Static Analysis and Runtime-Assertion Checking: Contribution to Security Counter-Measures

Dillon Pariente (Dassault Aviation)
Julien Signoles (CEA-List)

SSTIC'17, 7-9 Juin 2017, Rennes

Verification **E**ngineering of **S**afety and **SE**curity critical **D**ynamic **I**ndustrial **A**pplications

"This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731453."

Objectif de la présentation

Comment l'analyse statique de code (i.e. sans exécution)
peut aider à rendre plus robuste un code source C (et C++) ...

Méthode CURSOR :
- analyser du code source
- détecter des vulnérabilités
- générer des contextes d'appel de contre-mesure

=> génération ~automatique de *PROGRAMMATION DÉFENSIVE*

Démos "Live"

Plan

- Analyse Statique et Méthodes formelles
- Frama-C : Plate-forme d'Analyse Statique
  focus sur les plug-ins Value et E-ACSL
- Méthode CURSOR
- Démos
  Stack-Based Buffer Overflow
  Heartbleed dans OpenSSL
- Conclusion & Perspectives
- Questions

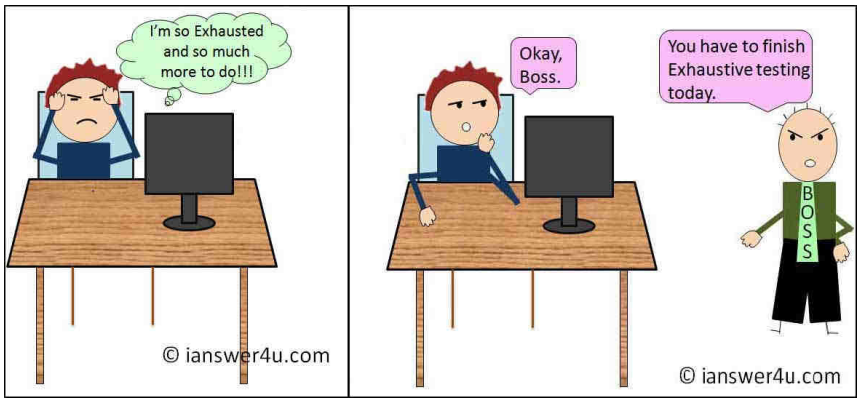
Analyse Statique et Méthodes formelles

- les **méthodes formelles** reposent sur des fondations théoriques pour vérifier des logiciels et des systèmes en offrant des garanties fortes

- l'**analyse statique** par interprétation abstraite est une méthode formelle approximant les comportements possibles du programme par examen du source pour en déduire des propriétés sans exécuter le programme

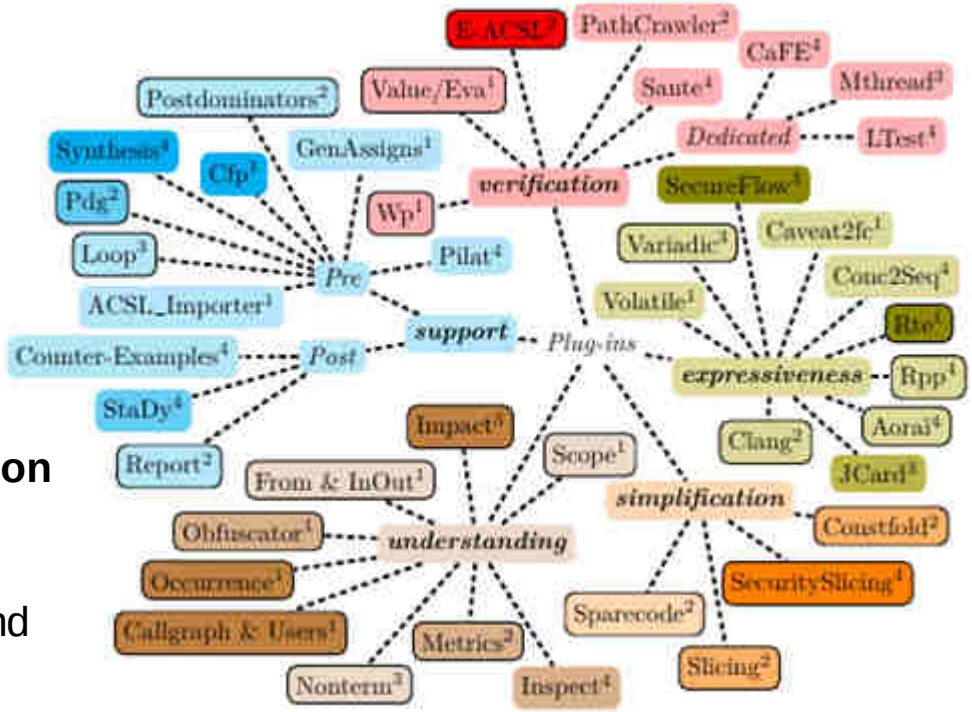
- **avantages** :
  - **exhaustivité** des vérifications pour des classes de propriétés données
  - **déjà utilisées avec succès** en sûreté (avionique, nucléaire), mais aussi en sécurité (hyperviseurs sécurisés, polarssl)

- **inconvénients**:
  - **complexité**
  - **coûts**
  - pas de large dissemination aujourd'hui



Frama-C

- **plateforme d'analyses de code C**
- *open source*
- **combinaisons** d'analyses facilitées



E-ACSL

- modifie le source original
- **vérifie des propriétés pendant l'exécution**
- notamment comportements indéfinis
  - accès mémoire, débordements
- inclut un **débogueur mémoire** à la Valgrind

Value/Eva

- sur-approximation des valeurs des variables en chaque point de programme
- **analyse statique** par interprétation abstraite
- **« alarme » sur les comportements indéfinis possibles**
- satisfait le critère d'Ockam du NIST à 100% (2014)
- utilisations industrielles

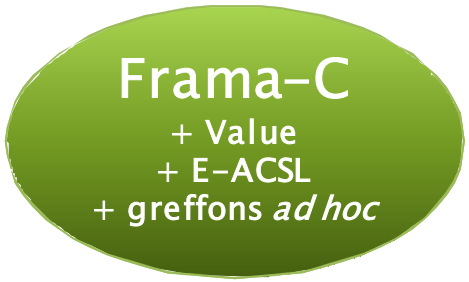
NIST
National Institute of Standards and Technology
U.S. Department of Commerce

MINISTÈRE DES AFFAIRES ÉTRANGÈRES

Détecter des alarmes (CWEs, ...) analysables sur du C :

- 119 to 127: buffer-related weaknesses
- 369: divide-by-zero
- 415: double-free, 416: use-after-free
- 457: use of uninitialized variable
- 476: null pointer dereference
- 562: return of stack variable address
- 690: unchecked return value to null pointer dereference

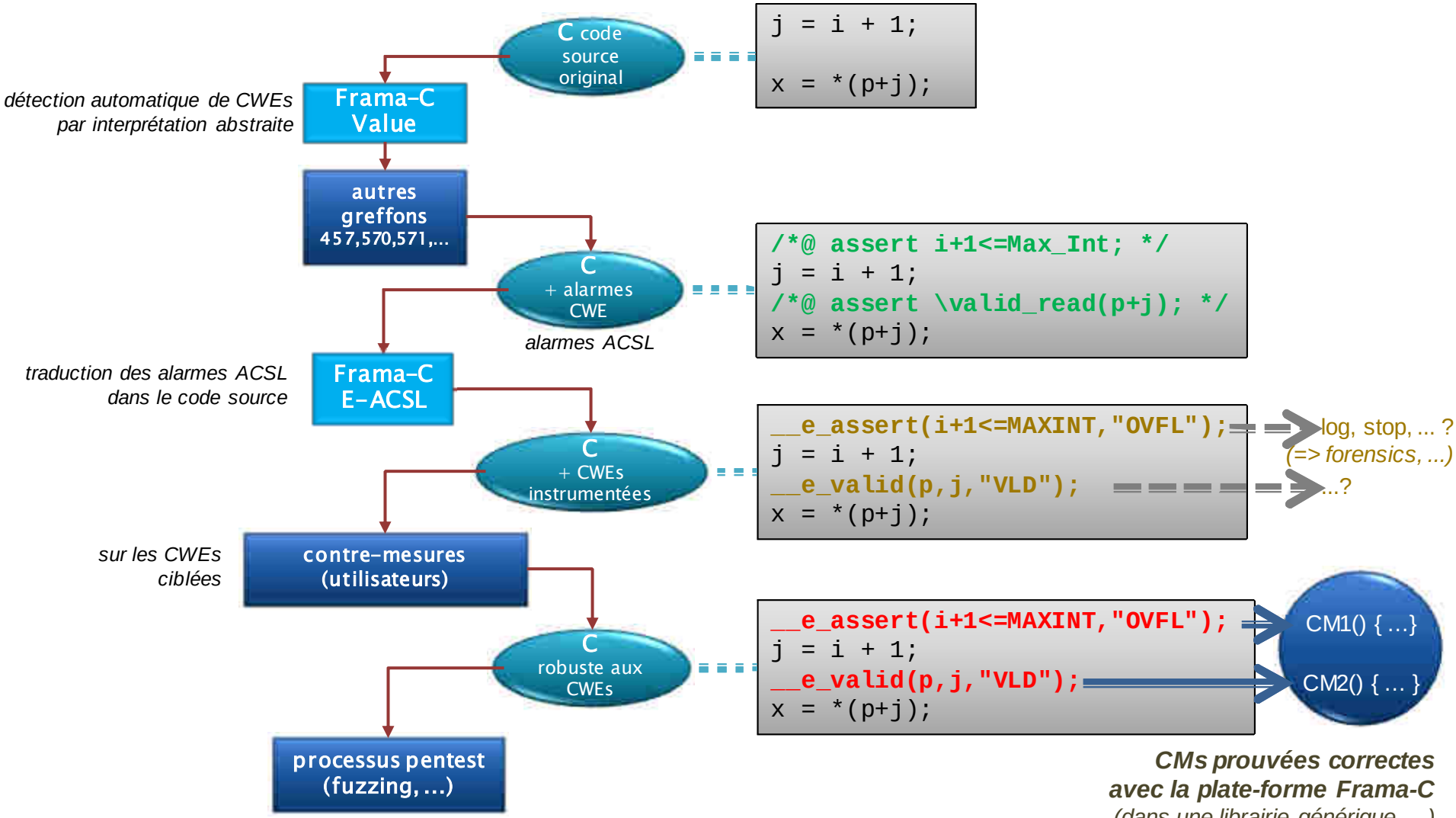
et avec quelques développements complémentaires :



- 174: double release of resource
- 457: uninitialized variable
- 570: always false conditions, 571: always true conditions
- ..., race conditions (à venir)
- propriétés plus fonctionnelles (*business logic*)

Définir les actions (contre-mesures) en cas d'activation d'une alarme

Méthode CURSOR



CMs prouvées correctes avec la plate-forme Frama-C
(dans une librairie générique, ...)

: code source
: outils CEA
: outils complémentaires

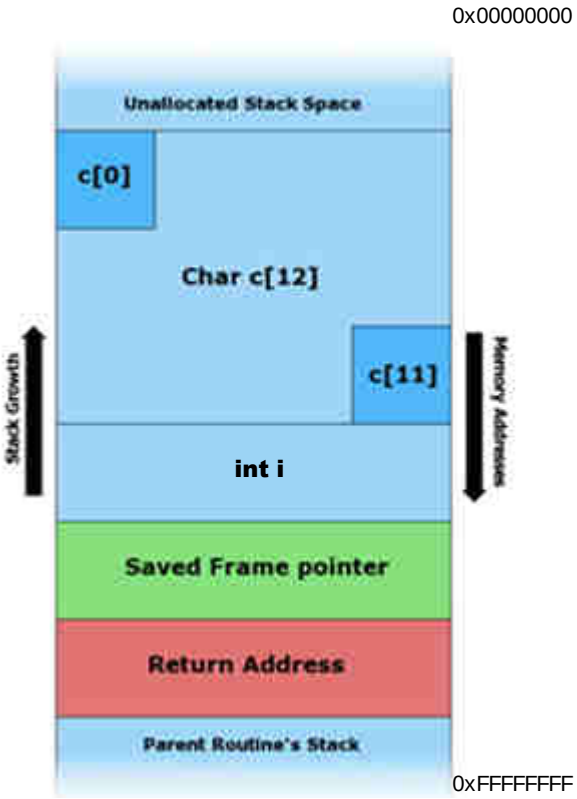
DEMOS

- Stack-Based Buffer Overflow
- OpenSSL/HeartBleed

Démo 1 : Stack-Based Buffer Overflow

```
int main(int argc, char**argv)
{
  int i = 12345;
  char c[12];

  strncpy(c, argv[1], strlen(argv[1]));
  ...
}
```

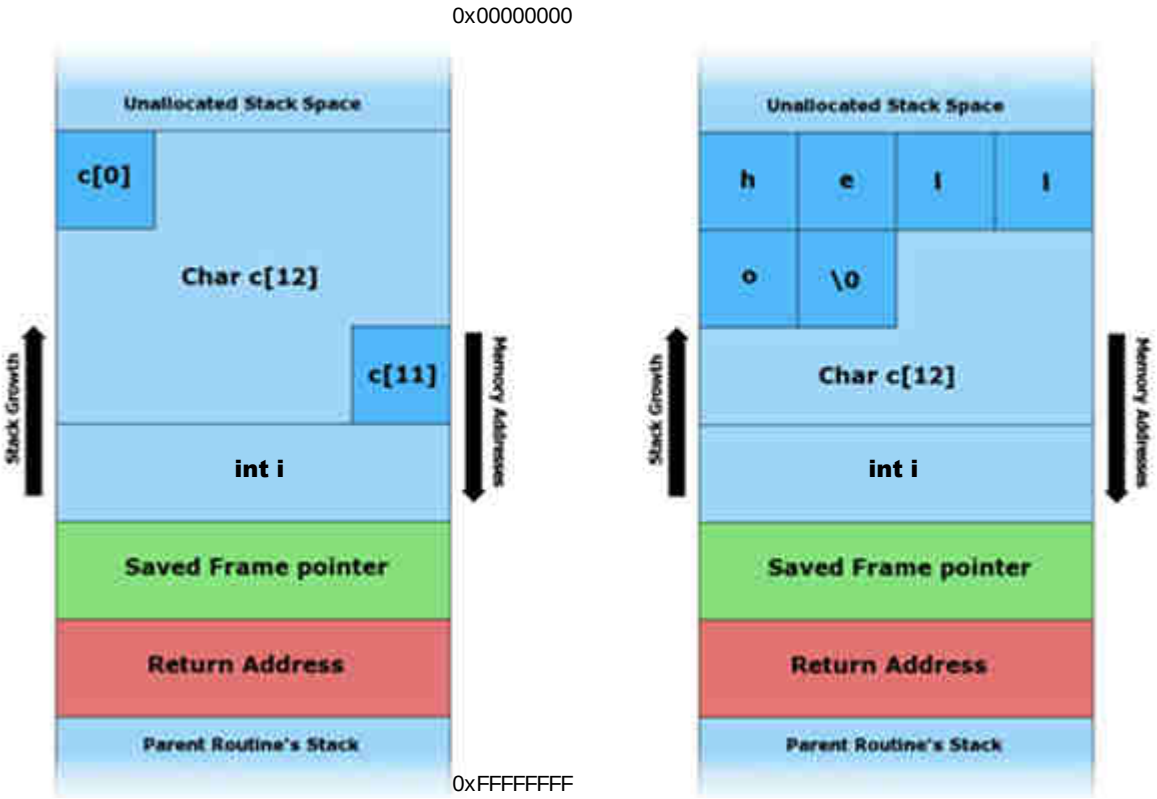


images Wikipedia

Démo 1 : Stack-Based Buffer Overflow

```
int main(int argc, char**argv)
{
  int i = 12345;
  char c[12];

  strncpy(c, argv[1], strlen(argv[1]));
  ...
}
```



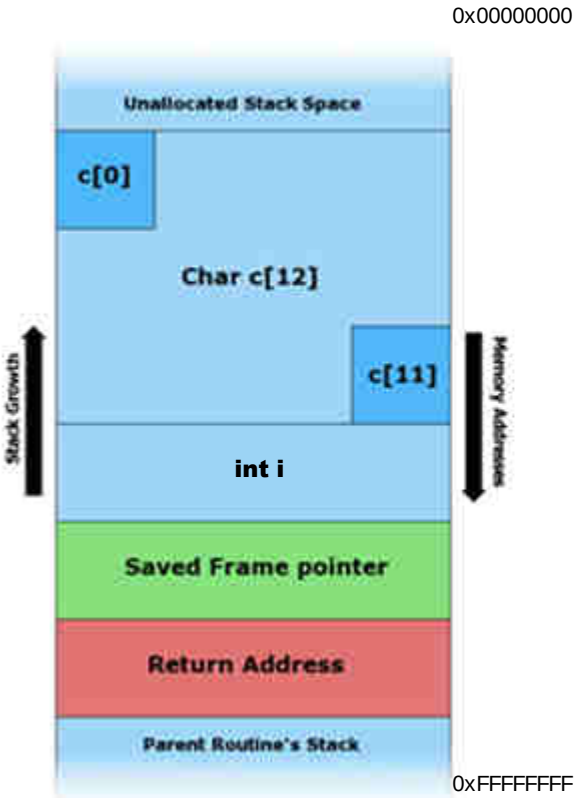
Appel avec la chaîne
"hello"

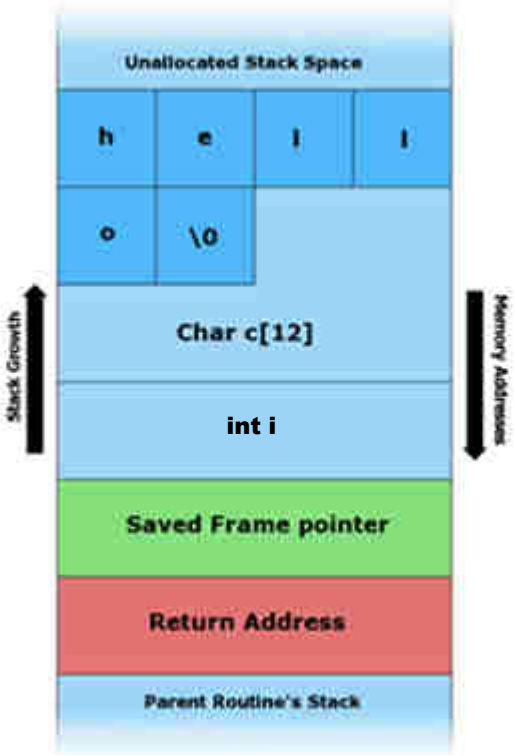
images Wikipedia

Démo 1 : Stack-Based Buffer Overflow

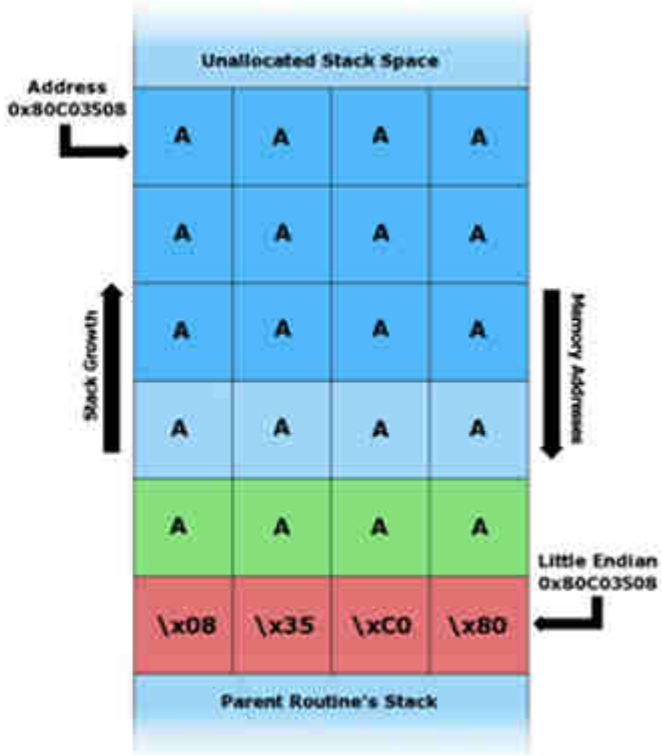
```
int main(int argc, char**argv)
{
  int i = 12345;
  char c[12];

  strncpy(c, argv[1], strlen(argv[1]));
  ...
}
```





Appel avec la chaîne
"hello"



Appel avec
"AA...A\x08\x35\xC0\x80"

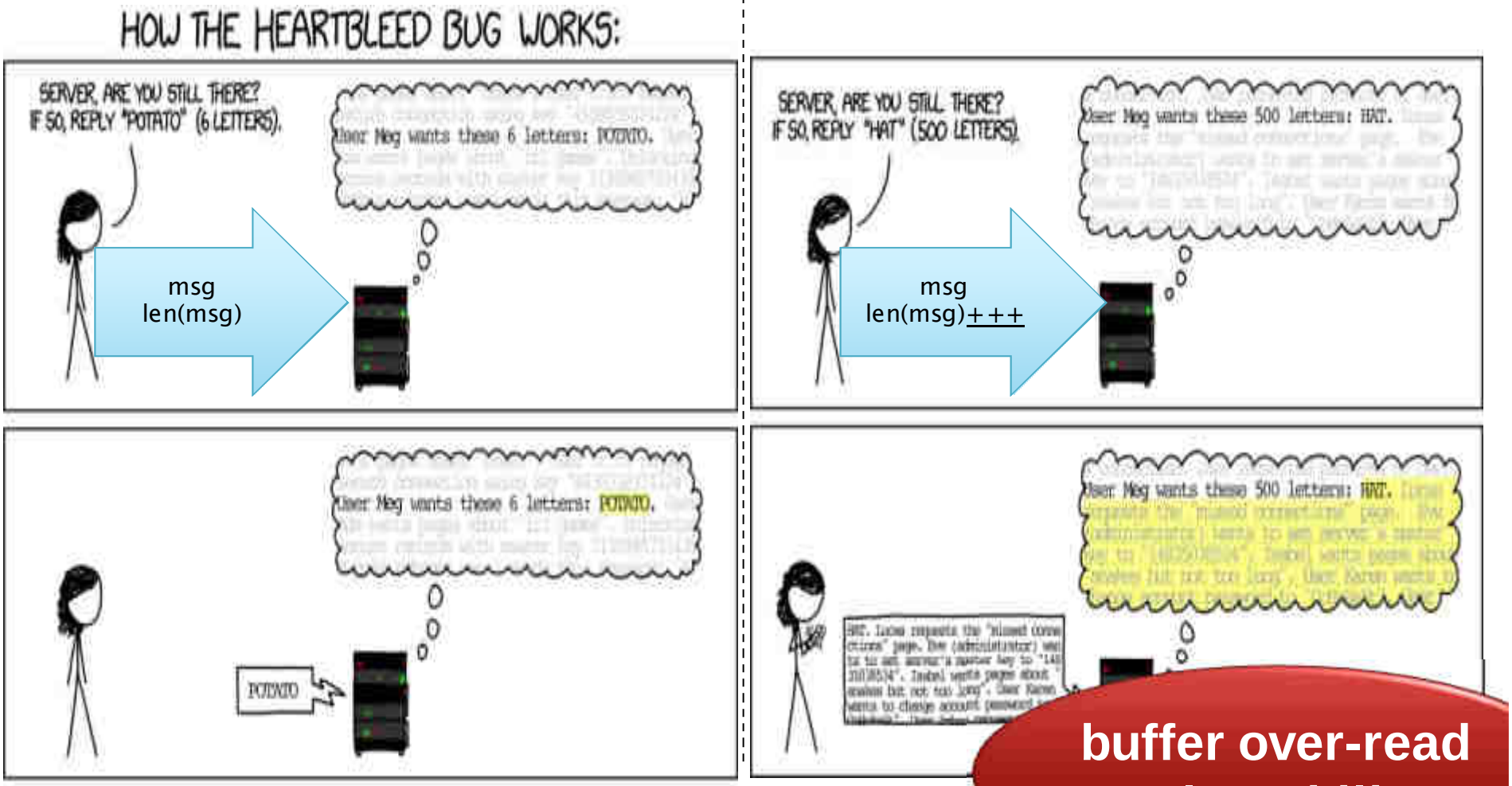
images Wikipedia

DEMOS

- Stack-Based Buffer Overflow
- OpenSSL/HeartBleed

Démo 2 : HeartBleed dans OpenSSL

OpenSSL versions<1.0.1g, CVE-2014-0160
sur Wikipédia, Amazon, GitHub, Yahoo!, Flickr, ...



buffer over-read vulnerability

source : xkcd.com

Démo 2 : HeartBleed dans OpenSSL

Code vulnérable dans la fonction `tls1_process_heartbit` :

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);   /* normally payload=16, padding=18 */
bp = buffer;

                                                /* enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;                             /* store 2-byte response header in buffer */
s2n(payload, bp);                                      /* store 2-byte payload length in buffer */
memcpy(bp, pl, payload);
bp += payload;

RAND_pseudo_bytes(bp, padding);
r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
```

Pour la démo, utilisation de ***Heartbit*** :

un simulateur instrumenté de la vulnérabilité HeartBleed

http://research.cs.wisc.edu/mist/heartbit/heartbit-1.0

Application de CURSOR, et expérimentation d'un couplage avec un fuzzer

Conclusion et Perspectives

- CURSOR = combinaison analyses statique et dynamique pour appliquer des contre-mesures
- **1ers résultats** expérimentaux avec CURSOR **plutôt conclusifs**
- **faible investissement** pour des résultats efficaces

- **questions ouvertes**
  - existe-t-il toujours des CM applicables ?
  - contribution de CURSOR au processus d'évaluation (à la CC) ?

- **travaux futurs**
  - passage à l'échelle d'E-ACSL :
    - améliorer les performances en temps (x3 à x17) et mémoire (x2.5)
  - prise en compte de plus d'alarmes (CWEs, …)
  - approfondir les couplages avec pentesting (fuzzing, ...) et techniques ASan, TSan, USan, …

VESSEDIA Grant Agreement No. 731453

"The project VESSEDIA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731453."

If you need further information, please contact the coordinator:

TECHNIKON Forschungs- und Planungsgesellschaft mbH
Burgplatz 3a, 9500 Villach, AUSTRIA
Tel: +43 4242 233 55 Fax: +43 4242 233 55 77
E-Mail: coordination@vessedia.eu