Through the Eyes of a Hardware Hacker

Looking at the Engineering Lifecycle from an Adversarial Perspective

Joe Grand aka Kingpin

GRAND IDEA STUDIO

GRAND IDEA STUDIO

Through the Eyes of a Hardware Hacker

- Hardware Hacking Overview
- Selected Hardware Attack Vectors
  - Exposed Interfaces
  - Firmware Extraction / Manipulation
  - Fault Injection
  - Supply Chain / Espionage
- Best Practices

Hardware Hacking Overview

Why Hardware Hacking?

- Cloning/counterfeiting
  - Specific theft of information/data/IP for marketplace advantage
- Theft of service/PII
  - Malicious intent, malware, data harvesting for future use
- Privilege escalation
  - Defeat protection measures/gain increased control of a system
  - Use as an entry point into a network to further an attack
- Forensic analysis/intelligence
  - What is that hardware? Who designed it? How to extract data?
- Security competency/product integrity
  - Test hardware security/process for failures/weaknesses
  - Ensure (sub)system has not been tampered with

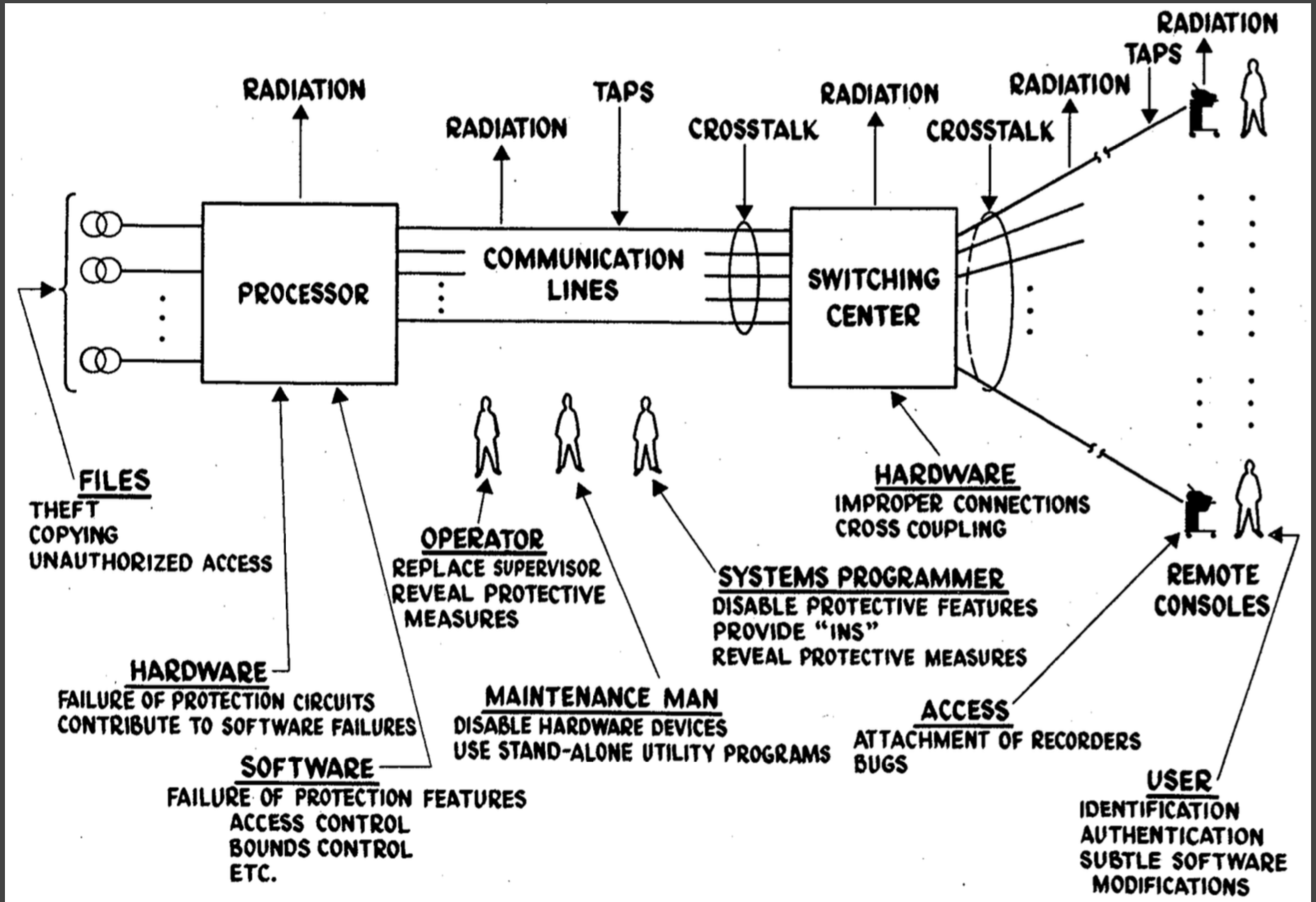
Hardware Hacking Process

- Information Gathering
  - Obtaining information about the target
- Teardown
  - Product disassembly, component/subsystem ID
- Buses & Interfaces
  - Signal monitoring/analysis/emulation/fault injection
- Memory & Firmware
  - Extract/modify/analyze/reprogram code or data
- Chip-Level
  - Silicon die modification/data extraction

Approaches

- Attack the hardware directly
  - Find a vulnerability and exploit it for access to system/data
- Attack *with* hardware
  - Mount an attack from the subverted hardware
  - Use hardware as a stepping stone to further attacks
- Implant the hardware
  - Add malicious hardware/firmware into an otherwise operable system

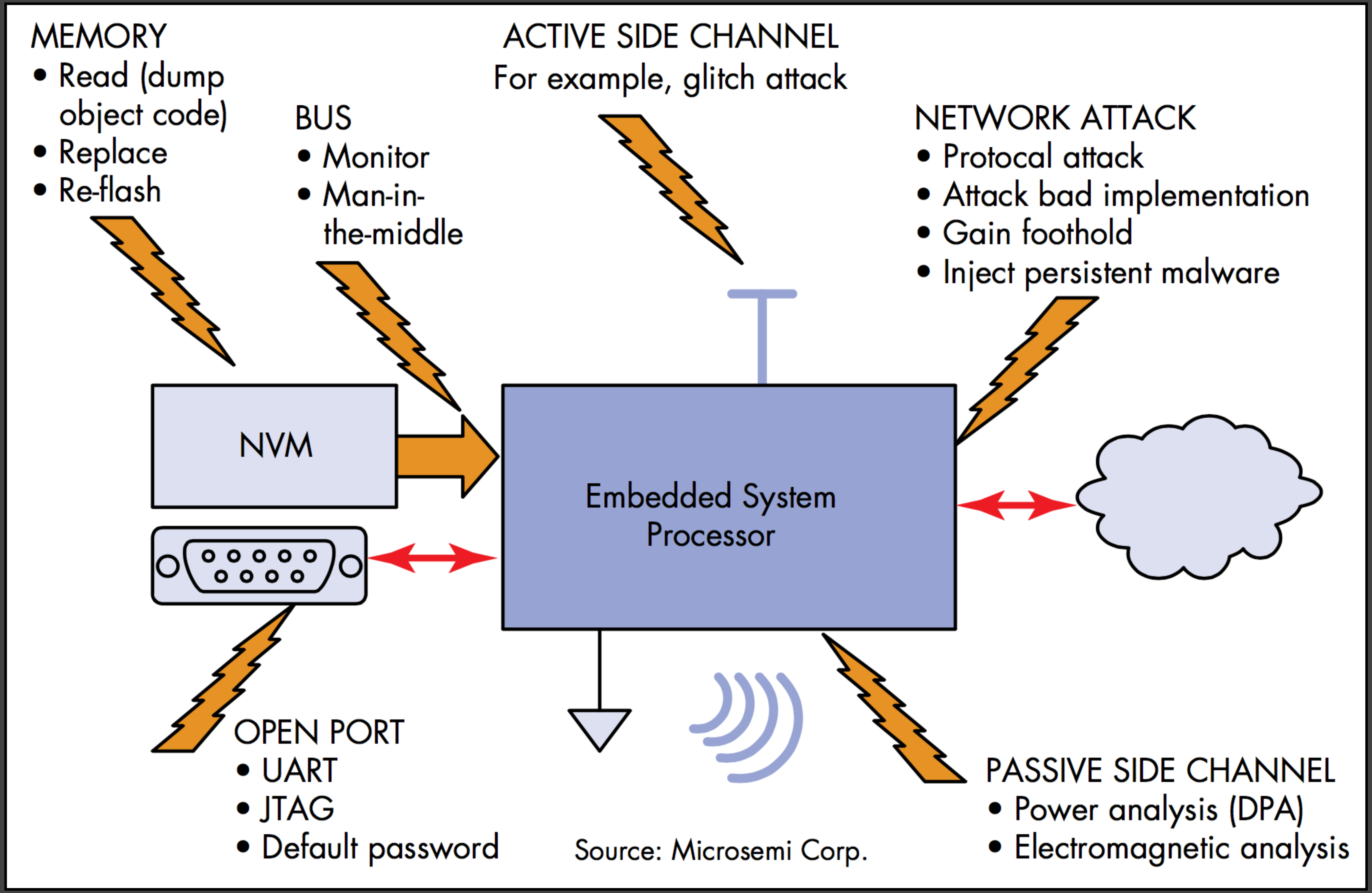
Common Attack Surfaces (1970)



Security Controls for Computer Systems: Report of the Defense Science Board Task Force on Computer Security, February 11, 1970

GRAND IDEA STUDIO

Common Attack Surfaces (Now)



The Biggest Security Threats Facing Embedded Designers, Electronic Design, June 2016

GRAND IDEA STUDIO

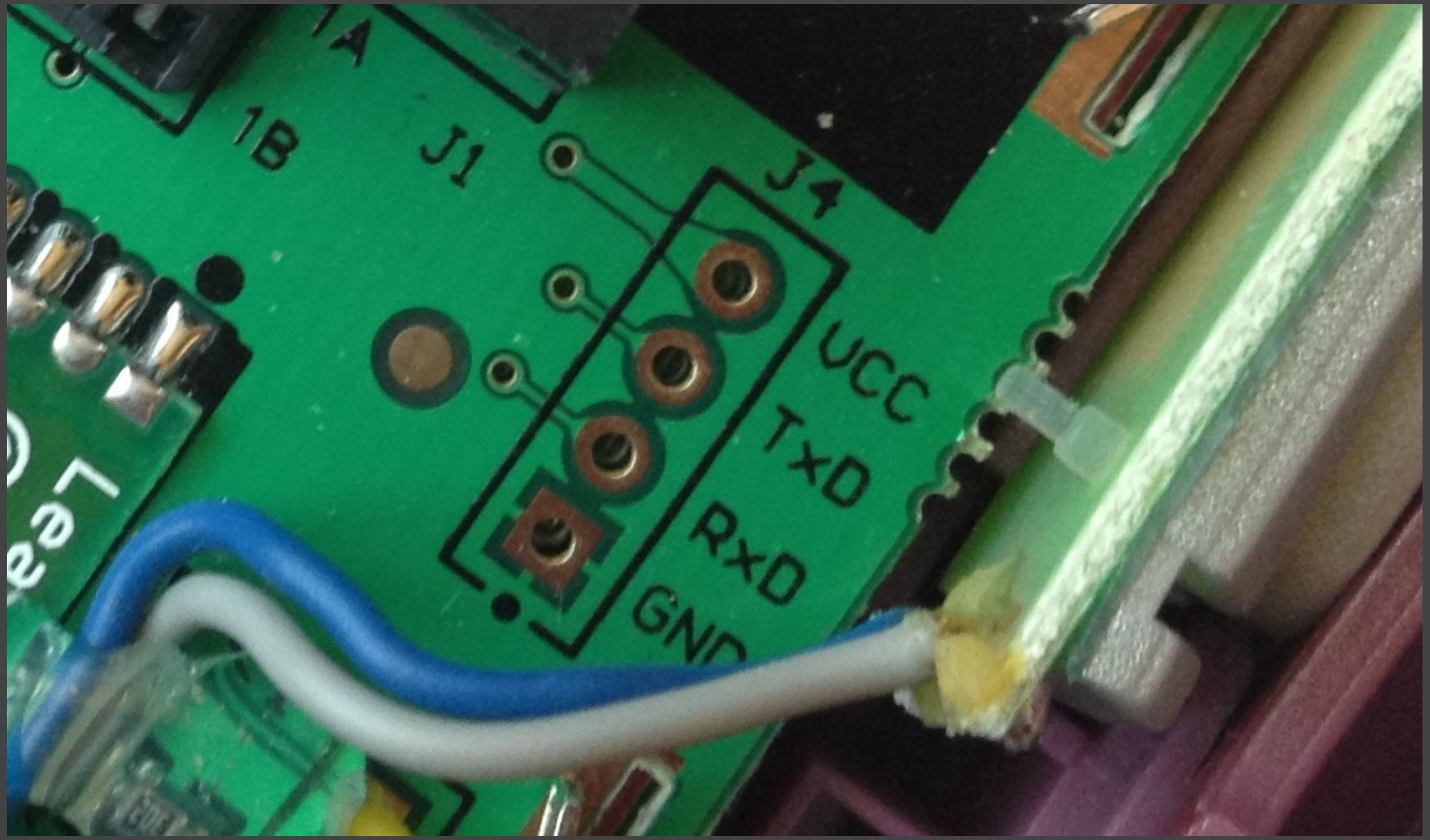
Threat Model / Risk Analysis

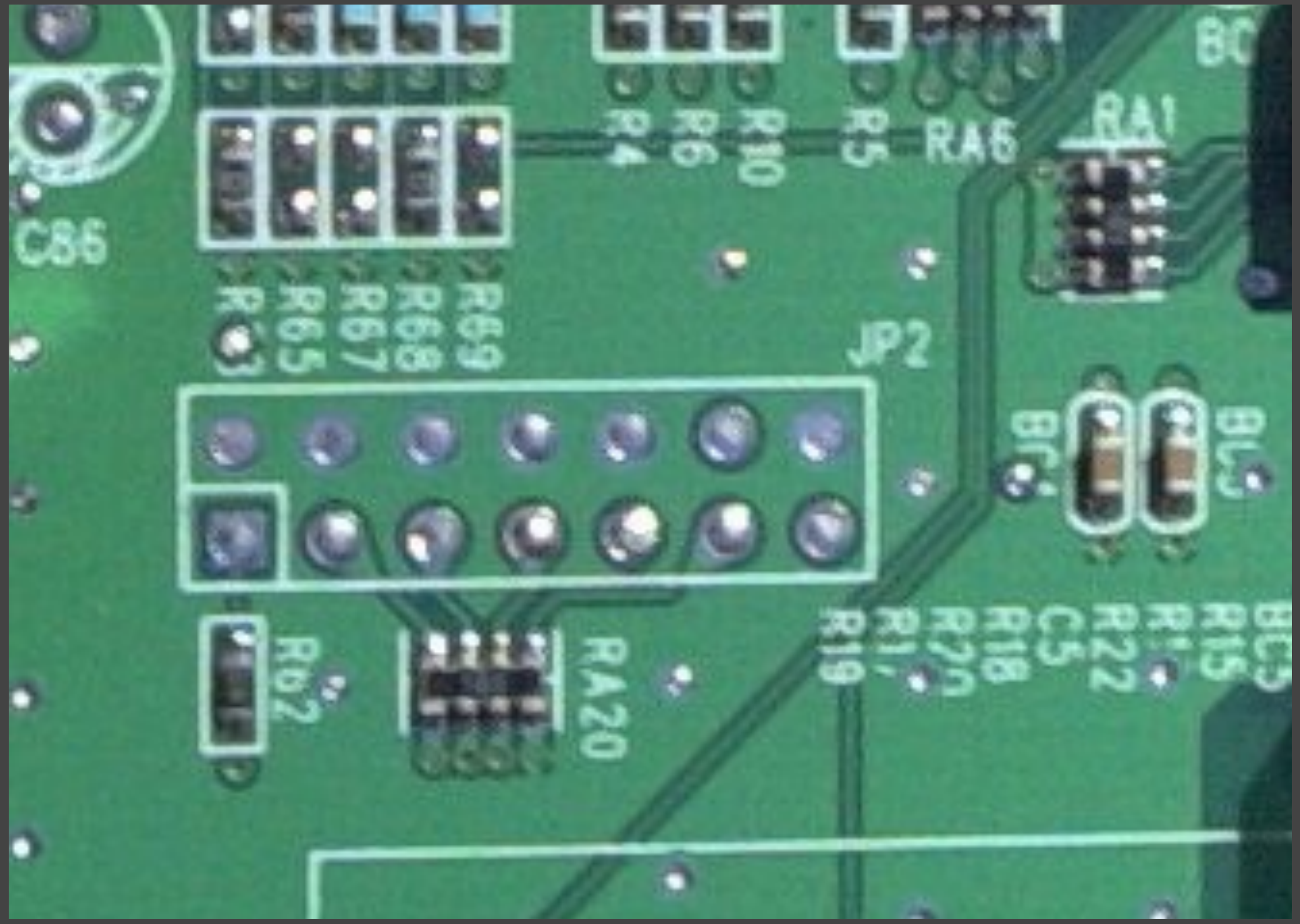
- You must understand your risk before you can protect yourself
  - What needs to be protected
  - Why it is being protected
  - Who you are protecting against (define your adversary)
- What features are absolutely necessary for system functionality?
  - Each new feature increases attack landscape
- Identify single points of failure across the lifecycle
- Security v. convenience/reality

Exposed Interfaces

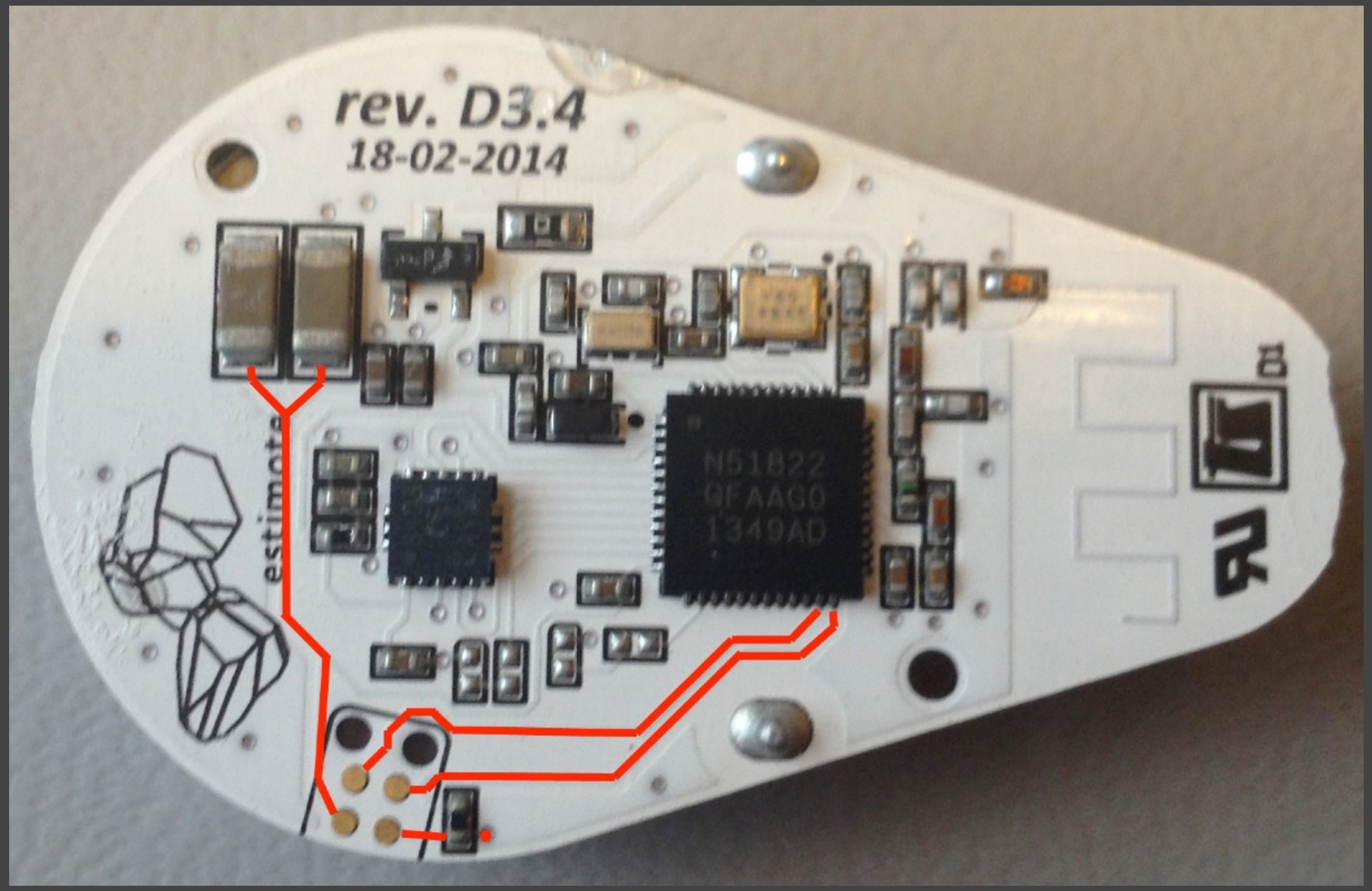
Exposed Interfaces

- Chip-to-chip (internal) or to the outside world (external)
- Signal manipulation
- Programming/debugging capabilities
- Many interfaces transmit data in the clear w/ no authentication
  - Engineers may not realize/be aware/care that data streams can be monitored/manipulated
  - Most chips do not have native support for secure communication

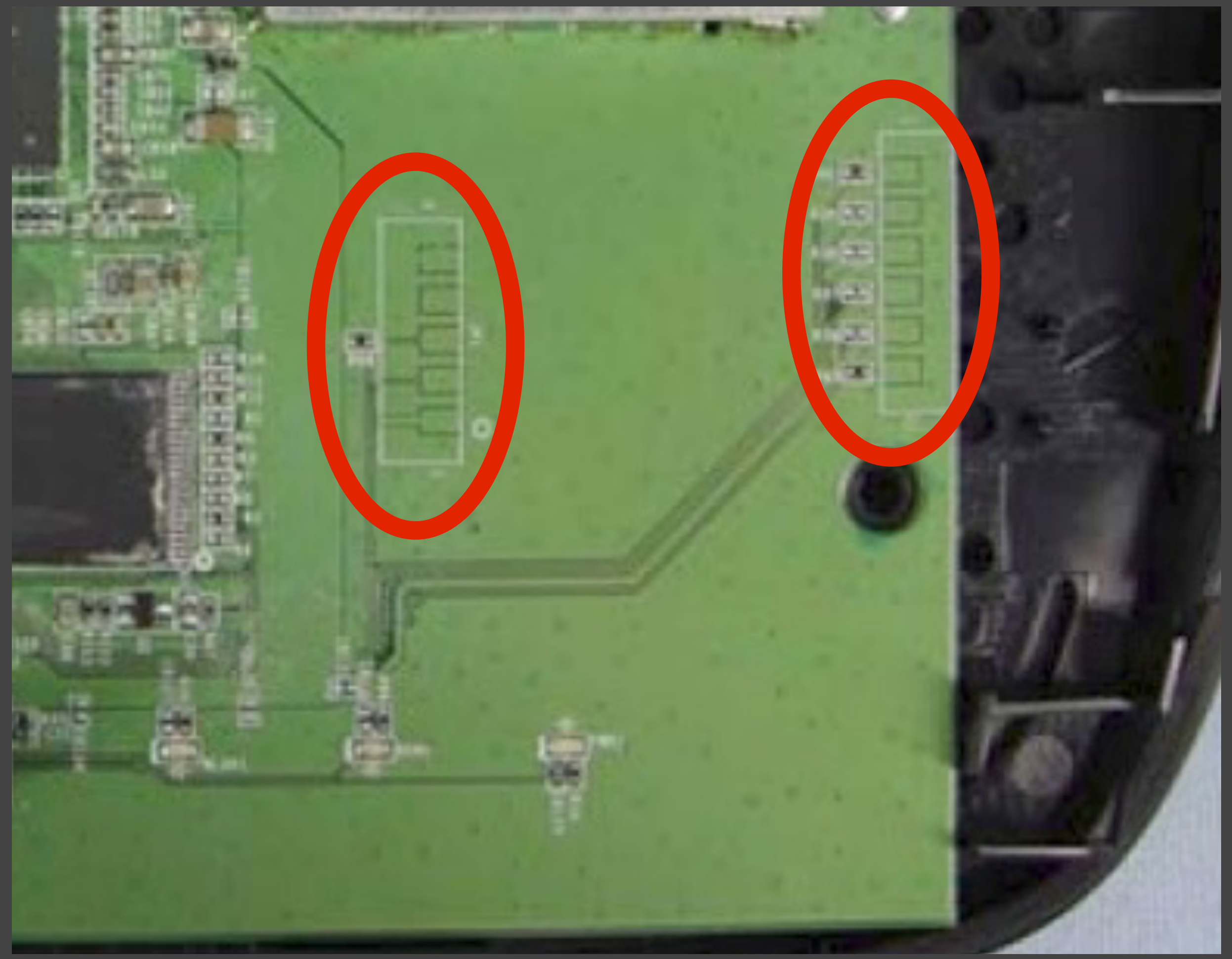

1B
J1
J4
VCC
TxD
RxD
GND

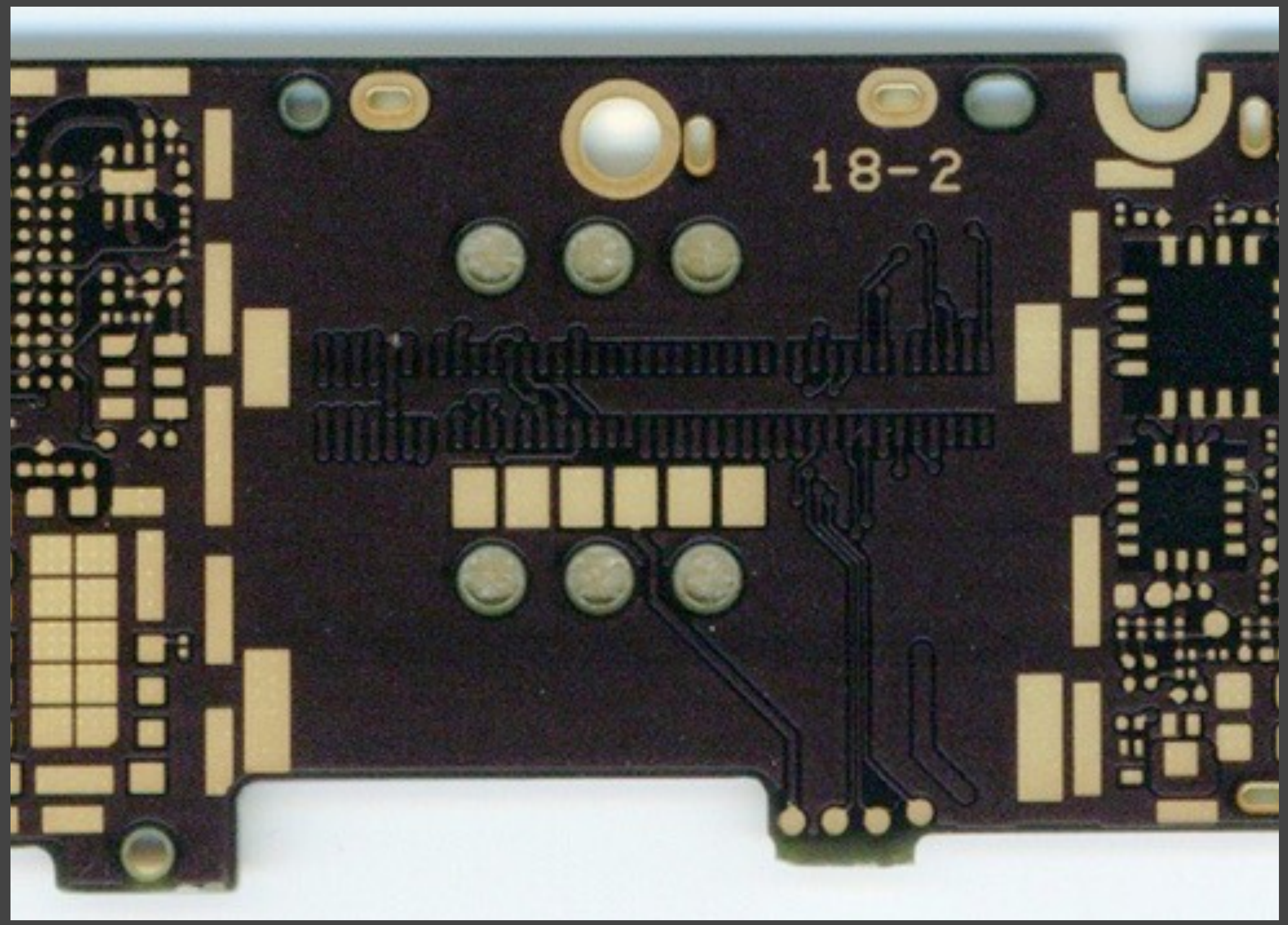

C86
R4
R6
R10
R5
RA6
RA1
R65
R67
R68
R69
JP2
RA20

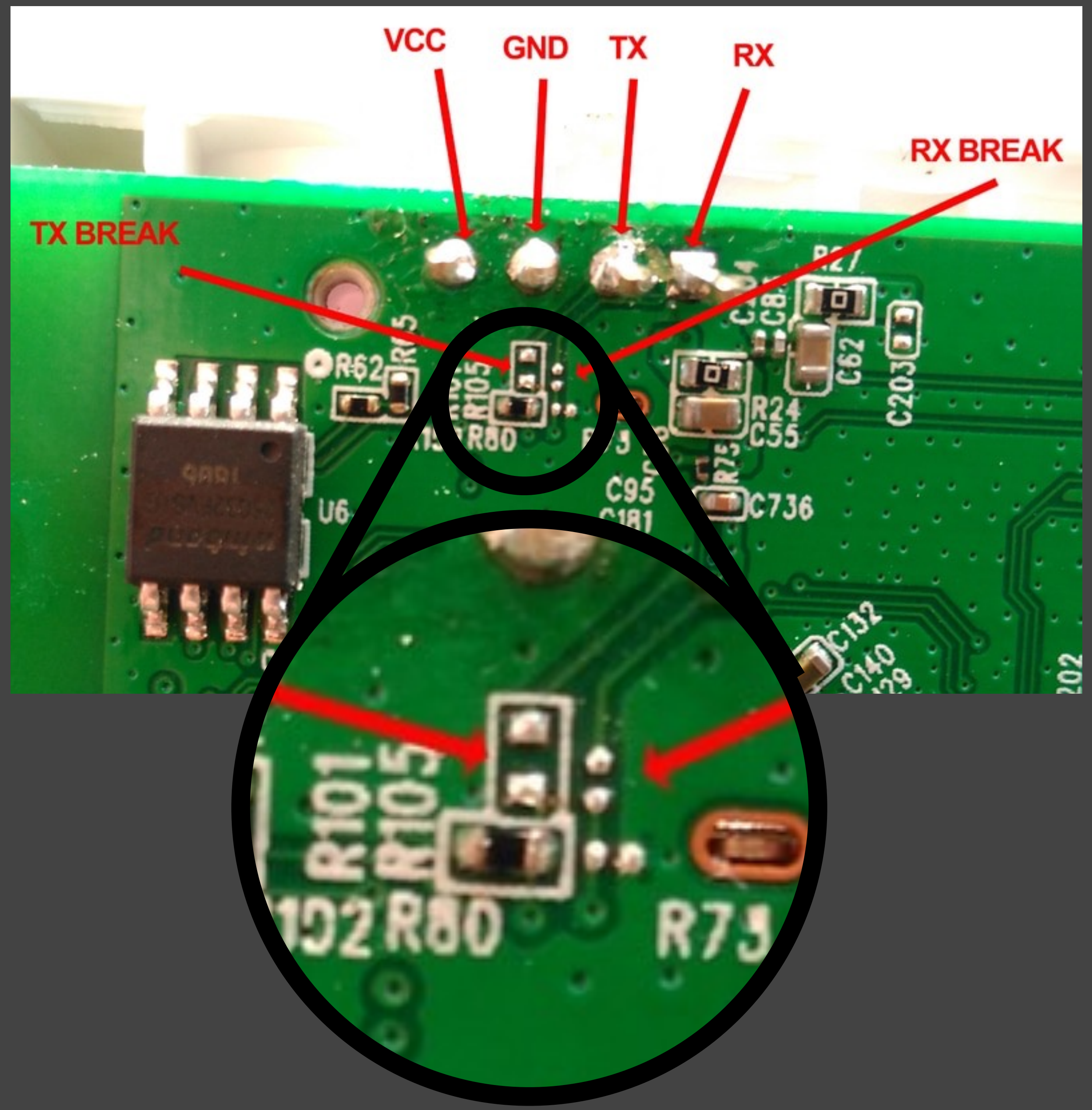

Rev A

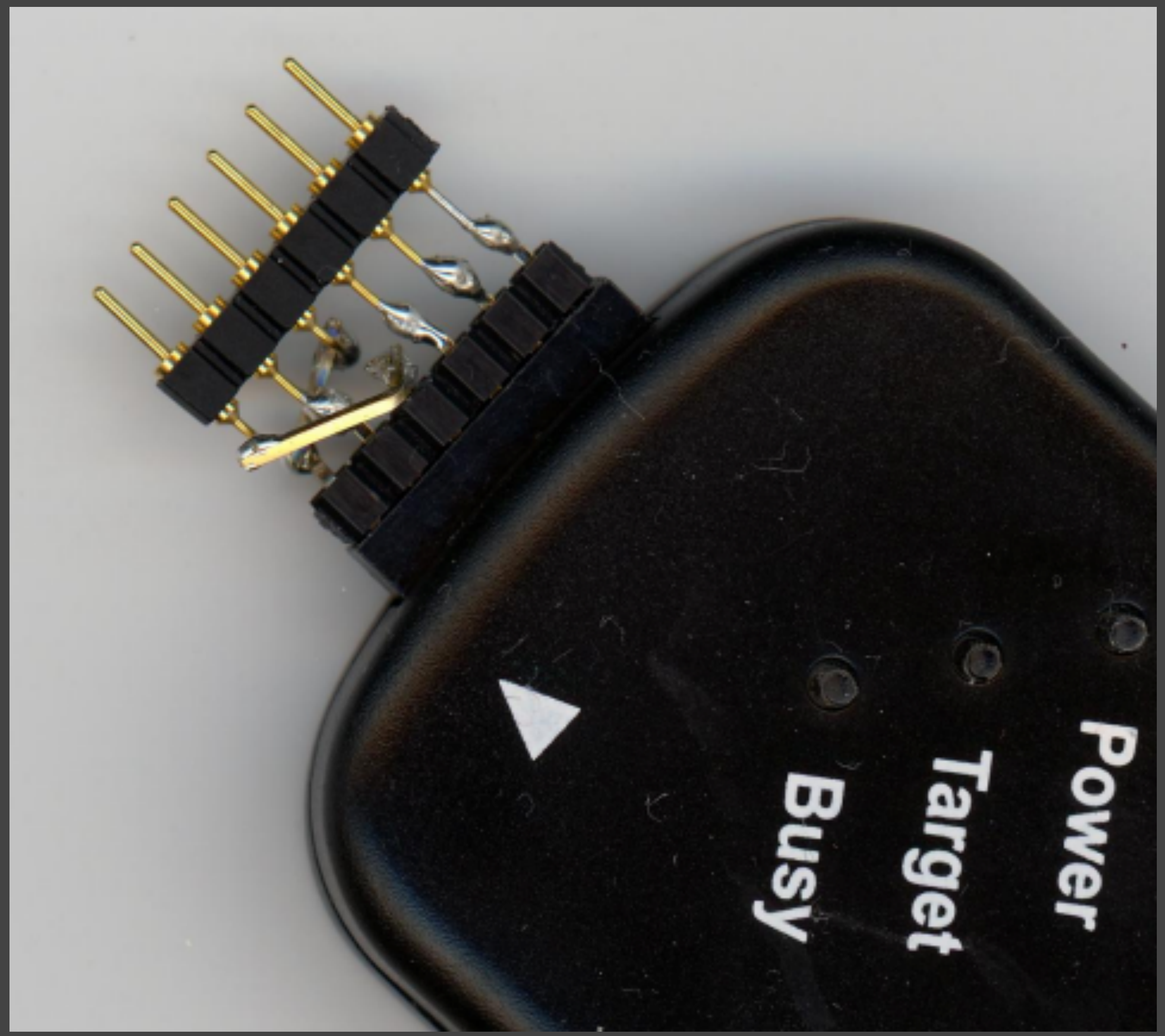

rev. D3.4
18-02-2014
estimote

GRAND IDEA STUDIO









GRAND IDEA STUDIO

Signal Manipulation

- Replay
  - Capture legitimate data, resend at a later date
- Spoofing
  - Masquerade as legitimate device, send falsified data to target
- Man-in-the-Middle (MITM)
  - Actively change data during interface/communication
  - Endpoints believe they are talking to each other
- Target doesn't know difference between real v. manipulated signal

www.grandideastudio.com/portfolio/smart-parking-meters/

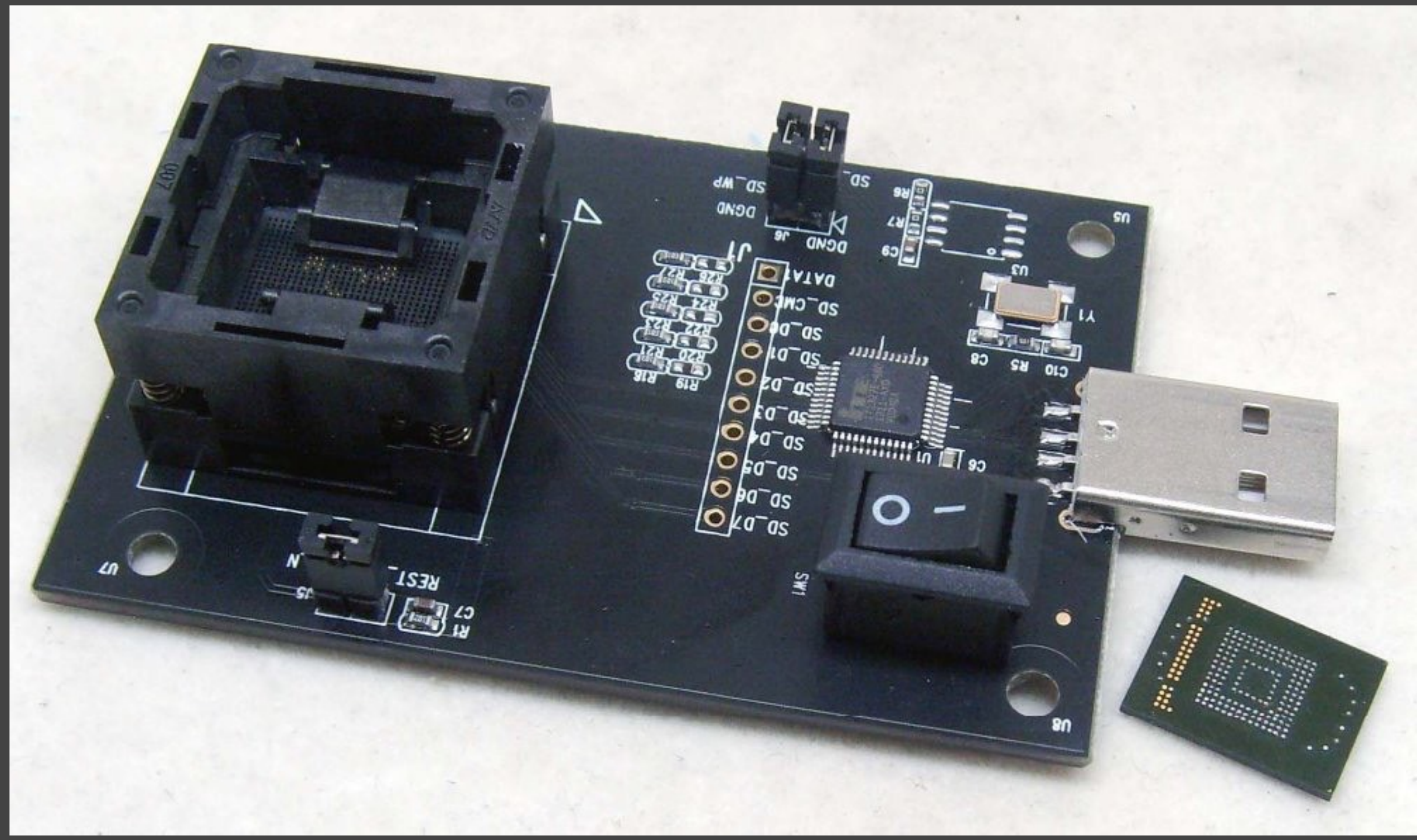
Firmware Extraction / Manipulation

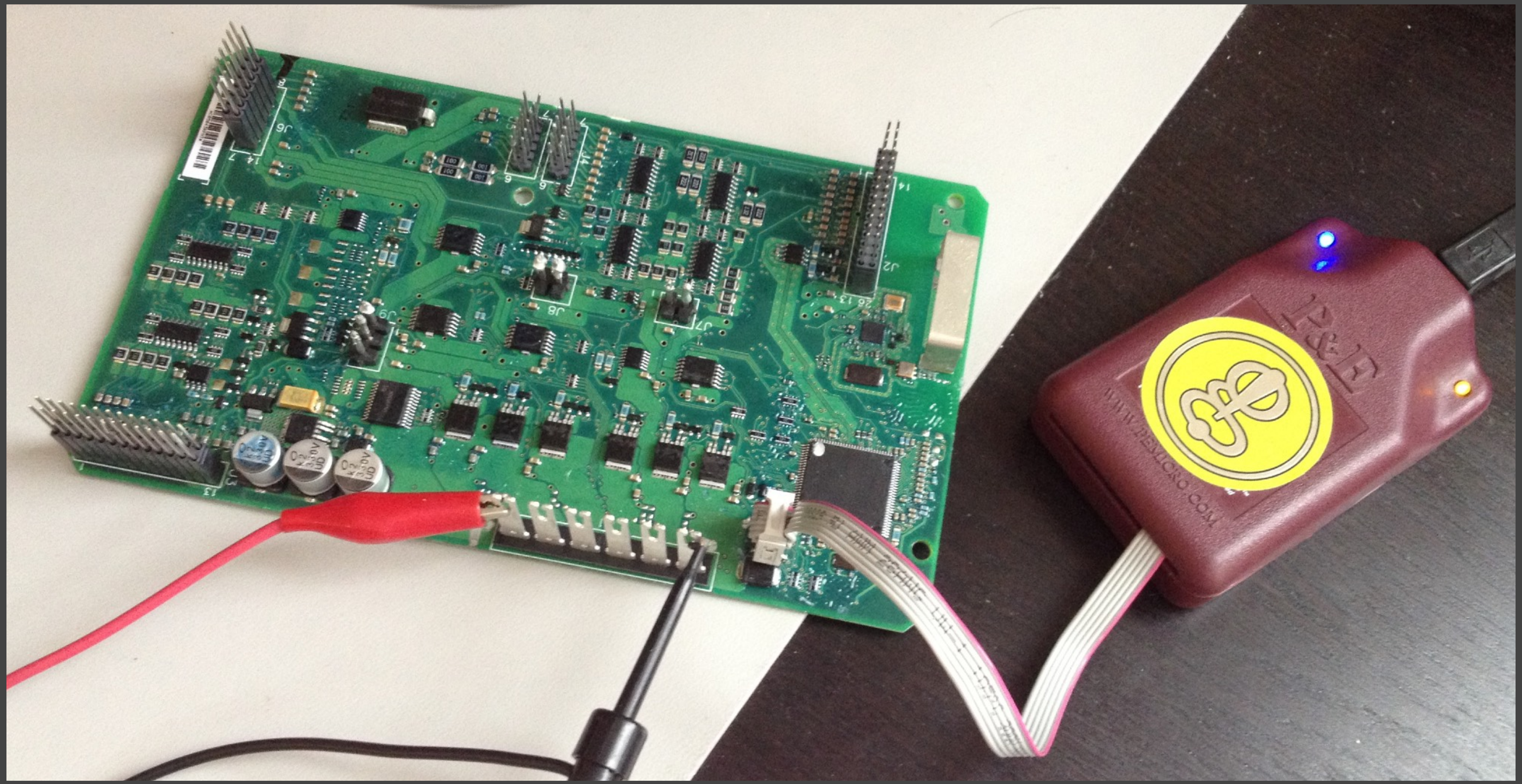
GRAND IDEA STUDIO

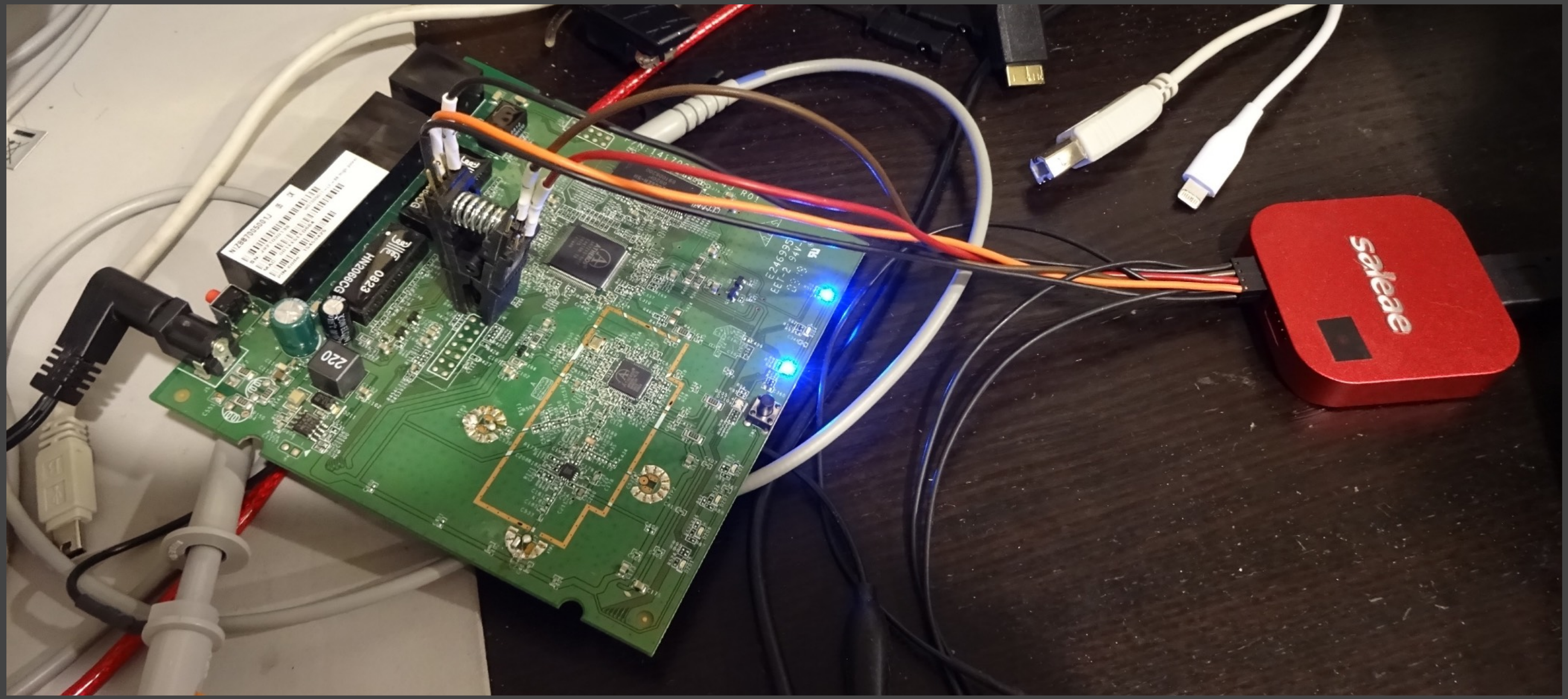
Firmware Extraction / Manipulation

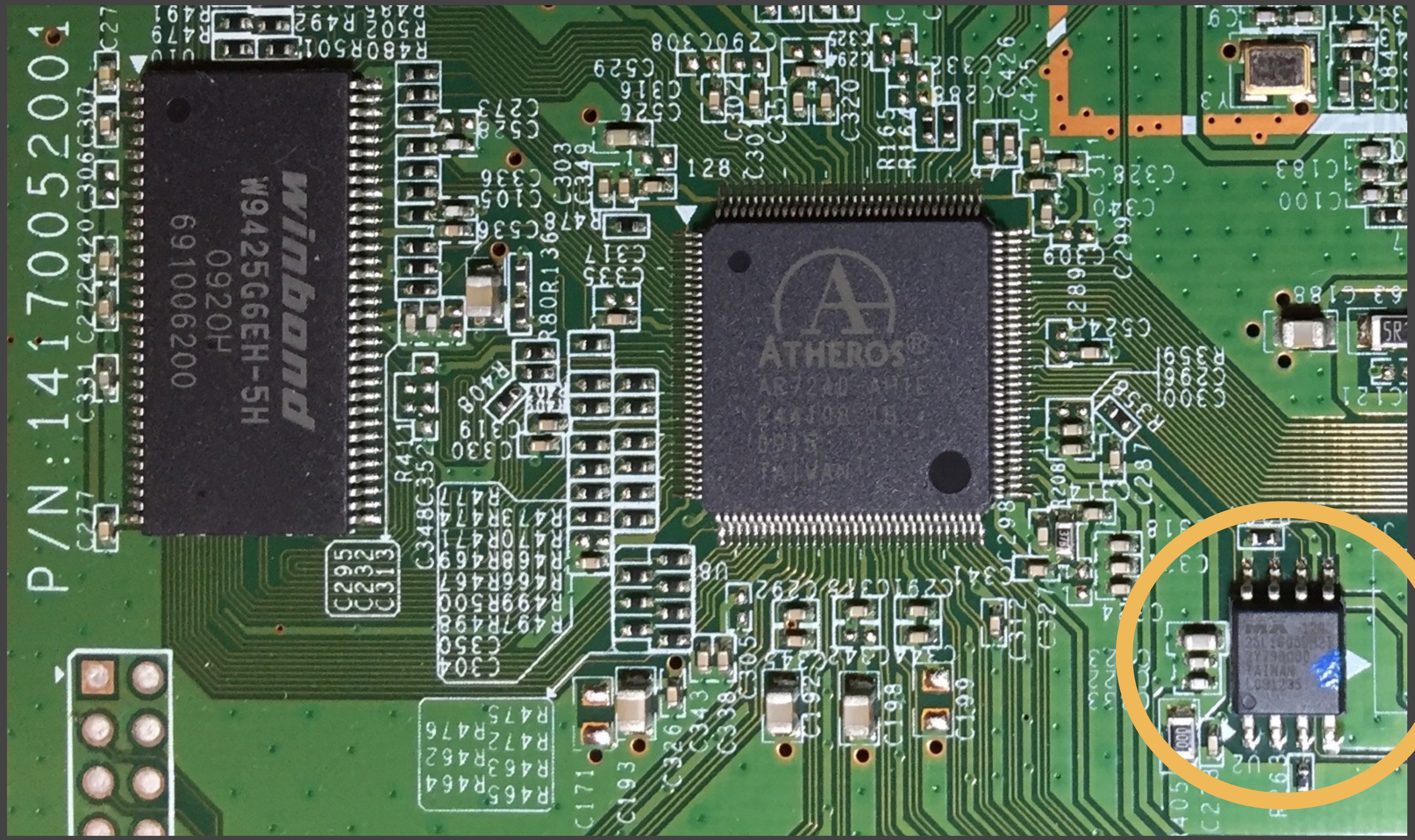
- Retrieve program code or other data from MCU/memory
  - Programming/debug interface
  - Device programmer (removed from board or in-circuit)
  - Passive capture during boot
  - Installing a custom program onto target
  - Firmware upgrade package (vendor website or proxy)
- Then, it becomes a software problem!
  - Identify system functionality (static/dynamic)
  - File system mounting/exploration
  - Search/extract user data, credentials, executables, etc.
  - Code modification/reprogramming

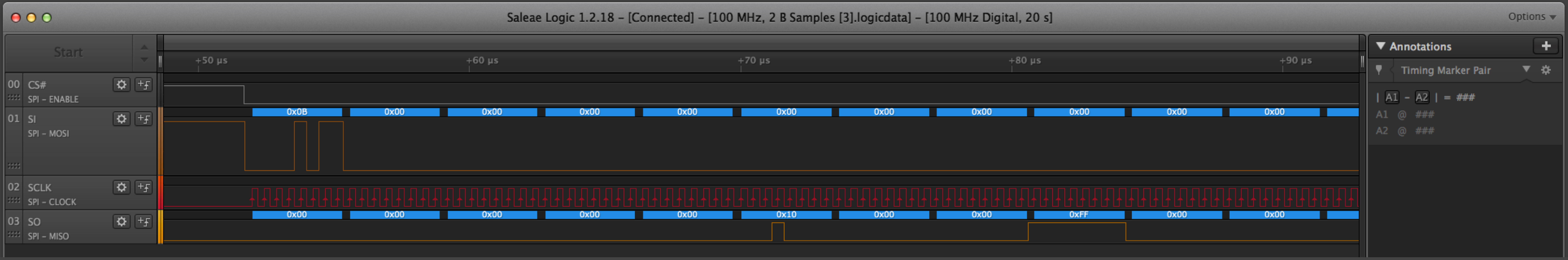
GRAND IDEA STUDIO






saleae

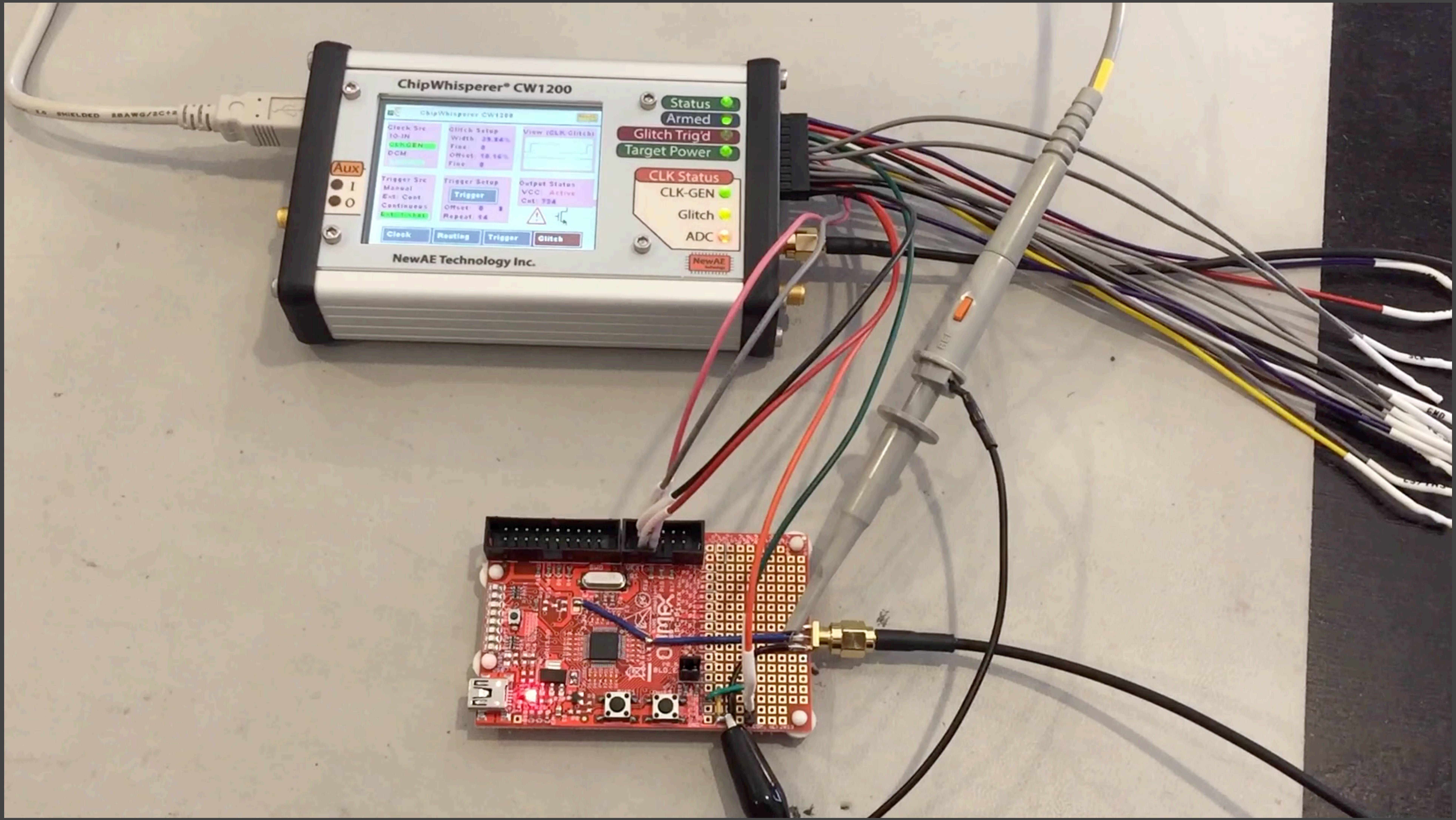

winbond
W9425G6EH-5H
ATHEROS


Saleae Logic 1.2.18 – [Connected] – [100 MHz, 2 B Samples [3].logicdata] – [100 MHz Digital, 20 s]
Options
Start
+50 µs
+60 µs
+70 µs
+80 µs
+90 µs
00 CS#
SPI - ENABLE
01 SI
SPI - MOSI
0x0B 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
02 SCLK
SPI - CLOCK
03 SO
SPI - MISO
0x00 0x00 0x00 0x00 0x00 0x10 0x00 0x00 0xFF 0x00 0x00
Annotations
Timing Marker Pair
| A1 - A2 | = ###
A1 @ ###
A2 @ ###

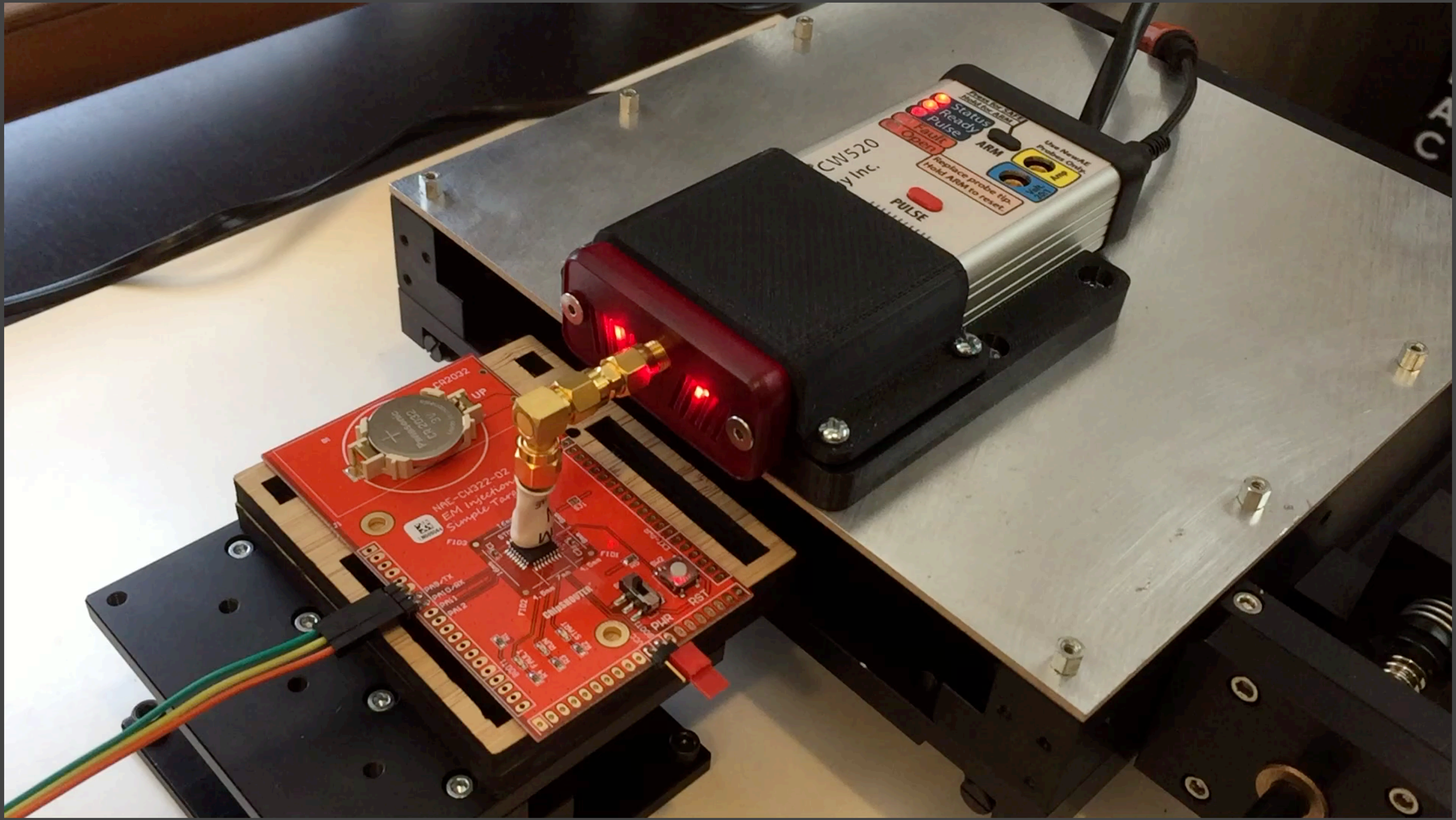
Fault Injection

Fault Injection

- Devices have defined operating parameters
  - Timing, voltage, electromagnetic (EM), temperature
- What happens if you intentionally & momentarily operate outside acceptable range?
- Could cause device to misbehave to attacker's advantage
  - Skip an instruction
  - Affect branching
  - Instruction decoding errors
  - Malformed data read/write
- Requires precise tuning to determine ideal glitch parameters



www.newae.com/chipwhisperer/



www.newae.com/chipshouter/

Supply Chain / Espionage

Supply Chain / Espionage

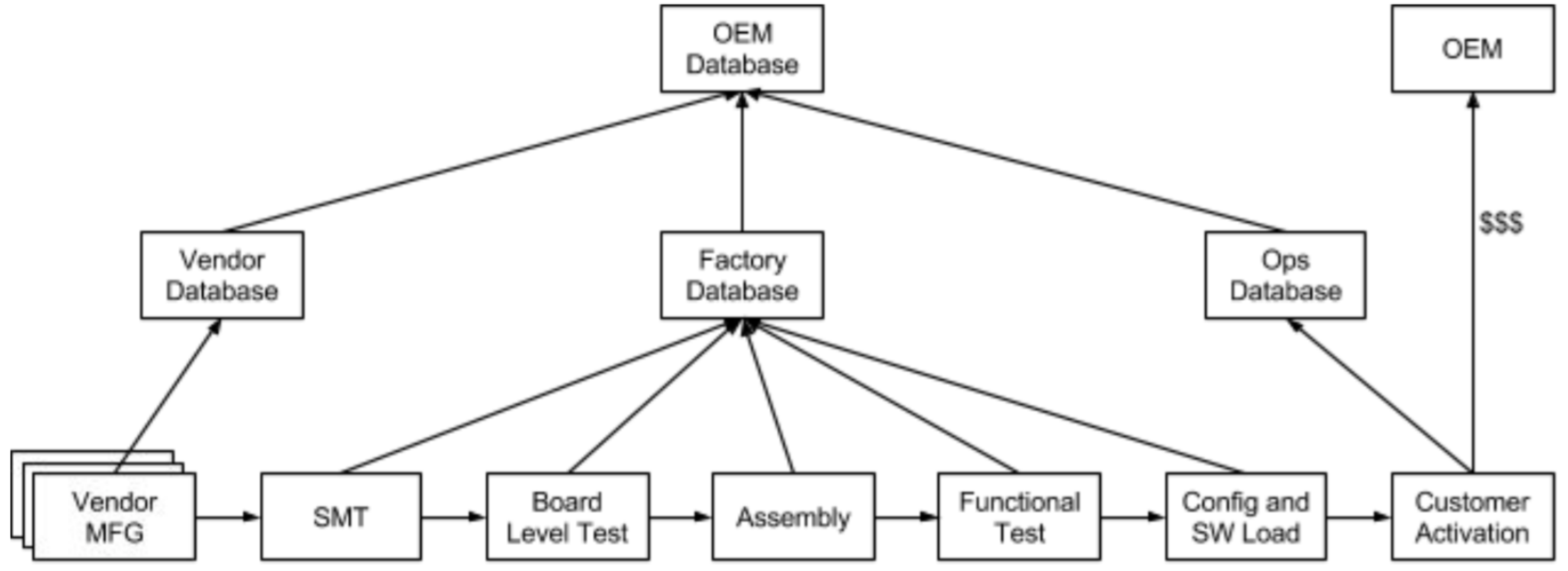
- Achieved at any layer of the product
  - HW, FW, or SW modification
  - Malicious/corrupt/deceived insiders
- Could be implemented at any part of the lifecycle
  - Design, fabrication, distribution, storage, integration, in-the-field

Development Tool Threats

- Engineering tools used during product development/manufacturing may be targeted
- Implant malicious code via compiler/programmer
  - Ex.: Infecting the Embedded Supply Chain, DEFCON 26, Miller & Kissinger
    - Multiple (remote) code execution vulnerabilities
    - Arbitrary downloading/flashing of code onto any devices connected to SEGGER J-Link
    - Load malicious firmware onto the J-Link itself

Factory Threats

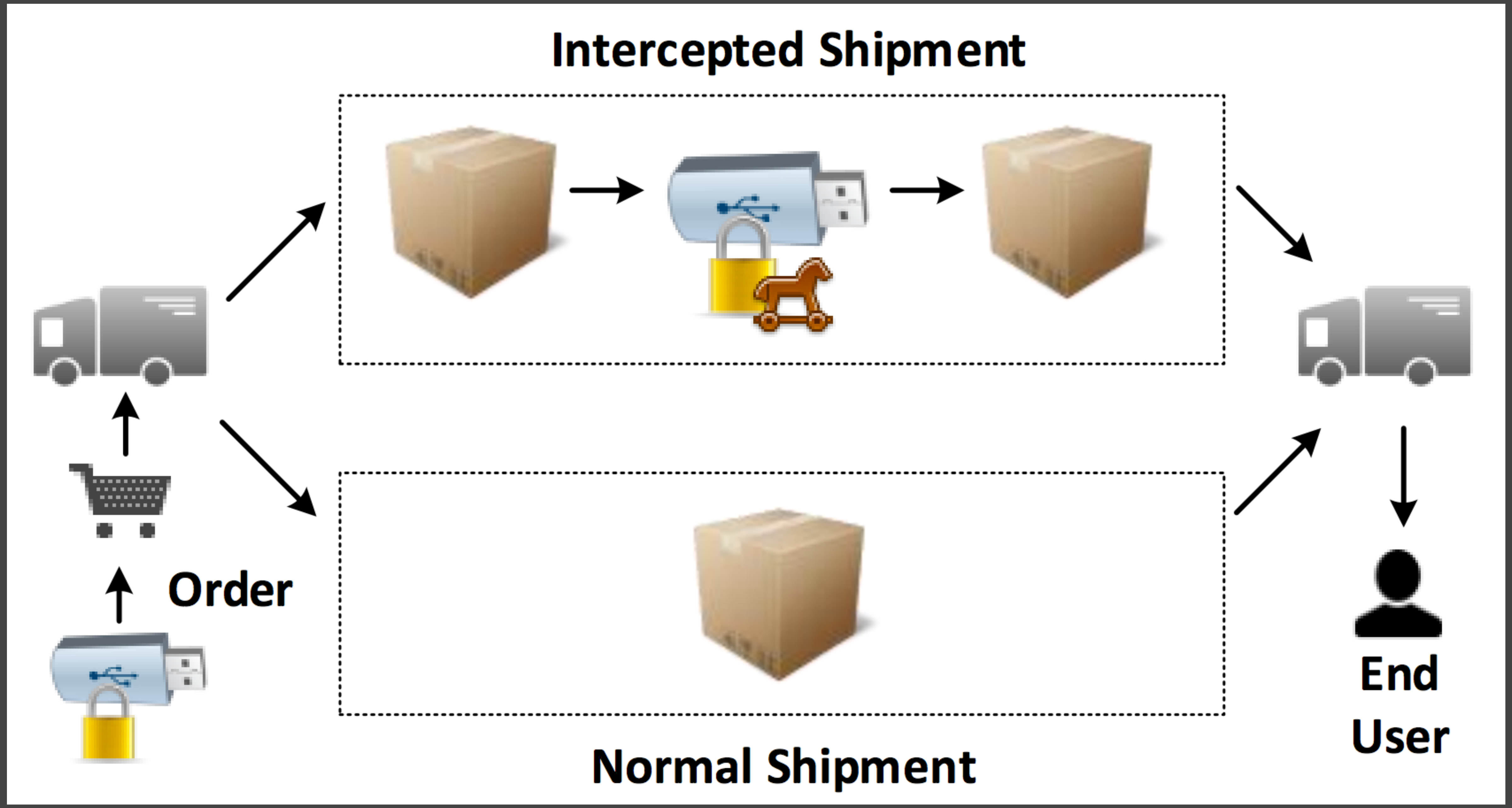
- Shadow supply chain (grey market runs)
- Firmware/data modification
- Unauthorized component replacement/PCB changes
- Leaked software/tools/schematics/data
- Targeted network access via malware/rogue devices



Secure Device Manufacturing: Supply Chain Security Resilience, NCC Group, 2015

Interdiction Threats

- Product intercepted between factory and intended customer/target
- Unauthorized field upgrades (modifications, implants)
- Repackaged and placed back into transit to original destination

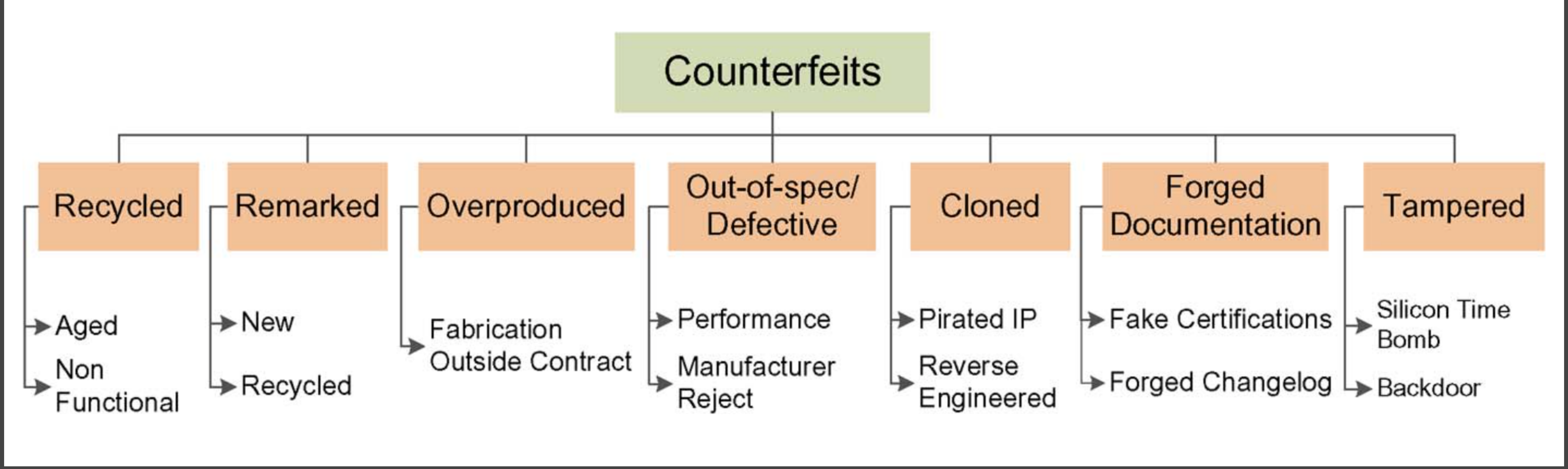


Interdiction in Practice - Hardware Trojan Against a High-Security USB Flash Drive, Swierczynski, et al., 2015

GRAND IDEA STUDIO

Silicon Threats

- Like dealing with circuitry, but at a microscopic level
- The semiconductor supply chain is ~~potentially~~ compromised
  - 15% of replacement semiconductors purchased by the Pentagon are estimated to be counterfeit (2013)



Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain, Guin et al., IEEE 2014

Best Practices

Easier Said Than Done

- Challenge of cost v. security v. convenience
- Implementation is product specific/resource dependent
  - No one-size-fits-all solution
  - Removing low-hanging fruit may increase attack difficulty
- However, security solutions/techniques/resources becoming more accessible
  - Incorporate features provided at a chip-level
  - Still requires some level of security competency
  - Be sure to independently verify what you're implementing

Best Practices

- Compartmentalization
  - Distribute design documentation on a need-to-know basis
  - Be aware of where/how documentation appears online (firmware update packages)
- Board-Level
  - Remove all non-essential information and test points
  - PCB silkscreen (designators, fab markings, logos)
  - Component/IC markings (part numbers, logos)
  - Hide critical signals on inner layers, use buried vias
  - Only obfuscation, but may increase reverse engineering time

Best Practices 2

- Security Fuses
  – Prevents full read-out or access to a specific memory area
  – Most commonly used on MCU internal memory
  – Easy to enable during code compilation or device programming
  – May still be exploited via brute force, glitch, die attack, off-shore services
- On-Chip Debug/Program/Diagnostic Interfaces
  – Disable or remove completely for production units
  – Implement password/authentication mechanism (may not be part of standard interface)
  – Possibly inconvenient for legitimate personnel (manufacturing, service/repair)

Best Practices 3

- Coding
  - Handle undefined behavior, memory leaks, buffer overflows/bounds checking, invalid data structures, off-by-one, etc.
  - Remove debug symbols/tables, enable optimization
  - Mechanism to update/patch vulnerable code/OS (if needed)
  - Source code review, static analysis
- Network Configuration
  - Don't use default login credentials (username/password)
  - Don't add hardcoded data/backdoors for future use
  - Close unused ports/daemons/configuration/management interfaces
  - Learn about common network/OS exploits

Best Practices 4

- Anti-Tamper
  - Prevent/deter/detect physical access or tampering of embedded system
  - Resistance, evidence, detection, response
  - See Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses, Weingart, CHES 2000
- Run-Time Diagnostics/Failure Modes
  - Ensure device is fully operational at all times (watchdog, periodic system/memory checks)
  - Detect when system is being operated outside of defined conditions (voltage, timing, thermal, optical glitching)
  - Determine how product handles failure (halt/shutdown system, erase critical memory areas)

Best Practices 5

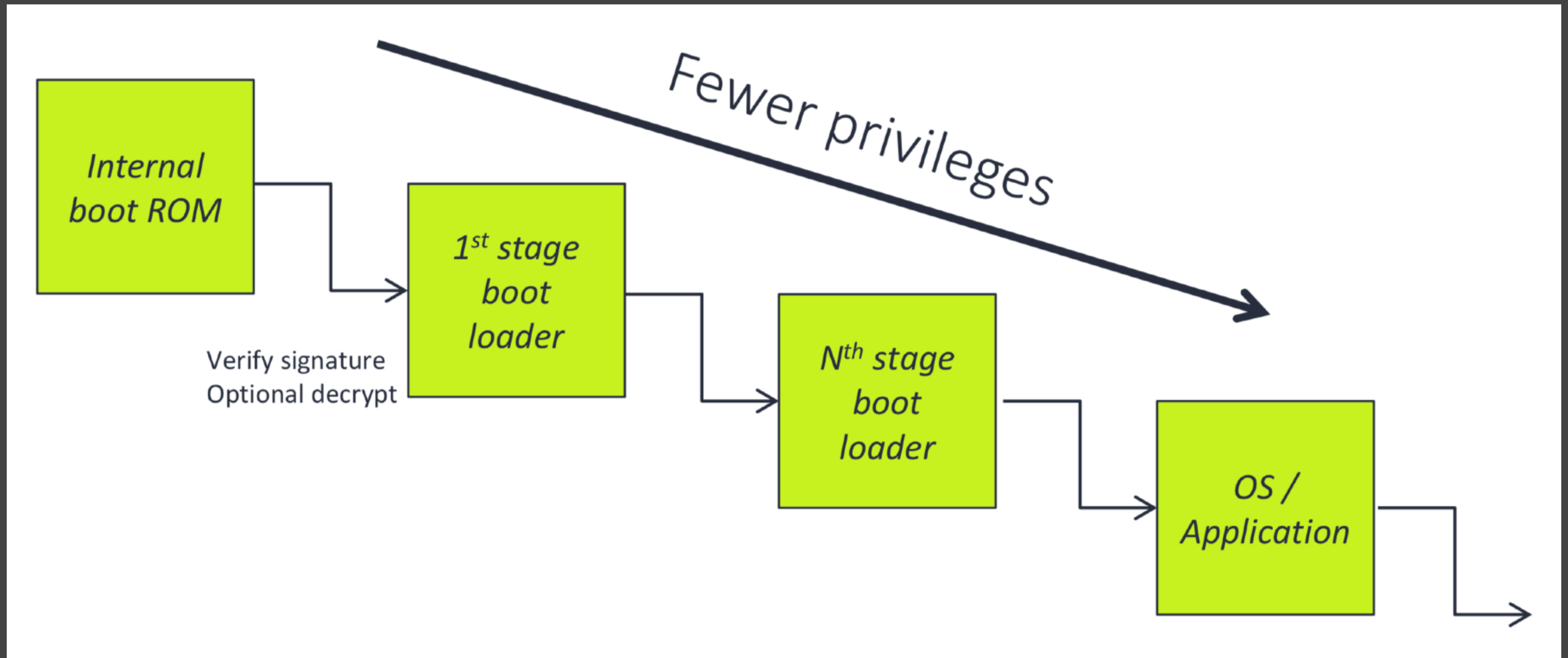
- Encryption
  - For both data at rest and in motion (including firmware, if possible)
  - Consider key management/storage, cipher type
  - Some vendors offer on-chip support for encrypted memory areas
  - Beware of how unencrypted data could be accessed during operation (chip-to-chip communication, debug interface to RAM)
  - For wireless systems, use available security features (check if protocol has already been broken)
  - Use industry standard, publicly scrutinized/analyzed/proven ciphers
    - Don't roll your own!
  - See CrypTech.is

Best Practices 6

- Side Channel / Fault Injection Countermeasures
  - Unintentional leakage from system
  - Consider power, EM/RF, timing, thermal
  - Many compilers generate side channels unintentionally
  - See www.newae.com/embedded-security-101

Best Practices 7

- Secure Boot Process
  - Each stage verifies the following stage (cryptographic signature)
  - Only execute trusted code (verified origin/integrity)
  - Prevents arbitrary code execution (unless defeated, commonly done via glitch/patch)
  - See Pew Pew Pew: Designing Secure Boot Securely, Timmers & Spruyt, Nullcon 2019



Top 10 Secure Boot Mistakes, van Woudenberg, Hardwear.io 2019

Best Practices 8

- Hardware Root of Trust
  - Ensure a trusted base on which to build your product
  - Open source for complete verification/validation
  - See OpenTitan, Tropic Square, Betrusted

Thanks for your time!