Fuzzing 101

Andrea Straforini

Objective

Questa presentazione vuole fare un overview sul fuzzing, spiegare come utilizzarlo in maniera efficace e mostrare alcuni casi d’uso.

Vuole quindi portarvi da così:

A così:

Five Ws

Who should use it?

Team agili, application security testers e pentesters

What is it for?

Sono tools che permettono di scovare vulnerabilità che generano un **crash** come:

- Buffer/Heap overflow
- Use after free/return
- Uninitialized memory read
- Integer overflow

Five Ws

When should we use it?

Per software che gestiscono infrastrutture critiche; DOS evento grave; per scovare vulnerabilità che generano un crash

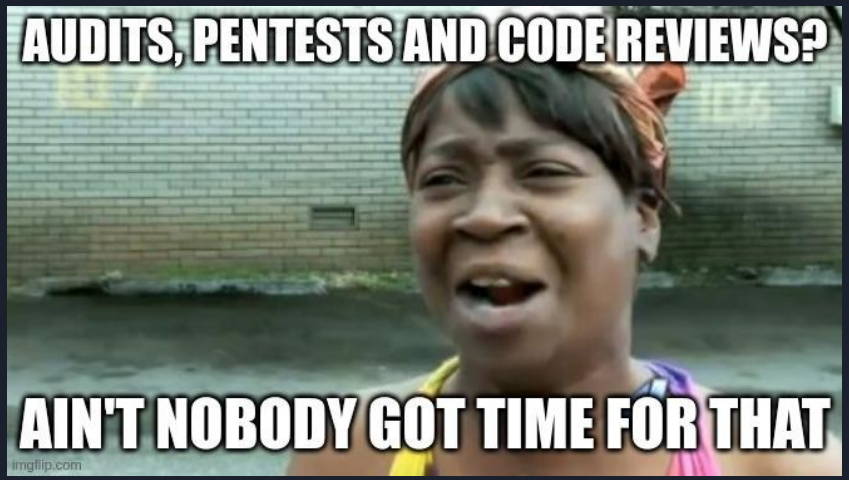
Where should we use it?

All'interno di pipeline CI/CD o comunque nelle fasi di test

Five Ws

Why should we use it?

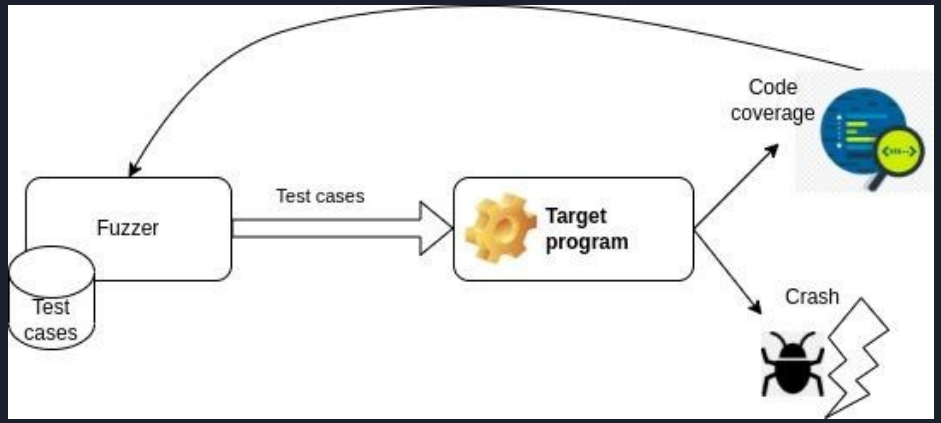
Garantisce buoni livelli di sicurezza a parità di costi e tempi



Introduction

What is fuzzing?

- Un approccio al software testing
- I casi di test sono generati **fuzzer**
- I casi di test sono iniettati nel **target program**
- Il target program viene monitorato per **rilevare i crashes**



Tassonomia

Esistono 3 tipi di fuzzers:

1. White box
2. Black box
3. Grey box

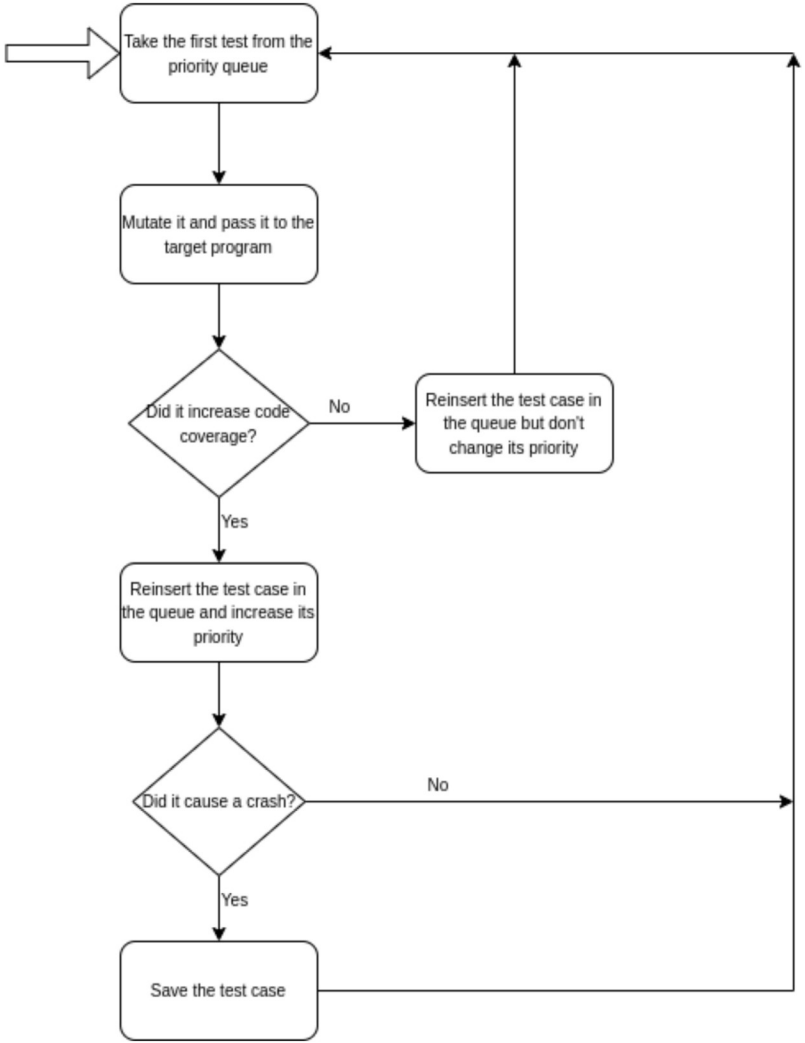
I fuzzer white-box utilizzano tecniche come la *symbolic execution* e il *constraint solving* ma purtroppo soffrono del *Path explosion problem* (Forti limitazioni d'uso)

I fuzzer black-box possono essere sempre utilizzati ma non analizzando lo stato interno del programma target sono spesso poco efficaci.

I fuzzer grey-box sono quelli più efficaci e funzionano analizzando lo stato interno del programma target durante l'esecuzione dei casi di test

Gray-box Internals

The internal operation of the fuzzer can be simplified as follows:



Tassonomia

I metodi per creare casi di test sono principalmente due:

1. Mutation based
2. Generation based

mutation based

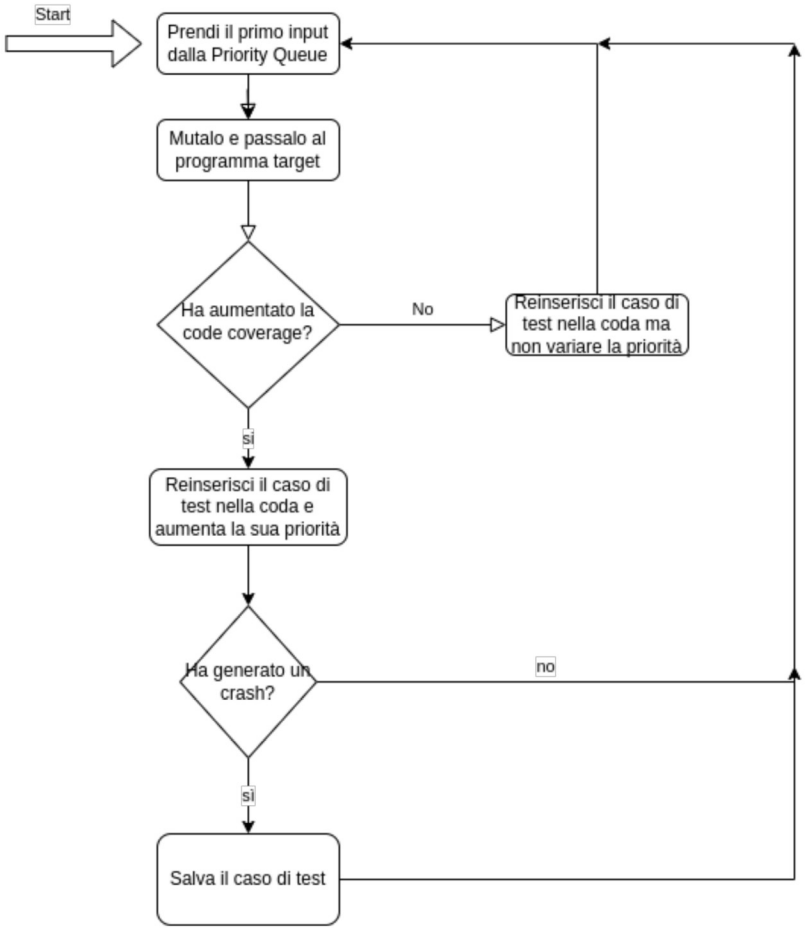
- variazione randomica dell’input.

generation-based invece

- muta l’input ma ne mantiene la struttura grazie ad una specifica fornita dall’utente.

Grey-box fuzzer

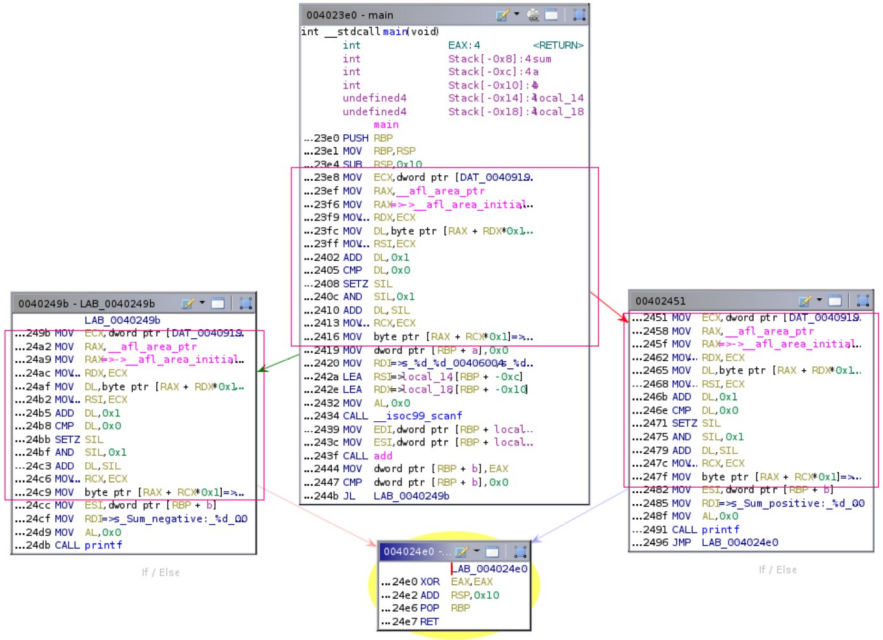
I grey-box funzionano grazie alla variazione della code coverage che viene calcolata grazie all'instrumentazione.



Instrumentazione

Quando il codice sorgente è presente la inseriamo durante la fase di compilazione.

L'instrumentazione nel fuzzing ci permette di sapere se l'esecuzione ha raggiunto un particolare basic-block (e quindi calcolare la coverage)



Nell'immagine sopra possiamo vedere tre basic-block instrumentati da AFL
I due basic block a SX e DX corrispondono ai due rami di un if-else

Road-Blocks

Ne esistono principalmente di 2 tipi:

- Magic number road-block
- Checksum road-block

Magic number road-block:

```
if input == "MAGIC":
        // buggy code
```

Checksum road-block:

```
if input[-1] == sum(input[:-1]):
        // buggy code
```

Road-Blocks

Questi road-block sono problematici nel black-box fuzzing e anche nel gray-box se utilizzato in modo *naive*.
Impediscono ai fuzzers di raggiungere parti di codice che si trovano in “profondità”

Abbiamo bisogno quindi di tecniche (per il gray-box fuzzing) aggiuntive che che ci permettano di sorpassarli

In breve:

Sub-instruction profiling

- Efficace contro i magic number road-block
- Non efficace per i checksum road-block

Implementata in CompareCoverage, laf-intel e integrata ad esempio da AFL++ e Honggfuzz

Original code:

```
if input == 0xDEADBEEF:
     // buggy code

// secure code
```

Sub-instruction profiling code:

```
if(input>>24 == 0xDE){
     if((input & 0xFF0000) >> 16 == 0xAD){
       if((input & 0xFF00) >> 8 == 0xBE){
          if((input & 0xFF) == 0xFF){
                    // buggy code
                    goto end;
          }
       }
    }
}
end: // secure code
```

Taint Analysis

La dynamic taint analysis traccia il flusso delle informazioni nel programma andando ad eseguire ed osservando quali computazioni sono affette dalle “taint sources”
Traccia quindi gli input non sanitizzati che raggiungono parti interne del programma e che possono generare vulnerabilità come SQL injection.

Alcuni fuzzers: Angora, VUzzer, TaintScope

Symbolic execution

- Tecnica di analisi statica
- Esplora i possibili path di esecuzione rappresentando le variabili e il contesto di esecuzioni con simboli e utilizzando constraint solvers
- Soffre del path explosion problem

Alcuni fuzzers: T-Fuzz

Concolic execution

Cerca di risolvere il path explosion problem andando ad utilizzare l'esecuzione simbolica insieme all'esecuzione concreta.
Esempi:

- Dynamic symbolic execution
- Selective symbolic execution

Nella dynamic symbolic execution la symbolic execution è guidata da una esecuzione concreta
La selective esegue l'SMT solver in specifiche porzioni del programma

Alcuni fuzzers: Driller, Qsym, TaintScope

Input to state correspondence

Permette spesso di sostituire il taint tracking e la symbolic execution.
E’ una versione lightweight di un taint-tracker.

Colorizza l’input con bytes randomici e analizza a run time gli argomenti utilizzati nelle istruzioni di compare.(senza fare il tracking)

Questi permettono di riconoscere quali parti dell’input sono interessanti e quindi da mutare

Alcuni fuzzers: AFL++

Fuzzing modes

- ***Simple mode***
- Fork-server mode
- Persistent mode

Simple mode

- Il processo da fuzzare viene creato da zero per ogni caso di test
- Rallentamenti causati da fork(),execve() e il linking process
- Garantisce uno stato interno corretto

Fuzzing modes

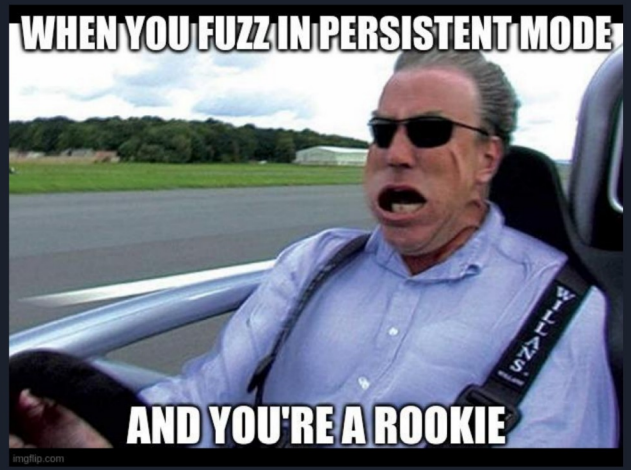
- Simple mode
- ***Fork-server mode***
- Persistent mode

Fork-server mode

- Il processo viene eseguito normalmente fino alla lettura deìll'input (esclusa)
- Il processo viene stoppato ed è così possibile generare nuovi processi già inizializzati forkando il processo.
- Garantisce uno stato interno corretto ed è più veloce della simple mode.

Fuzzing modes

- Simple mode
- Fork-server mode
- ***Persistent mode***



Persistent mode

- Numerosi casi di test vengono applicati ad un unico
- Necessario resettare lo stato interno del programma
- Soffre di problemi di stabilità
- Estremamente veloce

Instrumentation Optimization

AFL e altri fuzzers utilizzano la branch coverage che viene calcolata:

Listing 3.3: AFL instrumentation code [Zal]

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

- La shared memory region è 64KB ⟶ max 65.5K edges senza conflitti
- E.g. il 75% degli edges collide nell'instrumentazione di AFL su LibTorrent (206K edges)

Applications	Size	#ins.	#BB	#edges	collision
LAVA(base64)	193KB	5570	822	1308	0.8%
LAVA(uniq)	208KB	5285	890	1407	0.92%
LAVA(md5sum)	234KB	7397	1013	1560	1.02%
LAVA(who)	1.52MB	84648	1831	3332	1.8%
catdoc	202KB	6448	841	1322	1.29%
libtasn1	540KB	12511	2163	3820	2.72%
cflow	688KB	24655	4286	7001	5.2%
libncurses	338KB	21486	4646	7883	5.57%
libtiff+tiffset	1.77MB	61119	8974	14826	10.4%
libtiff+tiff2ps	1.97MB	65932	9632	15927	10.84%
libtiff+tiff2pdf	2.1MB	71530	10507	17603	12.31%
libming+listswf	4.04MB	87148	11456	19154	13.61%
libdwarf	3MB	73921	11698	20260	13.7%
tcpdump	4.62MB	127082	18781	32656	21.2%
nm	8.72MB	218326	31611	53652	36.06%
bison	3.28Mb	219268	42856	55658	32.8%
nasm	4.4MB	226665	41691	57411	33.38%
libpspp	5MB	259501	41323	71335	38.9%
objdump	11.88MB	305620	43935	74313	40.17%
clamav	11.35MB	347156	46140	81069	42.48%
exiv2+libexiv2	4.75MB	283284	59650	91287	45.87%
libsass+sassc	32.8MB	593570	68538	106738	50.7%
vim	14.7MB	478402	83877	153689	61.4%
libav	76.7MB	1776730	158009	255212	74.85%
libtorrent	97.5MB	1228513	164325	260485	75.29%

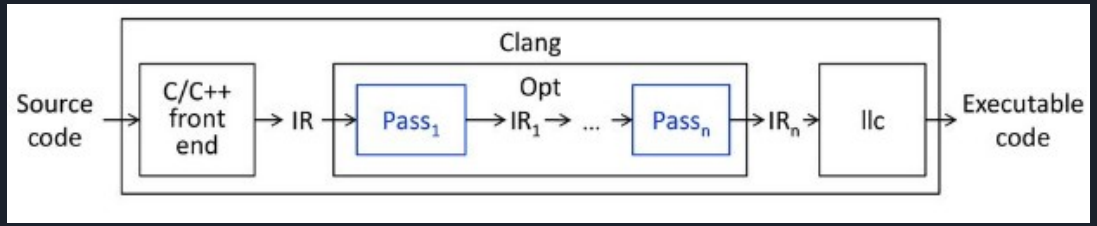
Instrumentation Optimization

AFL++ LTO:

- inseririsce l'instrumentazione a link time manipolando l'*Intermidiate representation*
- Tramite un *LLVM pass.*

Ogni edge:

- è associato ad uno specifico indirizzo nella shared memory
- garanzie di non collisione
- shared memory con una dimensione calcolata automaticamente



Instrumentation Optimization

- permette di scovare vulnerabilità che potrebbero non generare un crash.
- mantengono la campagna di fuzzing deterministica.

Se un crash infatti non avviene sempre con lo stesso caso di test, la coverage non viene calcolata correttamente e il processo diventa non deterministico

Alcuni dei sanitizers più importanti sono:

- AddressSanitizer (ASan)
- MemorySanitizer (MSan)
- UndefinedBehaviourSanitizer

Apache HTTP fuzzing

- Fuzzing con multiple istanze AFL++
- Persistent mode
- Per migliorare la stabilità abbiamo escluso delle librerie esterne
- Link time optimization
- Utilizzo dei sanitizers, laf-intel, input to state correspondence e altre tecniche
- 8 days fuzzing campaign
- Creazione di un harness per fuzzare via socket invece che stdin

Apache HTTP Fuzzing

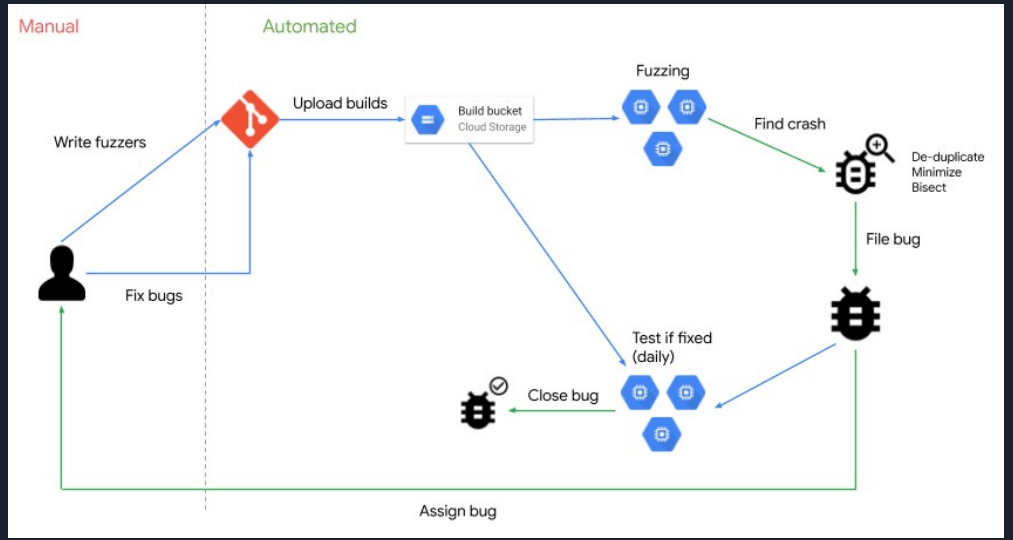
Nessun crash trovato perchè?

- Apache è fuzzato da OSSFuzz (cluster fuzz di Google)
- Molte attenzioni da parte della community Csec
- Mutation based approach invece che generation based

Cluster fuzz (Open source)

Creata da Google per integrare facilmente fuzzers con progetti software:

- Highly scalable. Can run on any size cluster (e.g. OSS-Fuzz instance runs on 100,000 VMs).
- Fully automatic bug filing, triage and closing for various issue trackers (e.g. Monorail, Jira).
- Supports multiple coverage guided fuzzing engines (libFuzzer, AFL++ and Honggfuzz) for optimal results
- Support for blackbox fuzzing.
- Easy to use web interface for management and viewing crashes



First approach

- Fork server mode
- Fuzz input passato usando AFL_PRELOAD (LD_PRELOAD)
- Gray box mutation based

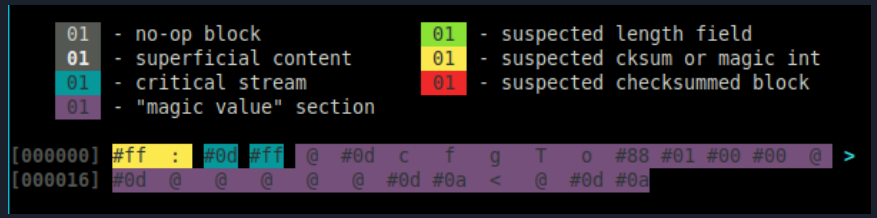
Lo shared object modifica il comportamento delle funzioni per i socket:

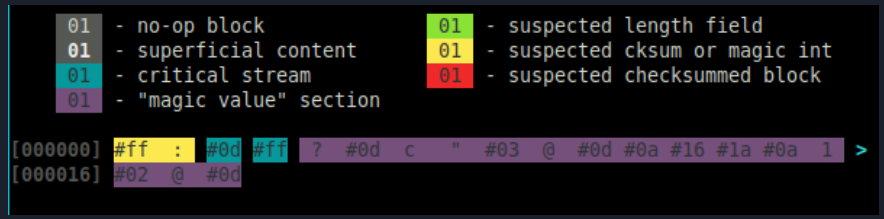
- close
- socket
- bind
- listen
- recv
- send
- ...

First approach

Prima vulnerabilità:

- Configuration Frame 2
- Non-sanitizzazione del field `FRAMESIZE`
- Undefined memory read del field `CRC`





First approach

Second vulnerability:

- Configuration Frame 2
- Non-sanitization of `NUM_PMU`
- Undefined memory read

No	Field	Size (bytes)	Short description
1	SYNC	2	Sync byte followed by frame type and version number.
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	Stream source ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2.
5	FRACSEC	4	Fraction of Second and Message Time Quality, defined in 6.2.
6	TIME_BASE	4	Resolution of FRACSEC time stamp.
7	NUM_PMU	2	The number of PMUs included in the data frame.
8	STN	16	Station Name—16 bytes in ASCII format.
9	IDCODE	2	Data source ID number identifies source of each data block.
10	FORMAT	2	Data format within the data frame.
11	PHNMR	2	Number of phasors—2-byte integer (0 to 32 767).
12	ANNMR	2	Number of analog values—2-byte integer.
13	DGNMR	2	Number of digital status words—2-byte integer.
14	CHNAM	16 × (PHNMR + ANNMR + 16 × DGNMR)	Phasor and channel names—16 bytes for each phasor, analog, and each digital channel (16 channels in each digital word) in ASCII format in the same order as they are transmitted. For digital channels, the channel name order will be from the least significant to the most significant. (The first name is for bit 0 of the first 16-bit status word, the second is for bit 1, etc., up to bit 15. If there is more than 1 digital status, the next name will apply to bit 0 of the second word and so on.)
15	PHUNIT	4 × PHNMR	Conversion factor for phasor channels.
16	ANUNIT	4 × ANNMR	Conversion factor for analog channels.
17	DIGUNIT	4 × DGNMR	Mask words for digital status words.
18	FNOM	2	Nominal line frequency code and flags.
19	*CFGCNT*	2	Configuration change count.
	Repeat 8–19		Fields 8—19, repeated for as many PMUs as in field 7 (NUM_PMU).
20+	DATA_RATE	2	Rate of data transmissions.
21+	CHK	2	CRC-CCITT.

First approach

AFL++ è stato in grado di generare un input in grado di passare il CRC check calcolato da un’area di memoria non inizializzata

- Fortuna
- Glu undefined behaviours non aiutanto il fuzzing a mantenere la code coverage stabile.

La campagna di fuzzing è continuata con i sanitizers attivati

Second approach

- Approccio Black box
- Radamsa
- Mutazione dei messaggi inviati da un PMU simulator

Cinque vulnerabilità trovate, sia nel CFG Frame che nel Data Frame

Le prime 2 già trovate con AFL++

Second approach

- Data Frame non valido -> free su puntatori non inizializzati
- Out of bound read: il campo `NUM_PMU` nel Configuration Frame non corrisponde con la dimensione reale del DataFrame
- Non sanitizzazione del field `FRAMESIZE` (Data Frame)

No.	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number.
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	Stream source ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2, for all measurements in frame.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2, for all measurements in frame.
6	STAT	2	Bit-mapped flags.
7	PHASORS	4 × PHNMR or 8 × PHNMR	Phasor estimates. May be single phase or 3-phase positive, negative, or zero sequence. Four or 8 bytes each depending on the fixed 16-bit or floating-point format used, as indicated by the FORMAT field in the configuration frame. The number of values is determined by the PHNMR field in configuration 1, 2, and 3 frames.
8	FREQ	2 / 4	Frequency (fixed or floating point).
9	DFREQ	2 / 4	ROCOF (fixed or floating point).
10	ANALOG	2 × ANNMR or 4 × ANNMR	Analog data, 2 or 4 bytes per value depending on fixed or floating-point format used, as indicated by the FORMAT field in configuration 1, 2, and 3 frames. The number of values is determined by the ANNMR field in configuration 1, 2, and 3 frames.
11	DIGITAL	2 × DGNMR	Digital data, usually representing 16 digital status points (channels). The number of values is determined by the DGNMR field in configuration 1, 2, and 3 frames.
	Repeat 6–11		Fields 6–11 are repeated for as many PMUs as in NUM_PMU field in configuration frame.
12+	CHK	2	CRC-CCITT

Third approach

- Black box
- Approccio generativo
- Libreria Scapy
- Creazione parziale della specifica per il C37.118

```
class ConfigurationFrame2(Packet):
  name = "ConfigurationFrame2"
  fields_desc = [
      XByteField("syncHead", 0xaa),
      XBitField("syncReserved", 0b0, 1),
      BitEnumField("syncFrameType", 0b011, 3,
                   CommonFieldsConstants.SyncFieldConstants.FRAME_TYPES),
      BitEnumField("syncVersion", 0b0001, 4,
                   CommonFieldsConstants.SyncFieldConstants.VERSIONS),
      ShortField("framesize", None),
      ShortField("idcode", 7734),
      IntField("soc", 1149577200),
      XBitField("fracsecReserved", 0b0, 1),
      BitEnumField("fracsecLeapSecDirection", 0b1, 1,
                   CommonFieldsConstants.FracSecFieldConstants.LEAP_SEC_DIRECTION),
      BitEnumField("fracsecLeapSecOccured", 0b0, 1,
                   CommonFieldsConstants.FracSecFieldConstants.LEAP_SEC_OCCURED),
      BitEnumField("fracsecLeapSecPending", 0b1, 1,
                   CommonFieldsConstants.FracSecFieldConstants.LEAP_SEC_PENDING),
      BitEnumField("fracsecTimeQuality", 0b0110, 4,
                   CommonFieldsConstants.FracSecFieldConstants.TQ_CODE),
      ThreeBytesField("fracsecValue", 0x071098),
      ByteField("timeBaseFlags", 0x00),
      ThreeBytesField("timeBaseValue", 0x0f4240),
      ShortField("numPmu", 1),
      PacketListField("configurationFrame2ListEntries",
                      [ConfigurationFrame2Entry(), ],
                      ConfigurationFrame2Entry),
      SignedShortField("dataRate", 30),
      ShortField("chk", None)
  ]

  def post_build(self, pkt: bytes, pay: bytes) -> bytes:
      if self.framesize is None:
          pkt = pkt[: 2] + struct.pack("!H", len(pkt)) + pkt[4:]
      if self.chk is None:
          pkt = pkt[: -2] + struct.pack("!H", crc16xmodem(pkt[:-2], 0xffff))
      return pkt + pay
```

The end