MINimum Failure

Stealing Bitcoins with Electromagnetic Fault Injection

Colin O’Flynn
C.T.O., NewAE Technology Inc.
Assistant Professor, Dalhousie University
My Dual Life

• C.T.O @ NewAE Technology Inc.
  • NewAE produces embedded tooling for hardware security validation
    • 700+ customers in 44 countries.
    • Started the open-source ChipWhisperer project.

• Assistant Professor @ Dalhousie University
  • Part of Electrical & Computer Engineering Dept.
  • Working on cybersecurity research + with local embedded startups.
The Talk With it All!! In no particular order:

• A true history of block chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass commercialization!
• Open-Source tool release!
• Fixing your crappy code!
Blockchain Background (1/2)
Blockchain Background (2/2)
Example: Blockchain Secures Voting Booth
History

• May 11, 2008: First concrete implementation. (H. Simpson)

• Oct 31, 2008: More well-known computer implementation. (S. Nakamoto)
• A true history of block chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass commercialization!
• Open-Source Tool Release!
• Fixing your crappy code!
Let’s Embed Those Blockchains
Why a Hardware Wallet?

- Hardware wallet shows what *actual request* is (display).
- Hardware wallet signs request.
- Hardware wallet prevents read-out of “private key” used to sign any request.
Trezor Features

Security measures of Trezor.

- **Firmware verification.**
  The bootloader verifies the firmware signature. The device only runs if the firmware is correctly signed by SatoshiLabs. Otherwise, the Trezor warns you.

- **Protected key operations.**
  Operations with private and public keys are only allowed after user authentication via PIN.

- **Additional passphrase support.**
  Trezor supports BIP39 passphrases, which are never stored or remembered on the device.

- **Reliable backup & recovery.**
  Your recovery seed protects you against theft, loss or destruction of your device. Simply restore the recovery seed, and your wallet is back.

- **Ultrasound hardware seal.**
  Trezor hardware case is ultrasonically welded, making it difficult to be restored after breakage.

- **Secure update procedure.**
  The bootloader erases memory on firmware update and requires it only if the firmware signature is verified in memory.

- **Write-protected bootloader.**
  The bootloader is write protected and safeguarded by the Memory Protection Unit.
Example: Ultrasonic Seal

<Hopefully my camera works>
Why Pick on Trezor?

• Trezor is Open Source
  • Anyone can validate the source code, modify it, etc.
  • Lessons learned on Trezor can be applied outside of bitcoin wallets.
    • Embedded systems, IoT, automotive, etc.

• This problem I’m disclosing has been fixed with issued firmware patch.
About Bitcoin Recovery Seeds

From Trezor documentation:

Understanding the recovery seed.

The recovery seed is a crucial element for the security of your Trezor hardware wallet. If your device is lost, damaged or stolen, you can use your recovery seed to restore access to your entire wallet, passwords and other data associated with it. The process is simple; you only have to enter the words of your seed into your new Trezor device. (You may also use any other wallets or applications that use the same standard as the Trezor.)

Important Caveat: You can also password-protect this seed, but it’s not done by default. If password protected, the attacks I’m going to describe don’t work!
What’s inside the Trezor?
How to Get Recovery Seeds?

Dmitry Nedospasov
Dmitry Nedospasov is a hardware design and security engineer, security researcher, trainer, speaker and reverse-engineer. In 2014 Dmitry received his PhD (Dr.-Ing.) in ICSecurity at TU Berlin.

Josh Datko
Josh Datko is an embedded systems engineer, security researcher and former submarine officer. Josh is best known for his 2017 presentation on insecures in cryptocurrency hardware wallets.

Thomas Roth
Thomas Roth was named as one of the 30 under 30 in Technology by the Forbes Magazine. His main focus is on mobile and embedded systems with published research on topics like TrustZone, payment terminals, and embedded security.
Wallet.Fail

- Vulnerabilities on several Bitcoin Wallets.
- For Trezor specifically:
  - How to copy recovering seed out of a backup stored in SRAM.
  - Required physical access to PCB (open enclosure), but can be performed with high reliability.
My Inspiration – Wallet.Fail Talk

|||~
pppp
>>>>>? 
[11]$ 

D
DEC 17, 2018 6:24 PM

Dmitry
Those 24 words are the seed
DEC 17, 2018 6:25 PM

Dmitry
And you can fucking show it with strings
DEC 17, 2018 6:25 PM

Dmitry
it's so epic
DEC 17, 2018 6:25 PM

Oh wow! So like rdp1 alone is su
DEC 17, 2018 6:25 PM
How else to get recovery seed?
Recovery seed, device PIN saved here!

What's going on up in here?

<table>
<thead>
<tr>
<th>Sector</th>
<th>Address Range</th>
<th>Size</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x08000000 - 0x08003FFF</td>
<td>16 KiB</td>
<td>bootloader code</td>
</tr>
<tr>
<td>1</td>
<td>0x08004000 - 0x08007FFF</td>
<td>16 KiB</td>
<td>bootloader code</td>
</tr>
<tr>
<td>2</td>
<td>0x08008000 - 0x0800BFFF</td>
<td>16 KiB</td>
<td>metadata area</td>
</tr>
<tr>
<td>3</td>
<td>0x0800C000 - 0x0800FFFF</td>
<td>16 KiB</td>
<td>metadata area</td>
</tr>
<tr>
<td>4</td>
<td>0x08010000 - 0x0801FFFF</td>
<td>64 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>5</td>
<td>0x08020000 - 0x0803FFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>6</td>
<td>0x08040000 - 0x0805FFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>7</td>
<td>0x08060000 - 0x0807FFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>8</td>
<td>0x08080000 - 0x0809FFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>9</td>
<td>0x080A0000 - 0x080BFFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>10</td>
<td>0x080C0000 - 0x080DFFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
<tr>
<td>11</td>
<td>0x080E0000 - 0x080FFFFF</td>
<td>128 KiB</td>
<td>application code</td>
</tr>
</tbody>
</table>
USB Descriptors
9.4.3 Get Descriptor

- This request returns the current device configuration value.

<table>
<thead>
<tr>
<th>bmRequest Type</th>
<th>bRequest</th>
<th>wValue</th>
<th>wIndex</th>
<th>wLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Direction 0x80

GET DESCRIPTOR

Descriptor Type (Hi) and Descriptor Index (LO)

- Low Byte: Descriptor Index

Zero or Language ID (9.6.7)

- String Descriptors, Language ID: Others: Zero

Descriptor Length

- The number of bytes to return.
Send MIN() of wLength & Struct Length

```c
} else if ((req->bmRequestType & USB_REQ_TYPE_RECIPIENT) == USB_REQ_TYPE_INTERFACE) &&
  (req->wIndex == WINUSB_REQ_GET_EXTENDED_PROPERTIES_OS_FEATURE_DESCRIPTOR) &&
  (usb_descriptor_index(req->wValue) == winusb_wcid.functions[0].bInterfaceNumber)) {

  *buf = (uint8_t*)(&guid);

  *len = MIN(*len, guid.header.dwLength);

  status = USBD_REQ_HANDLED;
```
Checking Implementation Details
Open Source FTW :) 

- I can be lazy since firmware is fully known & I can modify it even.

- We can ‘simulate’ the glitch to ensure things will work as we expect.
Validating This Will Work (1/2)

<table>
<thead>
<tr>
<th>Index</th>
<th>m:s.ms.us.ns</th>
<th>Len</th>
<th>Err</th>
<th>Dev</th>
<th>Ep</th>
<th>Record</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0:00.000.000.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[02/06/19 00:45:55]</td>
</tr>
<tr>
<td>1</td>
<td>0:00.000.000.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0:00.000.633.500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0:23.658.183.950</td>
<td>146</td>
<td>22</td>
<td>00</td>
<td>Ep</td>
<td>Control Transfer</td>
<td>92 00 00 00 00 00 01 05 00 01 00 08</td>
</tr>
<tr>
<td>24</td>
<td>0:06.791.576.583</td>
<td>146</td>
<td>22</td>
<td>00</td>
<td>Ep</td>
<td>Control Transfer</td>
<td>92 00 00 00 00 00 01 05 00 01 00 08</td>
</tr>
<tr>
<td>45</td>
<td>0:03.879.450.166</td>
<td>146</td>
<td>22</td>
<td>00</td>
<td>Ep</td>
<td>Control Transfer</td>
<td>92 00 00 00 00 00 01 05 00 01 00 08</td>
</tr>
<tr>
<td>66</td>
<td>1:58.972.722.583</td>
<td>65535</td>
<td>22</td>
<td>00</td>
<td>Ep</td>
<td>Control Transfer</td>
<td>92 00 00 00 00 00 01 05 00 01 00 08</td>
</tr>
<tr>
<td>4171</td>
<td>0:11.333.695.616</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[02/06/19 00:48:40]</td>
</tr>
</tbody>
</table>

Expected response (146 bytes)

Use debugger to skip MIN() check.
Validating This Will Work (2/2)
Generated and Induced Magnetic Field
Example of Coils
EMFI Setup

Force bootloader mode.

EMFI Tip.

ChipSHOUTER
Triggering EMFI

- **TotalPhase Beagle 480**
- **Hardware trigger on USB physical-layer packets!**
Beagle480 (trigger on USB packet)

ChipWhisperer

Target

USB Switch (hard reset required due to hard fault vectors)
PhyWhisperer-USB

- Cheapish ($250 USD) triggering device for USB physical-layer packets.
- Works as a sniffer too.
- Open-source HW/SW.
- Needs separate fault injection driver - ChipSHOUTER (EMFI), ChipWhisperer (Voltage glitching), or Mux (Voltage glitching)

https://github.com/newaetech/phywhispererusb
PhyWhisperer-USB

- USB 2.0 LS/FS/HS Phy
- Can switch power on/off to target (critical for glitching).
- Interpose with a real (external) USB host.

Stuff that works also:
- Sniffing USB packets (Wireshark as front-end).

Stuff you could do (but I’m way too lazy for):
- Generate USB packets.
• A true history of block chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass commercialization!
• **Open-Source Tool Release!**
• Fixing your crappy code!
PhyWhisperer on CrowdSupply

- Everyone loves OSHW.
- Nobody wants to actually build it.
- Solution: Search “PhyWhisperer” to see crowd funding campaign!
• A true history of block-chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass-commercialization!
• Open-Source Tool Release!
• Fixing your crappy code!
EMFI Demo

<Hopefully my camera works>
• A true history of block chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass commercialization!
• Open-Source Tool Release!
• Fixing your crappy code!
Wait – What About Authentication Tokens?

• Since submitting & preparing this I realized several other interesting targets, and rather than a second talk I give you...
How do these things work?

Authentication Token
- Private Key
- Public Key
- Signing Operation

Website/App
- Public Key
- Challenge
- Response
- Verify

Registration Phase (Once)
Authentication Phase (Many)
Who’s using these tokens?

Password-less protection

Reduce your risk exposure with password alternatives
Example: Solo Keys Authentication Token

TIP: You can find ECC private keys in memory if you have public key to compare with. If memory layout is unknown, we can register a new service (to get public key from device), perform attack, and figure out if any private keys in our memory dump.

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x200000bc</td>
<td>USBD_HID_Desc</td>
<td>HID Descriptor.</td>
</tr>
<tr>
<td>0x20001b0c</td>
<td>_signing_key</td>
<td>Pointer to key.</td>
</tr>
<tr>
<td>0x20001b10</td>
<td>master_secret</td>
<td>HMAC secret.</td>
</tr>
<tr>
<td>0x20001b50</td>
<td>privkey.8369</td>
<td>ECC Private Key.</td>
</tr>
<tr>
<td>0x20001b70</td>
<td>sha256_ctx</td>
<td>SHA256 context.</td>
</tr>
</tbody>
</table>

```c
else if( (req->wValue >> 8) == HID_DESCRIPTOR_TYPE){
    pbuf = USBD_HID_Desc;
    len = MIN(USBD_HID_DESC_SIZ, req->wLength);
}
```
What about other FIDO2 Keys?

e.g., Yubikey 5 *should* reach this

<table>
<thead>
<tr>
<th>SAMPLE DEVICE HARDWARE &amp; SOFTWARE REQUIREMENTS</th>
<th>DEFENDS AGAINST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection against chip fault injection and invasive attacks</td>
<td>L3+ Chip level attacks on captured devices</td>
</tr>
<tr>
<td>Circuit board potting, package on package memory, encrypted RAM…</td>
<td>L3 Circuit board attacks on captured devices</td>
</tr>
<tr>
<td>Device must support allowed Restricted Operating Environment (ROE) (e.g., TEE, Secure Element…), or intrinsically be an ROE (e.g., a USB token or Smart Card…)</td>
<td>L2+ Device OS compromise</td>
</tr>
<tr>
<td>Any device HW or SW</td>
<td>L1+ White Box Cryptography to defend against OS compromise</td>
</tr>
<tr>
<td></td>
<td>L1 Phishing, server credential breaches and MITM attacks (better than passwords)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIDO AUTHENTICATOR CERTIFICATION EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3+ USB U2F Token built on a CC-certified Secure Element Certification: L3+</td>
</tr>
<tr>
<td>L3 USB U2F Token built on a basic simple CPU, OS is certified. Good physical anti-tampering enclosure Certification: L3</td>
</tr>
<tr>
<td>L2+ FIDO2 making use of the Android keystrokes, Keystone runs in a TEE that is certified at L2+ Certification: L2+</td>
</tr>
<tr>
<td>L2 UAF implemented in a TA running on a certified TEE with PDP memory Certification: L2</td>
</tr>
<tr>
<td>L1+ L1+ U2F in downloadable app using white box and other techniques Certification: L1+</td>
</tr>
<tr>
<td>L1+ Downloaded app making use of Touch ID on iOS Certification: L1</td>
</tr>
<tr>
<td>L1 FIDO2 making use of the Android keystrokes. Keystone is not certified Certification: L1</td>
</tr>
</tbody>
</table>

https://fidoalliance.org/certification/authenticator-certification-levels/
• A true history of block chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass commercialization!
• Open-Source Tool Release!
• Fixing your crappy code!
How Concerned Should You Be?

Is your device using USB?

- YES
  - Is sensitive data in same memory space as USB?
    - YES
      - Within 64KB of any USB data?
        - YES

- NO
  - NO
  - NO

NO
How to Fix It?

• Why can you send back 64K of memory? No descriptors are that big!

• Devices have memory protection.
  • We can armour the sensitive data with invalid memory segments, or “disable” memory segments when read-out shouldn’t be needed.

• Move memory layout around so we can’t read into sensitive data.
  • Less useful as another fault might let us corrupt a pointer instead.

• Encrypt data in-place.
  • Don’t allow a “dumb” dump to figure out this critical data!
Trezor Fixes

• The disclosed problem has been fixed in latest firmware patch.

• The disclosed problem did not affect people using passphrases.

Solo Key Fixes

- The disclosed problem has been fixed in GIT!
• A true history of block chain!
• Me gambling with 0.3 BTC!
• Physical give-aways!
• Attacking bitcoin wallets & stealing funds!
• Stealing authentication credentials!
• Crass commercialization!
• Open-Source Tool Release!
• Fixing your crappy code!
Takeaways

• Tamper-resistant enclosures aren’t enough when discussing “near-physical” attacks. Commercially available EMFI tools exist (see - ChipSHOUTER) and can be purchased sometimes...

• If implementing a USB device, validate your response size makes sense to avoid many attacks!

• Testing against EMFI is useful to understand vulnerability – but you can do some testing with simulation/emulation and through code review.
Let’s Do It!

- I’ve got PhyWhisperer-USB PCBs here for you 😊 Join the CrowdSupply and make the full thing happen!

- See my WOOT’19 paper (linked from oflynn.com) & White Paper!

Blog   oflynn.com
Twitter @colinoflynn
Email  colin@oflynn.com
Dal Email coflynn@dal.ca
Company newae.com

I’m terrible on response times right now, please don’t take it personally if it takes weeks++ :/
For More, See:

• WOOT 2019 Paper: https://www.usenix.org/conference/woot19/presentation/oflynn

• My blog post: http://colinoflynn.com/2019/03/glitching-trezor-using-emfi-through-the-enclosure/

• PhyWhisperer-USB repo: https://github.com/newaetech/phywhispererusb