would have gone through the unpacking process and it would be reasonable to assume that it’s easier to reverse engineer.

```
user@ubuntu:~/Desktop/panda/qemu/i386-softmmu$ sudo ./qemu-system-i386 \
-m 256 -replay iclass_cloner -display none --monitor stdio \
-net nic,model=rtl8139 -panda memsavep:percent=95,file=iclass.dd
Adding PANDA arg memsavep:percent=95.
Adding PANDA arg memsavep:file=iclass.dd.
adding panda/qemu/i386-softmmu/panda_plugins/panda_memsavep.so to panda_plugin_files 0
loading panda/qemu/i386-softmmu/panda_plugins/panda_memsavep.so
Success
Warning: vlan 0 is not connected to host network
QEMU 1.0,1 monitor - type 'help' for more information
(qemu) loading snapshot
Unknown savevm section or instance 'slirp' 0
Unknown savevm section or instance '0000:00:01.2/uhci' 0
Unknown savevm section or instance '1/usb-host' 0
... done.
opening nondet log for read :   ./iclass_cloner-rr-nondet.log
iclass_cloner:    26118544 (  1.01%) instrs.    2.07 sec.  0.17 GB ram.
total_instr in replay: 2596690768
iclass_cloner:    56523437 (  2.18%) instrs.    3.20 sec.  0.20 GB ram.
iclass_cloner:    82856733 (  3.19%) instrs.    3.79 sec.  0.21 GB ram.
iclass_cloner:   106148319 (  4.09%) instrs.    4.09 sec.  0.21 GB ram.
iclass_cloner:   131014501 (  5.05%) instrs.    4.39 sec.  0.21 GB ram.
iclass_cloner:   155877170 (  6.00%) instrs.    5.02 sec.  0.22 GB ram.
iclass_cloner:   184977773 (  7.12%) instrs.    5.33 sec.  0.22 GB ram.
iclass_cloner:   208278260 (  8.02%) instrs.    5.77 sec.  0.22 GB ram.
iclass_cloner:   235690436 (  9.08%) instrs.    6.11 sec.  0.22 GB ram.
iclass_cloner:   262590380 ( 10.11%) instrs.    6.43 sec.  0.22 GB ram.
...omitted for brevity...
iclass_cloner:  2214259222 ( 85.27%) instrs.   33.52 sec.  0.23 GB ram.
iclass_cloner:  2237487850 ( 86.17%) instrs.   33.67 sec.  0.23 GB ram.
iclass_cloner:  2263666800 ( 87.18%) instrs.   33.88 sec.  0.23 GB ram.
iclass_cloner:  2289252184 ( 88.16%) instrs.   34.01 sec.  0.23 GB ram.
iclass_cloner:  2315753999 ( 89.18%) instrs.   34.20 sec.  0.23 GB ram.
iclass_cloner:  2337023256 ( 90.00%) instrs.   34.40 sec.  0.23 GB ram.
iclass_cloner:  2364365127 ( 91.05%) instrs.   34.90 sec.  0.23 GB ram.
iclass_cloner:  2389479880 ( 92.02%) instrs.   35.45 sec.  0.24 GB ram.
iclass_cloner:  2415390500 ( 93.02%) instrs.   36.17 sec.  0.24 GB ram.
iclass_cloner:  2442065576 ( 94.05%) instrs.   37.34 sec.  0.25 GB ram.
memsavep: Saving memory to iclass.dd.
iclass_cloner:  2468716146 ( 95.07%) instrs.   38.21 sec.  0.26 GB ram.
Replay completed successfully. 1
Time taken was: 41 seconds.
max_queue_len = 5889
```

```
5888 items on recycle list, 518144 bytes total
Replay terminated at user request.
```

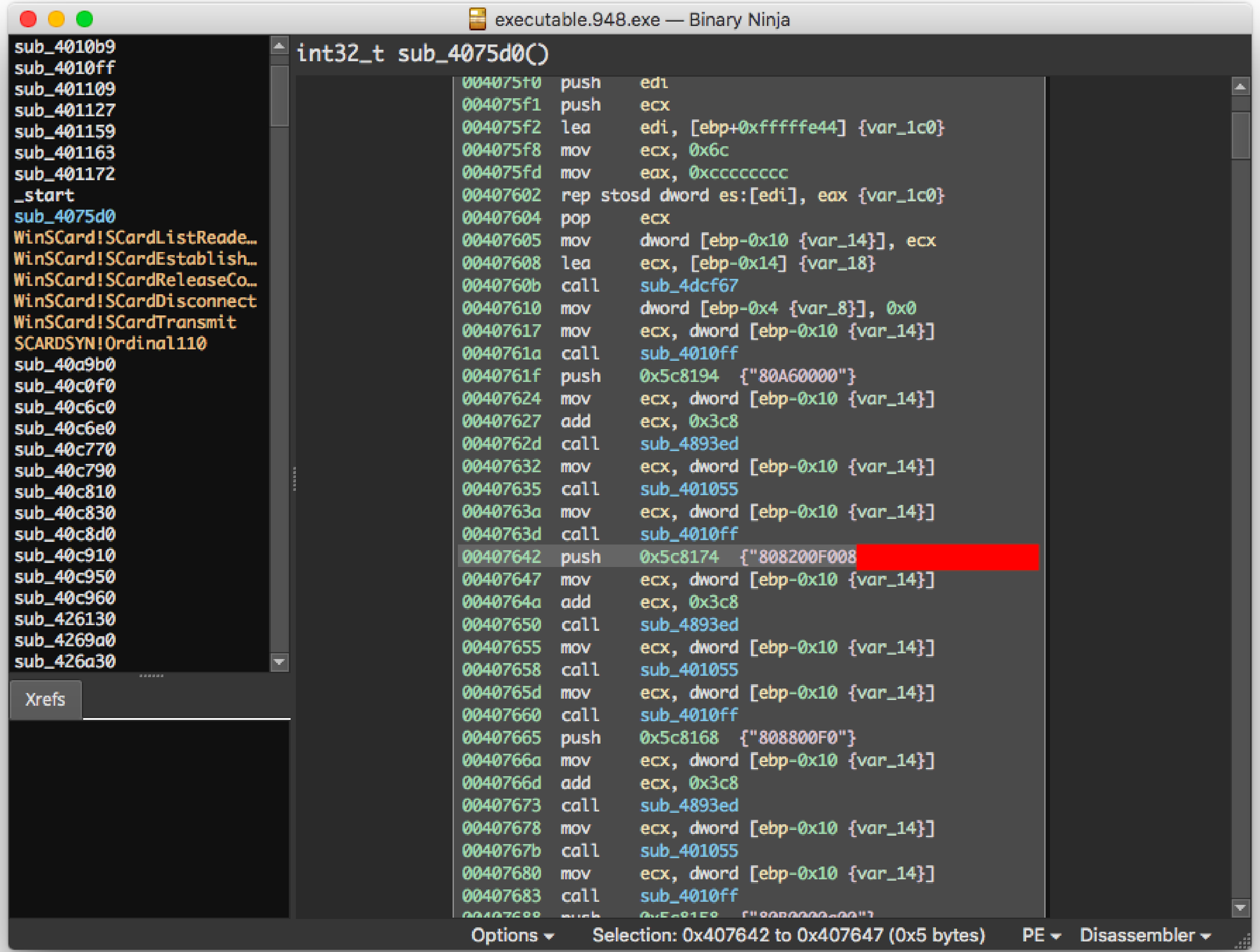
Technically, PANDA isn't really needed here as ScyllaHide and x64dbg allow us to bypass the application's anti-debugging and also dump application memory at runtime. But I initially could not run the application in a debugger and PANDA is an amazingly cool project that also allowed me to capture the USB dongle information for future analysis.

Now that we have a copy of memory *presumably* with a copy of the unpacked binary, we can use Volatility's `procdump` command to extract the program out of the RAM dump:

```
user@ubuntu:~/Desktop/iclass_cloner-rr/volatility$ python vol.py -f ../iclass.dd \
--profile=WinXPSP2x86 procdump -D dump/ --unsafe
Volatility Foundation Volatility Framework 2.5
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x817cca00 ---------- System               Error: PEB at 0x0 is unavailable
0x816e6020 0x48580000 smss.exe             OK: executable.312.exe
0x815e4da0 0x4a680000 csrss.exe            OK: executable.392.exe
0x81660da0 0x01000000 winlogon.exe         OK: executable.496.exe
0x81667da0 0x01000000 services.exe         OK: executable.540.exe
0x815e3020 0x01000000 lsass.exe            Error: ImageBaseAddress at 0x1000000 is unavailable
0x815e1020 0x01000000 svchost.exe          OK: executable.708.exe
0x816804f8 0x01000000 svchost.exe          OK: executable.756.exe
0x815f1560 0x01000000 svchost.exe          OK: executable.852.exe
0x815f44b8 0x01000000 svchost.exe          OK: executable.972.exe
0x8164e280 0x01000000 svchost.exe          OK: executable.1012.exe
0x81540020 0x01000000 explorer.exe         OK: executable.1212.exe
0x8167ac10 0x01000000 spoolsv.exe          OK: executable.1304.exe
0x81607980 0x01000000 alg.exe              OK: executable.1916.exe
```

```
0x816744b8 0x01000000 wscntfy.exe          OK: executable.1948.exe
0x81658448 0x00400000 wuauclt.exe          OK: executable.464.exe
0x814d7020 0x00400000 iClass Cloner 3      OK: executable.948.exe
```

Once a copy of the unpacked binary has been generated (executable.948.exe), we can do some light reverse engineering (string search) to identify the master key.



```
sub_4010b9
sub_4010ff
sub_401109
sub_401127
sub_401159
sub_401163
sub_401172
_start
sub_4075d0
WinSCard!SCardListReade…
WinSCard!SCardEstablish…
WinSCard!SCardReleaseCo…
WinSCard!SCardDisconnect
WinSCard!SCardTransmit
SCARDSYN!Ordinal110
sub_40a9b0
sub_40c0f0
sub_40c6c0
sub_40c6e0
sub_40c770
sub_40c790
sub_40c810
sub_40c830
sub_40c8d0
sub_40c910
sub_40c950
sub_40c960
sub_426130
sub_4269a0
sub_426a30
```

```
int32_t sub_4075d0()

004075f0  push    edi
004075f1  push    ecx
004075f2  lea     edi, [ebp+0xfffffe44] {var_1c0}
004075f8  mov     ecx, 0x6c
004075fd  mov     eax, 0xcccccccc
00407602  rep stosd dword es:[edi], eax {var_1c0}
00407604  pop     ecx
00407605  mov     dword [ebp-0x10 {var_14}], ecx
00407608  lea     ecx, [ebp-0x14] {var_18}
0040760b  call    sub_4dcf67
00407610  mov     dword [ebp-0x4 {var_8}], 0x0
00407617  mov     ecx, dword [ebp-0x10 {var_14}]
0040761a  call    sub_4010ff
0040761f  push    0x5c8194  {"80A60000"}
00407624  mov     ecx, dword [ebp-0x10 {var_14}]
00407627  add     ecx, 0x3c8
0040762d  call    sub_4893ed
00407632  mov     ecx, dword [ebp-0x10 {var_14}]
00407635  call    sub_401055
0040763a  mov     ecx, dword [ebp-0x10 {var_14}]
0040763d  call    sub_4010ff
00407642  push    0x5c8174  {"808200F008[redacted]
00407647  mov     ecx, dword [ebp-0x10 {var_14}]
0040764a  add     ecx, 0x3c8
00407650  call    sub_4893ed
00407655  mov     ecx, dword [ebp-0x10 {var_14}]
00407658  call    sub_401055
0040765d  mov     ecx, dword [ebp-0x10 {var_14}]
00407660  call    sub_4010ff
00407665  push    0x5c8168  {"808800F0"}
0040766a  mov     ecx, dword [ebp-0x10 {var_14}]
0040766d  add     ecx, 0x3c8
00407673  call    sub_4893ed
00407678  mov     ecx, dword [ebp-0x10 {var_14}]
0040767b  call    sub_401055
00407680  mov     ecx, dword [ebp-0x10 {var_14}]
00407683  call    sub_4010ff
00407688  push    0x5c8158  {"80B0000a00"}
```

```
Selection: 0x407642 to 0x407647 (0x5 bytes)
```

The extracted key (censored for my protection)

All of a sudden it seems way easier to just buy this Chinese software and extract the keys from memory! No fiddling with finicky bit-banging, having multiple vulnerable readers, or having a solid background in electrical engineering.

But alas, it is even easier to gain the master key as it was leaked online before I even began this research. I won't share where it is, but maybe you should go looking!

Throwing away the keys

While researching the HID iClass system and reading the proxmark forums, a few ideas became apparent:

- Very little specific information is available to reproduce the work of the higher up proxmark forum members, but a large amount of documentation is available from HID. Reading through this documentation is critical to getting anywhere with the systems.
- Some of this information has been purged from the internet by HID. Some of it can be recovered by asking the right people, and some I have been unable to find.

- A large amount of code released is based on code released by HID or the iclassified library released by the authors of Exposing iClass Key Diversification.
- A user purports to be able to write cards without needing access to the HID master key.

Obviously, being able to achieve all of this without the master key is desirable. You would no longer need to purchase vulnerable readers to get the ability to modify or clone iClass cards.

According to the iClass developer's documentation, it should not be possible to perform this without knowing the master key. Nonetheless, in a true testament to the insecurity of the HID iClass system, simple modifications to the iclassified example code allow a user to read and write to an iClass card without the master key.

The iclassified example code initializes what's known as Secure Mode and then proceeds to authenticate to the card. But, if after authentication, you were to issue the command to initialize Secure Mode **AGAIN**, you gain full read and write permissions to the card!

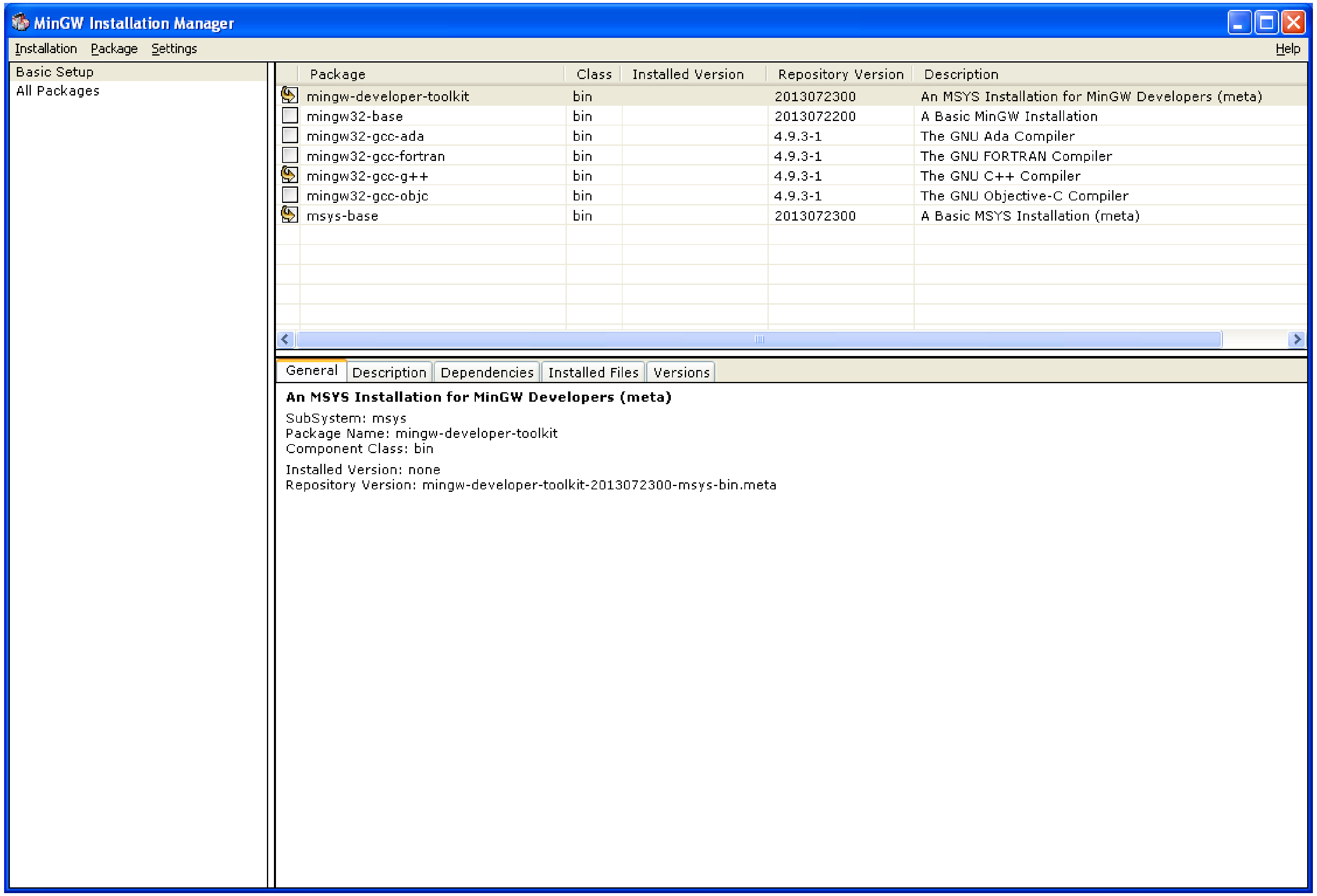
A modified version of the iclassified example code, full build instructions, drivers, and a GUI application to read and write iClass cards without using the master key is provided on Github.

In order to read and write to cards, you will, at the minimum, need a reader/writer. The simplest way to get one is to get a reader/writer from the HID OMNIKEY line. Either the HID OMNIKEY 5321 or 6321 will work and both are fairly cheap at around $50. The model number does not matter very much, contrary to what you may think. I used an HID OMNIKEY 6321 CLi and an HID OMNIKEY 5321 v2 CLi.

To build the software you want to start off by downloading the MinGW installer assistant. Install it and select:

- mingw-developer-toolkit
- mingw32-gcc-g++
- msys-base

It should look something like this:



Package	Class	Installed Version	Repository Version	Description
mingw-developer-toolkit	bin		2013072300	An MSYS Installation for MinGW Developers (meta)
mingw32-base	bin		2013072200	A Basic MinGW Installation
mingw32-gcc-ada	bin		4.9.3-1	The GNU Ada Compiler
mingw32-gcc-fortran	bin		4.9.3-1	The GNU FORTRAN Compiler
mingw32-gcc-g++	bin		4.9.3-1	The GNU C++ Compiler
mingw32-gcc-objc	bin		4.9.3-1	The GNU Objective-C Compiler
msys-base	bin		2013072300	A Basic MSYS Installation (meta)

An MSYS Installation for MinGW Developers (meta)

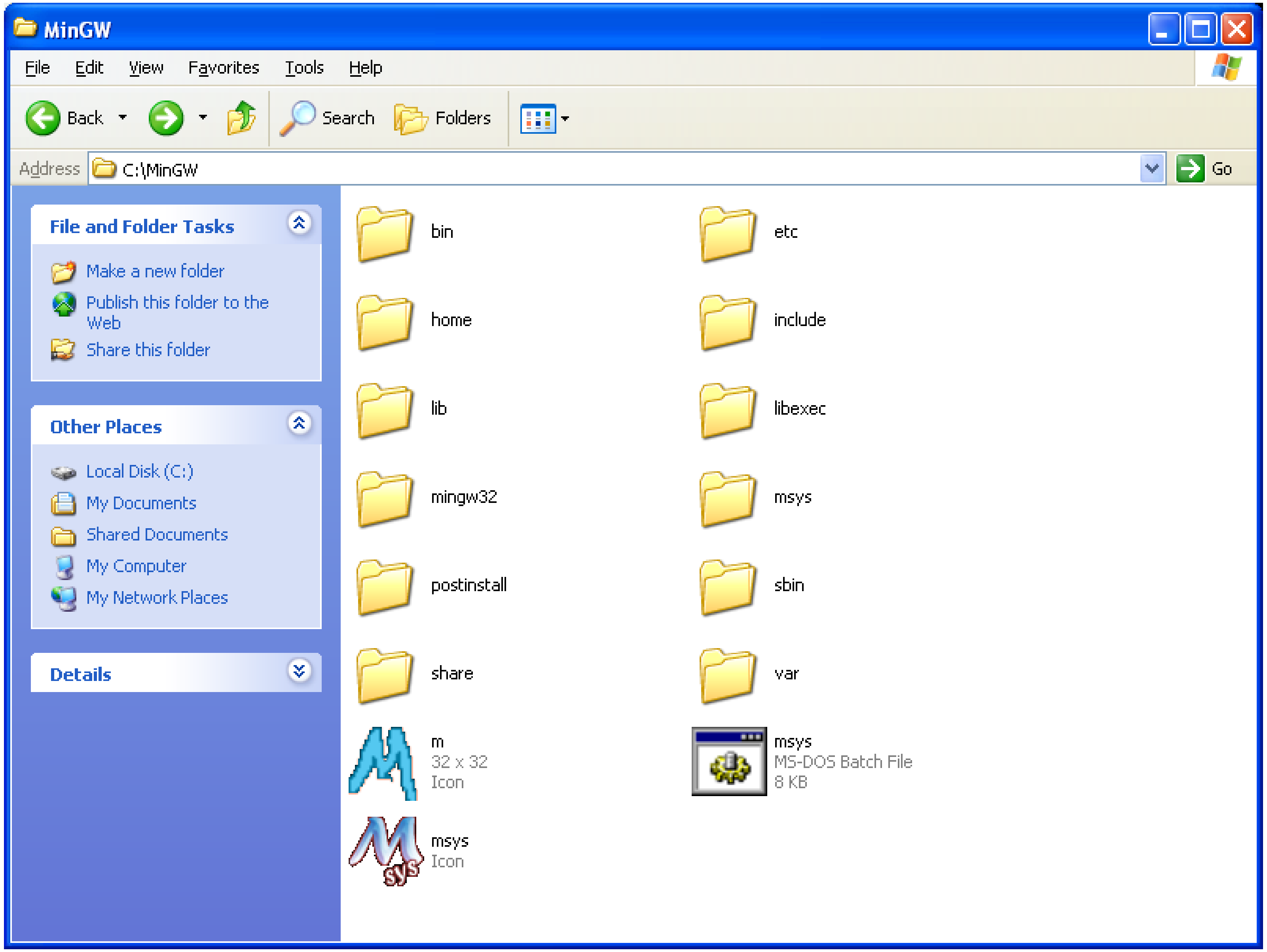
SubSystem: msys
Package Name: mingw-developer-toolkit
Component Class: bin

Installed Version: none
Repository Version: mingw-developer-toolkit-2013072300-msys-bin.meta

This was confusing but it was easier than most things in Windows…

Then go to `Installation > Apply Changes` on the menu bar to install part of your build environment.

Now go to `C:\MinGW\msys\1.0` and copy everything over to `C:\MinGW`.



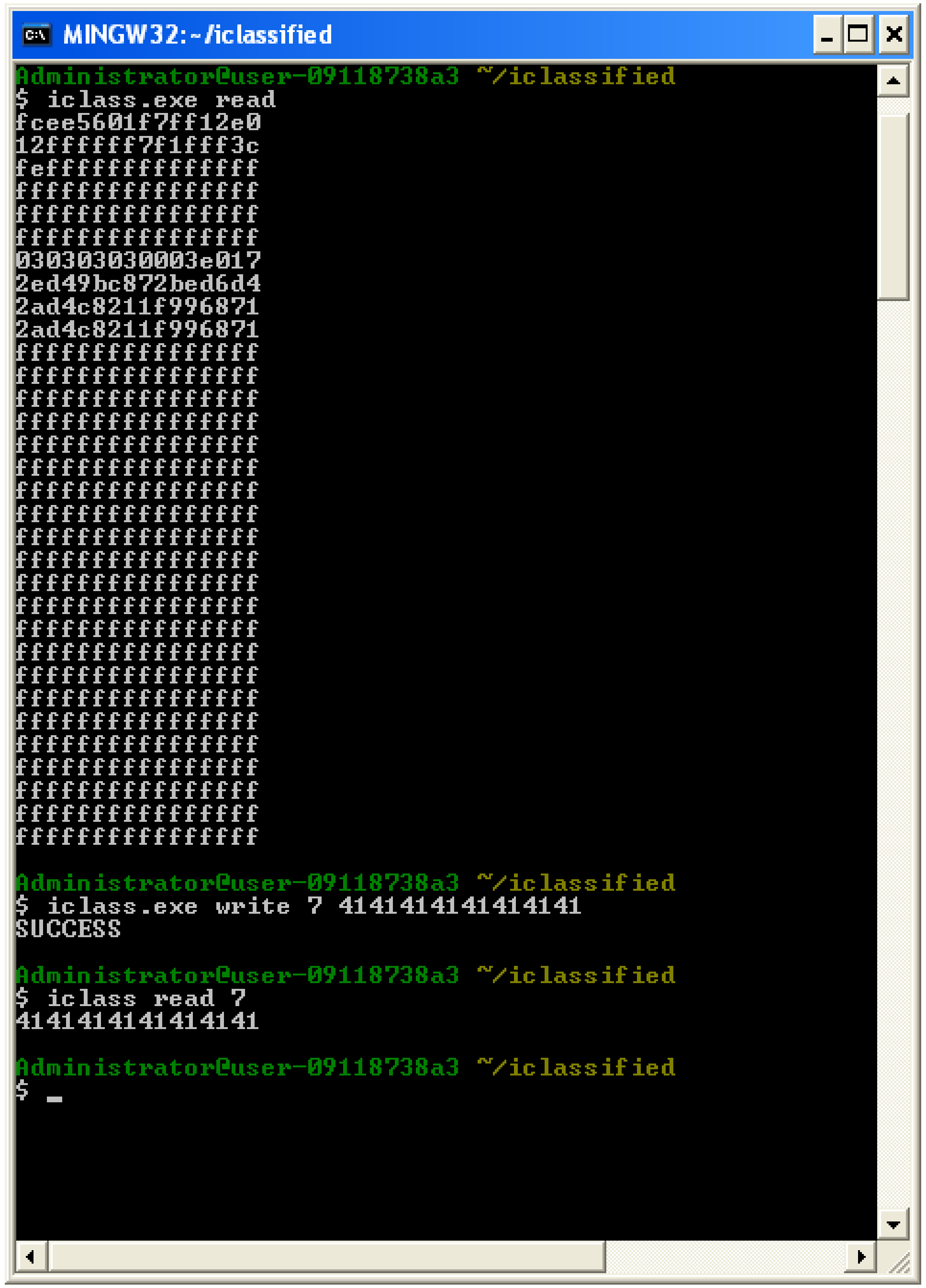
Your MinGW folder should look like this

Open the `msys.bat` file to get a small shell environment. Clone the provided source code into the home folder, go into the `iclassified` directory, and run `make`.

If everything runs well you should get `iclass.exe` and `iclassified.o`

At this point you should plug in your OMNIKEY reader and follow the instructions provided alongside the drivers to get the reader setup.

If all goes well you should be able to execute `iclass.exe read` to read a card and `iclass.exe write` to write to a card.



```
Administrator@user-09118738a3 ~/iclassified
$ iclass.exe read
fcee5601f7ff12e0
12ffffff7f1fff3c
feffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
030303030003e017
2ed49bc872bed6d4
2ad4c8211f996871
2ad4c8211f996871
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff
ffffffffffffffff

Administrator@user-09118738a3 ~/iclassified
$ iclass.exe write 7 4141414141414141
SUCCESS

Administrator@user-09118738a3 ~/iclassified
$ iclass read 7
4141414141414141

Administrator@user-09118738a3 ~/iclassified
$ _
```

Then you can read and write HID iClass cards without a master key!

Thanks to Carl, Brad Antoniewicz, Fran Brown, Brendan Dolan-Gavitt, and Evan Jensen.

Kevin Chung

I like doing computer stuff, playing chess, and playing video games. I ran CSAW CTF for a couple of years and I wrote CTFd which is a popular Capture The Flag framework.

New York City

https://ctfd.io

Follow me on Twitter

Share this post

READ THIS NEXT

API Keys, API Keys everywhere

YOU MIGHT ENJOY

Adding image captions to Ghost