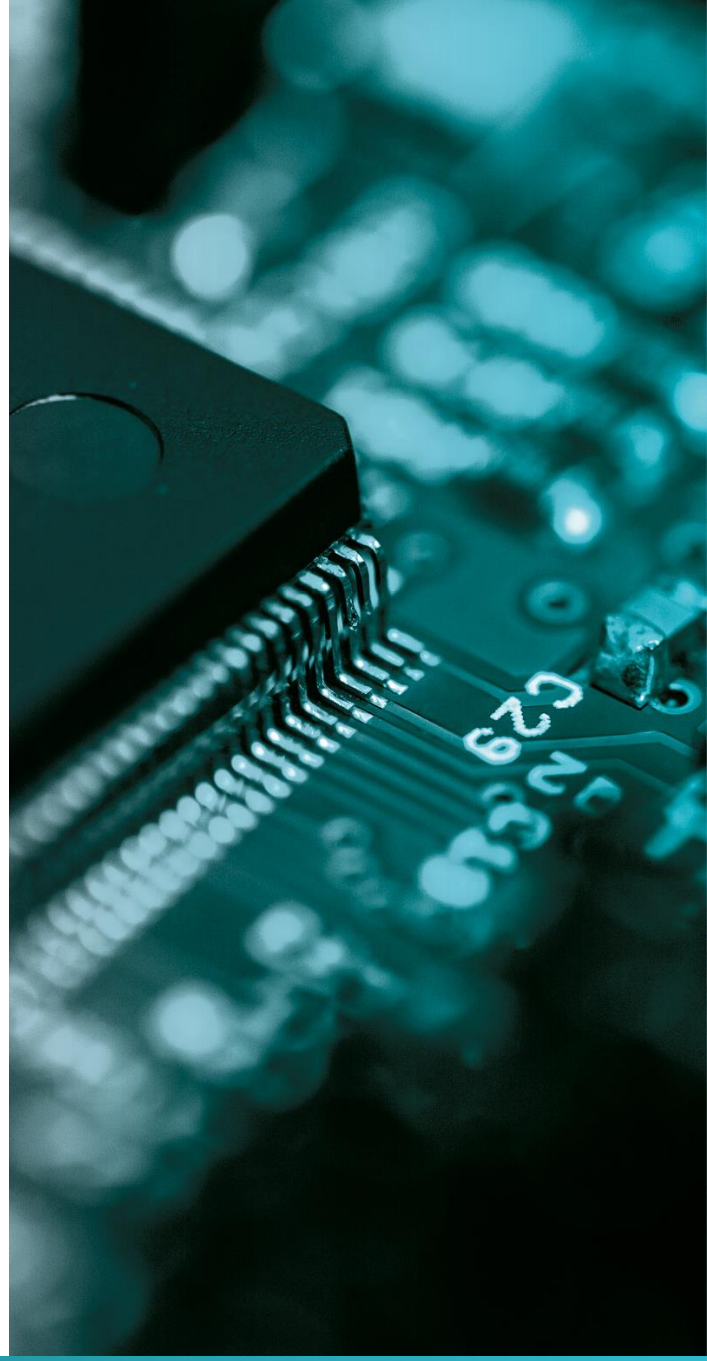


emproof
embedded security

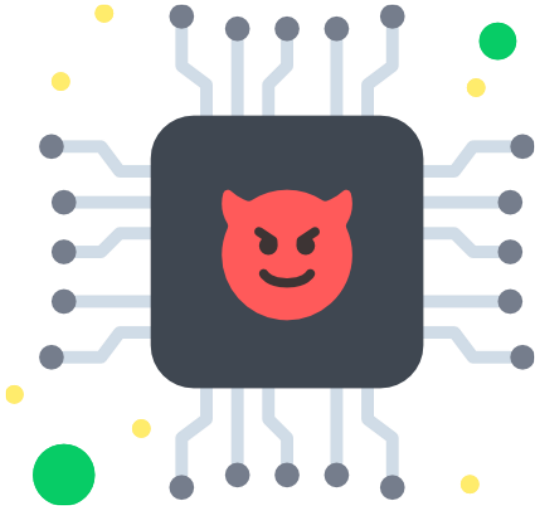
MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Introduction to Software and Hardware Reverse Engineering

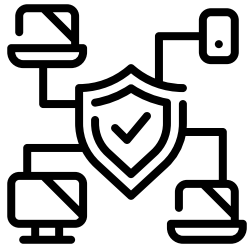
Prof. Dr. Christof Paar & Dr. Marc Fyrbiak

Embedded Systems are a Lucrative Target

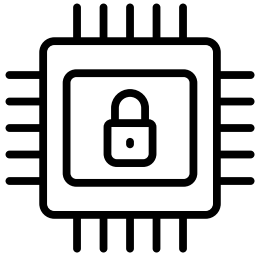
- Steal algorithms & clone devices
- Entry point to networks and systems
- Data breaches
- Eavesdropping & espionage
- Piracy and cracking
- Botnets, crypto mining, ransomware…



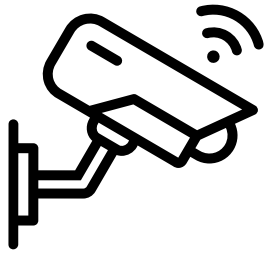
Embedded System Security Landscape



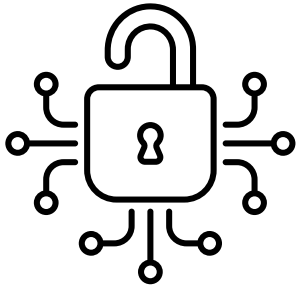
Network Security



Hardware Security

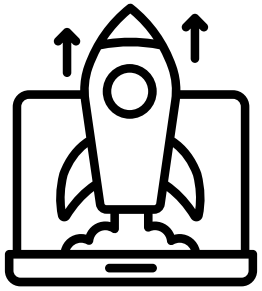


Physical Security



Cryptography

Auth. Security



Deployment Security

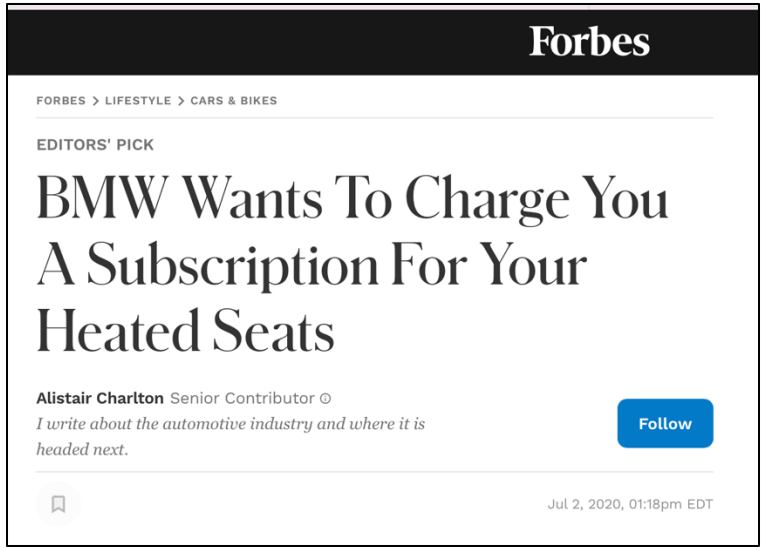
Licensing

Binary Software Security is often neglected but has the potential to bypass all other verticals

TESLA

Tesla’s new Models S and X have the same battery pack but with software-locked capacity

Fred Lambert | Aug 15 2023 - 5:50 am PT | 0 Comments



Forbes

FORBES > LIFESTYLE > CARS & BIKES

EDITORS' PICK

BMW Wants To Charge You A Subscription For Your Heated Seats

Alistair Charlton Senior Contributor

I write about the automotive industry and where it is headed next.

Follow

Jul 2, 2020, 01:18pm EDT

Source:
https://www.forbes.com/sites/alistaircharlton/2020/07/02/bmw-wants-to-charge-you-a-subscription-for-your-heated-seats/
https://electrek.co/2023/08/15/teslas-new-model-s-x-same-battery-pack-but-with-software-locked-capacity/

Security

Researchers jailbreak a Tesla to get free in-car feature upgrades

Lorenzo Franceschi-Bicchierai

@lorenzofb / 3:44 PM GMT+2 • August 3, 2023

Comment

Image Credits: CFOTO/Future Publishing / Getty Images

MOTHERBOARD
TECH BY VICE

BMW Wants to Charge for Heated Seats. These Grey Market Hackers Will Fix That.

A vibrant community of BMW “coders” has modified the luxury vehicles for years. Now they’re ready to unlock BMW’s controversial heated seat subscription.

By Joseph Cox

By Aaron Gordon

Sources:
https://tcrn.ch/45fXMlh
https://www.vice.com/en/article/7k8bv9/bmw-wants-to-charge-for-heated-seats-these-grey-market-hackers-will-fix-that

Motivation from an Attackers Perspective

emproof

Intellectual property theft

- Algorithm stealing or device cloning

Vulnerability analysis

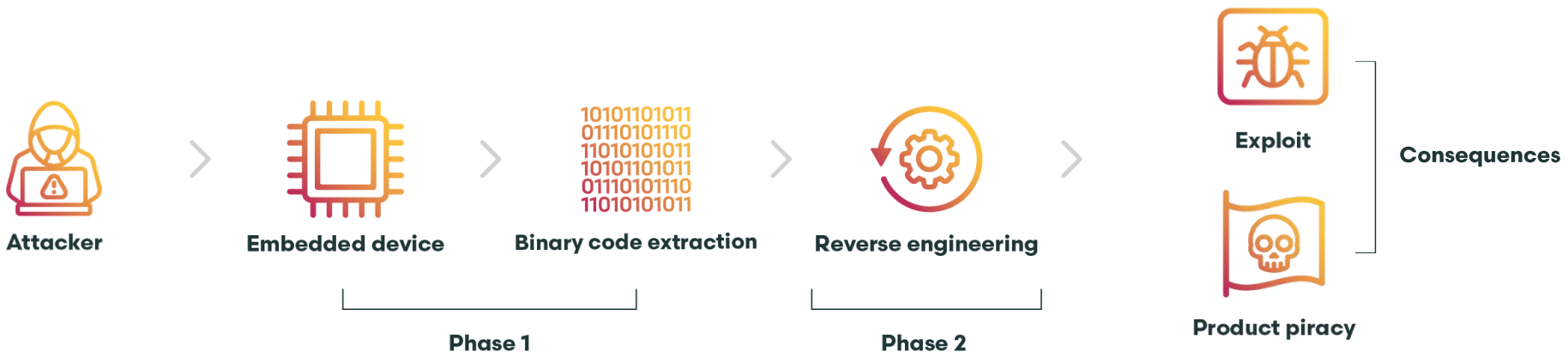
- Find exploitable weaknesses in systems

Piracy and cracking

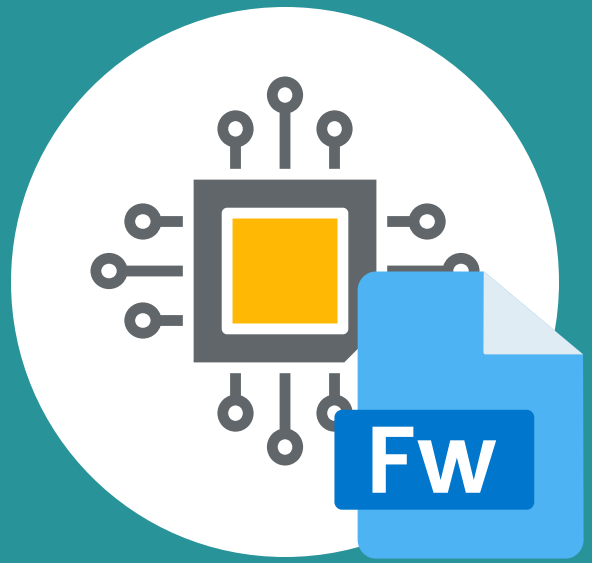
- Circumvent paid features or license checks

Misc.

- Writing a publication for a Ph.D. thesis ...
- Fun



Attack Flow of Embedded Devices



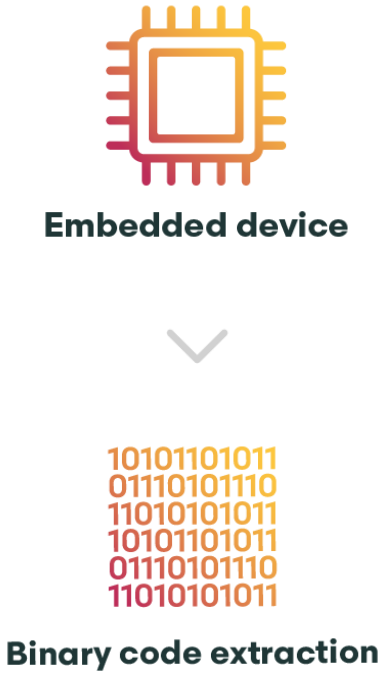
Phase 1: Firmware Extraction

Phase 1: Firmware Extraction

- **Goal:** Extract binary firmware image (from device)

- **How?**
  - **Read binary from non-volatile memory (Flash, EEPROM)**
    ... or download from internet
    ... or record (unencrypted) software updates
    ... or access parts of software from compiled libraries

  - However, modern embedded systems feature **read-out protection**

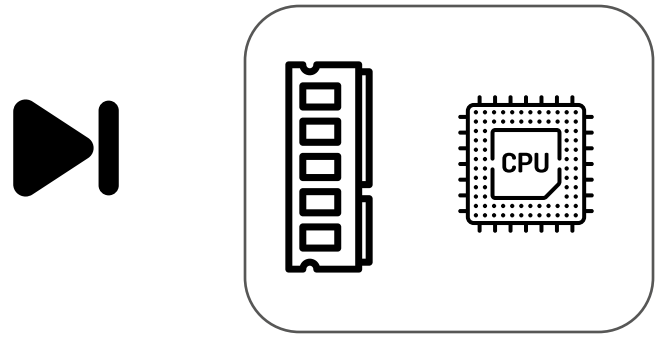


Firmware Extraction – Non-Invasive

- Embedded system bootloader are prone to **logic errors**
  - Incorrect or incomplete logic checks
  - No authentication or static passwords
  - Timing-side cannels of password validation

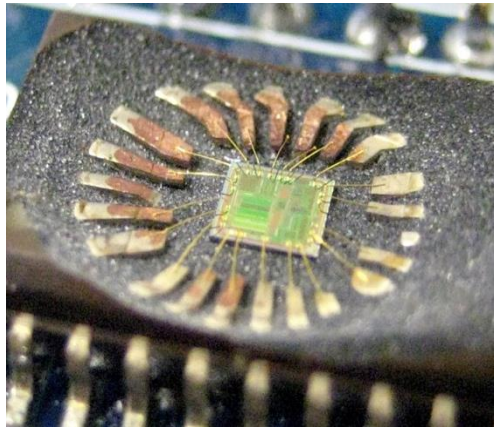
Firmware Extraction – Semi-Invasive

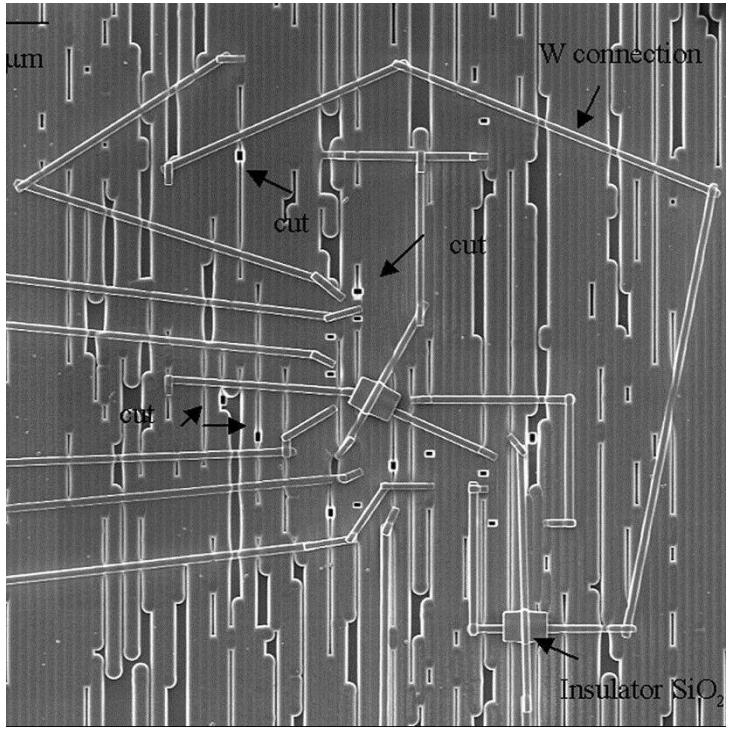
- Embedded system bootloader are prone to **fault injection**
  - **Goal:** Skip instructions that validate read-out protection
  - Digital circuits are designed to work at defined operating points
    - Voltage
    - Clock frequencies
    - Temperatures
  - Fault injection yields incorrect evaluation of CPU instruction(s)
    - Supply voltage
    - Clock signal
    - Electromagnetic radiation
    - Lasers



Firmware Extraction – Invasive

- **Embedded systems hardware is prone to chip-level editing**
  - **Goal:** Change the non-volatile protection level (fuse bits)
  - Microprobing to analyze chip signals
  - Chip edits to change silicon functionality



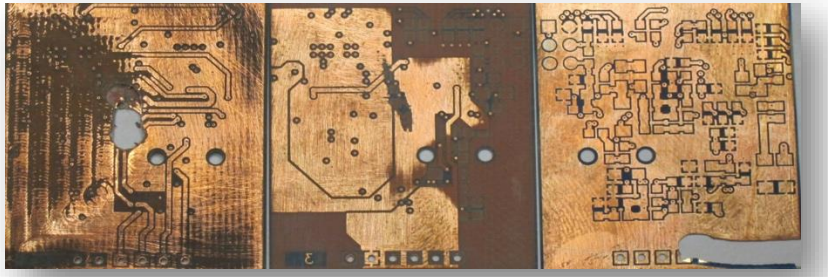


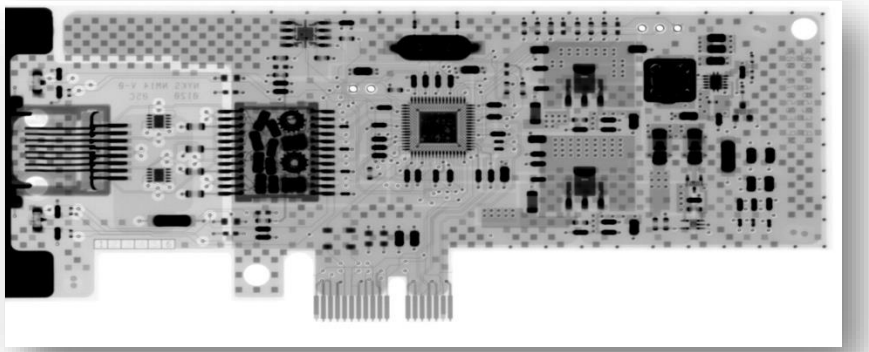
https://research.nccgroup.com/wp-content/uploads/2020/02/NCC-Group-Whitepaper-Microcontroller-Readback-Protection-1.pdf
https://www.eag.com/services/engineering/fib-circuit-edit-debug/

Excursus: Hardware Reverse Engineering

System-level Analysis

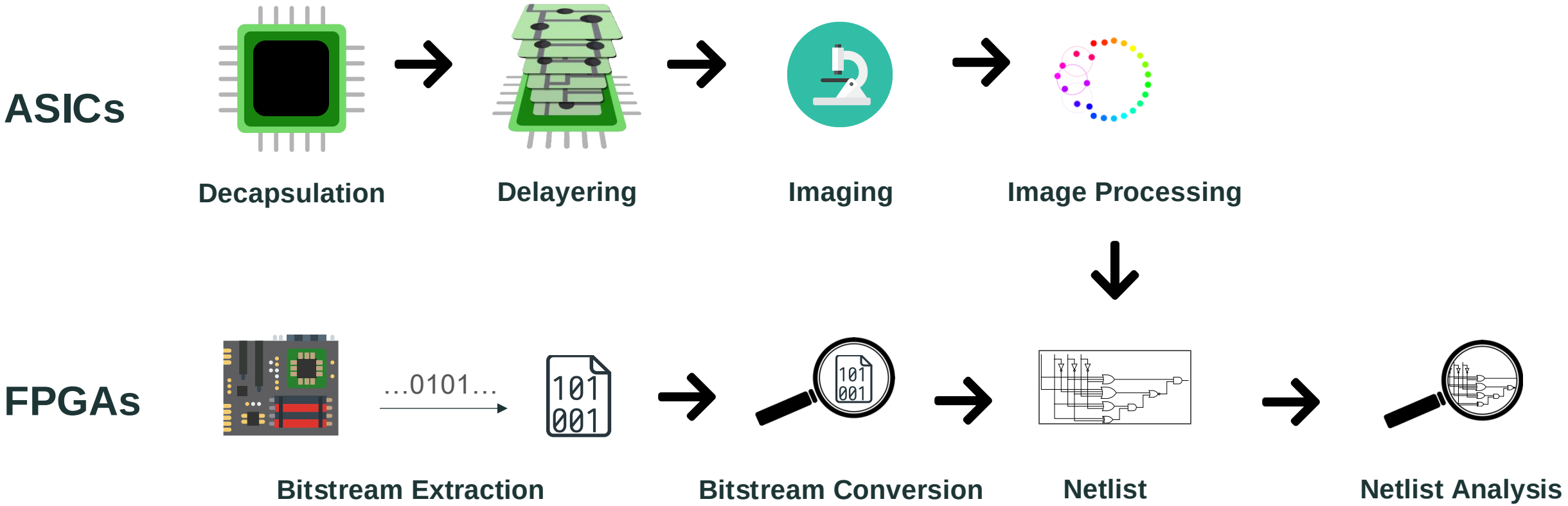
- **Goal:** Identify product, package, components, …
- Anti-tamper mechanism (detection / evidence)
- Printed-circuit board analysis
  - Mechanical
  - Chemical
  - X-ray



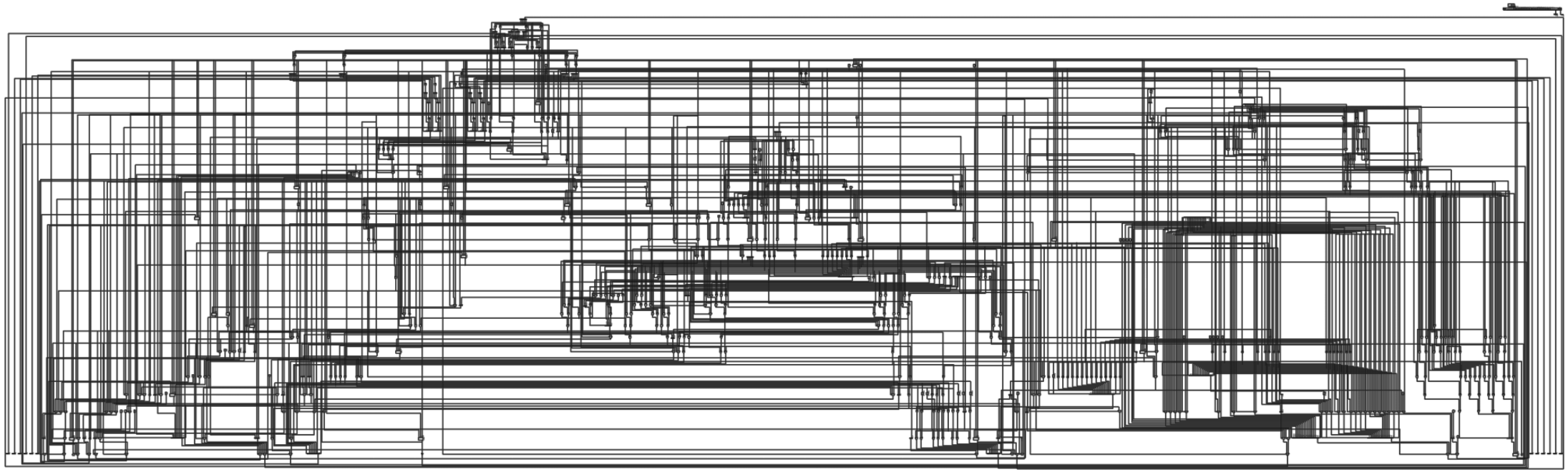


https://images.squarespace-cdn.com/content/v1/5ae6dd648f51303ed37b4cdd/55ab7178-a050-489b-a580-565ff9f0f31a/NIC_PCB.jpg?format=250
https://dchhv.org/assets/images/pcb_delayer/EMIC_carnage.jpg

Overview of the Hardware Reverse Engineering Process



Gate-Level Netlist Reverse Engineering

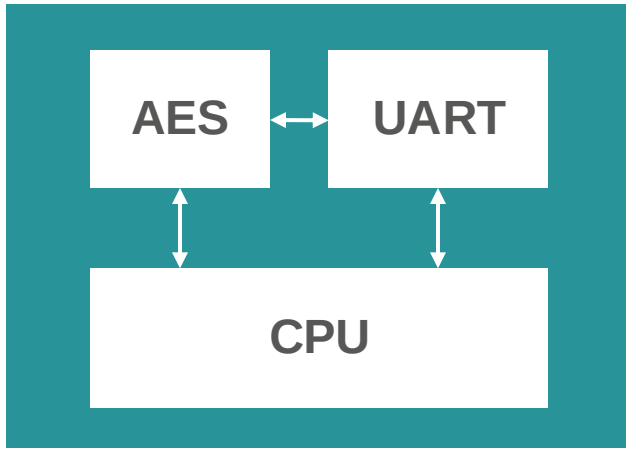


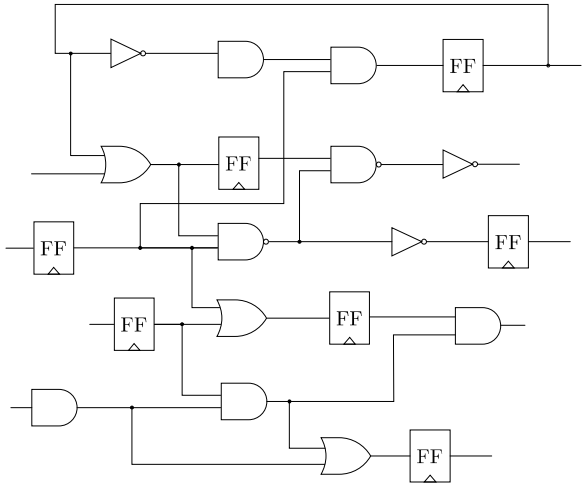
Unprocessed netlist

→ incomprehensible sea-of-gates with thousands to billions of gates

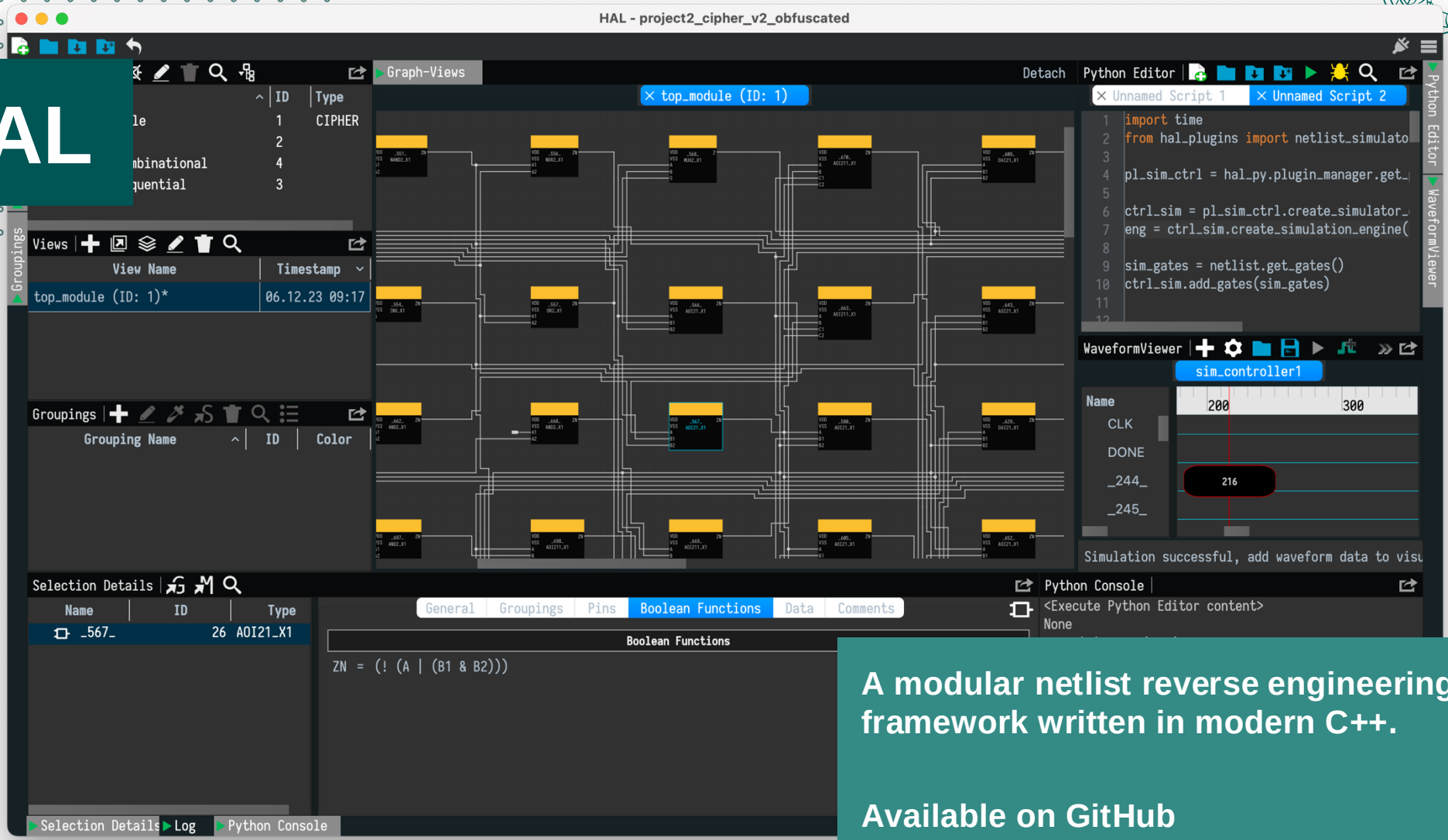
Gate-Level Netlist Reverse Engineering

- Loss of Hierarchy





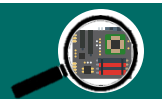
HAL



A modular netlist reverse engineering framework written in modern C++.

Available on GitHub

Hardware Reverse Engineering

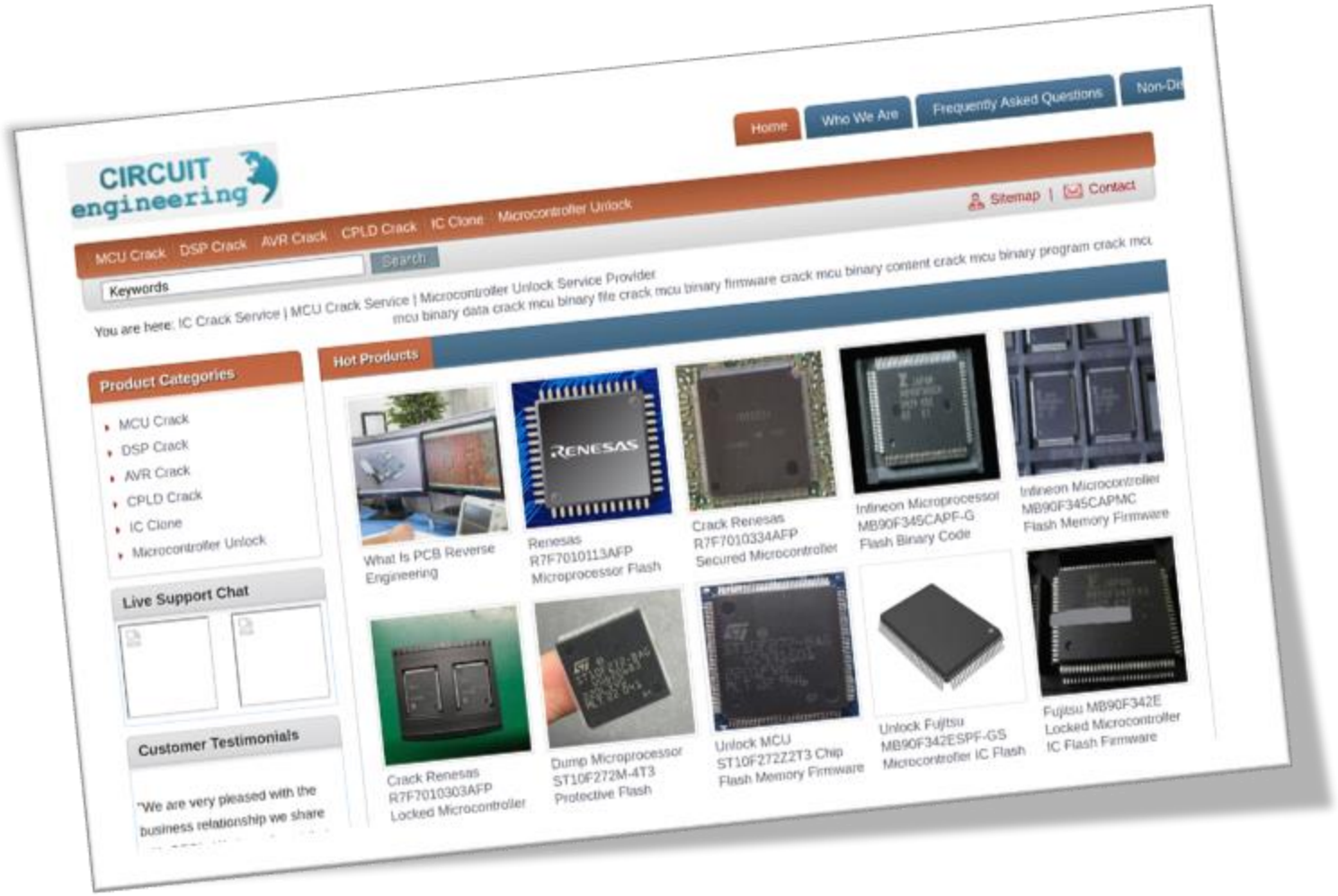


- **StarBleed – A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs** — **USENIX 2020**
  M. Ender, A. Moradi, and C. Paar
- **DANA - Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering** — **CHES 2020**
  N. Albartus, M. Hoffmann, S. Temme, L. Azriel, C. Paar
- **How Not to Protect Your IP - An Industry-Wide Break of IEEE 1735 Implementations** — **S&P 2022**
  J. Speith, F. Schweins, M. Ender, M. Fyrbiak, A. May, C. Paar
- **Stealing Maggie's Secrets - On the Challenges of IP Theft Through FPGA Reverse Engineering** — **CCS 2024**
  S. Klix, N. Albartus, J. Speith, P. Staat, A. Verstege, A. Wilde, D. Lammers, J. Langheinrich, C. Kison, S. Sester-Wehle, D. Holcomb, C. Paar
- **HAWKEYE – Recovering Symmetric Cryptography From Hardware Circuits** — **CRYPTO 2024**
  G. Leander, C. Paar, J. Speith, L. Stennes

Hardware Trojans

- **On the Design and Misuse of Microcoded (Embedded) Processors - A Cautionary Note** — **USENIX 2021**
  N. Albartus, C. Nasenberg, F. Stolz, M. Fyrbiak, C. Paar, R. Tessier
- **Red Team vs. Blue Team: A Real-World Hardware Trojan Detection Case Study** — **S&P 2023**
  E. Puschner, T. Moos, S. Becker, C. Kison, A. Moradi, C. Paar

Firmware Extraction as a Service

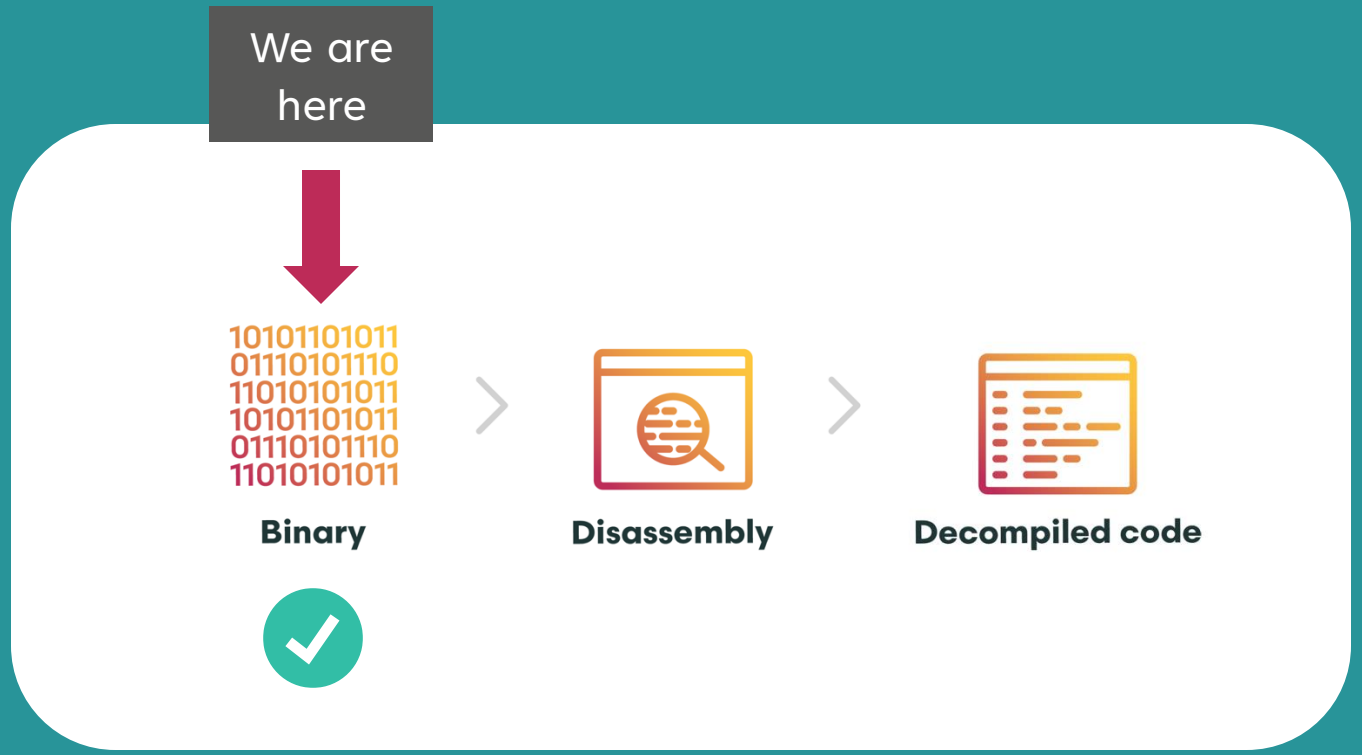


https://www.ic-cracker.com/

Firmware Extraction as a Service

https://www.break-ic.com/

Phase 2: Reverse Engineering



Binary and Machine Code

Binary

emproof

- executable file with code, data, and metadata
- contains CPU-specific machine code
- common formats: ELF (Linux), PE (Windows)

Binary

Machine Code

emproof

```
0a 01 0a 00 0b 02 de ad
```

Machine Code


```
0a 01 0a 00 0b 02 de ad
```

Machine Code


```
0a 01 0a 00 0b 02 de ad
```

```
add

mul
```

Machine Code


```
0a 01 0a 00 0b 02 de ad
```

```
add R1
mul R2
```

Machine Code


```
0a 01 0a 00 0b 02 de ad
```

```
add R1, 0x0a00
mul R2, 0xdead
```

Machine Code


```
0a 01 0a 00 0b 02 de ad
```

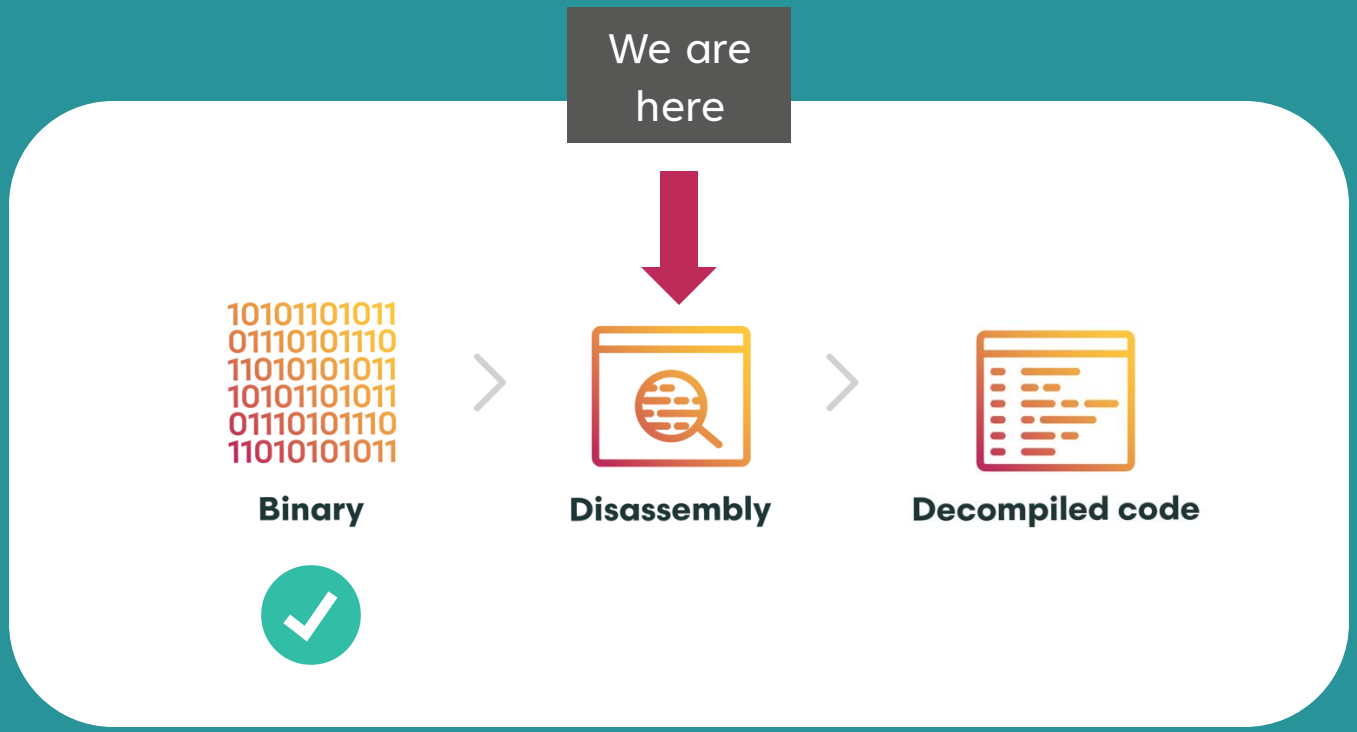
```
add R1, 0x0a00
mul R2, 0xdead
```

The decoded machine code is called assembly code.

Static Analysis

Static Analysis

- examines binary without executing it
- reveals structure and functionality
- common tools: disassembler & decompiler

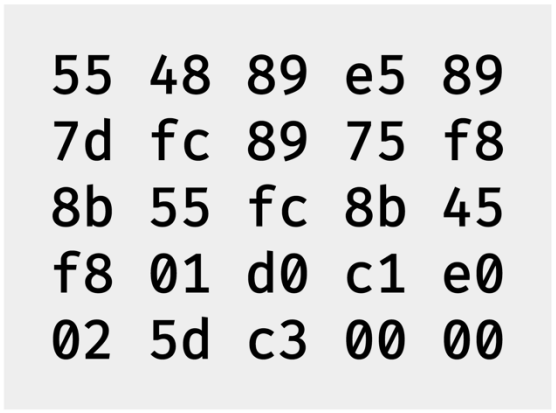


Disassembler

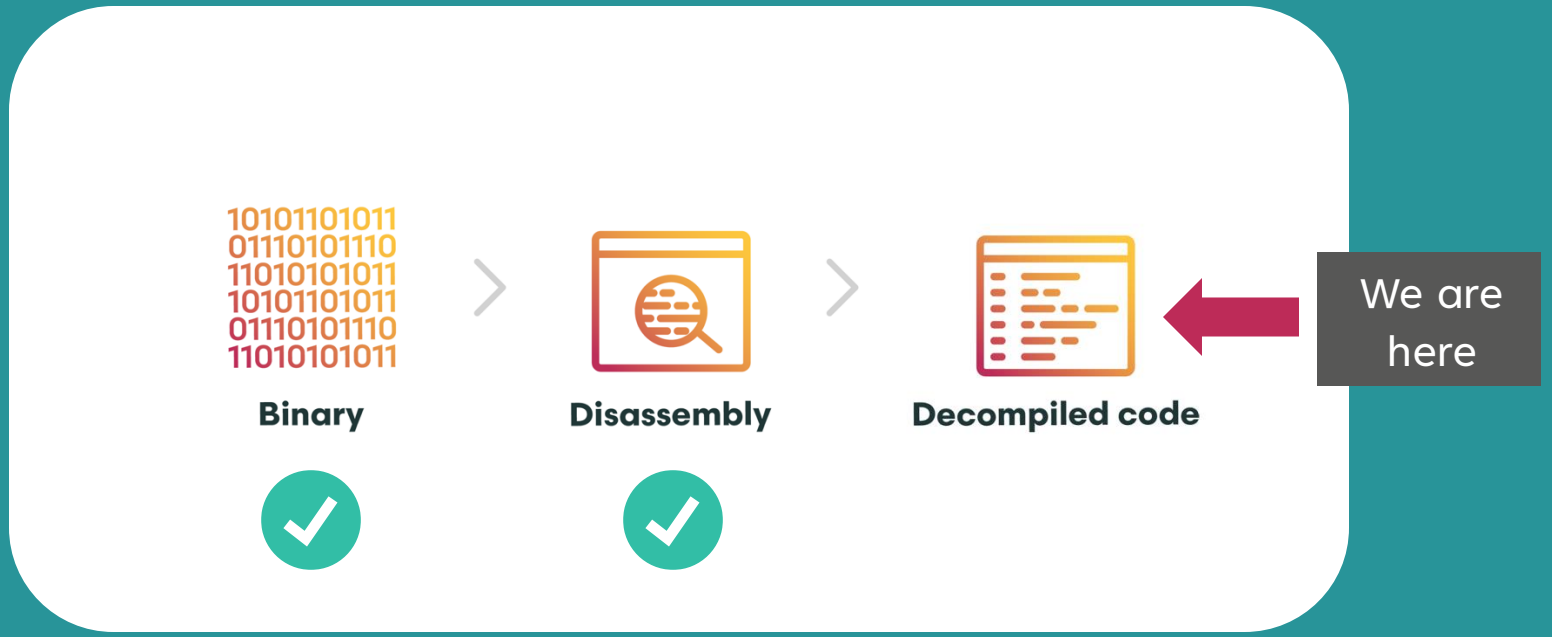
Disassembler

- converts machine code into assembly code
- useful for understanding binary's behavior
- key tool in reverse engineering workflows

Disassembler



```
push    rbp
mov     rbp, rsp
mov     [rbp+var_4], edi
mov     [rbp+var_8], esi
mov     edx, [rbp+var_4]
mov     eax, [rbp+var_8]
add     eax, edx
shl     eax, 2
pop     rbp
retn
```


10101101011
01110101110
11010101011
10101101011
01110101110
11010101011
Binary
Disassembly
Decompiled code
We are here

Decompiler

Decompiler

converts machine code back to high-level code

aids in understanding program logic and structure

produces approximate source code from binaries

Decompiler: Reconstructs High-Level Code

emproof

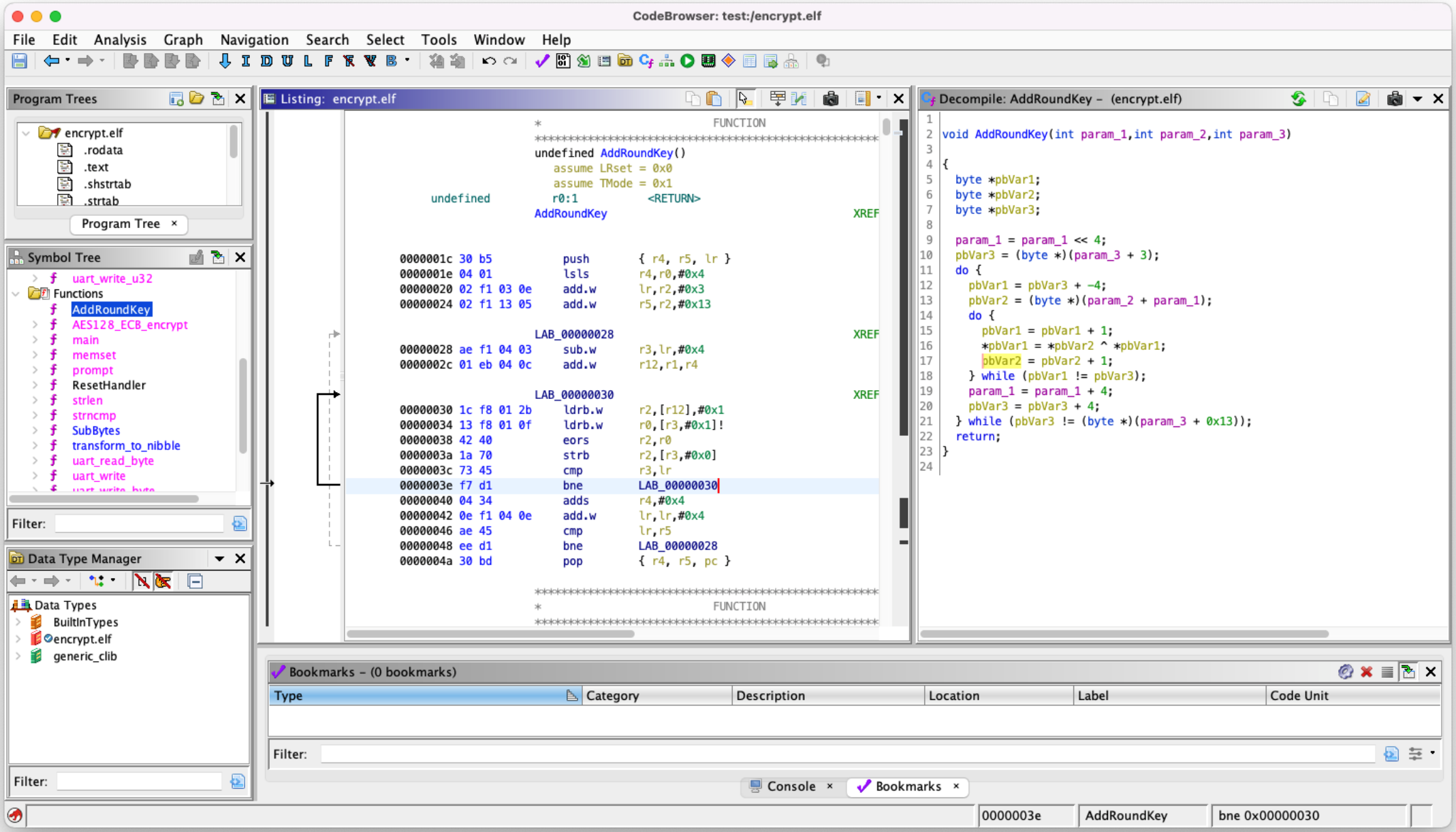
```
push    rbp
mov     rbp, rsp
mov     [rbp+var_4], edi
mov     [rbp+var_8], esi
mov     edx, [rbp+var_4]
mov     eax, [rbp+var_8]
add     eax, edx
shl     eax, 2
pop     rbp
retn
```

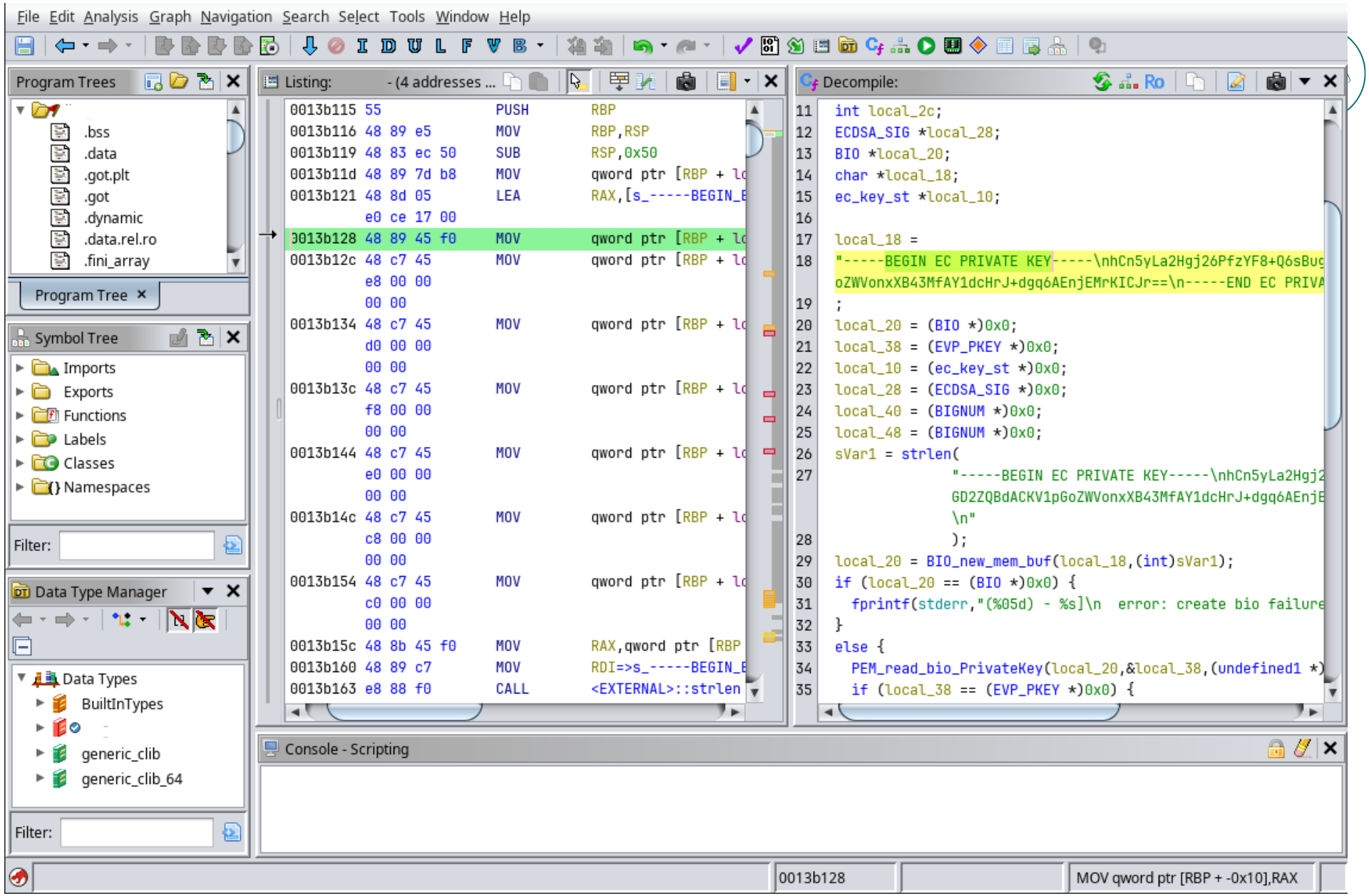
```
ulong calculate(int param_1,int param_2)
{
    return (ulong)(uint)((param_2 + param_1) * 4);
}
```

Tools

- Community with advanced tooling

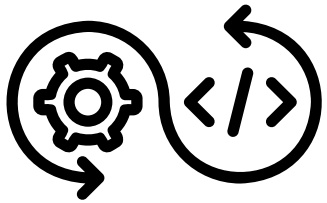
- Many open-source and commercial frameworks
  - Ghidra
  - Ida Pro
  - Binary Ninja
  - Radare2
  - WinDBG
  - OllyDBG


CodeBrowser: test:/encrypt.elf
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
encrypt.elf
.rodata
.text
.shstrtab
.strtab
Program Tree
Symbol Tree
uart_write_u32
Functions
AddRoundKey
AES128_ECB_encrypt
main
memset
prompt
ResetHandler
strlen
strncmp
SubBytes
transform_to_nibble
uart_read_byte
uart_write
Filter:
Data Type Manager
Data Types
BuiltInTypes
encrypt.elf
generic_clib
Filter:
Listing: encrypt.elf
FUNCTION
undefined AddRoundKey()
assume LRset = 0x0
assume TMode = 0x1
undefined r0:1 <RETURN>
AddRoundKey XREF
0000001c 30 b5 push { r4, r5, lr }
0000001e 04 01 lsls r4,r0,#0x4
00000020 02 f1 03 0e add.w lr,r2,#0x3
00000024 02 f1 13 05 add.w r5,r2,#0x13
LAB_00000028 XREF
00000028 ae f1 04 03 sub.w r3,lr,#0x4
0000002c 01 eb 04 0c add.w r12,r1,r4
LAB_00000030 XREF
00000030 1c f8 01 2b ldrb.w r2,[r12],#0x1
00000034 13 f8 01 0f ldrb.w r0,[r3,#0x1]!
00000038 42 40 eors r2,r0
0000003a 1a 70 strb r2,[r3,#0x0]
0000003c 73 45 cmp r3,lr
0000003e f7 d1 bne LAB_00000030
00000040 04 34 adds r4,#0x4
00000042 0e f1 04 0e add.w lr,lr,#0x4
00000046 ae 45 cmp lr,r5
00000048 ee d1 bne LAB_00000028
0000004a 30 bd pop { r4, r5, pc }
FUNCTION
Decompile: AddRoundKey - (encrypt.elf)
void AddRoundKey(int param_1,int param_2,int param_3)
{
byte *pbVar1;
byte *pbVar2;
byte *pbVar3;
param_1 = param_1 << 4;
pbVar3 = (byte *)(param_3 + 3);
do {
pbVar1 = pbVar3 + -4;
pbVar2 = (byte *)(param_2 + param_1);
do {
pbVar1 = pbVar1 + 1;
*pbVar1 = *pbVar2 ^ *pbVar1;
pbVar2 = pbVar2 + 1;
} while (pbVar1 != pbVar3);
param_1 = param_1 + 4;
pbVar3 = pbVar3 + 4;
} while (pbVar3 != (byte *)(param_3 + 0x13));
return;
}
Bookmarks - (0 bookmarks)
Type
Category
Description
Location
Label
Code Unit
Filter:
Console
Bookmarks
0000003e
AddRoundKey
bne 0x00000030


File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
.bss
.data
.got.plt
.got
.dynamic
.data.rel.ro
.fini_array
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
generic_clib
generic_clib_64
Filter:
Listing: - (4 addresses ...
0013b115 55 PUSH RBP
0013b116 48 89 e5 MOV RBP,RSP
0013b119 48 83 ec 50 SUB RSP,0x50
0013b11d 48 89 7d b8 MOV qword ptr [RBP + lo
0013b121 48 8d 05 LEA RAX,[s_-----BEGIN_E
e0 ce 17 00
0013b128 48 89 45 f0 MOV qword ptr [RBP + lo
0013b12c 48 c7 45 MOV qword ptr [RBP + lo
e8 00 00
00 00
0013b134 48 c7 45 MOV qword ptr [RBP + lo
d0 00 00
00 00
0013b13c 48 c7 45 MOV qword ptr [RBP + lo
f8 00 00
00 00
0013b144 48 c7 45 MOV qword ptr [RBP + lo
e0 00 00
00 00
0013b14c 48 c7 45 MOV qword ptr [RBP + lo
c8 00 00
00 00
0013b154 48 c7 45 MOV qword ptr [RBP + lo
c0 00 00
00 00
0013b15c 48 8b 45 f0 MOV RAX,qword ptr [RBP
0013b160 48 89 c7 MOV RDI=>s_-----BEGIN_E
0013b163 e8 88 f0 CALL <EXTERNAL>::strlen
Decompile:
11 int local_2c;
12 ECDSA_SIG *local_28;
13 BIO *local_20;
14 char *local_18;
15 ec_key_st *local_10;
16
17 local_18 =
18 "-----BEGIN EC PRIVATE KEY-----\nhCn5yLa2Hgj26PfzYF8+Q6sBuc
oZWVonxXB43MfAY1dcHrJ+dgq6AEnjEMrKICJr==\n-----END EC PRIVA
19 ;
20 local_20 = (BIO *)0x0;
21 local_38 = (EVP_PKEY *)0x0;
22 local_10 = (ec_key_st *)0x0;
23 local_28 = (ECDSA_SIG *)0x0;
24 local_40 = (BIGNUM *)0x0;
25 local_48 = (BIGNUM *)0x0;
26 sVar1 = strlen(
27 "-----BEGIN EC PRIVATE KEY-----\nhCn5yLa2Hgj2
GD2ZQBdACKV1pGoZWVonxXB43MfAY1dcHrJ+dgq6AEnjE
\n"
28);
29 local_20 = BIO_new_mem_buf(local_18,(int)sVar1);
30 if (local_20 == (BIO *)0x0) {
31 fprintf(stderr,"(%05d) - %s]\n error: create bio failure
32 }
33 else {
34 PEM_read_bio_PrivateKey(local_20,&local_38,(undefined1 *)
35 if (local_38 == (EVP_PKEY *)0x0) {
Console - Scripting
0013b128
MOV qword ptr [RBP + -0x10],RAX

Consequences

Consequences



Software Reverse Engineering

Intellectual Property Theft:
- Theft of valuable algorithms
- Cloning of devices

Key or Token Stealing:
- Identify hardcoded keys or tokens
- Access to sensitive data or infrastructure

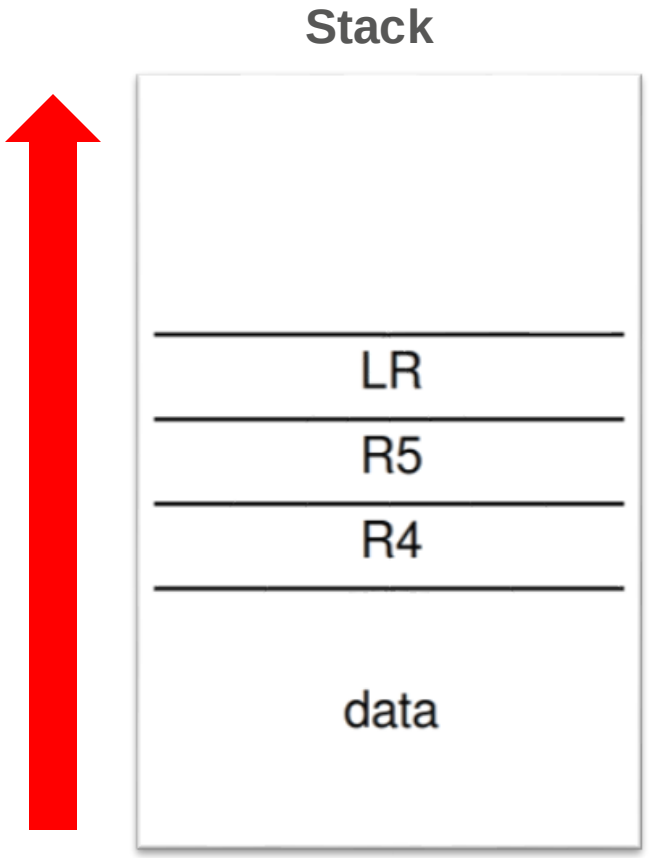
Cracking:
- Bypass license checks
- Paid feature activation

Vulnerability Analysis:
- Identify exploitable weaknesses
- Undermine system security

Software Vulnerabilities – Buffer overflow

- Buffer overflow is a data write beyond its allocated memory

```
unsigned int i = 0;
while (1) {
    data[i] = 0;

    uart_write(instance, "\r[+] plaintext ('ENTER' as last cha
    uart_write(instance, data);

    char value = 0;
    while (uart_read_byte(instance, &value) != UART_OK) {}

    if ((value == '\n') || (value == '\r')) {
        break;
    }

    if (value == 'q') {
        return 1;
    }

    data[i] = value;

    i++;
}
```



Buffer Overflow – Why?

emproof

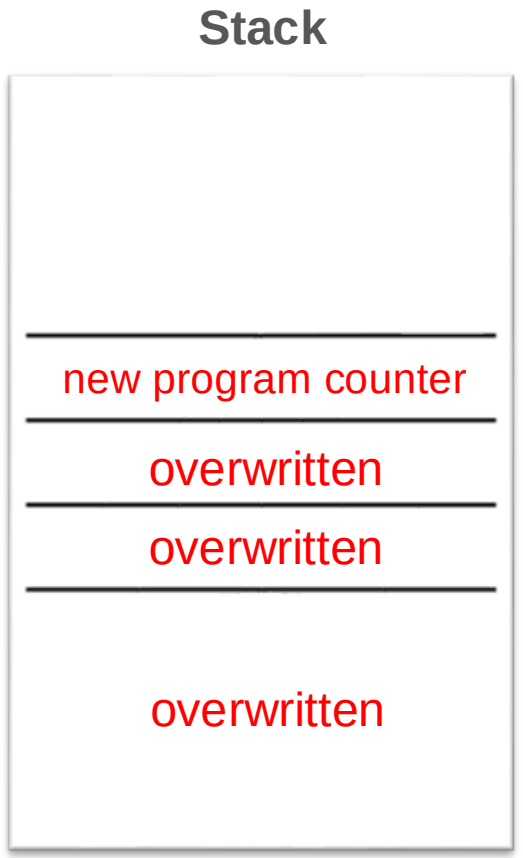
- Function prologue prepares stack and registers

```
push    {r4, r5, lr}
sub     sp, #0x10
```

- Function epilogue restores stack and registers

```
add     sp, #0x10
pop     {r4, r5, pc}
```

- **Control-flow re-direction**



Hands-On Part

Slides, Setup & Samples

Check it out on GitHub:

https://github.com/emproof-com/workshop_software_reverse_engineering_escar24

Task 1: Hello World

emproof

Open `hello_world` in Ghidra.

- Load and analyze the binary.
- Get used to the analysis window (functions, assembly, decompiler).
- Inspect the strings in the binary.
- Locate the `main` function. What does it do?
- Repeat the task in Binary Ninja.

Task 2: Game Reverse Engineering

- Execute `game` and check how it works.
- Open it in Ghidra and inspect the strings.
- Locate the main function and inspect it in the decompiler.
- Locate the hardcoded license to unlock the full version.

Task 3: Game Cracking

emproof

- Open game in Binary Ninja and open the `main` function.
- Identify the variable that hardcodes the number of trials.
- Patch the binary such that it accepts 5 trials.
- Patch the code such that it accepts any input as valid license.

Task 4: Feature Unlocking

emproof

Open `license_check` in Binary Ninja.

- Locate the hardcoded secret to unlock the premium feature.
- Patch the binary such that the feature is always unlocked

Task 5: Embedded Firmware

emproof

Open the embedded firmware `car_demo.elf` in Ghidra.

- Inspect the strings. What functionality might the firmware implement?
- The firmware hardcodes Wifi credentials (SSID & password). Find them.
- What is the IP address of the router?

Software Hardening

Software Hardening

Reverse Engineering Protection

Safeguards software from unauthorized analysis, preventing intellectual property theft and security breaches

Exploit Mitigation

Security measures to block attackers from exploiting known vulnerabilities

Techniques:

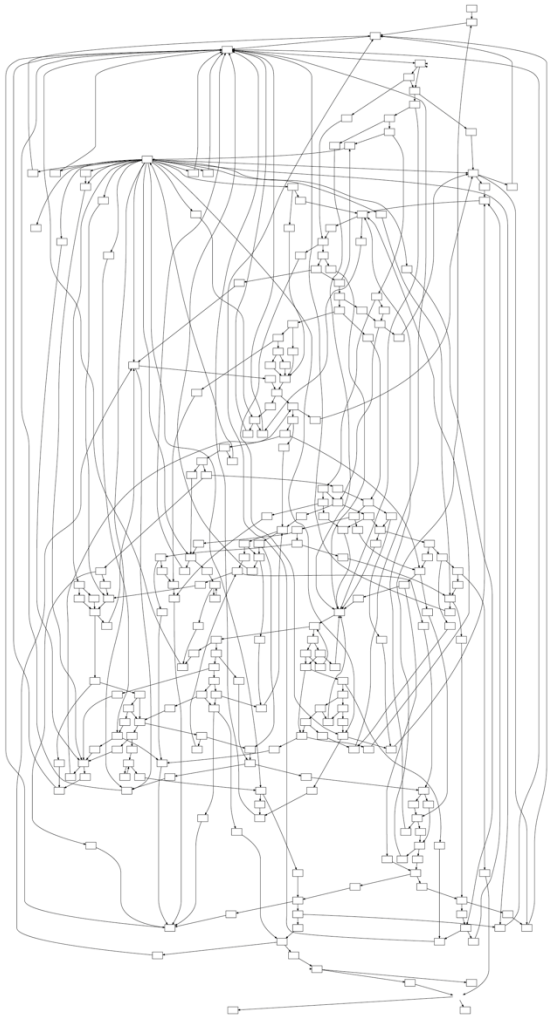
- Code Obfuscation & Data Encoding
- Anti-Debug/Emulation
- Anti-Tamper

Techniques:

- Stack Canaries
- Control-flow Integrity

Reverse Engineering Protection

Code Obfuscation & Data Encoding



- Increase **code complexity** to impede **reverse engineering**
- **Hide keys and credentials** and decode them at **runtime**

Anti-Debug & Anti-Tamper

- observe execution environment for debuggers (anti-debug)
- detect code modifications (patching) by code checksumming (anti-tamper)

Exploit Mitigation

Challenges

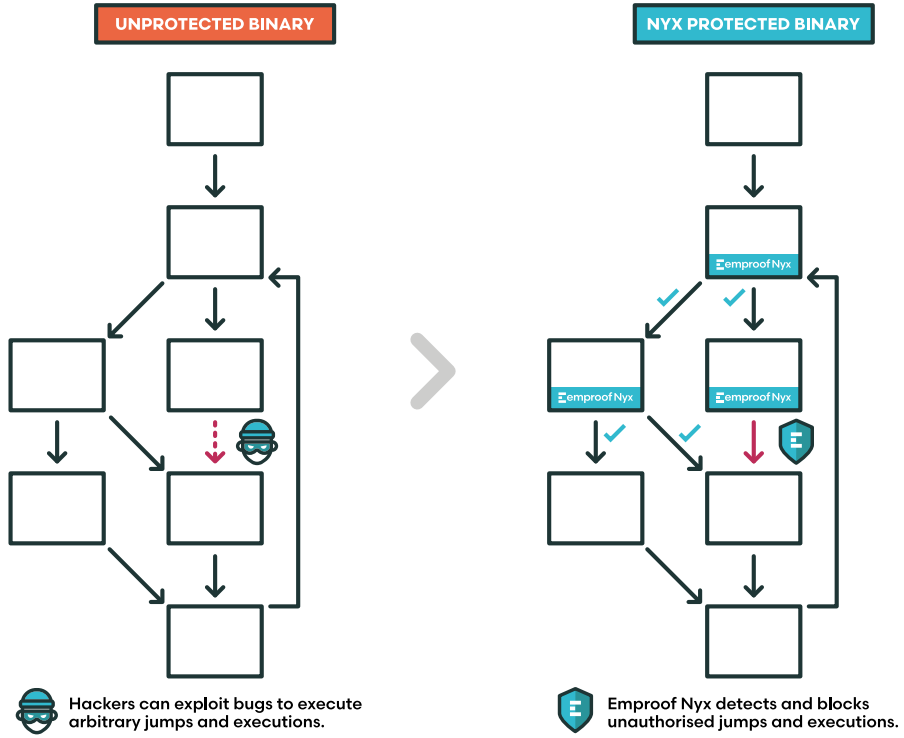
- Exploit mitigations are standard on desktop / mobile systems

- Software hardening is **a challenge on embedded systems**

  - Old development stacks for functional safety reasons (confidence of use)
  - Compilers, like gcc or clang, often do not support basic mitigations, such as canaries
  - Bare-metal systems or some RTOS often do not offer support
  - Little overhead available

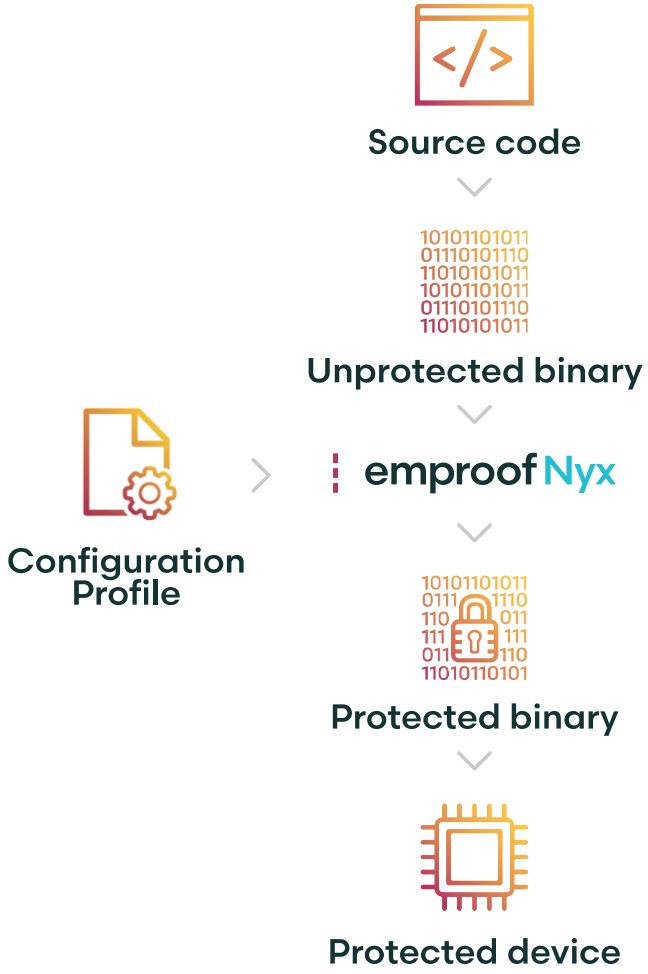
Exploit Mitgations

- **Stack Canaries**
  - Guards that are placed on the stack that make sure that the stack variables have not been overwritten
- **Control-Flow Integrity**
  - Detect control-flow transfers to unintended locations



Emproof Nyx

- **State-of-the-art embedded software security**
  Exploit mitigation, obfuscation, anti-fuzzing, anti-tampering and platform security
- **Integrates with customer's existing design flow**
  No disruption - Emproof Nyx works on the binary
- **Easy customization**
  Fine-granular control of protections, enables targeted security for critical functions
- **Proven solution**
  Faster time to market and saves costs as no in-house expertise required
- **Achieve compliance**
  Assures adherence to upcoming regulations, including the CRA, U.S. Cyber Trust Mark, and RED act.



Emproof Nyx Software Suite

Two pillars of security

Code protection
Safeguards embedded firmware and applications from theft of intellectual property, competitor analysis and unauthorized software modifications.

- IP theft
- Key extraction
- Reverse engineering

Security hardening
Prevents the detection and exploitation of known and unknown software vulnerabilities as well as malware and privilege escalation attacks.

- Denial of service
- Non-fixable bugs
- Zero-day exploits
- Malware

Protections

Code obfuscation
Visual manipulation of the data using both static and dynamic analysis to provide total protection.

Anti-fuzzing
Code Transformation techniques used to slow down and increase the randomized data making fuzzing ineffective when used to attack the device.

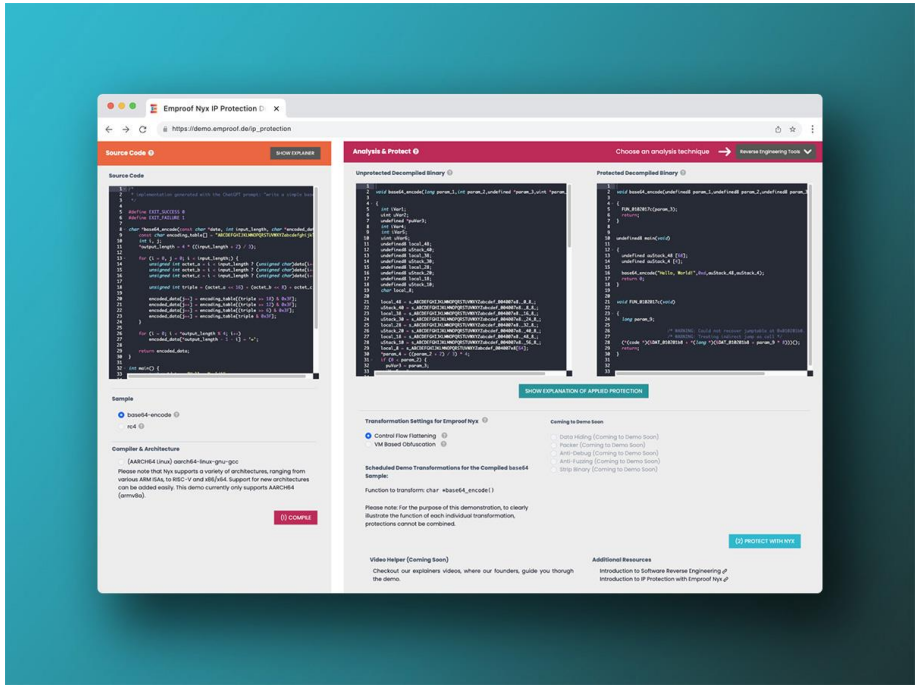
Platform security
If additional security measures such as unique identifier data or hardware security modules are available these are incorporated to provide enhanced security.

Anti-tamper
On start-up the deployed software image is checked against a known verified version enabling a customer-defined action (e.g., set device to safe-state).

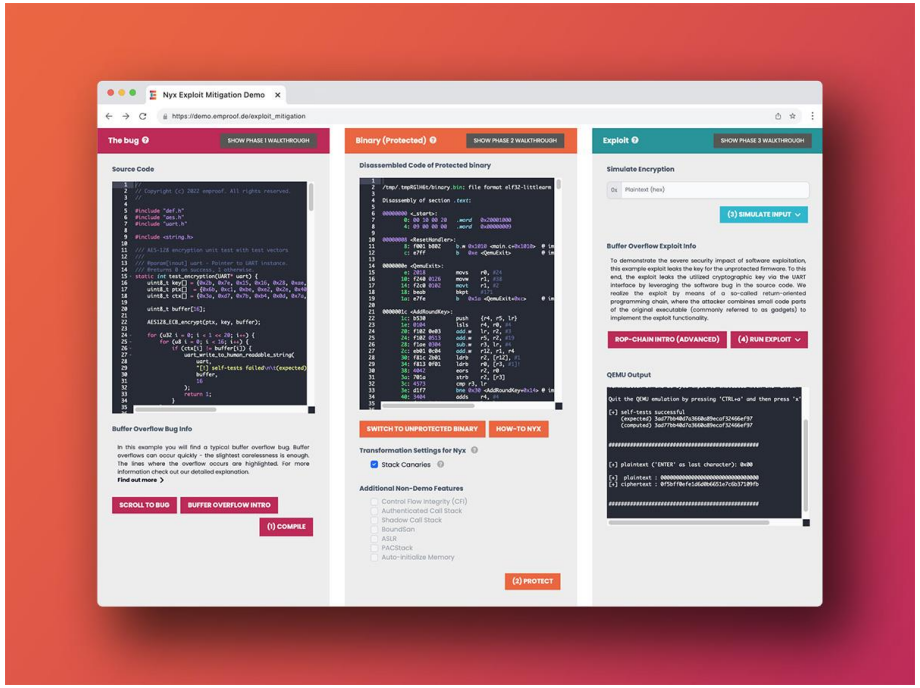
Exploit mitigation
Little guards such as stack canaries and control-flow integrity used to detect attacks and have a customer-defined action (e.g., reset the system...).

Try Our Online Demo

IP Protection



Exploit Mitigation



Visit: demo.emproof.com

Thank you for your attention!

emproof.com