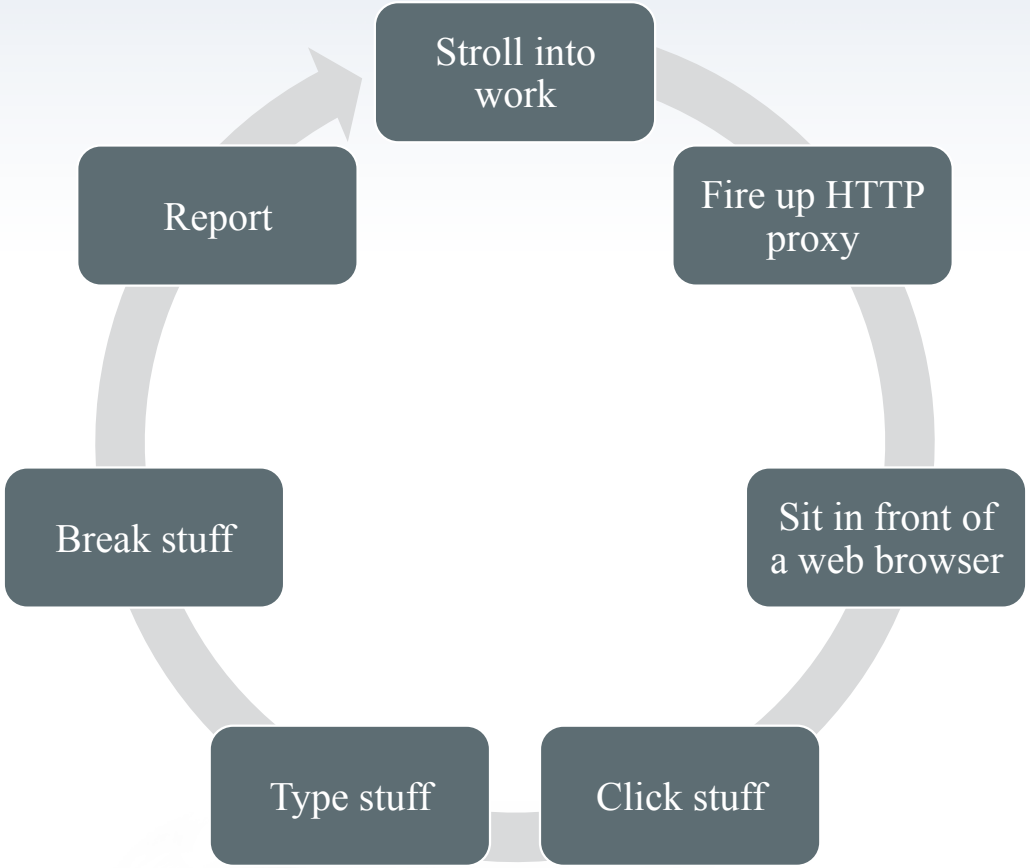
Exploiting USB Devices with Arduino

THE LIFE OF GREG



WHY AM I HERE?

"PROTECT THE PRIVACY OF YOUR COMPUTER MONITOR"!!!!11

"SAFE, SECURE, AUTOMATIC"

WHY ARE YOU HERE?

- Crypto, forensics, SSL, or mobile phones not interesting?
- Learn how to approach assessing USB devices
- Learn about some protocol level / implementation issues
- See devices get exploited
- Finally trying to justify buying an Arduino

A USB PRIMER

- A well established protocol (no oscilloscope required!)
- With protocols (classes)
  - Inside of protocols
    - We can go deeper
- Great reference: http://www.beyondlogic.org/usbnutshell/usb1.shtml
- Tools exist to parse these protocols (even some free ones!)
- What we are interested in is the application protocol

GO WITH WHAT YOU KNOW

- No different than a web app
- Break it down into familiar steps
  - Threat Modeling
  - Use Case Analysis
  - Stimulus / Response Testing
  - Exploitation

THREAT MODELING

- Identify the components of the underlying architecture
- Identify security relevant use cases
- Identify explicit and implicit trust boundaries

USE CASE ANALYSIS

- Identify the inputs and outputs of the enumerated use cases
- Identify the protocol and methods for these inputs
- Identify how security relevant use cases are executed

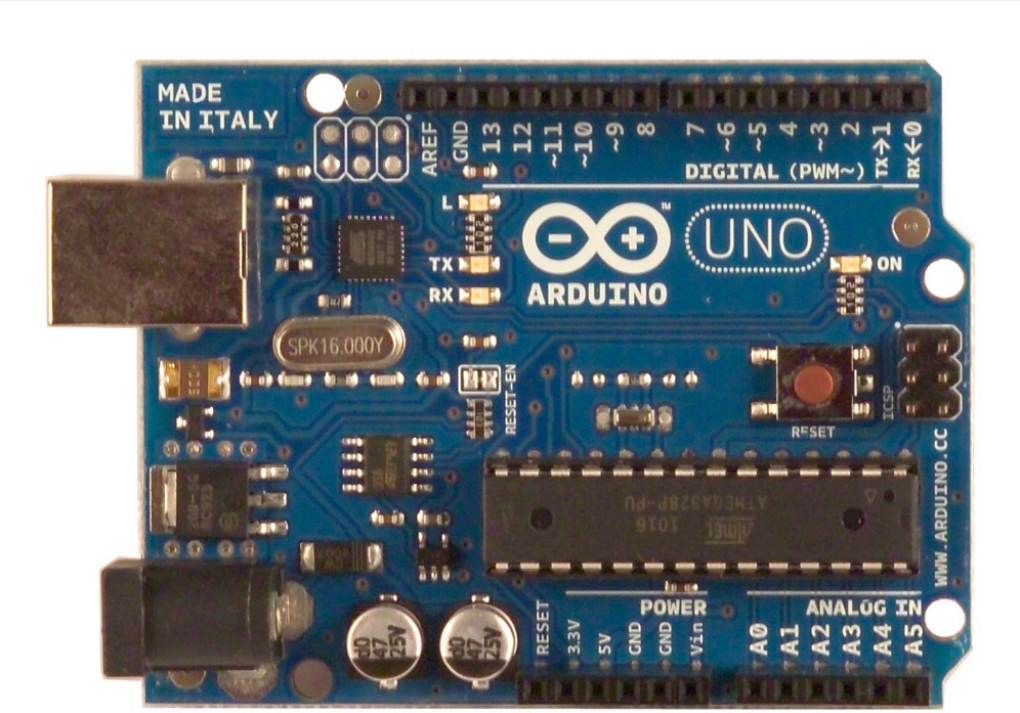
STIMULUS / RESPONSE TESTING

- Produce instrumentation to execute the identified use cases
- Perform testing of the identified use cases with unexpected input to yield unexpected outputs

EXPLOITATION

- Instrumentation of any identified vulnerabilities
- Automation of this exploitation
- Pwnin suckaz

THIS IS ALL REALLY BORING, WHERE IS THE ARDUINO



… pwnin suckaz

THREAT MODELING THE SCREEN KEEPER

- Components
  - Wireless token
  - USB dongle
  - Host software

TYPICAL INSTALLATION AND USAGE

- Installation (software & hardware)
- Screen Locking
  - Walk out of range
  - Turn wireless token off
- Screen Unlocking
  - Walk back in range
  - Turn on token
  - Enter override password

USE CASES AND TRUST BOUNDARIES

- Pretty limited security relevant use cases
  - Device installation and registration
  - Host screen lock
  - Host screen unlock via token
  - Host screen unlock via password
- Assumed trust boundaries
  - Host to USB receiver
  - USB receiver to wireless token
- Assumed compromised components
  - Physical host computer
  - USB receiver

USE CASE ANALYSIS

- How do I go about really testing and seeing what is going on?
- You wouldn't assess a web app without an HTTP proxy, so we need the equivalent tools setup
- USB traffic analyzer
  - Hardware
  - Software
  - Virtual Host

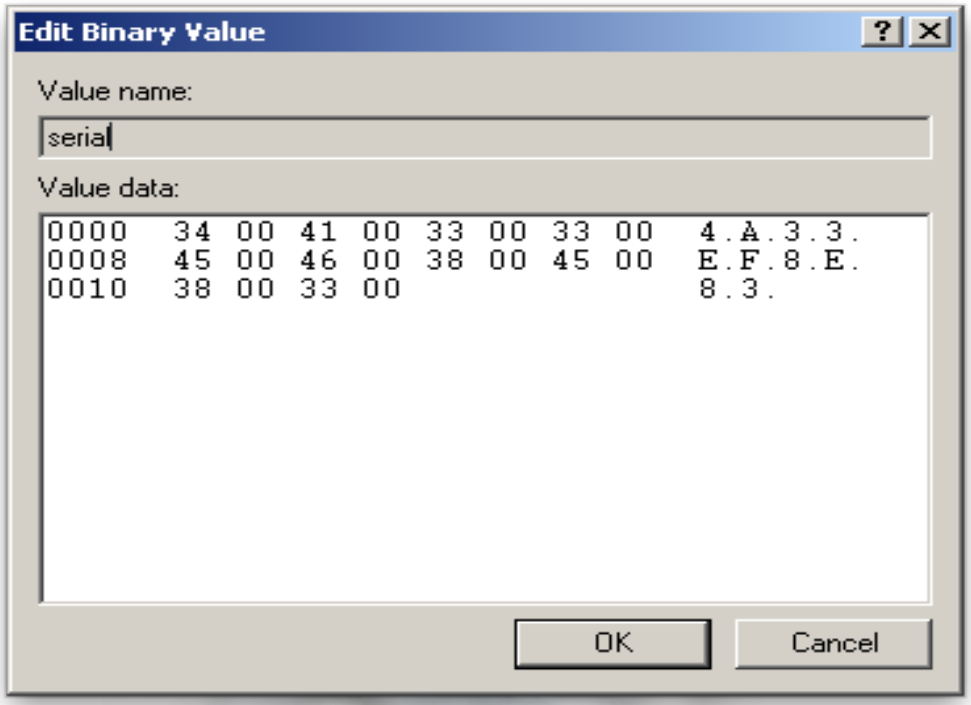
CONFIGURATION FOR VIRTUAL USB ANALYZER

- Set some VMWare configuration options:
  - monitor = "debug"
  - usb.analyzer.enable = TRUE
  - usb.analyzer.maxLine = 8192
  - mouse.vusb.enable = FALSE
- Get USB traffic:

```
May 15 14:59:57.911: vmx| USBIO: GetDescriptor(string, 2, langId=0x0409)
May 15 14:59:57.911: vmx| USBIO: Down dev=1 endpt=0 datalen=255 numPackets=0 status=390052272 1a54dbb0
May 15 14:59:57.911: vmx| USBIO:  000: 80 06 02 03 09 04 ff 00                          ........
May 15 14:59:57.912: vmx| USBIO: Up dev=1 endpt=0 datalen=38 numPackets=0 status=0 0
May 15 14:59:57.912: vmx| USBIO:  000: 80 06 02 03 09 04 ff 00                          ........
May 15 14:59:57.912: vmx| USBIO:  000: 26 03 73 00 63 00 72 00 65 00 65 00 6e 00 20 00 &.s.c.r.e.e.n. .
May 15 14:59:57.912: vmx| USBIO:  010: 6b 00 65 00 65 00 70 00 65 00 72 00 20 00 31 00 k.e.e.p.e.r. .1.
May 15 14:59:57.912: vmx| USBIO:  020: 2e 00 30 00 41 00                              ..0.A.
```

ANSWERING IMPORTANT QUESTIONS: DEVICE INSTALLATION / REGISTRATION

- Can a USB receiver be swapped out from a locked screen and replaced with another USB receiver and in-range token?
  - Nope, each wireless token seems registered or linked per USB dongle
- How is a USB receiver registered with the host computer?
  - A per-device identifier is stored within the Windows registry:



ANSWERING IMPORTANT QUESTIONS: DEVICE INSTALLATION / REGISTRATION

» What information is sent from the USB receiver when inserted into the host computer?

Field	Value	Meaning
bLength	18	Valid Length
bDescriptorType	1	DEVICE
bcdUSB	0x0200	Spec Version
bMaxPacketSize0	32	Max EP0 Packet Size
idVendor	0x1915	Nordic Semiconductor ASA
idProject	0x001F	Unknown
bcdDevice	0x0100	Device Release Number
iManufacturer	1	Index to Manufacturer String (Not known)
iProduct	2	Index to Product String “screen keeper 1.0A”
iSerialNumber	3	Index to Serial Number String “4A33EF8E83”
bNumConfigurations	1	Number of Possible Configurations

ANSWERING IMPORTANT QUESTIONS: HOST SCREEN LOCK

- What USB traffic is sent when the wireless device is out of range or turned off to indicate that the screen should be locked?
  - None, lack of traffic == lock screen
- Does the host remain locked when the physical USB device is removed?
  - Yes, oh well, would have been a lolz finding
- Can the host be unlocked after the physical USB receiver has been removed and reinserted?
  - Yes, this means that an attack can compromise the USB receiver and the software will still allow the receiver to unlock the host

ANSWERING IMPORTANT QUESTIONS: HOST SCREEN UNLOCK VIA TOKEN

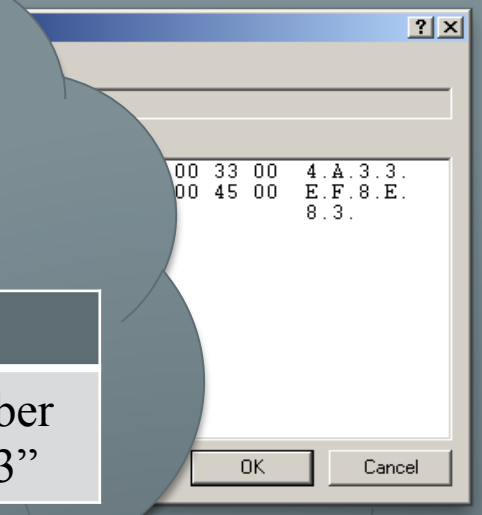
» What USB traffic is sent when the wireless device is in range?

#413212...413214 17.545,904 s	FS	Interrupt Transfer	Addr	Endp	Data (24 bytes)	Status
	⇦	HID Report In	0x02	0x1	34 00 41 00 33 00 33 00...	OK
#595629...595631 19.570,298 s	FS	Interrupt Transfer	Addr	Endp	Data (24 bytes)	Status
	⇦	HID Report In	0x02	0x1	34 00 41 00 33 00 33 00...	OK
#775290...775292 21.562,685 s	FS	Interrupt Transfer	Addr	Endp	Data (24 bytes)	Status
	⇦	HID Report In	0x02	0x1	34 00 41 00 33 00 33 00...	OK
#955715...955717 23.563,074 s	FS	Interrupt Transfer	Addr	Endp	Data (24 bytes)	Status
	⇦	HID Report In	0x02	0x1	34 00 41 00 33 00 33 00...	OK

ANSWERING IMPORTANT HOST

»

Field	Value	Meaning
iSerialNumber	3	Index to Serial Number String “4A33EF8E83”



4	A	3	3	E	F	8
E	8	3	0	1		

ANSWERING IMPORTANT QUESTIONS: HOST SCREEN UNLOCK VIA TOKEN

- Wait… what was the secret sauce used to unlock the host?

```
The serial number...
                          from...
                                        the USB descriptor.
```

IN SUMMARY

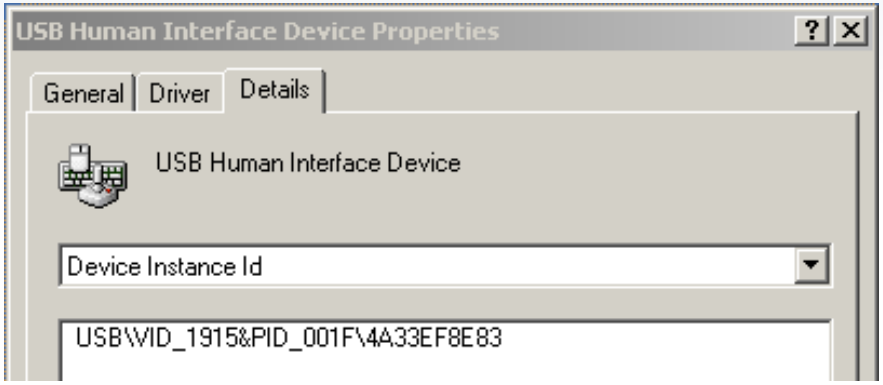
- Host does not care if USB receiver has been removed or replaced
- Host decides to lock based on lack of USB traffic
- Host decides to unlock based on USB traffic sent by receiver
- Host authenticates and unlocks host based on USB serial number

STIMULUS / RESPONSE TESTING

- We know what we want to do:
  - unplug victim USB token
  - read USB serial number from it
  - play unlock message containing this serial
  - profit
- How do we go about doing this?

STIMULUS / RESPONSE TESTING

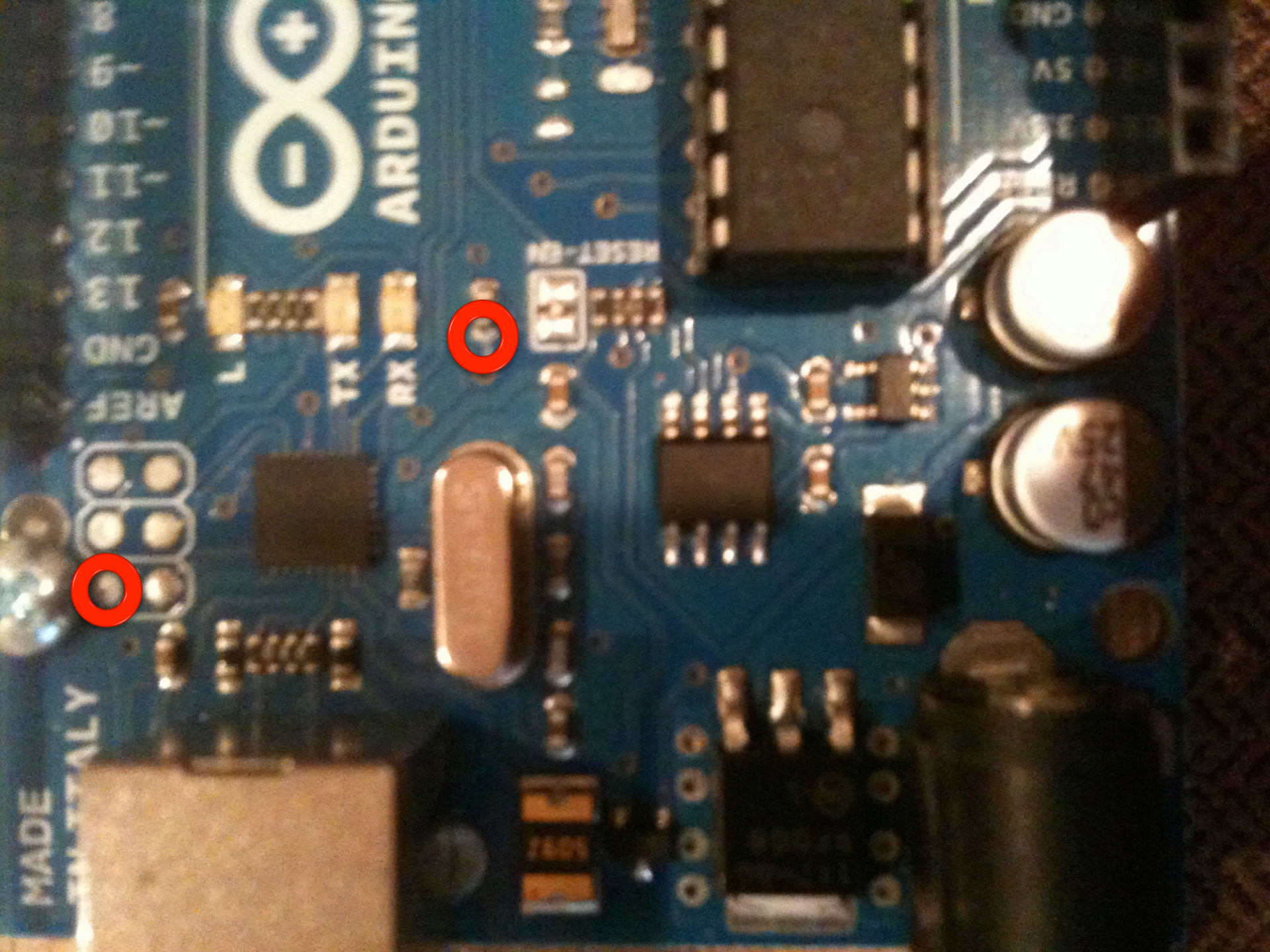
» Reading serial number

- 

- `$ system_profiler SPUSBDataType | grep -A 10 keeper | grep Serial | cut -f 2 -d ':'`
  - `4A33EF8E83`
- We are real USB hackers now…

SETTING UP TEST ENVIRONMENT

- Arduino Uno
- ATmega8U2
- Can use firmware developed using the open source LUFA (Lightweight USB Framework for AVR) library
- Firmware can be built using and AVR GNU compiler suite
- Firmware can be flashed using dfu-programmer after super secret Arduino handshake


ARDUIN
13
GND
AREF
TX
RX
MADE

CREATING A CUSTOM FIRMWARE

- Start with Arduino distributed source
- Descriptors.c
  - Defines the device descriptors used when enumerating the device
  - We will modify these to enumerate to the values of the Screen Keeper device
- Arduino-usbserial.c
  - The actual main() loop of the firmware
  - This will need to send the device serial ID to the host

EVIL DESCRIPTORS.C

- Since we are a HID device, base off of provided LUFA HID device demo
- Setup descriptors to match device
- Modify HID Report to match (24-bytes)
- Extend descriptor table to include the serial number
  - Table se for this
  - It's a
  - Add it to the enumeration function - CALLBACK_USB_GetDescriptor

Field	Value	Meaning
iSerialNumber	3	Index to Serial Number String "4A33EF8E83"

EVIL ARDUINO-SCREENKEEPER.C

- Modify main function to remove all unnecessary code
- Modify the main loop to send the HID Report message to the host

```
for (;;) {
Endpoint_Write_PStream_LE(
        SERIAL_NUMBER+”01”,
        SERIAL_NUMBER_LEN+4,
        NO_STREAM_CALLBACK
);
...
```

DEMO

A NEW VERSION APPEARS

NEW DEVICE DESCRIPTOR

Field	Value	Meaning
bLength	18	Valid Length
bDescriptorType	1	DEVICE
bcdUSB	0x0200	Spec Version
bMaxPacketSize0	32	Max EP0 Packet Size
idVendor	0x1915	Nordic Semiconductor ASA
idProject	0x001F	Unknown
bcdDevice	0x0100	Device Release Number
iManufacturer	1	Index to Manufacturer String “SEMI-LINK”
iProduct	2	Index to Product String “screen keeper 1.1A”
iSerialNumber	3	Index to Serial Number String “Screen Lock”
bNumConfigurations	1	Number of Possible Configurations

SO WHAT'S THE DIFFERENCE REALLY?

- How is the device now registered?
  - Still uses a serial number
  - Registered after first message received (from HID report), not from device descriptor
  - Registration locked until reset within the software
- Serial number is no longer sent unless the USB dongle unless token is in range
- Other ways can we get this?

MITIGATED?

- Well how can we get the key now?
  - Wait for someone to leave their computer
  - Grab their USB dongle
  - Go to a meeting with them
  - Record serial number sent in HID report
  - Leave meeting early
  - See earlier slides
- That's all too "Mission Impossible" for me
- Please give me a real vulnerability
- Brute force? (kind of real I guess…)

BRUTE FORCE ANALYSIS

- Looking at Key Space
- Possible keys = Number of possible symbols ^ Length

Interpretation	Possible Symbols	Length	Possible Keys
Full HID Report	255	24	Huge
Printable Unicode String	94	12	Huge
Hex String	16	12	Maybe possible for brute force?

ENTROPY ANALYSIS

- We have 16^12 possible keys, but are all 12 actually unpredictable?
- Some sample keys (limited sample size)

4	A	3	8	D	B	E	E	D	5	0	1
4	A	3	3	E	F	8	E	8	3	0	1
4	B	3	7	A	8	E	C	1	1	0	1
4	B	0	4	A	3	6	1	F	4	0	1

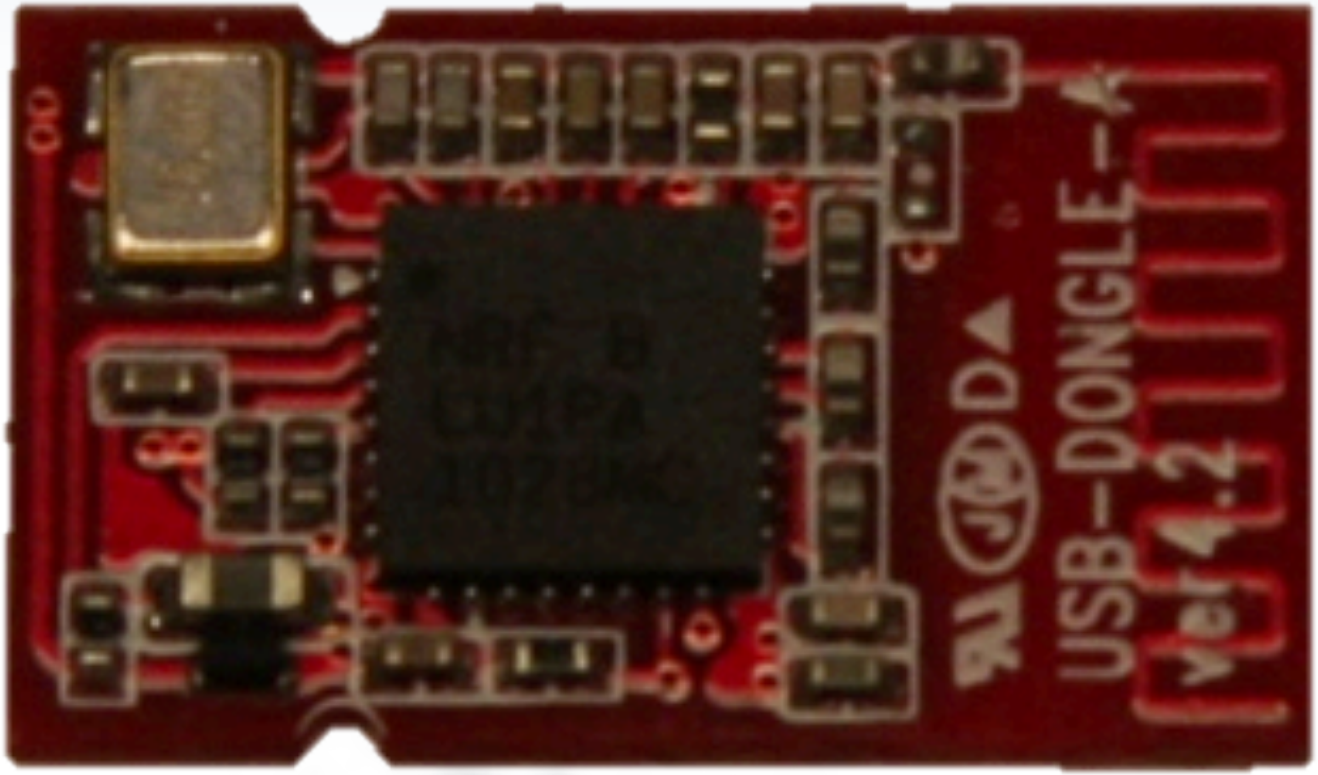
- Now only 16^7

BRUTE FORCE CONCLUSION

- We can whittle down to 7 hex digits
- 16 ^ 7 = 268,435,456 = 268 million
- On average we would have to search ~134 million keys
- Rate
  - HID report was sent every 0.032 seconds
- Conclusion
  - We can send 31.25 messages / second
  - 134M / 31.25 = 4,288,000 seconds
  - 49.6 days

MITIGATED!?!

» Cracked open the USB dongle



CHIPSET

- Nordic Semi nRF24LU1+ chip
  - http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24LU1P-OTP
- Uses OTP memory, programmable over SPI
- OTP memory also contains a 5-byte pseudo random Chip ID
- We could maybe read from SPI if the SPI readback hasn't been disabled (RDISMB)

SO WHAT COULD BE DONE BETTER?

- Boils down to an issue with trust boundaries and the storage of the secret
- The secret needs to be only on the token and host software, only components not considered compromised
- Both the token and the host need to authenticate to each other

A BETTER IMPLEMENTATION?

» Took a look at another device in the previous few weeks
  - Same idea, but with a generic wireless receiver
  - Multiple tokens to the same receiver
    - Sounds like a better solution, generic USB device
    - Secret sauce must be on wireless token!
» How do we identify what this secret sauce is?

THREAT MODELING

- Components
  - Generic receiver
  - Wireless token
- Installation
  - User plugs in USB receiver
  - User uses software to register tokens identified in range or enters token serial number printed on back
  - Token is now associated with a system user account
  - Can configure things like signal strength at which to lock the host

USE CASES

- Device registration
- Unlock
- Lock

USE CASE ANALYSIS

- Application protocol more complicated than before
- Registers as a generic USB device, doesn't utilize the HID device class
- No serial number-ish things in the initial registration of the USB dongle

ANALYSIS OF PROTOCOL MESSAGES

» Device lock

- When the device is locked, messages are sent
- Every 0.03 seconds heartbeat messages are sent with no data
- Every 2 seconds, a version string is also sent

2A	56	65	72	20	4C	53	32	2E	30	36
*	V	e	r		L	S	2	.	0	6

ANALYSIS OF PROTOCOL MESSAGES

- Device unlock
  - Like the Screen Keeper, the transmission of new USB messages signal that the token is in range
  - So what do these look like?

Time	M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]	M[7]	M[8]	M[9]
116.45	32	28	00	FD	FA	40	00	04	F8	CF
116.46	32	29	00	FD	FA	41	00	04	F8	CF
117.44	32	2A	00	FE	FC	42	00	04	F8	CF
117.44	32	2B	00	FE	FB	43	00	04	F8	CF
118.43	32	2C	00	FE	FB	44	00	04	F8	CF
118.43	32	2D	00	FE	FB	45	00	04	F8	CF
119.42	32	2E	00	FE	FC	46	00	04	F8	CF
	32	M[1]+1	00	??	??	M[5]+1	00	04	F8	CF

ANALYSIS OF PROTOCOL MESSAGES

» Message Fields

- M[0] = Message Type
  - 32 – Token in range
  - F2 – No token in range, just keeps heartbeat
- M[1] = USB Counter
  - Increments +1 every message
  - Rolls over to 00 at FF
  - Starts at 00 when USB receiver re-plugged

MESSAGE ANALYSIS

- M[2] = 00
- M[3 - 4] ???
- M[5] = Token Counter
  - Per token
  - Counts from 40 – 4F
  - Rolls over to 40
  - Different values when on/off token events occur
- M[6] = 00

MESSAGE ANALYSIS – M[7-9]

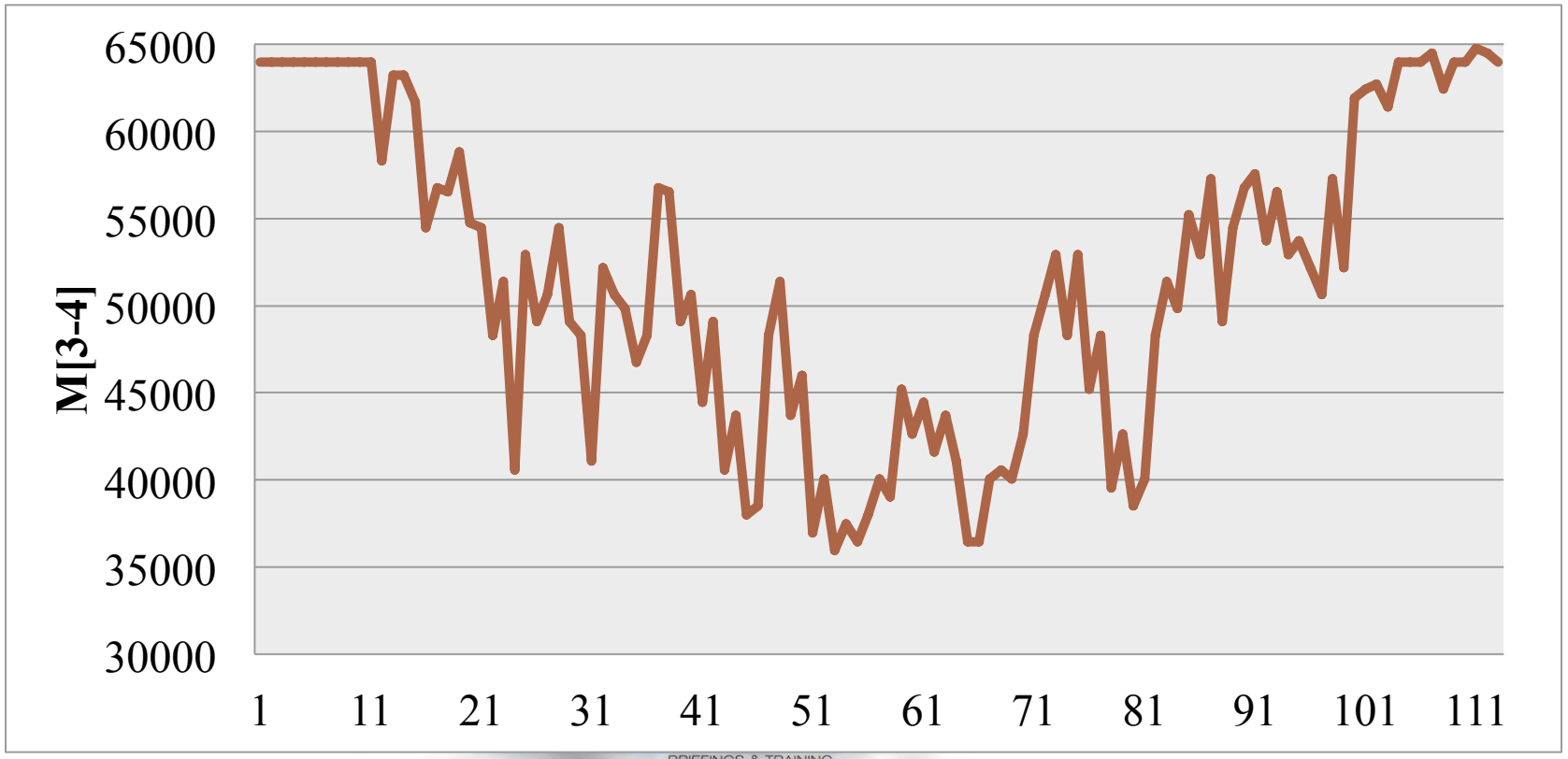
- 3-byte identifier
- Unique and static per token
- From Serial Number on token / seen in configuration UI:

Token ID	M[7]	M[8]	M[9]
325626	04	F7	FA
325839	04	F8	CF
331431	05	0E	A7

- Hrmmmmm
- Well this could be bad, but we still have 2 unknown bytes M[3-4]

MESSAGE ANALYSIS – M[3-4]

- Seems to vary randomly in each message
- But remember we can view the token signal strength in the UI!
- What happens to this value when I walk away and back to my desk?



MESSAGE ANALYSIS SUMMARY

- To unlock a machine, the only secret information is the M[7-9] (token ID)
- Is this really secret?
  - Its printed on the back of the token
  - Prior to registration of a token, we could get a list of all tokens in range
- This secret is broadcast to anyone in range!

A NEW PLAN OF ATTACK

- Sniff for tokens in range
- Wait for user to leave
- Plug in malicious device that replays USB registration (nothing unique) and replay messages with the last 3 bytes set as the sniffed ID
- Tested this in practice (not using Arduino, but USB traffic generator)

SO WHAT SHOULD IT DO

- Boils down to client-side control of the secret bits, disclosure of the password to anyone who cares
- You must assume the USB device has been compromised
- A secret needs to be established to authenticate between the host and the wireless token
- This secret needs to be secret to everything else, including the USB device

QUESTIONS? / CONTACT

- Greg Ose
- greg@nullmethod.com
- @gose1