Evil Mass Storage

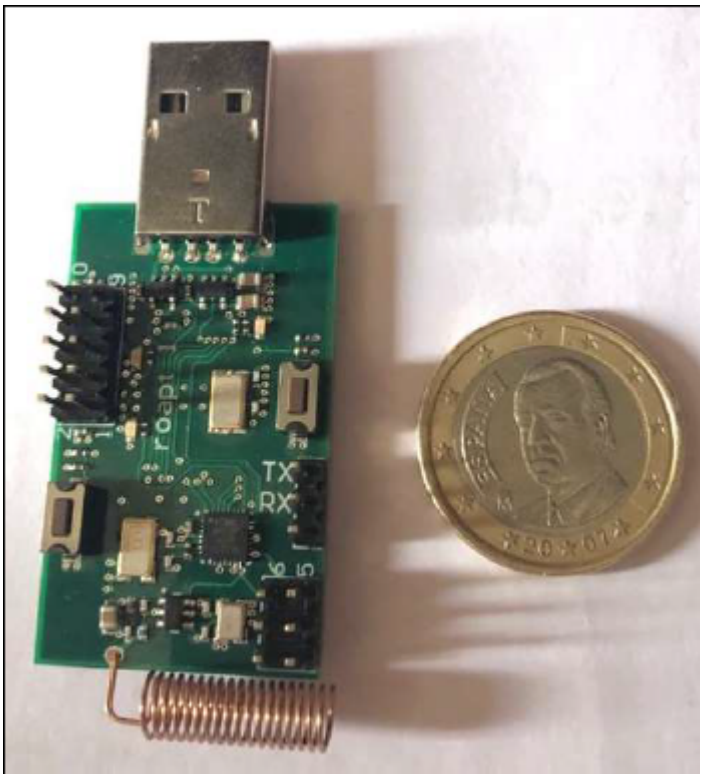
dreg 18 Sep, 21

Tags: Windows Security Implant

45 0 0

Introduction

Evil Mass Storage is a proof of concept USB composite device which demonstrates an end-to-end solution that infiltrates an isolated-offline-network and covertly extracts data over both radio frequency or close access covert storage while hiding from forensic analysis.



Evil Mas Storage Proof of Concept

This article discusses how Evil Mass Storage achieves its nefarious goals. The article is is broken down into Persistence, Exfiltration, Infection, Forensic Evasion, Targeting, Clean up, and Conclusion. Following this, there is a Developer Details sections with links to code and developer details.

Official post to ask about this project: https://www.driverentry.com/node/104

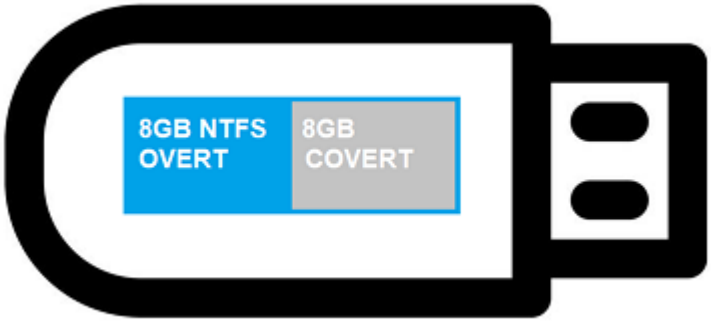
Persistence

Mass storage works out of the box with Windows - no need to install additional drivers. This device can have the form factor of a USB flash drive.

The malware can write to the mass storage device similar to writing to any drive. The microcontroller on Evil Mass Storage has been programmed to intercept writes, using the Small Computer System Interface (SCSI) interface from Windows to detect certain flags.

For example, write a file with a magic word EXFIL:DATA and the microcontroller will know to exfiltrate data over Radio Frequency (RD), thus not requiring an Internet connection to exfiltrate data.

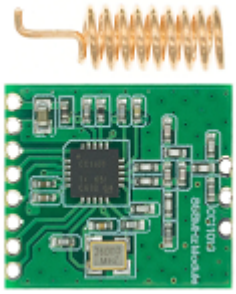
The Mass Storage device has a Micro SD. We divide the space on the Micro SD card into one overt partition which is formatted as FAT, and the remainder is reserved as covert unpartitioned space. In this way we can have one 16GB Micro SD card with 4+4GB on the overt FAT driver and 4GB as covert storage in unpartitioned space. The attached POC uses 2GB and 12GB as covert storage because SD CARD communication via SPI is very slow, smaller FAT means more speed.



Micro SD with covert partition

Remote Exfiltration

To exfiltrate data, Evil Mass Storage uses a commercial 433 MHz RF transmitter module. RF 433 is a pair of small RF electronic modules used to send radio signals to another device device. Evil Mass Storage does not have a 433 receiver module so information is sent several times to try to ensure the arrival.



RF 433 Module

Using radio frequency 433MHz ASK allows Evil Mass Storage to exfiltrate small amounts of information such as digital certificates without the need for an internet connection and at a considerable distance with great penetration (unlike 2.4GHz).

Close Access / Covert Storage Exfiltration

To covertly exfiltrate data back onto the the Evil Mass Storage Device, EvilMassStorage.exe can write encrypted data back onto the Micro SD card into the covert unpartitioned space. Even formatting the drive will not erase the data in the unpartitioned space. Windows is completely unaware of this space - only the Evil Mass Storage microcontroller knows about this space. The victim data written back onto the drive is protected in two ways. Firstly, the data is cloaked and completely hidden in the unpartitioned space. The only way to reveal the data is to send the microcontroller the right flags/key. Secondly, the data is encrypted. In the POC example a simple XOR is used.

Close exfiltration can be useful for large volumes of data or when remote exfiltration is not practical or prohibited due to use within a faraday cage or the like.

Evil Mass Storage Infection

There are a number of ways to execute the malware on the target device, including using zero-day vulnerabilities, however for the purposes of this article, Evil Mass Storage will register as a composite device. A composite device is a device composed of other devices. These devices address the case of hardware-level composition, in which a "device" (from the user's perspective) is implemented by several distinct hardware blocks. In short, Evil Mass Storage registers as a composite device encompassing a Mass Storage device and a Keyboard device. A composite device is very different to using a USB HUB.

Using a composite device allows Evil Mass Storage to register the Mass Storage device and the Keyboard device. Once plugged in, Evil Mass Storage does not know what drive letter has been assigned to it by Windows. To overcome this the Keyboard device starts executing the following commands: WIN+R, followed by a brute force of all driver letters looking for the malware to run: A:\EvilMassStorage.exe, B:\EvilMassStorage.exe, C:\EvilMassStorage.exe, D:\EvilMassStorage.exe, ...

Forensic Evasion

Once installed, EvilMassStorage.exe is executed in memory from the Micro SD card. To hide Evil Mass Storage from forensic inspection, the microcontroller talks directly to the Micro SD card and deletes the specific sectors. Because this is happening at the hardware layer it avoids any file handle locks in Windows allowing EvilMassStorage.exe to run in memory without any file. Even if the victim disconnects the device before the file has been deleted, the microcontroller will continue to delete the files once powered on (even without a computer). Evil Mass Storage only needs power current to finish the job.

As a safety net, if the Evil Mass Storage device is plugged in 'n' times without success the microcontroller will delete the files.

The microcontroller also randomizes PID, VID and serial disk to make it difficult to create a forensic profile.

EvilMassStorage.exe is also encrypted on the Micro SD. When the microcontroller reads the sectors it decrypts the buffers. The attached POC uses a simple XOR cipher.

Targeting

To target a specific machine, EvilMassStorage.exe can be installed "cloaked" on the Micro SD card. When Windows tries to read the location of EvilMassStorage.exe, Evil Mass Storage returns 0x00000000. For example, if EvilMassStorage.exe is located on sector 669 when Windows tries to read that sector, Evil Mass Storage returns 0 appearing to be a completely empty drive.

We then put a validator program, lets call this Happy.exe, to profile the target’s machine and, when it matches the victim, Happy.exe sends a flag to the microcontroller to "uncloak" and execute EvilMassStorage.exe.

Clean up

Once the attack is finished and EvilMassStorage.exe sends the flag to the microcontroller, the Keyboard device is disabled and the USB device changes VID, PID etc. At this point Evil Mass Storage will only work as a standard Mass Storage Device.

Conclusion

The Evil Mass Storage proof of concept shows an end-to-end solution to infiltrate an isolated-offline-network and covertly extract data over RF or physical access and hide from forensic analysis. Evil Mass Storage is a proof of concept designed for learning and highlighting security issues. The POC is a prototype and may crash your computer.

Development Details

All the source code its available in my github:
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2

Demo video (in Spanish): https://youtu.be/-K6MMVyKEv0?t=346

Steps to reproduce an attack:

1. The victim connects the USB device.

2. To make forensic work more difficult the device can randomize the VID/PID, serial disk and all relevant forensic-USB-data in each connection (the uploaded POC only changes this info in some stages):
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Descriptors.c#L112
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SCSI.c#L88
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SDCardManager.c#L150

3. The USB device is a USB composite device (not a USB HUB, again... read the books!!). Windows will detect it as a new keyboard and a new mass storage device.
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Descriptors.c#L193

4. The keyboard-device opens a run window (WIN + R) and starts to brute-force the assigned letter for mass storage in order to execute the stored .exe in our mass storage. This exe is not the malware, its the first stage. It retrieves useful information like the user name and writes it into the mass storage.
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/MassStorageKeyboard.c#L342
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/stage1/stage1/main.c#L90

5. The microcontroller gets the SCSI command and if the info is correct it resets the USB connection, and at this moment the malware is at the mass storage. This malware is decrypted (the POC uploaded is only a crap-XOR) using the information written in the mass storage... if evil mass storage is not connected to the target computer, malware won't be in it.

https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/MassStorageKeyboard.c#L317
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SDCardManager.c#L381
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SCSI.c#L370
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SDCardManager.c#L587

6. The malware is executed and the microcontroller removes all sectors of the malware from the SD.
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SDCardManager.c#L163
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/MassStorageKeyboard.c#L125

7. From this moment, the USB device will only work as a regular USB mass storage (keyboard part is removed). The VID-PID + other USB info gets changed again.
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Descriptors.c#L97
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Descriptors.c#L288

8. The malware exfiltrates data writing to the mass storage device and the microcontroller resends the information via rf 433MHz ASK (helped by a **atmega328p**). It also supports exfiltration via the SD card (encrypting the information first).
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilmass/evilmass/Lib/SDCardManager.c#L438
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/evilard/evilard.ino#L176
https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/stage2/stage2/main.c#L183

NOTE: This attack is only useful to steal a little info because SPI uses slow, RF 433MHz bandwidth...

I'm working on a new version. Currently experimenting with two ARM Cortex-M4 32 bit boards: **FRDM-K66F** MK66**FN2**M0VMD18 and **Teensy 3.6** MK66**FX1**M0VMD18 (Paul J Stoffregen + an awesome community pjrc + a lot of code).

What I am looking for:

Fast microcontroller ARM Cortex-M4 at 180 MHz

A real SDIO interface (fast SD access)

Cryptographic Acceleration & Random Number Generator (I want to use AES to encrypt/decrypt sectors...).

NOTE: ARM Cortex-M4 is very, very complex compared to AVR-8 bit, you should read this (hard) book:

The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors Third Edition by Joseph Yiu. ARM Ltd., Cambridge, UK. https://amzn.to/3tMbfzC

Teensy 3.6 ARM Cortex-M4 (NXP Kinetis MK66**FX1**M0VMD18) 180MHz:

ARM Cortex-M4 at 180 MHz

Float point math unit, 32 bits only

1024 Flash, 256K RAM, 4K EEPROM

USB device 12 Mbit/sec, USB host 480 Mbit/sec

64 digital input/output pins, 22 PWM output pins

25 analog input pins, 2 analog output pins, 11 capacitive sense pins

6 serial, 3 SPI, 4 I2C ports

1 I2S/TDM digital audio port

2 CAN bus

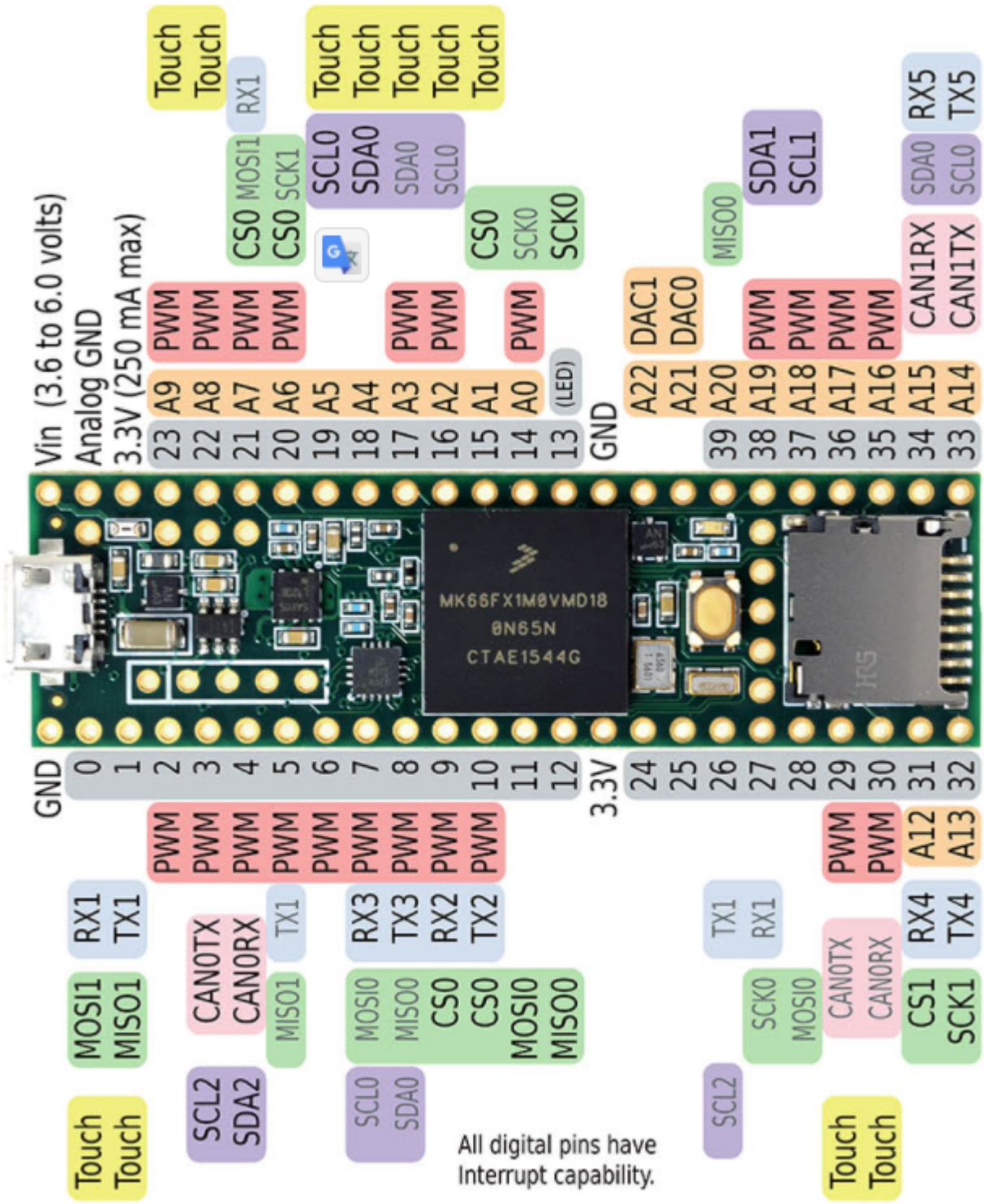
1 SDIO (4 bit) native SD Card port

32 general purpose DMA channels

Cryptographic Acceleration & Random Number Generator

RTC for date/time

https://www.pjrc.com/store/teensy36.html



FRDM-K66F (NXP Kinetis MK66**FN2**M0VMD18):

Performance

- **Up to 180 MHz ARM Cortex-M4 based core with DSP instructions and Single Precision Floating Point unit**

Precision Floating Point unit

System and Clocks

Multiple low-power modes to provide power optimization based on application requirements

Memory protection unit with multi-master protection

3 to 32 MHz main crystal oscillator

32 kHz low power crystal oscillator

48 MHz internal reference

Security

Hardware random-number generator

Supports DES, AES, SHA accelerator (CAU)

Multiple levels of embedded flash security

Timers

Four Periodic interrupt timers

16-bit low-power timer

Two 16-bit low-power timer PWM modules

Two 8-channel motor control/general purpose/PWM timers

Two 2-ch quad decoder/general purpose timers

Real-time clock

Human-machine interface

Low-power hardware touch sensor interface (TSI)

General-purpose input/output

Memories and memory expansion

Up to 2 MB program flash memory on nonFlexMemory devices with 256 KB RAM

Up to 1 MB program flash memory and 256 KB of FlexNVM on FlexMemory devices

4 KB FlexRAM on FlexMemory devices

FlexBus external bus interface and SDRAM controller

Analog modules

Two 16-bit SAR ADCs and two 12-bit DAC

Four analog comparators (CMP) containing a 6-bit DAC and programmable reference input

Voltage reference 1.2V

Communication interfaces

Ethernet controller with MII and RMII interface to external PHY and hardware IEEE 1588 capability

USB high-/full-/low-speed On-the-Go with on-chip high speed transceiver

USB full-/low-speed OTG with on-chip transceiver

Two CAN, three SPI and four I2C modules

Low Power Universal Asynchronous Receiver/ Transmitter 0 (LPUART0) and five standard UARTs

Secure Digital Host Controller (SDHC)

I2S module

https://www.nxp.com/docs/en/data-sheet/K66P144M180SF5V2.pdf

https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k66-k65-and-k26-mcus:FRDM-K66F

My pull request adding new ClassDriver MassStorageSDKeyboard Demo for LUFA - the Lightweight USB Framework for AVRs:

https://github.com/abcminiuser/lufa/pull/158

My talk in english (translated by who knows):

https://www.youtube.com/watch?v=5-ly4IyrD1Q

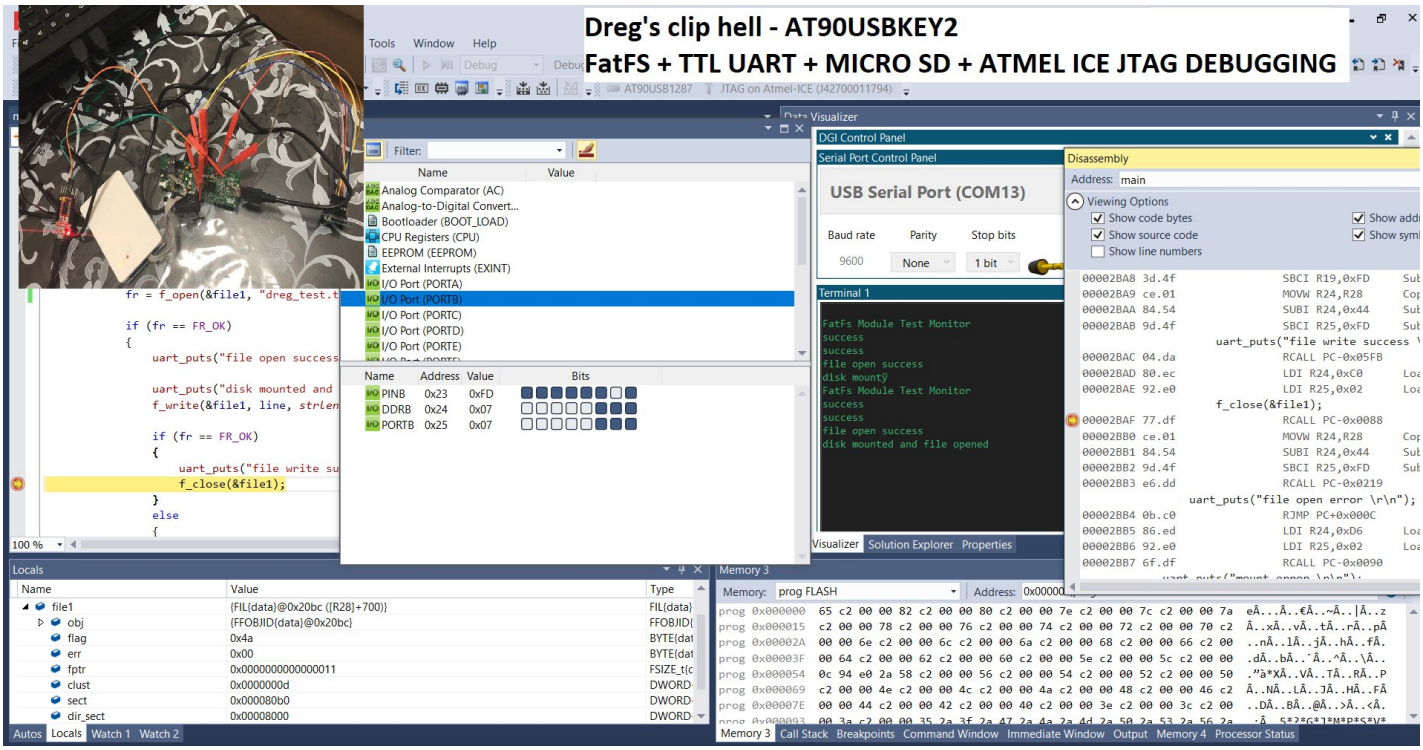
Just my own adaptation for mass storage sd card and keyboard for AT90USBKEY2:

https://github.com/David-Reguera-Garcia-Dreg/lufa-sdcard-mass-storagekeyboard-fatfs-AT90USBKEY2

Presentation:

https://github.com/David-Reguera-Garcia-Dreg/evilmass_at90usbkey2/blob/master/Roapt%20evil%20mass%20storage.pdf

FatFS + TTL UART + MICRO SD + ATMEL ICE JTAG DEBUGGING:



NOTE: I have no plans to make/sell more roapt v1 boards. I don't want to spend money on this xD.

ARM POC version is coming

Thanks to Daniel Brooks for his patience and for writing this awesome article.

Add new comment

Format Source

Text format

About text formats

Basic HTML

Save

Terms and Conditions

Data Privacy Policy

info@driverentry.com

DriverEntry © 2021. All Rights Reserved.

Contact