Supply-Chain Security : Trust but Verify

Jonathan Brossard

13th of March 2025

Agenda

- Motivation
- Problem Statement
- Introduction to ELF Libification
- Procedural Debugging
- Partial Proofs of Vulnerabilities
- Conclusion & Future Work

le cnam

Conformité réglementaire

MOABI

IGI 1300

ii 901

im 900

NIS2 Directive

EXPORT CONTROLS

EXPORT ADMINSTRATION REGULATIONS

LPM 2024 -2030
Les grandes orientations

Cyber Resilience Act

Motivation

Motivation : SBOMs vulnerability assessments don't scale well

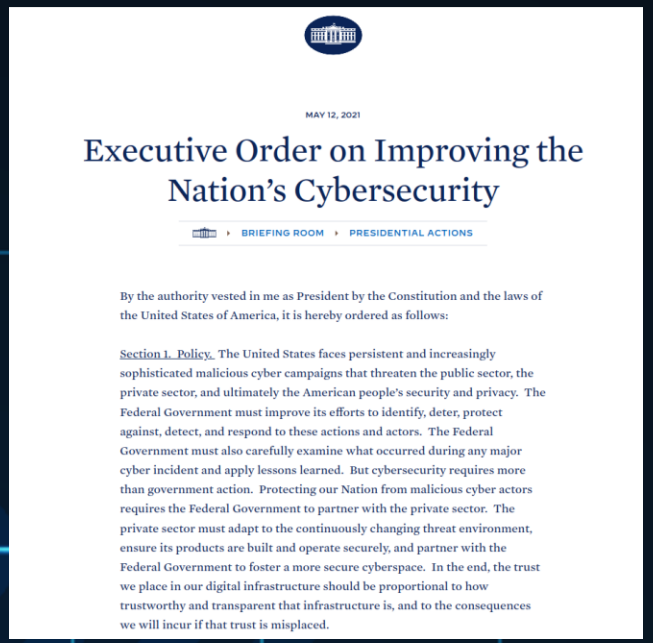
Software Bill of Materials (SBOMs) contain lists of CPEs or Package URLS (purl) describing all the components of a given Software.

They allow to perform vulnerability assessments by comparing the CPEs to the dictionaries published by the NIST for each CVE.


```
"components": [
  {
    "type": "application",
    "name": "sshd",
    "hashes": [{
      "alg": "MD5",
      "content": "0e6e1c30f8e10e45647ba5d9d4ea6948"
    },{
      "alg": "SHA-1",
      "content": "399e5d00e2d91628792eefe066489f64a6310486"
    },{
      "alg": "SHA-256",
      "content": "5bd9544d2da6daf7519ee8d6efcd71d6edc16a507f86a90e233174bbebb843c2"
    }]
  }
  ,
  {
    "type": "application",
    "description": "sshd",
    "group": "openbsd",
    "name": "openssh",
    "version": "9.6p1",
    "cpe": "cpe:2.3:a:openbsd:openssh:9.6p1:*:*:*:*:*:*:*",
    "externalReferences": [{
        "type": "website",
        "url": "https://www.openssh.com/"
    }],
    "licenses": [{
      "license": {
        "id": "BSD-2-clause"
      },
      "license": {
        "id": "BSD-3-clause"
      },
```


Software Bill of Materials are becoming mandatory

Motivation : SBOMs vulnerability assessments don't scale well



MAY 12, 2021

Executive Order on Improving the Nation's Cybersecurity

BRIEFING ROOM › PRESIDENTIAL ACTIONS

By the authority vested in me as President by the Constitution and the laws of the United States of America, it is hereby ordered as follows:

Section 1. Policy. The United States faces persistent and increasingly sophisticated malicious cyber campaigns that threaten the public sector, the private sector, and ultimately the American people's security and privacy. The Federal Government must improve its efforts to identify, deter, protect against, detect, and respond to these actions and actors. The Federal Government must also carefully examine what occurred during any major cyber incident and apply lessons learned. But cybersecurity requires more than government action. Protecting our Nation from malicious cyber actors requires the Federal Government to partner with the private sector. The private sector must adapt to the continuously changing threat environment, ensure its products are built and operate securely, and partner with the Federal Government to foster a more secure cyberspace. In the end, the trust we place in our digital infrastructure should be proportional to how trustworthy and transparent that infrastructure is, and to the consequences we will incur if that trust is misplaced.

EU Cyber Resilience Act

New EU cybersecurity rules ensure safer hardware and software.

From baby-monitors to smart-watches, products and software that contain a digital component are omnipresent in our daily lives. Less apparent to many users is the security risk such products and software may present.

The Cyber Resilience Act (CRA) aims to safeguard consumers and businesses buying or using products or software with a digital component. The Act would see inadequate security features become a thing of the past with the introduction of mandatory cybersecurity requirements for manufacturers and retailers of such products, with this protection extending throughout the product lifecycle.

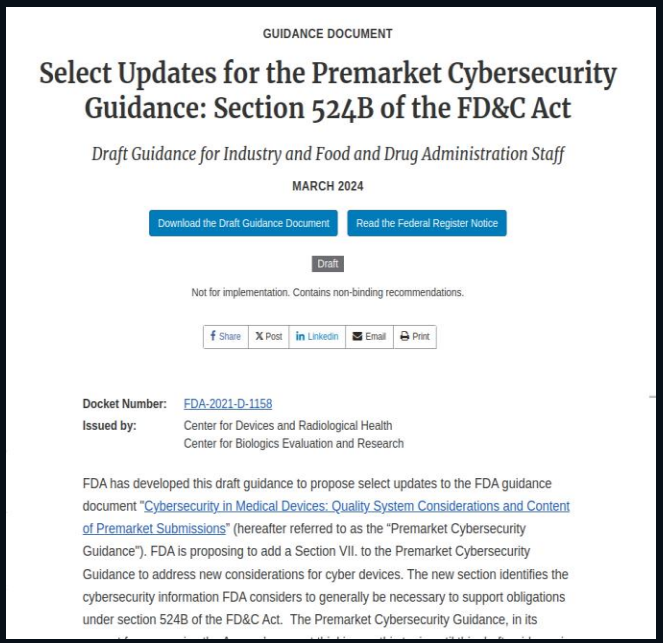
The problem addressed by the Regulation is two-fold.

First is the inadequate level of cybersecurity inherent in many products, or inadequate security updates to such products and software.

Second is the inability of consumers and businesses to currently determine which products are cybersecure, or to set them up in a way that ensures their cybersecurity is protected.

The Cyber Resilience Act will guarantee:

- harmonised rules when bringing to market products or software with a digital component;



GUIDANCE DOCUMENT

Select Updates for the Premarket Cybersecurity Guidance: Section 524B of the FD&C Act

Draft Guidance for Industry and Food and Drug Administration Staff

MARCH 2024

Download the Draft Guidance Document | Read the Federal Register Notice

Draft

Not for implementation. Contains non-binding recommendations.

Share | Post | Linkedin | Email | Print

Docket Number: FDA-2021-D-1158
Issued by: Center for Devices and Radiological Health
Center for Biologics Evaluation and Research

FDA has developed this draft guidance to propose select updates to the FDA guidance document "Cybersecurity in Medical Devices: Quality System Considerations and Content of Premarket Submissions" (hereafter referred to as the "Premarket Cybersecurity Guidance"). FDA is proposing to add a Section VII. to the Premarket Cybersecurity Guidance to address new considerations for cyber devices. The new section identifies the cybersecurity information FDA considers to generally be necessary to support obligations under section 524B of the FD&C Act. The Premarket Cybersecurity Guidance, in its

Software Bill of Materials are becoming mandatory

Motivation : SBOMs vulnerability assessments don't scale well

MOABI

243 SOFTWARE MONITORED

SOFTWARE	SCAN DATE	NEW CVEs	Last Modified
Hyundai.update.zip	10 August 2022	1230	8 August 2024
psa_rcc.zip	10 August 2022	1151	8 August 2024
Firmware.bin	17 October 2022	952	8 August 2024
Tesla_2019.20.4.2.model3.squashfs	20 June 2023	467	8 August 2024

SBOMs provide possible CVES.
For each vulnerability : is it true ?

CONTINUOUS MONITORING

1019 new vulnerabilities identified for software Tesla_2019.20.4.2.model3.squashfs

Updated report for **dnsmasq**

CVE-2023-50387 — CVSS v3 : 7.5

CVE-2023-50387

Description	Certain DNSSEC aspects of the DNS protocol (in RFC 4033, 4034, 4035, 6840, and related RFCs) allow remote attackers to cause a denial of service (CPU consumption) via one or more DNSSEC responses, aka the "KeyTrap" issue. One of the concerns is that, when there is a zone with many DNSKEY and RRSIG records, the protocol specification implies that an algorithm must evaluate all combinations of DNSKEY and RRSIG records.
Impact	CVSS v3 : 7.5 HIGH
Type	CWE-770
Attack Vector	CVSSv3 Vector : CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

le cnam

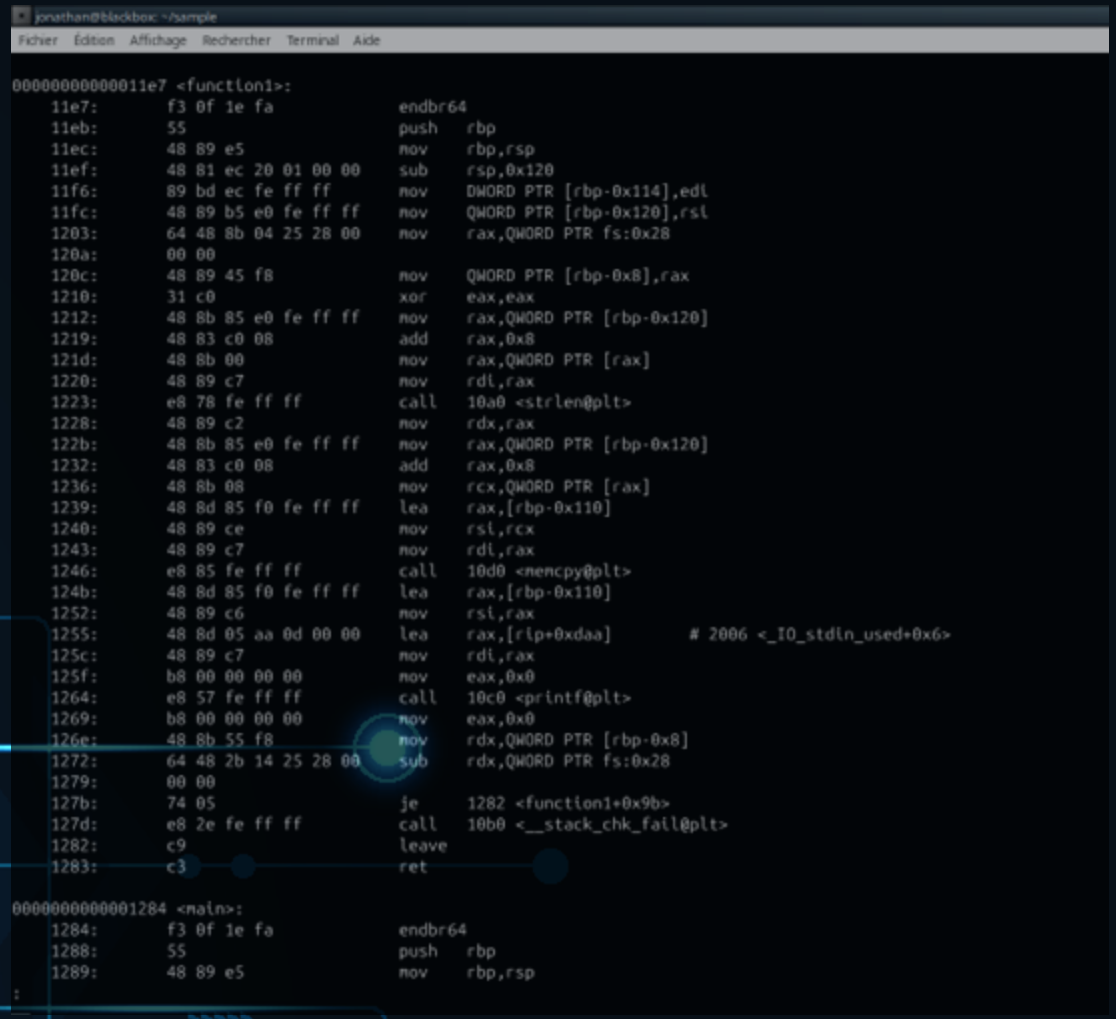
Motivation (bis)

Zerodays : No patch, am I vulnerable ?

MOABI

Taint analysis on binary
(no source code)

Calling convention SYSTEM V:
RDI, RSI, RDX, RCX, R8, R9



le cnam

Zerodays : No patch, am I vulnerable ?

MOABI

Taint analysis on binary
(no source code)

ASLR	YES
FORTIFY	NO
FULL RELOCATIONS	YES
NX (STACK)	YES
STACK COOKIES	YES
W^X	YES

Vulnerability — **CWE-120 - BUFFER OVERFLOW** — **Score : 7**

Impact: 9	Confidence: 9	Risk: 5

Type	**CWE-120 - BUFFER OVERFLOW**
Address	00001246
Function	000011e7
Description	Vulnerability when calling function memcpy() : * function argument 3 has the following properties: user controled, command line argument, arithmetic operation, upper function argument 2, stack variable at offset -0x30, return value from strlen(), * source has the following properties: user controled, command line argument, arithmetic operation, upper function argument 2, stack variable at offset -0x30, * destination has the following properties: stack variable at offset -0x10, ↳ buffer overflow in third argument
Backtrace	#00 <1246> void * __cdecl memcpy(__out_bcount_full_opt(_Size) void * _Dst, __in_bcount_opt(_Size) const void * _Src, __in size_t _Size) at: /home/jonathan/sample/sample.c:22 #01 <11e7> function1() at: /home/jonathan/sample/sample.c:19 #02 <1284> int main (int argc, char **argv, char **envp) at: /home/jonathan/sample/sample.c:29

le cnam

Problem Statement :

Partial Proofs of Vulnerabilities through Procedural Debugging

Proving Exploitability

Industry standard to Prove exploitability : Write an exploit

This bar is too high.

If we decompose an exploit into 3 problems:

- Reach the vulnerable function
- Trigger the vulnerability
- Achieve code execution/Weaponize

The first step alone is already undecidable ("reachability problem").

Undecidable

Proving Exploitability : Partial Proof of Vulnerability

Let's do only step 2:

If we decompose an exploit into 3 problems:

- ~~Reach the vulnerable function~~
- Trigger the vulnerability
- ~~Achieve code execution/Weaponize~~

This a reasonable heuristic to determine vulnerability of the application.

We'd like to be able to call the vulnerable function directly.
Problem : How to do this out of context ?
Proposal : Let's turn the vulnerable application into a shared library !

Proving Exploitability : Procedural Debugging

ELF Executable
or Library

Load in Memory
(+ load dependencies
& solve relocations)

function1()
...
function...()
...

function_vulnerable()

....

Address Space

We would like to map the ELF in memory, solve dependencies/relocations if the ELF is dynamically linked.

Then call the potentially vulnerable function with chosen arguments, with just enough context to trigger the vulnerability.

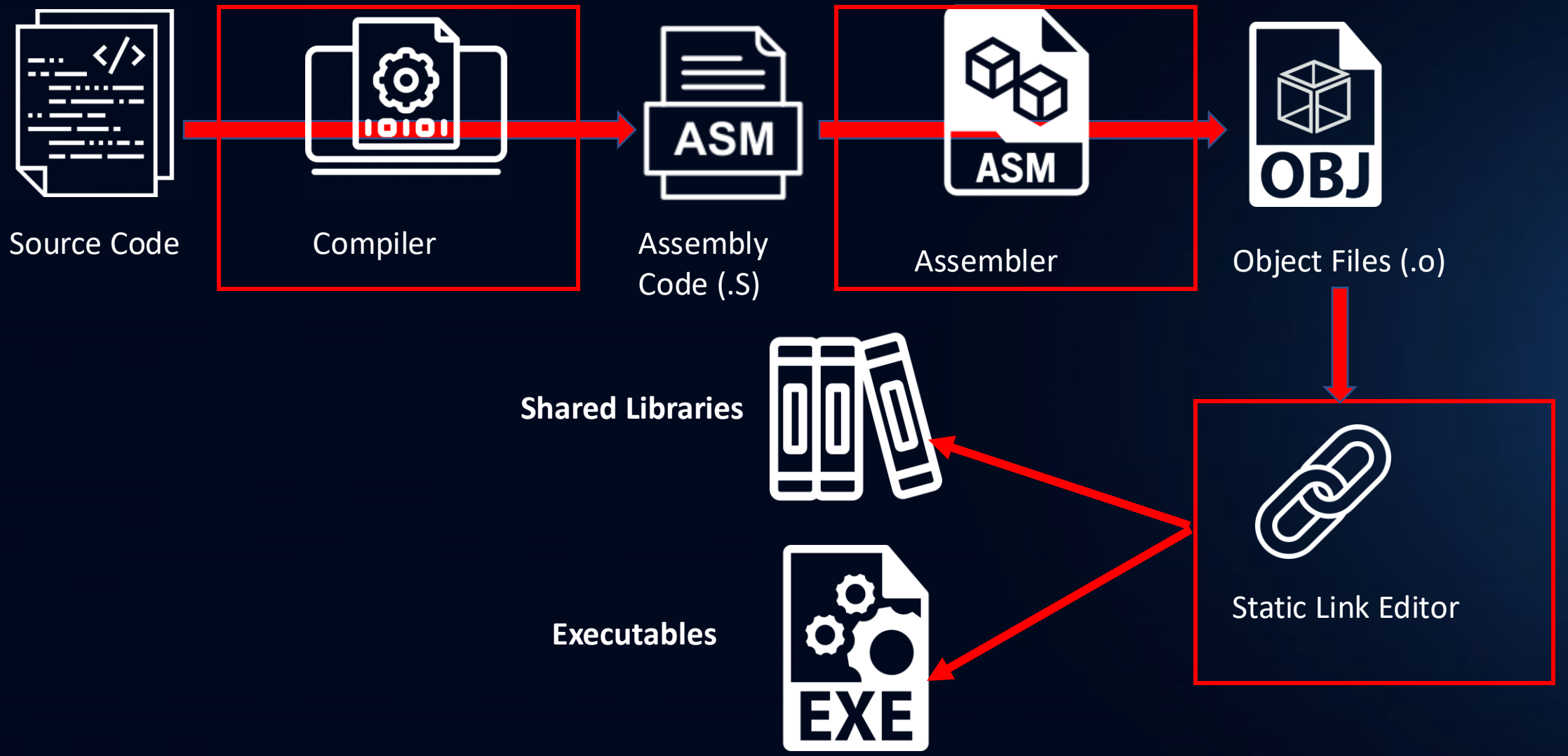
Note: If the ELF is statically linked, loading its segments is enough.
We simply need a userland ELF loader.
Confere userland exec() techniques (eg: The Grugq, phrack, 2004)

MOABI

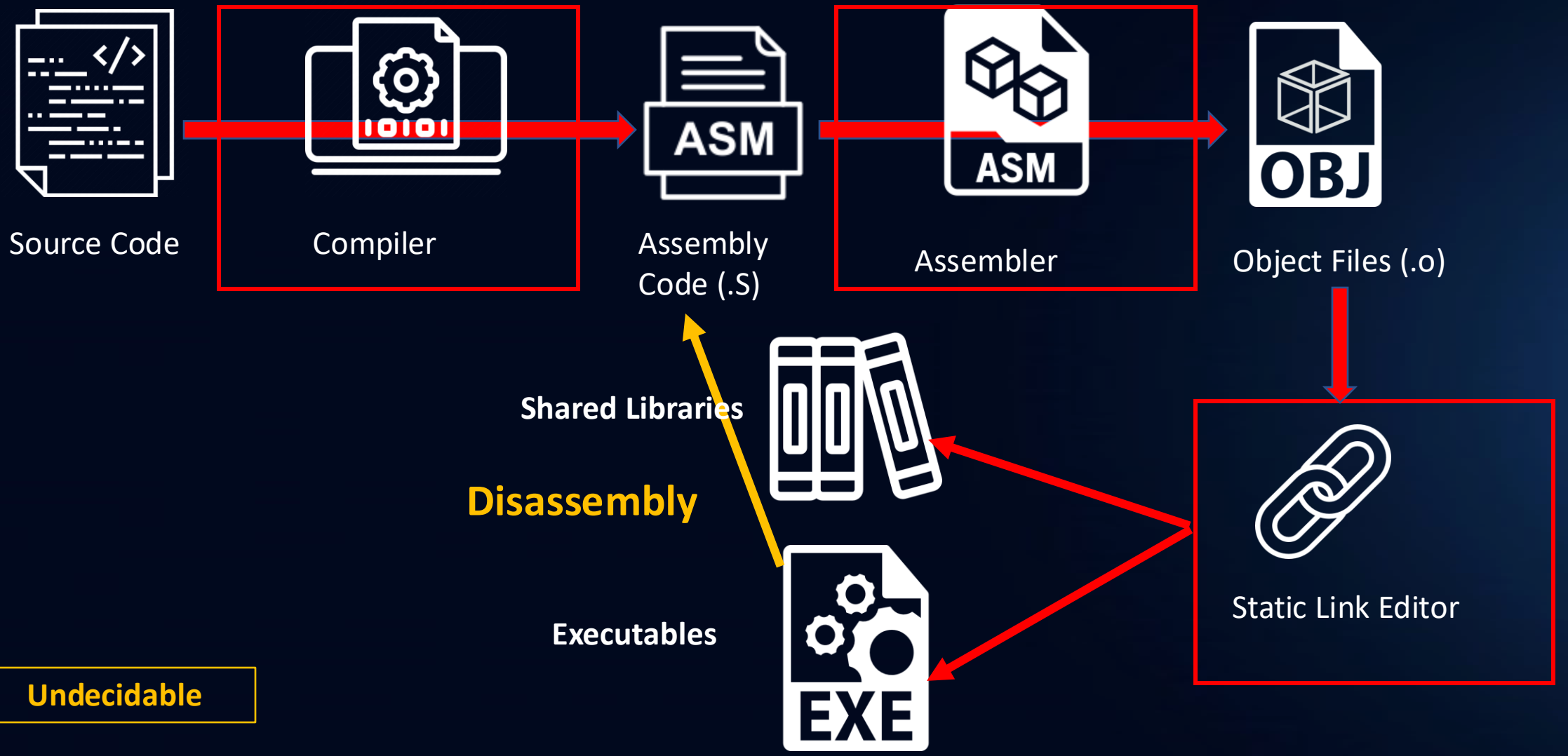
Loading Dynamic Executables : Introduction to ELF Libification

le cnam

Reverse Engineering : Compilation Process

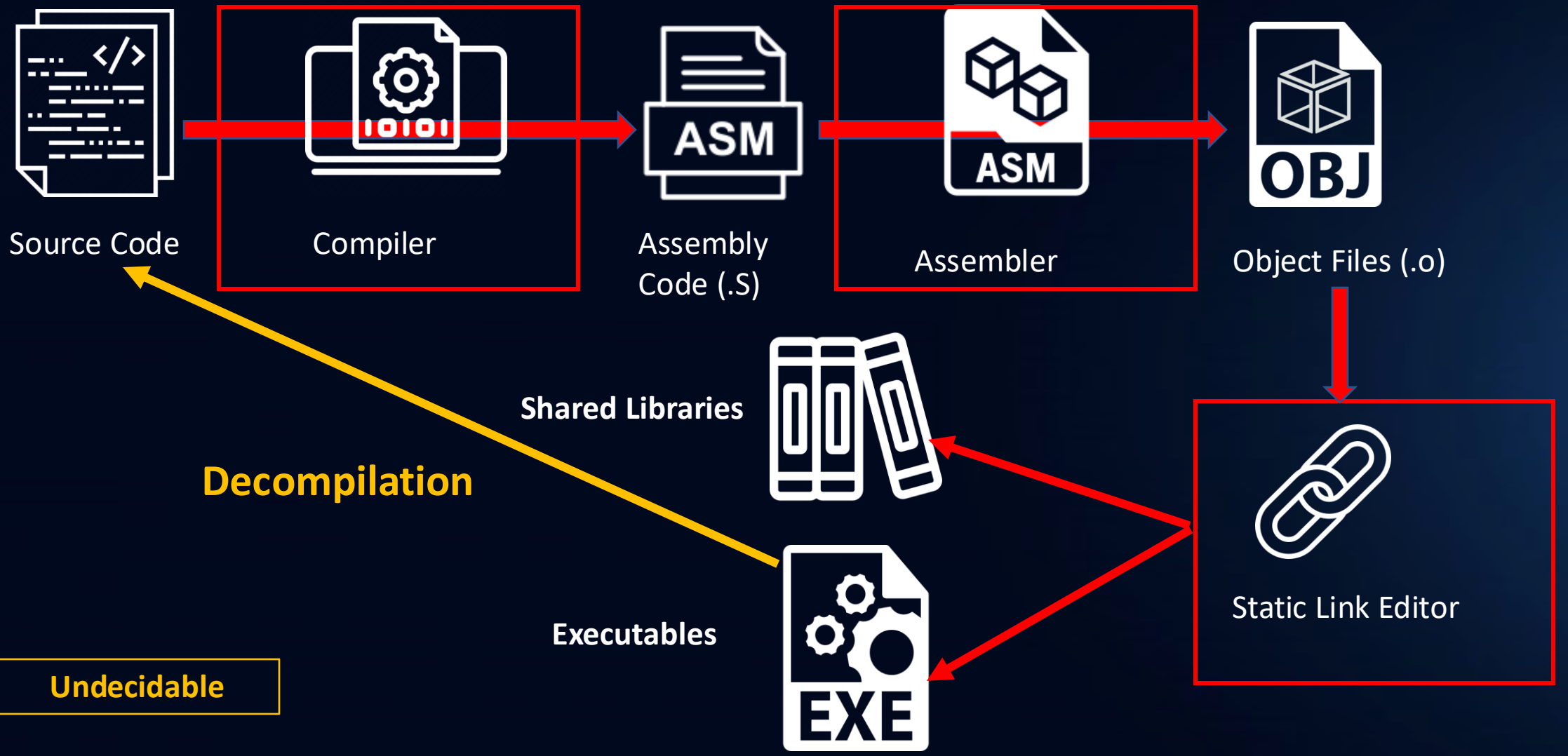


Reverse Engineering : Disassembly

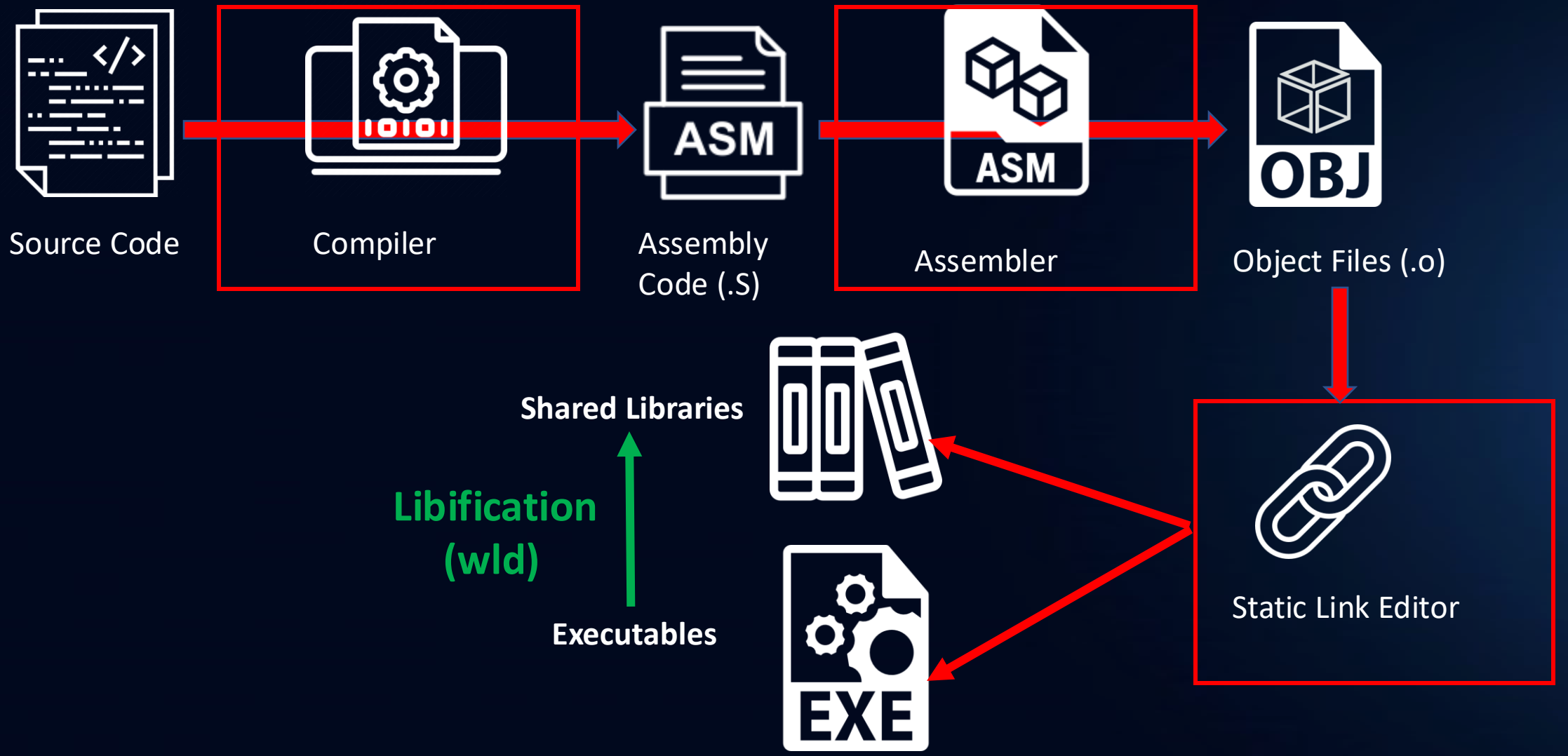


Undecidable

Reverse Engineering : Decompilation



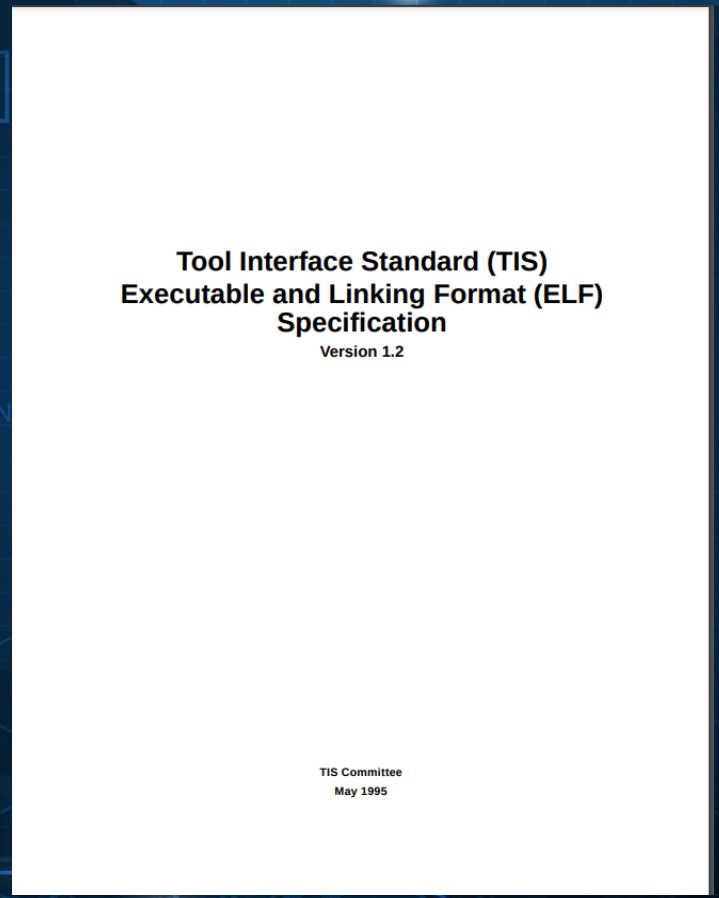
Reverse Engineering : Libification



Executables vs Shared Libraries : How Do They Differ ?

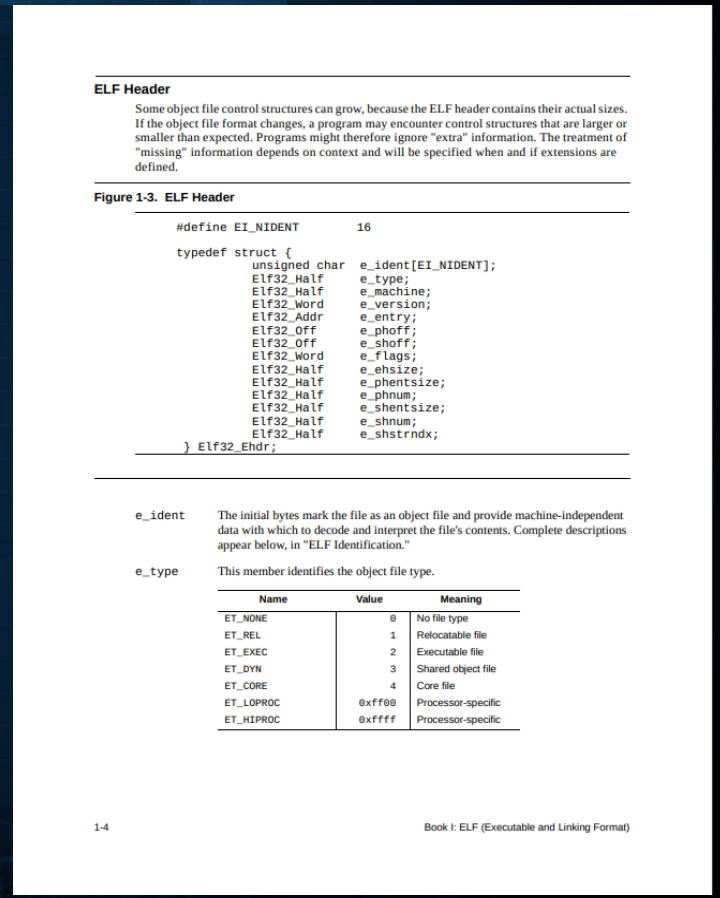
MOABI

Same headers, same segments, same sections. They mostly differ through their **metadata (various ELF headers)**



Tool Interface Standard (TIS)
Executable and Linking Format (ELF)
Specification
Version 1.2

TIS Committee
May 1995



ELF Header

Some object file control structures can grow, because the ELF header contains their actual sizes. If the object file format changes, a program may encounter control structures that are larger or smaller than expected. Programs might therefore ignore "extra" information. The treatment of "missing" information depends on context and will be specified when and if extensions are defined.

Figure 1-3. ELF Header

```
#define EI_NIDENT       16

typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

`e_ident` The initial bytes mark the file as an object file and provide machine-independent data with which to decode and interpret the file's contents. Complete descriptions appear below, in "ELF Identification."

`e_type` This member identifies the object file type.

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

1-4 Book I: ELF (Executable and Linking Format)

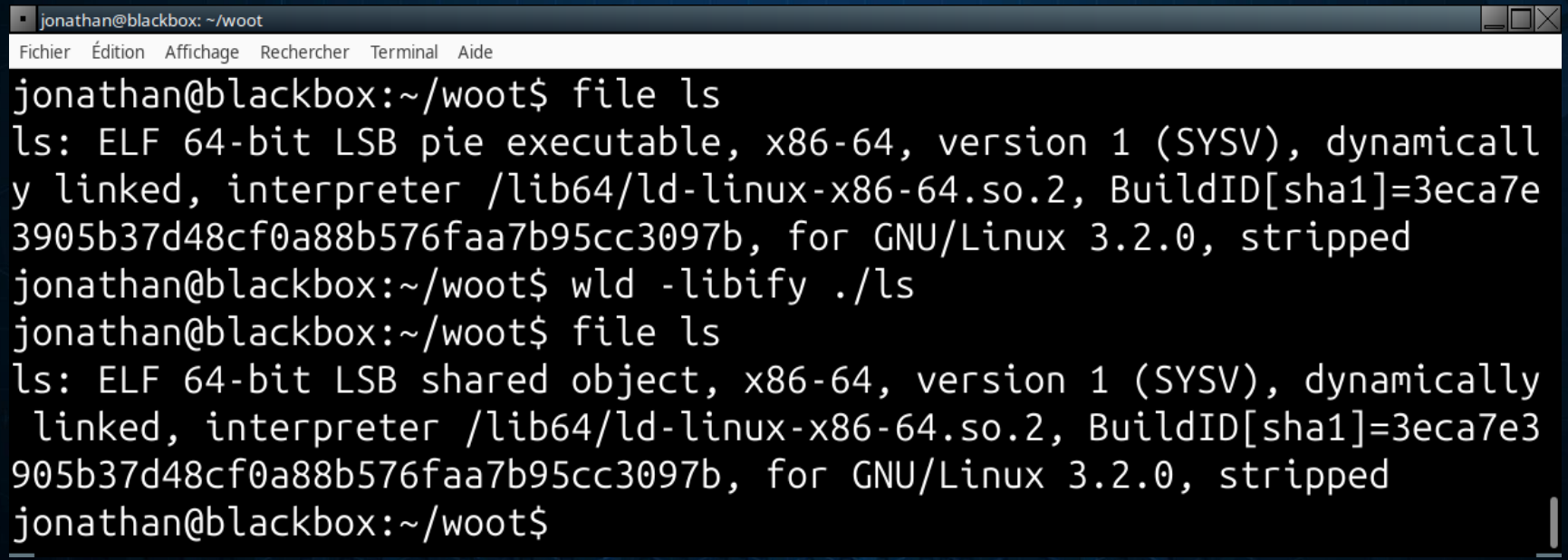
The work to be done:

Modify the various ELF headers to turn an Executable into a Shared Library

le cnam

Automated Libification : the Witchcraft Linker

URL: https://github.com/endrazine/wcc
License: MIT/BSD-2



ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

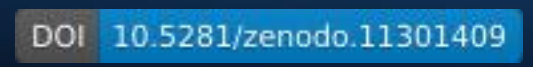
DOI 10.5281/zenodo.11405214

le cnam

Validation

Test Repository:
https://github.com/endrazine/wcc-tests

Version	Codename	Year	GCC Version	Passed	Failed
24.04	noble	2024	12.3.0	440	0
22.04	Jammy	2022	11.2.0	435	0
20.04	focal	2020	9.3.0	435	0
18.04	bionic	2018	7.3.0	369	0
16.04	xenial	2016	5.3.1	363	0
14.04	trusty	2014	4.8.2	352	0



Procedural Debugging

Procedural Debugging : Libification based Debugging

ELF Executable
or Library

Load in Memory
(+ load dependencies
& solve relocations)

function1()
...
function...()
...

function_vulnerable()

....

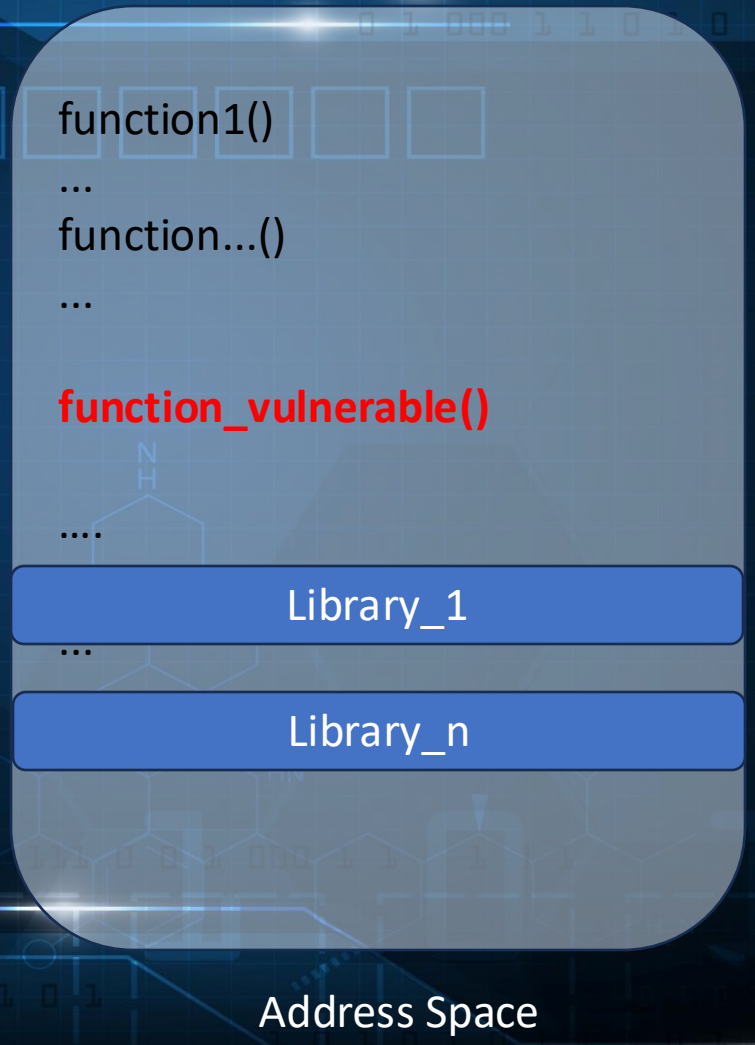
Address Space

If the ELF is dynamically linked, libify it and load via dlopen().

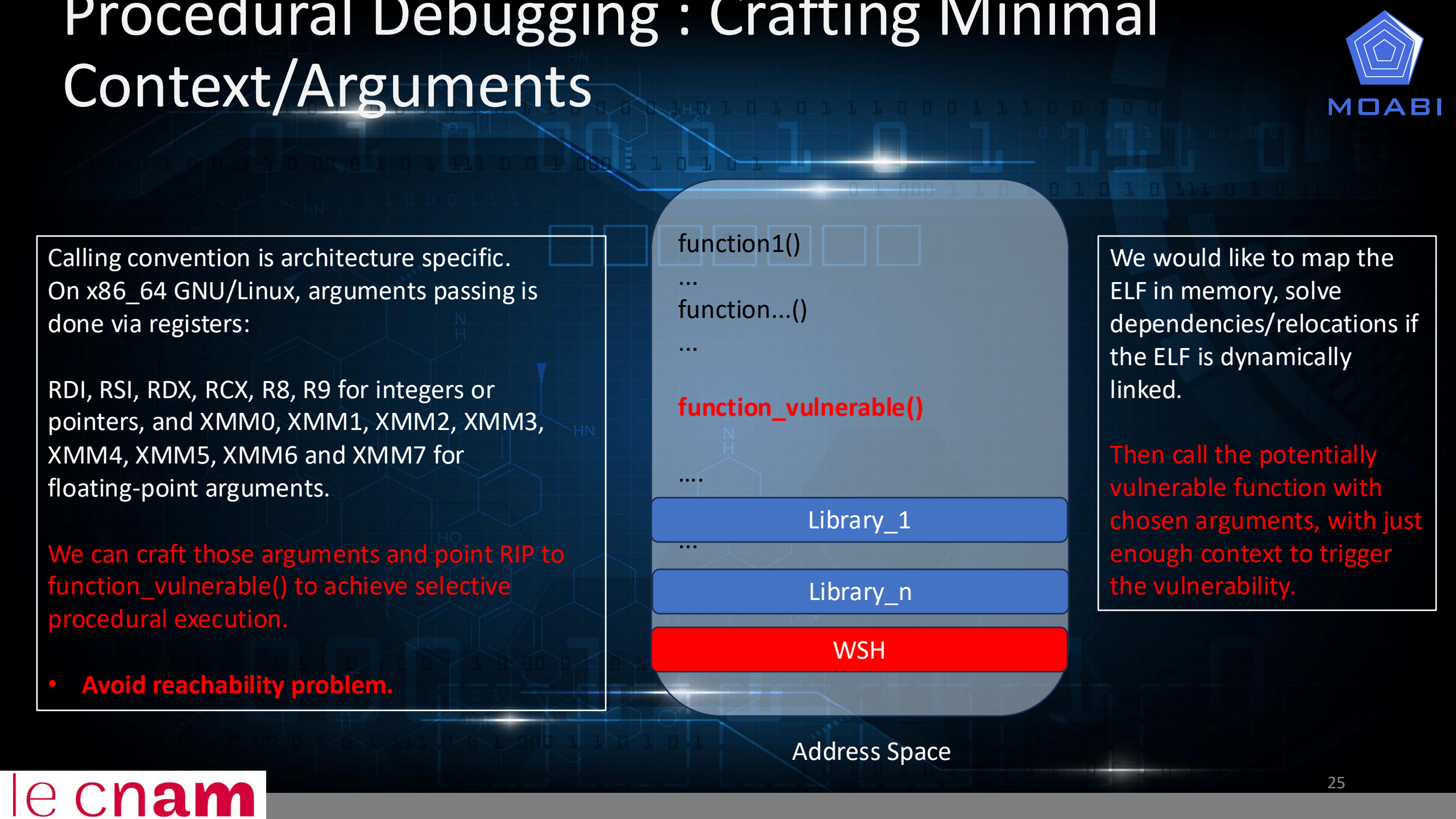
If the ELF is statically linked, loading its segments is enough.
We simply need a userland ELF loader.
Confere userland exec() techniques (eg: The Grugq, phrack, 2004)

(Without Control Flow Analysis, which is Undecidable)

In Dynamically linked ELFs, parse the link_map
- Provides a list of all ELFs in memory
- Dynamic linker exports iterators to symbols
- Use dlsym() function from the Dynamic Linker to obtain address of symbols.



Procedural Debugging : Crafting Minimal Context/Arguments



Calling convention is architecture specific.
On x86_64 GNU/Linux, arguments passing is done via registers:

RDI, RSI, RDX, RCX, R8, R9 for integers or pointers, and XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 and XMM7 for floating-point arguments.

We can craft those arguments and point RIP to function_vulnerable() to achieve selective procedural execution.

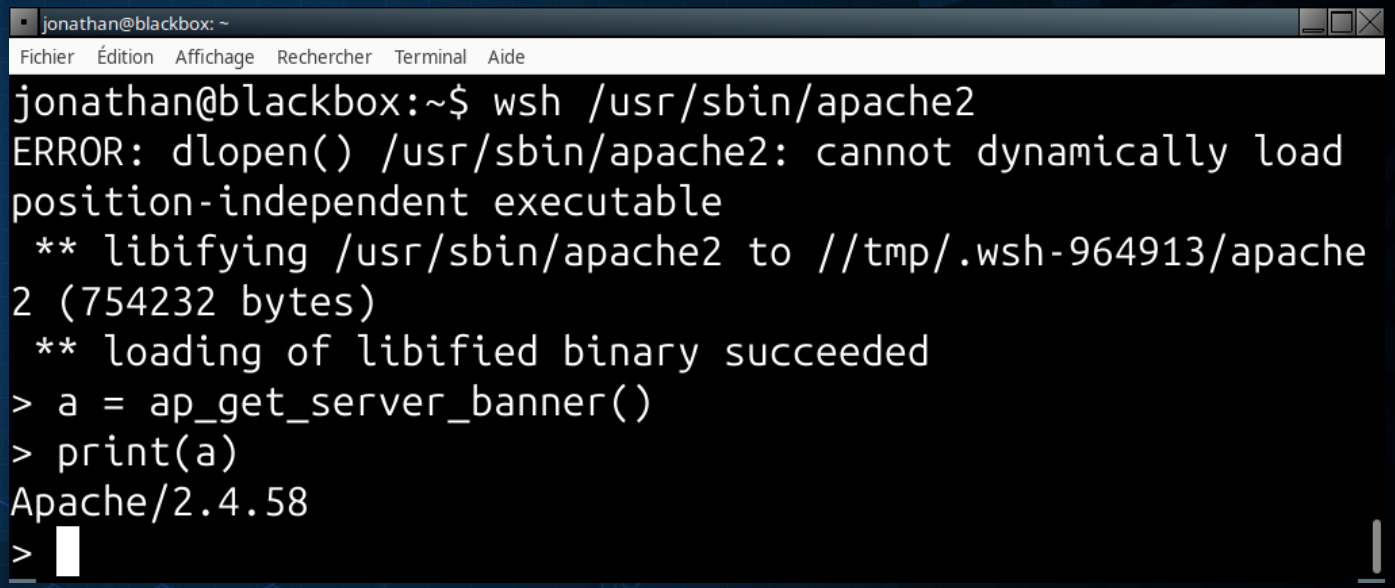
- **Avoid reachability problem.**

We would like to map the ELF in memory, solve dependencies/relocations if the ELF is dynamically linked.

Then call the potentially vulnerable function with chosen arguments, with just enough context to trigger the vulnerability.

Procedural Debugging

- Libify ELF executables
- Make ELF executables scriptable
- Call arbitrary functions (procedural debugging)



URL: https://github.com/endrazine/wcc
License: MIT/BSD-2

DOI 10.5281/zenodo.13902925

Partial Proofs of Vulnerabilities

Partial Proofs of Vulnerabilities

MOABI

The ability to turn ELF executables into libraries will allow us to create partial proofs of vulnerabilities in the form of WSH test scripts.

<u>Examples:</u>

https://github.com/endrazine/wcc-tests/tree/main/scripts

root@b80e8606fba6: ~/wcc-tests

Fichier Édition Affichage Rechercher Terminal Aide

```
root@b80e8606fba6:~/wcc-tests# cat ./scripts/CVE-2022-3786.wsh
print(" [*] testing for CVE-2022-3786")

-- Input arguments:
teststring = "a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--
.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--"
out = calloc(1,16)
outlen = calloc(1,2)
memset(outlen,0x10,1)

-- Trigger the stack overflow:
res = ossl_a2ulabel(teststring, out, outlen)

if res == 0 then print(" [*] Not vulnerable to CVE-2022-3786") end
root@b80e8606fba6:~/wcc-tests# wsh ../openssl/build-3.0.8/libcrypto.a < ./scripts/CVE-2022-3786.wsh
 -- relinked static library (ar archive) ../openssl/build-3.0.8/libcrypto.a into dynamic shared library /tmp/.wsh-51/libcrypto.a.so
 [*] testing for CVE-2022-3786
 [*] Not vulnerable to CVE-2022-3786
root@b80e8606fba6:~/wcc-tests# wsh ../openssl/build-3.0.7/libcrypto.a < ./scripts/CVE-2022-3786.wsh
 -- relinked static library (ar archive) ../openssl/build-3.0.7/libcrypto.a into dynamic shared library /tmp/.wsh-56/libcrypto.a.so
 [*] testing for CVE-2022-3786
 [*] Not vulnerable to CVE-2022-3786
root@b80e8606fba6:~/wcc-tests# wsh ../openssl/build-3.0.6/libcrypto.a < ./scripts/CVE-2022-3786.wsh
 -- relinked static library (ar archive) ../openssl/build-3.0.6/libcrypto.a into dynamic shared library /tmp/.wsh-61/libcrypto.a.so
 [*] testing for CVE-2022-3786
free(): invalid next size (fast)

[SIGABRT]       Read    0x3d        (Error code -6)
        0x7a20993c89fc    /lib/x86_64-linux-gnu/libc.so.6(pthread_kill+0x12c)
        0x7a2099374476    /lib/x86_64-linux-gnu/libc.so.6(raise+0x16)
        0x7a209935a7f3    /lib/x86_64-linux-gnu/libc.so.6(abort+0xd3)
        0x7a20993bb676    /lib/x86_64-linux-gnu/libc.so.6(+0x89676)
        0x7a20993d2cfc    /lib/x86_64-linux-gnu/libc.so.6(+0xa0cfc)
        0x7a20993d4a9d    /lib/x86_64-linux-gnu/libc.so.6(+0xa2a9d)
*** longjmp causes uninitialized stack frame ***: terminated
```

Partial Proofs of Vulnerabilities: CVE-2022-3602


```
print(" [*] Testing for CVE-2022-3602")

-- Input Arguments
teststring = "hello! -gr25faaaaaaaaaaaaa"
decoded = calloc(80)
decodedlen = calloc(1,2)
memset(decodedlen, 0x14,1)


res = ossl_punycode_decode(teststring, 26, decoded, decodedlen)

if res > 0  then  free(res) else  print(" [*] Not vulnerable to CVE-2022-3602") ;  exit(0) ; end
```

Partial Proofs of Vulnerabilities: CVE-2022-3786


```
print(" [*] testing for CVE-2022-3786")

-- Input arguments:
teststring = "a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--
.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--.xn--
.xn--.xn--.xn--.xn--.xn--"
out = calloc(1,16)
outlen = calloc(1,2)
memset(outlen,0x10,1)

-- Trigger the stack overflow:
res = ossl_a2ulabel(teststring, out, outlen)

if res == 0 then print(" [*] Not vulnerable to CVE-2022-3786") end
```

le cnam

Bonus

Binary Reflection

The Witchcraft Shell implements:

- An exposition of binary functions from the LUA interpreter.

Thanks to lua, we then implement:

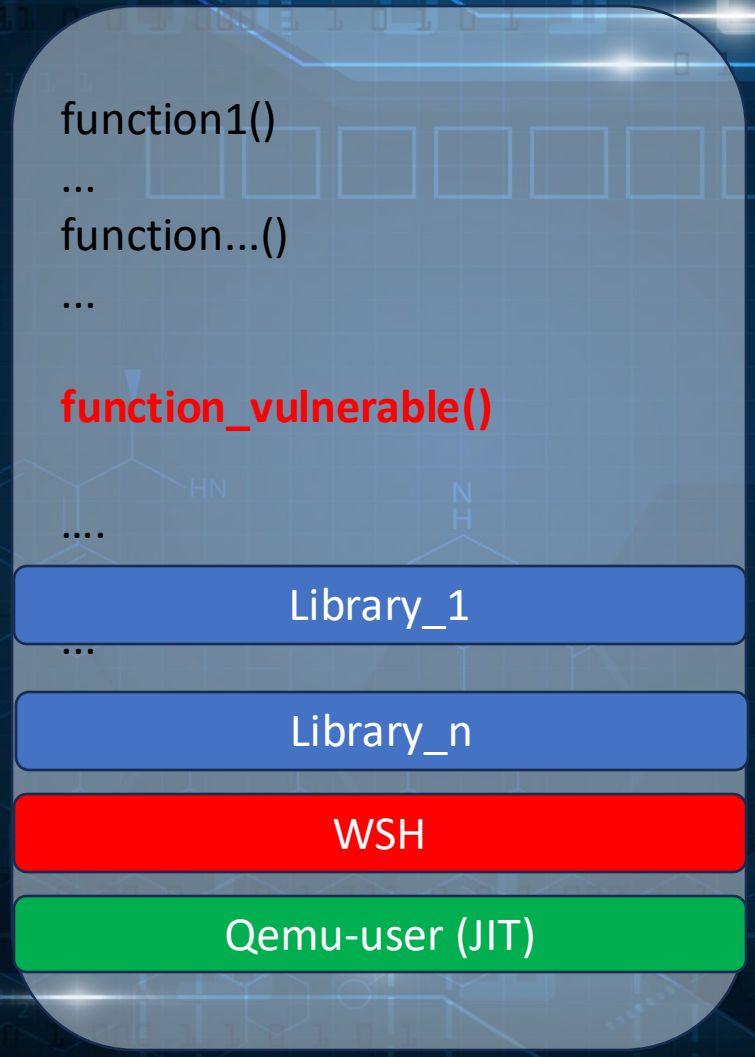
- Binary introspection
- Binary intercession

=> We implement binary reflection, **without source code**.

DOI 10.5281/zenodo.13902925

Cross Architecture Procedural Debugging (JIT)

By cross-compiling WSH to other architectures and running it within qemu-user, we benefit from JIT translation to the host architecture (eg: Aarch64 to x86_64) without a Virtual Machine, and in a single process (debugger + debuggee + JIT translator)



DOI 10.5281/zenodo.13902925

Conclusion & Future Work

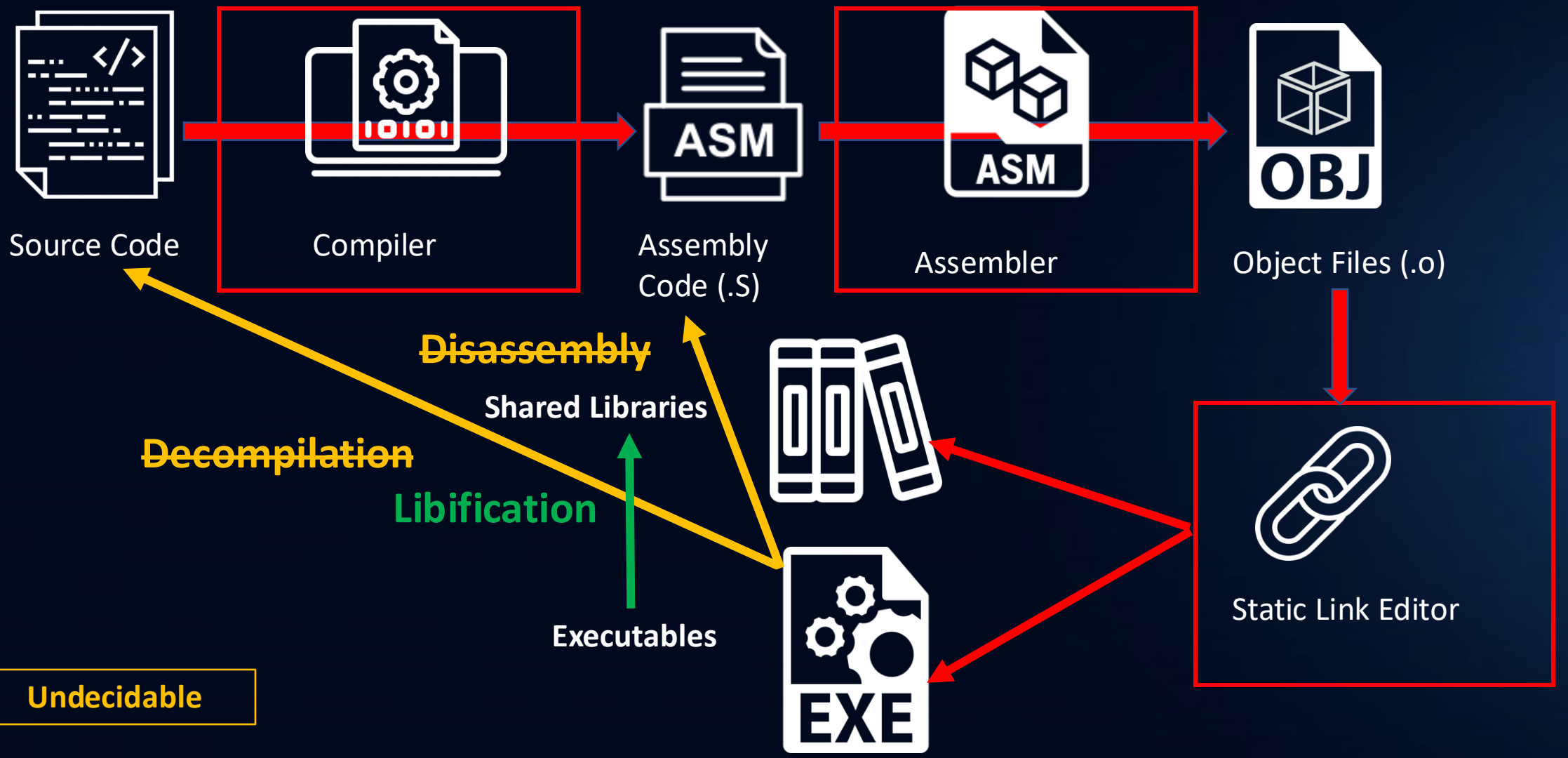
Partial Proofs of Vulnerabilities

- Turning Dynamically linked ELFs into shared libraries is practical (Witchcraft Linker)
- The Witchcraft Shell can perform procedural debugging.
- Selective execution of vulnerable functions is practical via procedural debugging.

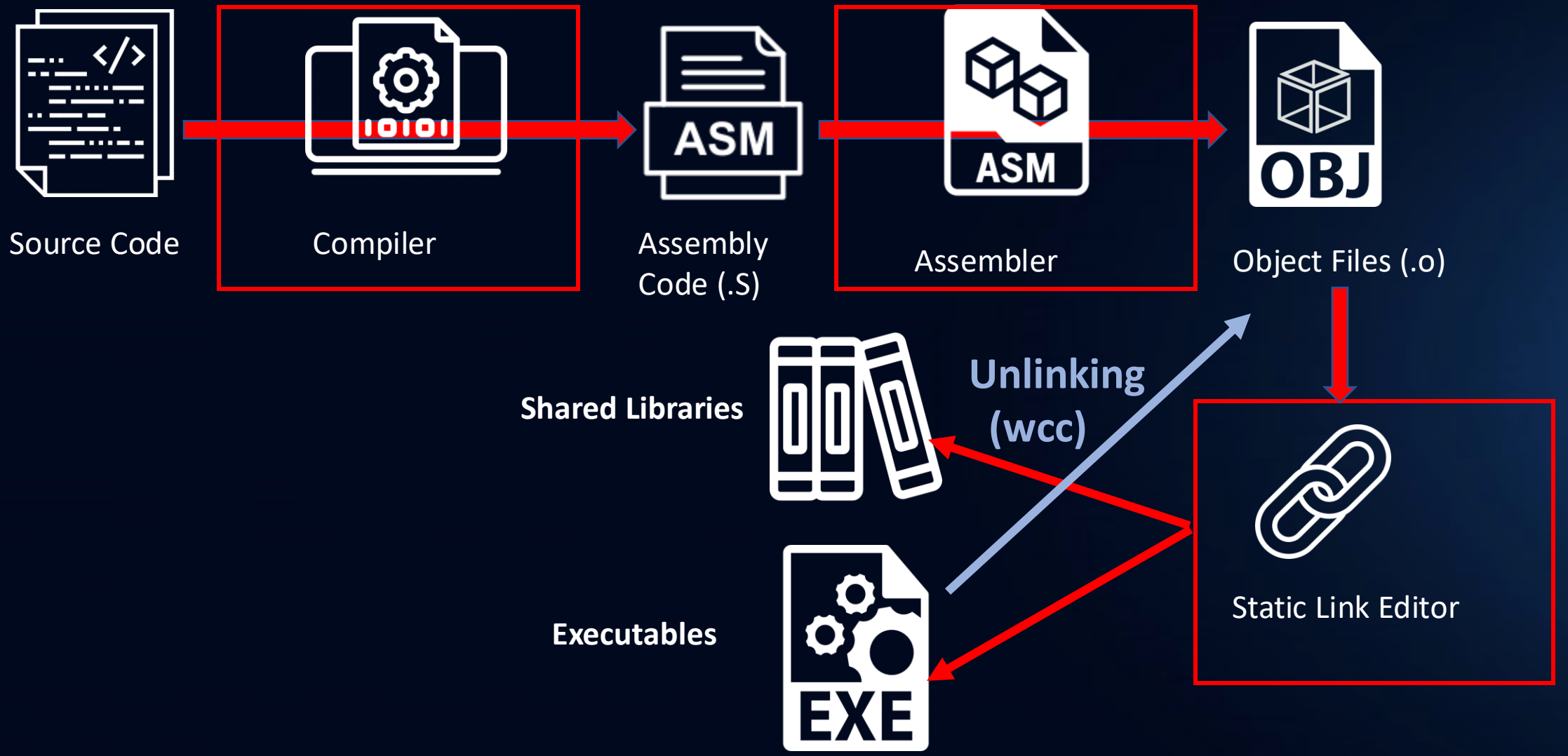
- Work to be done: Automatically write the scripts to trigger CVEs ?

DOI 10.5281/zenodo.13902925

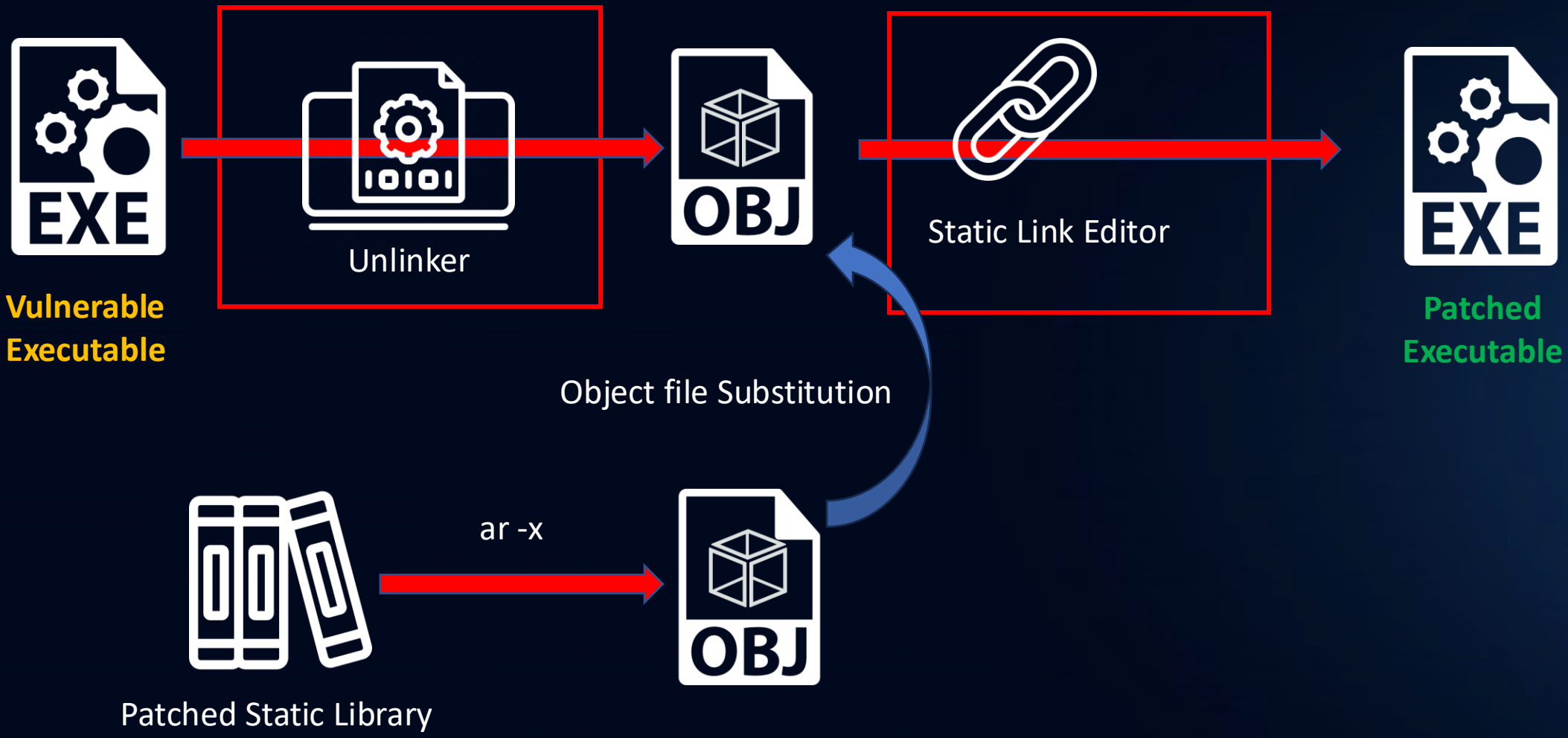
Witchcraft Linker : Libification



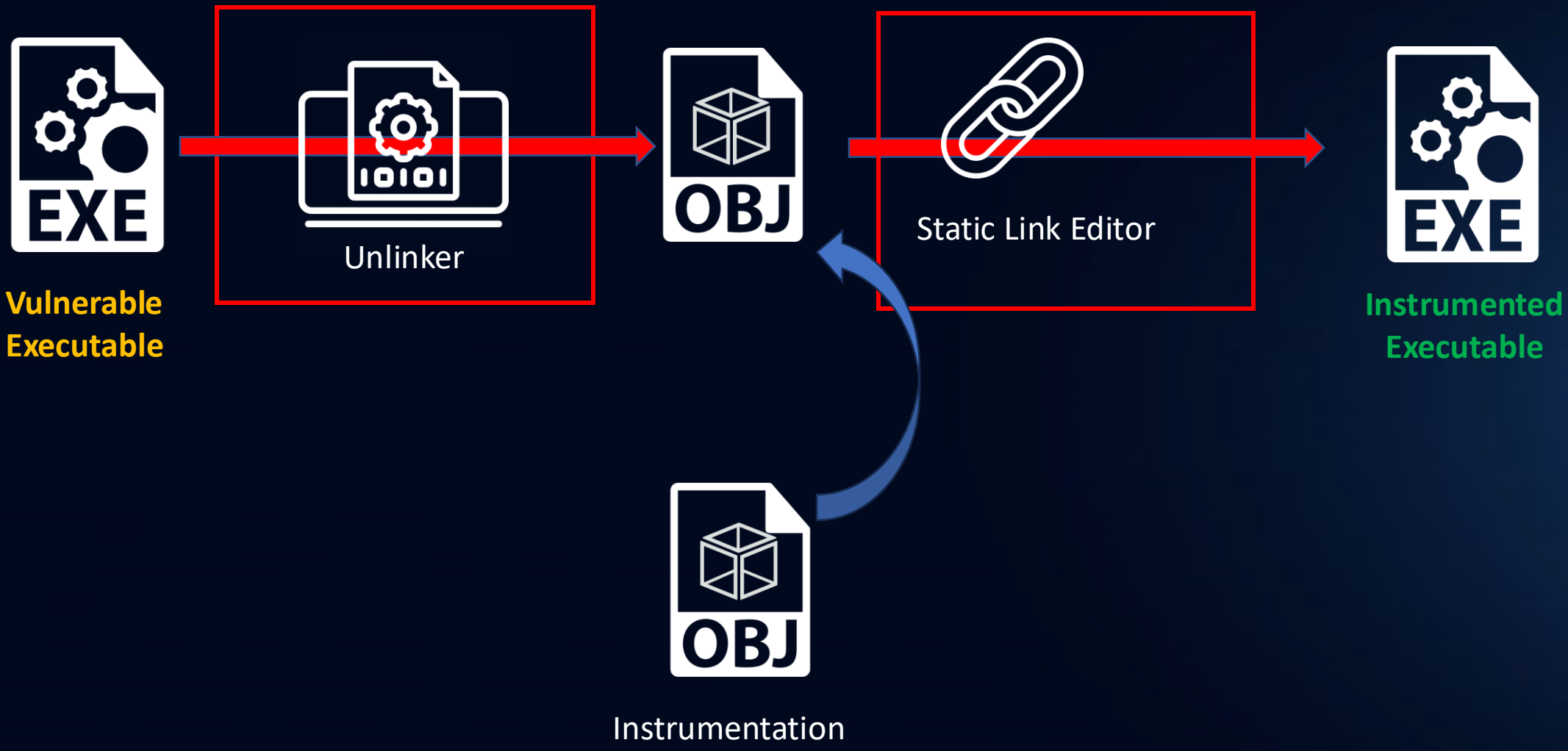
Witchcraft Compiler: Unlinking



Witchcraft Compiler: Unlinking + Relinking = Assured Patching



Witchcraft Compiler: Unlinking + Instrumentation = Fuzzing



MOABI

le cnam

THANK YOU FOR YOUR ATTENTION

PROGRAMME DE TRANSFERT AU CAMPUS CYBER

FRANCE 2030 PROGRAMME DE RECHERCHE CYBERSÉCURITÉ

Jonathan Brossard
jonathan.brossard@moabi.com

Systematic
Paris Region Deep Tech Ecosystem