Heisenbugs & Nasal Demons

How UB broke Chrome and saved Tor

Donato Barone

CyberChallenge Workshop 2026

SPEAKER

Donato Barone (@fwame)

Matematica, cybersecurity, CTF

01 1° anno Matematica · Sapienza Università di Roma

02 Tutor CyberChallenge.IT 2026 · UniRM1

03 ECSC 2025 · TeamItaly

04 CTF player · TRX & mhackeroni

CyberChallenge.IT 2024 · TeamItaly 2025

L’assunzione nascosta

“Ciò che scrivo determina interamente ciò che esegue la macchina.”

L’assunzione nascosta

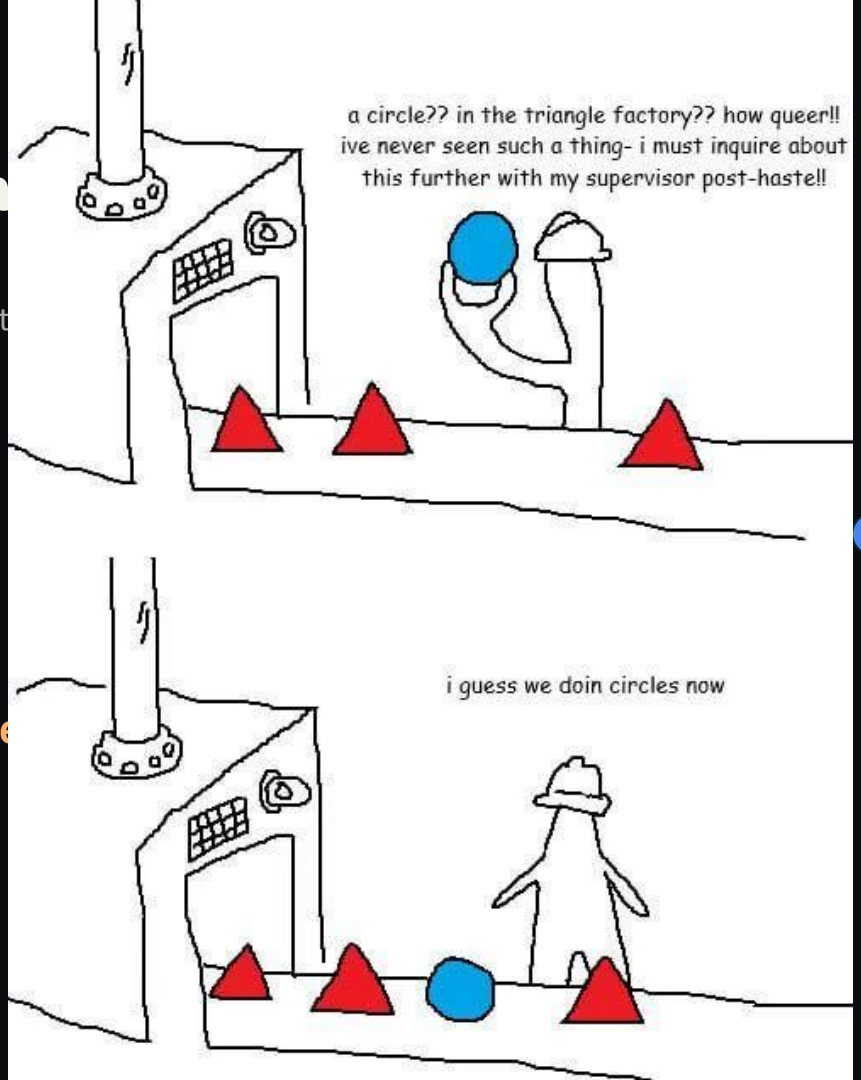
“Ciò che scrivo determina interamente ciò che esegue la macchina.”

È vero solo finché restiamo dentro il contratto del linguaggio.

L’assunzione n

“Ciò che scrivo determina int

È vero solo finché re



Undefined Behaviour = nessun vincolo sul risultato

Non significa “crash”.

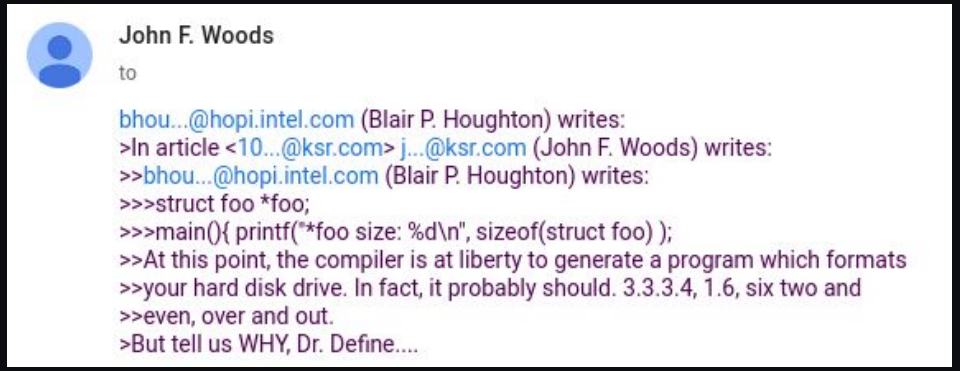
Significa che lo standard non impone più alcun comportamento.

Aaaaaaaaaaaaaalso il compilatore può assumere che quel percorso non accada.

There are levels to this game

DEFINED	un risultato prescritto
IMPLEMENTATION-DEFINED	scelta documentata dal compilatore
UNSPECIFIED	una tra più scelte valide
UNDEFINED	**nessun requisito**

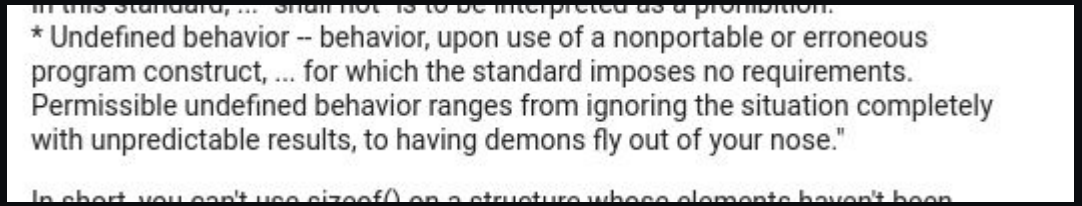
UB Classica



John F. Woods

to

bhou...@hopi.intel.com (Blair P. Houghton) writes:
>In article <10...@ksr.com> j...@ksr.com (John F. Woods) writes:
>>bhou...@hopi.intel.com (Blair P. Houghton) writes:
>>>struct foo *foo;
>>>main(){ printf("*foo size: %d\n", sizeof(struct foo));
>>At this point, the compiler is at liberty to generate a program which formats
>>your hard disk drive. In fact, it probably should. 3.3.3.4, 1.6, six two and
>>even, over and out.
>But tell us WHY, Dr. Define....



* Undefined behavior -- behavior, upon use of a nonportable or erroneous program construct, ... for which the standard imposes no requirements. Permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to having demons fly out of your nose."

Guess the UB Time!

Chi ha orchestrato tutto questo?

- Salvatore Giuseppe Abello, 2016

Signed overflow

```
int f(int x) {
  return x + 1 > x;
}
```

Signed overflow

```
int f(int x) {
  return x + 1 > x;
}
```

Per ogni esecuzione definita:

f(x) = 1

INT_MAX non è “il caso speciale”.
È fuori dal contratto.

Magic Loops

```
void wait(int ready) {
  while (!ready) {
    // nessun side effect
  }
  enter_critical_section();
}
```

Magic Loops

```
void wait(int ready) {
  while (!ready) {
    // nessun side effect
  }
  enter_critical_section();
}
```

Se il loop non può produrre effetti osservabili…

il compilatore può assumerne la terminazione.

Unintentional assumptions

```
void meow(int input) {

  int x = input + 2147483580;

  If (input > 0x420)
    panic();

  miagola(x);
}
```

Unintentional assumptions

```
void meow(int input) {

  int x = input + 2147483580;

  If (input > 0x420)
    panic();

  miagola(x);
}
```

Se input > 67, calcolare x produce UB.

Dunque input < 67

Dunque input < 0x420

Dunque il check è superfluo! 🧠

L’UB non accade “dopo”: cambia ciò che può accadere prima

programma scritto ≠ programma macchina ≠ intenzione

La sicurezza deve sopravvivere alle trasformazioni lecite.
- ChatGPT 5.6 Sol Ultra /fast, 2026

Case study: V8 Sandbox

Brrrrrrrrrrrrrrrrr VROOM VROOM

Case study: V8 Sandbox

01 Legge una entry dalla External Pointer Table

02 Estrae il tag effettivo

03 Verifica che il tag sia nel range atteso

04 Restituisce il payload oppure 0

Case study: V8 Sandbox

```
if (tag_range.Contains(actual_tag)) {
  return entry & kExternalPointerPayloadMask;
} else {
  return 0;
}
```

tag valido

→ **payload**

tag errato

→ **nullptr**

Case st

```
if (tag_range
  return entry
} else {
  return 0;
}
```

ad

r

Case study: V8 Sandbox

```
auto a = *ReadExternalPointerField(...);
auto b = *ReadExternalPointerField(...);
auto c = *ReadExternalPointerField(...);
```

***nullptr**

è UB

Compiler reasoning

01 Il chiamante dereferenzia (quasi) sempre il risultato

02 Un’esecuzione definita non dereferenzia nullptr

03 Quindi ReadExternalPointerField non restituisce 0

04 Il ramo “tag errato → 0” è irraggiungibile

Compiler

01 Il chiamante

02 Un’esecuzio

03 Quindi Reac

04 Il ramo “tag

Case study: V8 Sandbox (Fix)

```
volatile Address safe_entry;

if (tag_range.Contains(actual_tag))
  safe_entry = entry & kExternalPointerPayloadMask;
else
  safe_entry = 0;

return safe_entry;
```

volatile

impedisce la DCE di questa logica

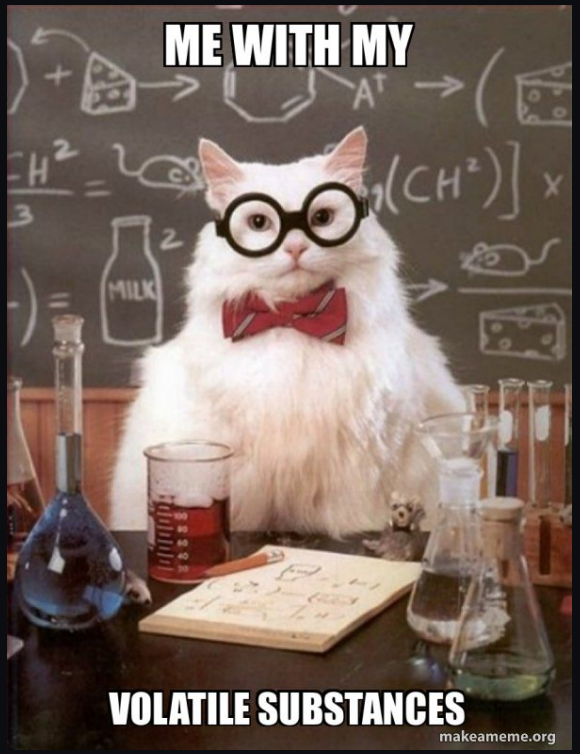
Fix un po’ cosi, ma così è V8.

What even is `````volatile`````?

01 Intended per contesti di concurrent access

02 Se non sei l’unico con accesso a una variabile, non modificarla!

03 Utilizzato anche per prevenire ottimizzazioni moleste



Case study: Tor

Non si può vincere sempre
- Sapienza, 2026

Case study: Tor

```
STATIC void
channel_tls_process_certs_cell(var_cell_t *cell, channel_tls_t *chan)
{

  ...

  started_here = chan->conn->handshake_state->started_here;

  if (chan->conn->base_.state != OR_CONN_STATE_OR_HANDSHAKING_V3)
    ERR("We're not doing a v3 handshake!");

  ...

}
```

Case study: Tor

```
STATIC void
channel_tls_process_certs_cell(var_cell_t *cell, channel_tls_t *chan)
{

  ...

  started_here = chan->conn->handshake_state->started_here;

  if (chan->conn->base_.state != OR_CONN_STATE_OR_HANDSHAKING_V3)
    ERR("We're not doing a v3 handshake!");

  ...

}
```

Dopo l'autenticazione:

handshake_state = NULL

La prima istruzione
è una nullptr dereference.

Case study:

```
STATIC void
channel_tls_process_certs_c
{

  ...

  started_here = chan->conn-

  if (chan->conn->base_.state
    ERR("We're not doing a v3

  ...

}
```

utenticazione:

shake_state = NULL

na istruzione
nullptr dereference.

Case study: Tor

```
started_here = chan->conn->handshake_state->started_here;

if (chan->conn->base_.state != OR_CONN_STATE_OR_HANDSHAKING_V3)
  ERR("We're not doing a v3 handshake!");
if (chan->conn->link_proto < 3)
  ERR("We're not using link protocol >= 3");

if (chan->conn->handshake_state->received_certs_cell)
  ERR("We already got one");
if (chan->conn->handshake_state->authenticated) {
  /* Should be unreachable, but let's make sure. */
  ERR("We're already authenticated!");
}

started_here = chan->conn->handshake_state->started_here;
```

LOAD

↓

spostato

più tardi

Su Linux x64 il crash sembra sparire

sorgente → code motion → ERR prima del load

La build più testata nasconde accidentalmente il bug.

Ma l’ottimizzazione non è una patch

LINUX x64	safe (in quelle versioni)
WINDOWS	**vulnerabile**
ARM	
BUILD ESOTICHE	**/sede**

Stesso sorgente ≠ stessa manifestazione.

Ma l’ottimizzazione non è una patch

```
o Major bugfixes (relay):
  - Fix null pointer dereference when receiving a CERT cell out of
    order. TROVE-2026-006. Found by Fwame. Fixes bug 41240; bugfix
    on 0.2.4.4-alpha.
```

LINUX x64	safe (in c
WINDOWS	**vulnerabile**
ARM	
BUILD ESOTICHE	**/sede**

Stesso sorgente ≠ stessa manifestazione.

Ma l’ottimizzazione non è una patch

```
o Major bugfixes (relay):
  - Fix null pointer dereference when receiving a CERT cell out of
    order. TROVE-2026-006. Found by Fwame. Fixes bug 41240; bugfix
    on 0.2.4.4-alpha.
```

LINUX x64

safe (in c

CVE-2026-44602 Detail

Description

Tor before 0.4.9.7 has a NULL pointer dereference when a CERT cell is received out of order, aka TROVE-2026-006.

Metrics

CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

NIST: NVD **Base Score:** 7.5 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

Come si evita?

La comunicazione è la parte più importante di una relazione

- GCC, 1998

Compilatore, sanitizer, IR: servono tutti e tre

WARNINGS

-Wall -Wextra
-Wconversion

RUNTIME

UBSan · ASan
test e fuzzing

OPTIMIZER

-O2 / -O3
IR · assembly

La sicurezza vive nel programma definito

1. L’UB è assenza di garanzie, non un risultato particolare.
2. L’optimizer può propagare quella assenza nel tempo.
3. Un check esiste solo se la semantica gli permette di esistere.

DOMANDE? :3