

Runtime Verification of Security Properties with E-ACSL

Julien Signoles

Software Reliability and Security Lab



Sound Static Analysis for Security Workshop

2018/6/28

(long m
(for i < 0
C1); if (b
tmp2 =
at the
tmp2[i][0] = -1; i < (NBi - 1); else if (tmp1[i][0] >= (1 - i) * (NBi - 1)) tmp2[i][0] = (1 - i) * (NBi - 1) - 1; else tmp2[i][0] = tmp1[i][0] - 1; then have good pass condition like this first one:
tmp1[i][0] >= 0; k < 0; k < i; tmp1[i][k] = tmp2[i][k];" The [i][j] coefficient of the matrix product $MC_2^T \cdot TMP_2$, that is, " $(MC_2^T \cdot TMP_2)_{ij} = MC_2^T_{ik} \cdot (MC_1 \cdot M_1)_{kj} = MC_2^T_{ik} \cdot M_{1,j}$ ".
i++ = 1; tmp1[i][0] >= 1; /* Final rounding, tmp2[i][0] is now represented on 9 bits. */ if (tmp1[i][0] > 255) m2[i][0] = 255; else if (tmp1[i][0] < -255) m2[i][0] = -255; else m2[i][0] = tmp1[i][0];



Frama-C

- ▶ opensource framework for analyzing code written in ISO C99
- ▶ used in many critical domains (avionic, energy, telecom, health)
- ▶ provides lots of analyzers as plug-ins
- ▶ C code may be annotated with ACSL formal specifications

This talk: E-ACSL

- ▶ Runtime verification with Frama-C by means of E-ACSL
- ▶ as automatic as possible
- ▶ for finding security weaknesses

(long m
for i < 0
C); if (b
tmp2, i
at the
tmp2[i][
0 < i < (NBi - 1) & else if (tmp1[i][0] >= 0 & i < (NBi - 1)) tmp2[i][0] = 0 & (tmp1[i][0] - 1) <= tmp2[i][0] & tmp1[i][0] >= -1) & then have good pass. Code like this fails.
tmp1[i][0] = 0 & k < 0 & k < i) tmp1[i][0] = mc2[i][k]; tmp2[i][0] = mc2[i][k]; The [i][j] coefficient of the matrix product MC2^T * MP2, that is, $(MC2^T)^T \cdot MP2 = MC2^T \cdot (MC2^T)^T \cdot MP2 = MC2^T \cdot MP2$.
i++ == 1 & tmp1[i][0] >= 1 /" Final rounding: tmp2[i][0] is now represented on 9 bits. "/ if (tmp1[i][0] > 255) m2[i][0] = 255; else if (tmp1[i][0] < -256) m2[i][0] = -256; else m2[i][0] = tmp1[i][0];



E-ACSL, as a specification language

- ▶ Executable-ACSL: executable subset of ACSL
- ▶ `\forall integer i; 0 <= i < len-1 ==> t[i] <= t[i+1]`

E-ACSL, as a tool

- ▶ convert formal annotations into C code to be validated at runtime
- ▶

```
int res = 1;
for(int i = 0; i < len-1; i++)
    if (t[i] > t[i+1]) { res = 0; break; }
e_acsl_assert(res);
```
- ▶ runtime assertion checker



Automatic Usages of E-ACSL for Security

some Frama-C plug-ins generate E-ACSL annotations

- ▶ RTE + E-ACSL for preventing undefined behaviors to be executed
- ▶ SecureFlow + E-ACSL for detecting information flow leakage
- ▶ Aorai + E-ACSL for API conformance checking

(long m
(for i < 0
C1); if (b
tmp2, i
at the
if (tmp2[i][0] = -1 <= i < (NBi - 1)) else if (tmp1[i][0] >= (1 <= i < (NBi - 1))) tmp2[i][0] = (1 <= i < (NBi - 1)) - 1 else tmp2[i][0] = -tmp1[i][0]; /* Then have a good pass condition like this first */ if (tmp1[i][0] >= 0 & k < 0 & k < i) tmp1[i][0] = -tmp2[i][0]; /* The [i][j] coefficient of the matrix product MC2^T * MP2, matrix * MC2^T * TMP2 = MC2^T * (MC1^W1) = MC2^T * W1 * MC1^W1 = MC2^T * W1 */ i+= 1; tmp1[i][0] >= 1; /* Final rounding, tmp2[i][0] is now represented on 9 bits. */ if (tmp1[i][0] > 255) m2[i][0] = -255; else if (tmp1[i][0] > 255) m2[i][0] = 255; else m2[i][0] = tmp1[i][0];



Usage 1: Detecting Undefined Behaviors

- ▶ Plug-in **RTE** automatically generates E-ACSL annotations for preventing potential undefined behaviors
- ▶ Combining **RTE + E-ACSL** allows to **detect undefined behaviors at runtime**

```
(long m[
```

```
{ for (j = 0;
```

```
    C1); if (b[
```

```
    tmp2 =
```

```
    at the
```

```
tmp2][j][0] = -1; j < (NBj - 1); else if (tmp1[0][0] >= 1 < (NBj - 1)) tmp2[0][0] = 0; j < (NBj - 1); else if (tmp2[0][0] >= 0 & j < (NBj - 1)) tmp2[0][0] = 1; else if (tmp2[0][0] >= 0 & j < (NBj - 1)) tmp2[0][0] = 0; k < (Nkj - 1); tmp1[0][0] = mC2[0][k]; tmp2[0][0] = mC2[0][k]; } The [j][0] coefficient of the matrix product MC2^T * MP2, that is,  $(MC2^T)^T \cdot MP2 = MC2^T \cdot (MC2^T)^{-1} \cdot MP2 = MC2^T \cdot MP2$ . MC2^T is now represented on 9 bits. If (tmp1[0][0] >= -255 & m2[0][0] <= 255) else if (tmp1[0][0] > 255 & m2[0][0] < -255) else if (tmp1[0][0] >= 1 & m2[0][0] <= 1) "Final rounding: tmp2[0][0]
```



Usage 1: Detecting Undefined Behaviors

- ▶ Plug-in RTE automatically generates E-ACSL annotations for preventing potential undefined behaviors
- ▶ Combining RTE + E-ACSL allows to detect undefined behaviors at runtime

```
int incr(int x) {  
    return x++;  
}
```

```
(long m  
(for i < 0  
C1); if (b  
tmp2 =
```



Usage 1: Detecting Undefined Behaviors

- ▶ Plug-in **RTE** automatically generates E-ACSL annotations for preventing potential undefined behaviors
- ▶ Combining **RTE + E-ACSL** allows to **detect undefined behaviors at runtime**

```
int incr(int x) {    int incr(int x) {  
    return x++;        RTE      /*@ assert x+1 <= 2147483647; */  
}  
}
```

(long m
(for i < 0
C1); if (b
tmp2, i
at the
)

tmp2[i][0] = -1, i < (NBi - 1); else if (tmp1[i][0] >= 1, i < (NBi - 1)) tmp2[i][0] = 0, i < (NBi - 1), else tmp2[i][0] = tmp1[i][0] - 1; else if (tmp1[i][0] >= 0, k < 0, k < i) tmp2[i][0] = -tmp2[i][0]; tmp2[0][0] = tmp2[0][0] / 2^n; The [i][j] coefficient of the matrix product MC2^T * MP2, that is, $(MC2^T)^T \cdot MP2 = MC2^T \cdot (MC1^T \cdot M1) = MC2^T \cdot M1 \cdot MC1^T = MC2^T \cdot MP2$.
i++ = 1; tmp1[i][0] >= 1; /* Final rounding, tmp2[0][0] is now represented on 9 bits. */ if (tmp1[0][0] > 255) m2[0][0] = 255; else if (tmp1[0][0] < -255) m2[0][0] = -255; else m2[0][0] = tmp2[0][0];

Usage 1: Detecting Undefined Behaviors

- ▶ Plug-in **RTE** automatically generates E-ACSL annotations for preventing potential undefined behaviors
- ▶ Combining **RTE + E-ACSL** allows to **detect undefined behaviors at runtime**

```
int incr(int x) {    int incr(int x) {  
    return x++;           RTE          /*@ assert x+1 <= 2147483647; */  
}  
                      return x++;  
}  
  
int incr(int x) {  
    /*@ assert x+1 <= 2147483647; */  
    e_acsl_assert(x+1L <= 2147483647L);  
    return x++;  
}
```

E-ACSL

(long m
(for i < 0
C1); if (b
tmp2 =
at the
tmp2[i][0] = -1 <= (NBi - 1); else if (tmp2[i][0] > 1 <= (NBi - 1)) tmp2[i][0] = 0 <= (NBi - 1); else if (tmp2[i][0] > -1 <= (NBi - 1)) tmp2[i][0] = -1 <= (NBi - 1); then have second pass (code like this first)
tmp2[i][0] = 0; k < NBi - 1; tmp2[i][k] = tmp2[i][k+1]; tmp2[NBi][0] = 0; k < NBi - 1; tmp2[NBi][k] = tmp2[NBi][k+1]; then the [i][j] coefficient of the matrix product MC2^T * MP2, that is, MC2^T * TMP2 = MC2^T * (MC1 * M1) = MC2^T * M1 * M1^T * MC1^T = MC2^T * M1^T * MC1^T = MC2^T * MC1^T. Final rounding: tmp2[0][0] is now represented on 9 bits. If (tmp2[0][0] > 255) m2[0][0] = 255; else if (tmp2[0][0] < -256) m2[0][0] = -256; else if (tmp2[0][0] > 255) m2[0][0] = 255; else m2[0][0] = tmp2[0][0].



Usage 1: Detecting Undefined Behaviors

- ▶ Plug-in **RTE** automatically generates E-ACSL annotations for preventing potential undefined behaviors
- ▶ Combining **RTE + E-ACSL** allows to **detect undefined behaviors at runtime**

```
int incr(int x) {           int incr(int x) {  
    return x++;             RTE             /*@ assert x+1 <= 2147483647; */  
}  
                           return x++;  
}  
  
int incr(int x) {           E-ACSL  
    /*@ assert x+1 <= 2147483647; */  
    e_acsl_assert(x+1L <= 2147483647L);  
    return x++;  
}
```

- ▶ above, why does it convert **1** to **1L**?

Usage 1: Detecting Undefined Behaviors

- ▶ Plug-in **RTE** automatically generates E-ACSL annotations for preventing potential undefined behaviors
- ▶ Combining **RTE + E-ACSL** allows to **detect undefined behaviors at runtime**

```
int incr(int x) {           int incr(int x) {  
    return x++;             RTE      /*@ assert x+1 <= 2147483647; */  
}                           return x++;  
  
                                }  
  
                                ↓  
                                E-ACSL  
int incr(int x) {           int incr(int x) {  
    /*@ assert x+1 <= 2147483647; */  
    e_acsl_assert(x+1L <= 2147483647L);  
    return x++;  
}
```

- ▶ above, why does it convert **1** to **1L**?
 - ▶ for being **efficient**, while remaining **sound**

Usage 1: E-ACSL Memory Monitoring

cont'd

```
void f(void) {
    int x, y, z, *p;

    p = &x;
    x = 0;
    y = 1;
    z = 2;
    /*@ assert \valid(p); */

    *p = 3;

}
```

(long m
(for i < 0
C1); if (b
tmp2 =

tmp2)[i][0] = -1, i < (NBi - 1); else if (tmp1[0][0] >= 1, i < (NBi - 1)) tmp2[0][0] = 0, i < (NBi - 1), else if (tmp1[0][0] >= 1, i < (NBi - 1)) then have good pass condition for first row. Then
tmp2[i][j] = 0, k < 0, k < i, tmp1[0][0] >= m2[0][j][k] * tmp2[0][j][k]; The [i][j] coefficient of the matrix product MC2^TTM2, matrix "MC2^TTM2" = MC2^T*(MC1⁻¹W1)=MC2^TW1*MC1⁻¹.
i+= 1; tmp1[0][0] >= 1; /* Final rounding, tmp2[0][0] is now represented on 9 bits. */ if (tmp1[0][0] >= 255) m2[0][0] = 255; else if (tmp1[0][0] < -255) m2[0][0] = -255; else m2[0][0] = tmp1[0][0].

Usage 1: E-ACSL Memory Monitoring

cont'd

```
void f(void) {
    int x, y, z, *p;
    // local variable allocation
    store_block((void *)(&p), 4U); store_block((void *)(&z), 4U);
    store_block((void *)(&y), 4U); store_block((void *)(&x), 4U);
    full_init((void *)(&p)); p = &x; // initialization of p
    full_init((void *)(&x)); x = 0; // initialisation de x
    full_init((void *)(&y)); y = 1; // initialisation de y
    full_init((void *)(&z)); z = 2; // initialisation de z
    /*@ assert \valid(p); */
    // validity check
    { int e_acsl_valid;
        e_acsl_valid = valid((void *)p, sizeof(int));
        e_acsl_assert(e_acsl_valid); }
    *p = 3;
    // memory deallocation
    delete_block((void *)(&p)); delete_block((void *)(&z));
    delete_block((void *)(&y)); delete_block((void *)(&x));
}
```

}

```
(long m
for i < 0
C1); if (t
tmp2 =
```

if ($tmp2[0][0] = -1 \leq - (NB - 1)$) else if ($tmp1[0][0] = (1 \leq - (NB - 1)) \wedge tmp2[0][0] = (1 \leq - (NB - 1))$) then have second pass code like this first one. Note that $tmp1[0][0] = 0 \wedge k < 0 \wedge k > n$ $\Rightarrow tmp1[0][0] = -tmp2[0][0]$. The $[i][j]$ coefficient of the matrix product $MC2^T / MP2$, that is $^T MC2^T \cap MP2 = MC2^T \cap MC1 \cap MC1^T \cap MP2 = MC2^T \cap MC1^T \cap MP2$.
 $i \leftarrow 1 : tmp1[0][0] \geq -1$ Final rounding: $tmp2[0][0] = -256$ else if ($tmp1[0][0] > 255$) $m2[0][0] = -255$ else $m2[0][0] = tmp2[0][0]$.



Usage 1: E-ACSL Memory Monitoring

cont'd

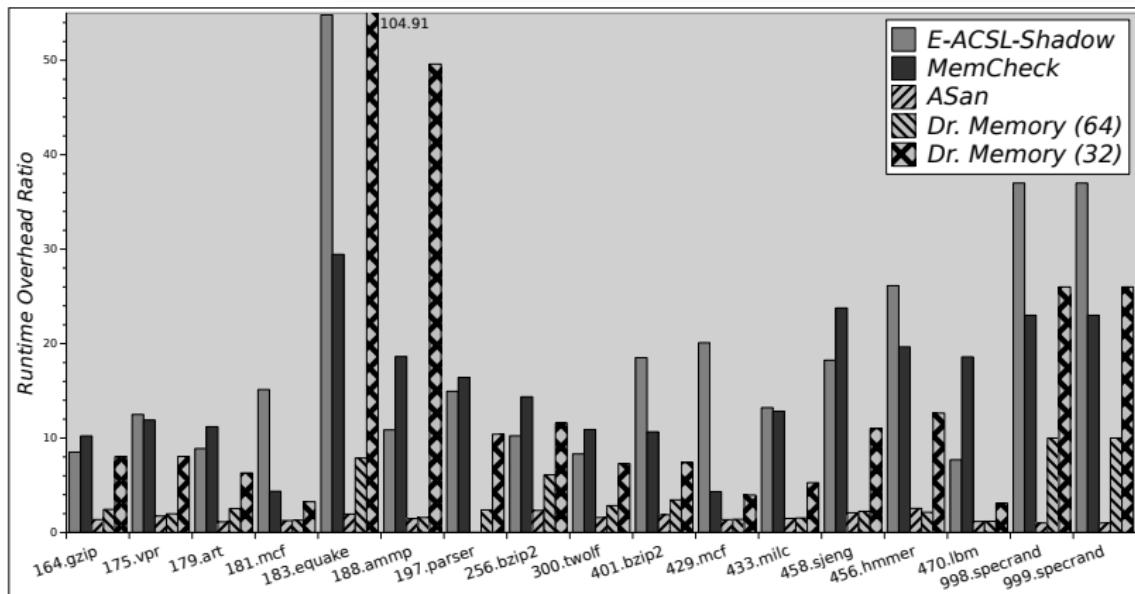
```
void f(void) {
    int x, y, z, *p;
    // local variable allocation
    store_block((void *)(&p), 4U); store_block((void *)(&z), 4U);
    store_block((void *)(&y), 4U); store_block((void *)(&x), 4U);
    full_init((void *)(&p)); p = &x; // initialization of p
    full_init((void *)(&x)); x = 0; // initialization of x
    full_init((void *)(&y)); y = 1; // initialization of y
    full_init((void *)(&z)); z = 2; // initialization of z
    /*@ assert \valid(p); */
    // validity check
    { int e_acsl_valid;
        e_acsl_valid = valid((void *)p, sizeof(int));
        e_acsl_assert(e_acsl_valid); }
    *p = 3;
    // memory deallocation
    delete_block((void *)(&p)); delete_block((void *)(&z));
    delete_block((void *)(&y)); delete_block((void *)(&x));
}
```

}

```
(long m
for i < 0
C1; if (t
tmp2 =
```

Efficient Generated Code





×17 time-overhead; ×2.4 memory overhead on SPEC-CPU

comparable to Valgrind; still slower than AddressSanitizer
less memory-overhead than these tools



Defect Type	E-ACSL	Google's Sanitizers
Dynamic Memory	94% (81/86)	78% (67/86)
Static Memory	✓ (67/67)	96% (64/67)
Pointer-related	56% (47/84)	32% (27/84)
Stack-related	35% (7/20)	70% (14/20)
Resource	99% (95/96)	60% (58/96)
Numeric	93% (100/108)	59% (64/108)
Miscellaneous	94% (33/35)	49% (17/35)
Inappropriate Code	– (0/64)	– (0/64)
Concurrency	– (0/44)	73% (32/44)
Overall	71% (430/604)	57% (343/604)

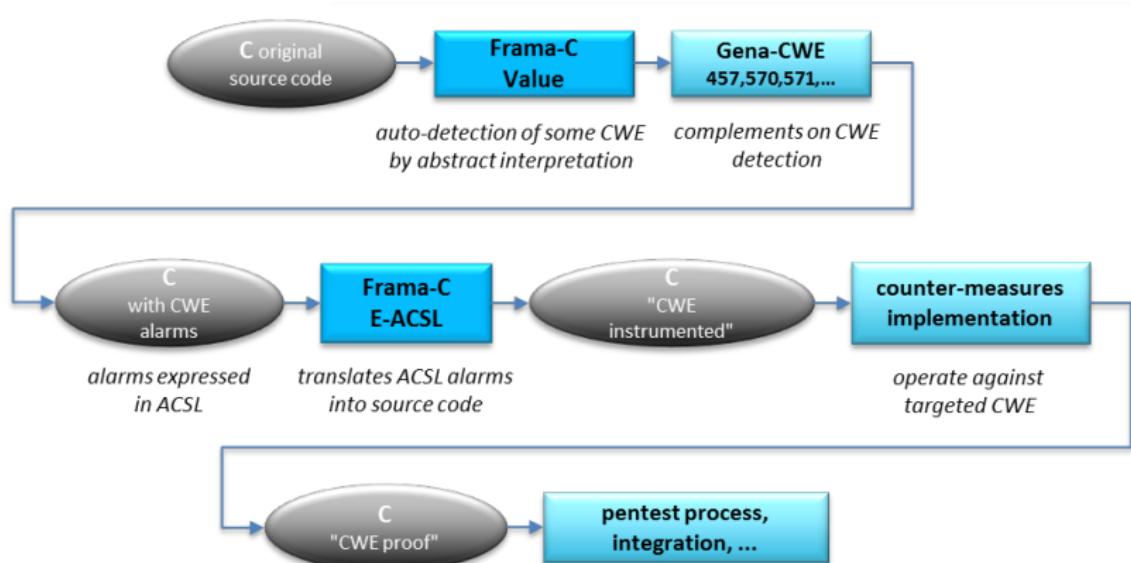
Detection Capabilities over Toyota ITC Benchmark:
more expressive than the mainstream tools

```
(long m
  for i < 0
    C1; if (t
      tmp2 =
```



Application to Generating Counter-Measures

by Dassault Aviation



First, use **automatic static analysis** to detect vulnerabilities

Then, switch to fast runtime monitoring

Experimented on modules from **Apache / OpenSSL**



Usage 2: Information Flow Tracking

- ▶ SecureFlow encodes program's information flow to C code
- ▶ allow usual code analyzers to verify information flow properties with standard techniques
 - ▶ Frama-C/Eva
 - ▶ Frama-C/E-ACSL
- ▶ does any private key leak on a public channel?
- ▶ may also detect some time-channel attacks
- ▶ experimented on LibTomCrypt (60,000-line C crypto library)
 - ▶ prove all the 14 symmetric cryptosystems as secure
 - ▶ detect one known vulnerability in 1 asymmetric cryptosystem
 - ▶ detect that the patch is unsecure

```
(long m[  
for i = 0;  
C1); if (b[  
tmp2[  
at the  
if (tmp2[0][0] <= (NB-1) && (i < (NB-1)) (tmp2[0][0] < 0 < (NB-1)) (tmp2[1][0] < 0 < (NB-1)) (tmp2[2][0] < 0 < (NB-1)) && (tmp2[3][0] < 0 < (NB-1)) {  
tmp2[1][0] = 0; k = 0; k<=i) (tmp2[1][0]) <= mc2[0][0]; } tmp2[2][0] = 0; k = 0; k<=i) (tmp2[2][0]) <= mc2[1][0]; } tmp2[3][0] = 0; k = 0; k<=i) (tmp2[3][0]) <= mc2[2][0]; }  
tmp2[1][0] = -256; else if ((tmp2[0][0] > 255) (m2[0][0] < -255) else m2[0][0] = tmp2[1][0]; }  
tmp2[2][0] = -256; else if ((tmp2[1][0] > 255) (m2[1][0] < -255) else m2[1][0] = tmp2[2][0]; }  
tmp2[3][0] = -256; else if ((tmp2[2][0] > 255) (m2[2][0] < -255) else m2[2][0] = tmp2[3][0]; }  
}
```



Usage 3: API Conformance Checking

- ▶ Aorai encodes temporal properties / automata to C code
- ▶ once again, allow usual code analyzers to be used
 - ▶ Frama-C/WP
 - ▶ Frama-C/E-ACSL
- ▶ is my crypto function always called after being initialized and eventually cleaned?
- ▶ experimented on a Linux driver to check that a sequence of hardware events does never happen from API calls

```
(long m  
for i < 0  
C1); if  
tmp2 <  
at the  
tmp2[i][0] = -1 <= (NBi - 1); else if (tmp1[i][0] >= 1 <= (NBi - 1)) tmp2[i][0] = 0 <= (NBi - 1) <= 1 else tmp2[i][0] = -tmp1[i][0]; if (tmp2[i][0] <= 0 && k <= 0 && k <= i) tmp2[i][0] = m2[i][k][0] * tmp2[i][0];  
The [i][j] coefficient of the matrix product MC2^T * MP2, matrix "MC2^T * MP2" = MC2^T * (MC1^W1) = MC2^T * W1 * MC1^W1  
i++ = 1; tmp1[i][0] >= 1; /* Final rounding, tmp2[i][0] is now represented on 9 bits */ if (tmp1[i][0] > 255) m2[i][0] = 255; else if m2[i][0] = tmp2[i][0];
```



- ▶ E-ACSL: Frama-C's runtime verification tool
- ▶ may detect undefined behaviors
 - ▶ reasonably efficient
 - ▶ better detection power than Google's sanitizer and Valgrind
- ▶ may detect incorrect information flows
- ▶ may detect incorrect API usages
- ▶ future works include
 - ▶ supporting additional safety and security properties
 - ▶ generating much more efficient code
 - ▶ time-overhead may still be reduced by 2×, at the least

(long m
(for i < 0
C1); if (b
tmp2 =

)(tmp2[i][0] = -1, i < -(NB-1)); else if (tmp1[0][0] > -(1, i < -(NB-1)) tmp2[0][0] = 0, i < -(NB-1), 1); else if (tmp2[i][0] > 0, k < 0, k < i) tmp1[0][0] = -mc2[i][k]; tmp2[0][0] = mc2[0][k]; The [i][j] coefficient of the matrix product $MC2^T \cdot MP2$, that is, $MC2^T(MC1 \cdot MP1) = MC2^T(MC1 \cdot MC1 \cdot MP1) = MC2^T(MC1 \cdot MC1 \cdot MC1 \cdot MP1)$, is given by the formula $MC2^T(MC1 \cdot MC1 \cdot MC1 \cdot MP1)_{ij} = \sum_k (-1)^{i+k} (MC2^T)_{ik} (MC1 \cdot MC1 \cdot MP1)_{kj}$. The first term in the sum is $(MC2^T)_{00} (MC1 \cdot MC1 \cdot MP1)_{0j} = -mc2[0][0] (MC1 \cdot MC1 \cdot MP1)_{0j}$. The second term is $(MC2^T)_{10} (MC1 \cdot MC1 \cdot MP1)_{1j} = mc2[1][0] (MC1 \cdot MC1 \cdot MP1)_{1j}$. The third term is $(MC2^T)_{20} (MC1 \cdot MC1 \cdot MP1)_{2j} = mc2[2][0] (MC1 \cdot MC1 \cdot MP1)_{2j}$. The fourth term is $(MC2^T)_{30} (MC1 \cdot MC1 \cdot MP1)_{3j} = mc2[3][0] (MC1 \cdot MC1 \cdot MP1)_{3j}$. The fifth term is $(MC2^T)_{40} (MC1 \cdot MC1 \cdot MP1)_{4j} = mc2[4][0] (MC1 \cdot MC1 \cdot MP1)_{4j}$. The sixth term is $(MC2^T)_{50} (MC1 \cdot MC1 \cdot MP1)_{5j} = mc2[5][0] (MC1 \cdot MC1 \cdot MP1)_{5j}$. The seventh term is $(MC2^T)_{60} (MC1 \cdot MC1 \cdot MP1)_{6j} = mc2[6][0] (MC1 \cdot MC1 \cdot MP1)_{6j}$. The eighth term is $(MC2^T)_{70} (MC1 \cdot MC1 \cdot MP1)_{7j} = mc2[7][0] (MC1 \cdot MC1 \cdot MP1)_{7j}$. The ninth term is $(MC2^T)_{80} (MC1 \cdot MC1 \cdot MP1)_{8j} = mc2[8][0] (MC1 \cdot MC1 \cdot MP1)_{8j}$. The tenth term is $(MC2^T)_{90} (MC1 \cdot MC1 \cdot MP1)_{9j} = mc2[9][0] (MC1 \cdot MC1 \cdot MP1)_{9j}$. The eleventh term is $(MC2^T)_{100} (MC1 \cdot MC1 \cdot MP1)_{10j} = mc2[10][0] (MC1 \cdot MC1 \cdot MP1)_{10j}$. The twelfth term is $(MC2^T)_{110} (MC1 \cdot MC1 \cdot MP1)_{11j} = mc2[11][0] (MC1 \cdot MC1 \cdot MP1)_{11j}$. The thirteenth term is $(MC2^T)_{120} (MC1 \cdot MC1 \cdot MP1)_{12j} = mc2[12][0] (MC1 \cdot MC1 \cdot MP1)_{12j}$. The fourteenth term is $(MC2^T)_{130} (MC1 \cdot MC1 \cdot MP1)_{13j} = mc2[13][0] (MC1 \cdot MC1 \cdot MP1)_{13j}$. The fifteenth term is $(MC2^T)_{140} (MC1 \cdot MC1 \cdot MP1)_{14j} = mc2[14][0] (MC1 \cdot MC1 \cdot MP1)_{14j}$. The sixteenth term is $(MC2^T)_{150} (MC1 \cdot MC1 \cdot MP1)_{15j} = mc2[15][0] (MC1 \cdot MC1 \cdot MP1)_{15j}$. The seventeenth term is $(MC2^T)_{160} (MC1 \cdot MC1 \cdot MP1)_{16j} = mc2[16][0] (MC1 \cdot MC1 \cdot MP1)_{16j}$. The eighteenth term is $(MC2^T)_{170} (MC1 \cdot MC1 \cdot MP1)_{17j} = mc2[17][0] (MC1 \cdot MC1 \cdot MP1)_{17j}$. The nineteenth term is $(MC2^T)_{180} (MC1 \cdot MC1 \cdot MP1)_{18j} = mc2[18][0] (MC1 \cdot MC1 \cdot MP1)_{18j}$. The twentieth term is $(MC2^T)_{190} (MC1 \cdot MC1 \cdot MP1)_{19j} = mc2[19][0] (MC1 \cdot MC1 \cdot MP1)_{19j}$. The twenty-first term is $(MC2^T)_{200} (MC1 \cdot MC1 \cdot MP1)_{20j} = mc2[20][0] (MC1 \cdot MC1 \cdot MP1)_{20j}$. The twenty-second term is $(MC2^T)_{210} (MC1 \cdot MC1 \cdot MP1)_{21j} = mc2[21][0] (MC1 \cdot MC1 \cdot MP1)_{21j}$. The twenty-third term is $(MC2^T)_{220} (MC1 \cdot MC1 \cdot MP1)_{22j} = mc2[22][0] (MC1 \cdot MC1 \cdot MP1)_{22j}$. The twenty-fourth term is $(MC2^T)_{230} (MC1 \cdot MC1 \cdot MP1)_{23j} = mc2[23][0] (MC1 \cdot MC1 \cdot MP1)_{23j}$. The twenty-fifth term is $(MC2^T)_{240} (MC1 \cdot MC1 \cdot MP1)_{24j} = mc2[24][0] (MC1 \cdot MC1 \cdot MP1)_{24j}$. The twenty-sixth term is $(MC2^T)_{250} (MC1 \cdot MC1 \cdot MP1)_{25j} = mc2[25][0] (MC1 \cdot MC1 \cdot MP1)_{25j}$. The twenty-seventh term is $(MC2^T)_{260} (MC1 \cdot MC1 \cdot MP1)_{26j} = mc2[26][0] (MC1 \cdot MC1 \cdot MP1)_{26j}$. The twenty-eighth term is $(MC2^T)_{270} (MC1 \cdot MC1 \cdot MP1)_{27j} = mc2[27][0] (MC1 \cdot MC1 \cdot MP1)_{27j}$. The twenty-ninth term is $(MC2^T)_{280} (MC1 \cdot MC1 \cdot MP1)_{28j} = mc2[28][0] (MC1 \cdot MC1 \cdot MP1)_{28j}$. The thirtieth term is $(MC2^T)_{290} (MC1 \cdot MC1 \cdot MP1)_{29j} = mc2[29][0] (MC1 \cdot MC1 \cdot MP1)_{29j}$. The thirtieth term is $(MC2^T)_{300} (MC1 \cdot MC1 \cdot MP1)_{30j} = mc2[30][0] (MC1 \cdot MC1 \cdot MP1)_{30j}$.

- ▶ J. Signoles, N. Kosmatov, and K. Vorobyov.
[E-ACSL, a Runtime Verification Tool for Safety and Security of C Programs \(tool paper\)](#).
RV-CuBES 2017.
- ▶ K. Vorobyov, J. Signoles, and N. Kosmatov.
[Shadow state encoding for efficient monitoring of block-level properties](#).
ISMM 2017.
- ▶ D. Pariente and J. Signoles.
[Static Analysis and Runtime Assertion Checking: Contribution to Security Counter-Measures](#).
SSTIC 2017.
- ▶ G. Barany and J. Signoles.
[Hybrid Information Flow Analysis for Real-World C Code](#).
TAP 2017

```
(long m)
for i = 0
C1; if (i
tmp2 =
```

```
tmp2[i][0] = -1 < -(NB-1) else if (tmp1[i][0] = 0 < -(NB-1)) tmp2[0][0] = 0 < -(NB-1); else if (tmp2[i][0] = tmp1[i][0]) then the second pass fails like the first one. Then
tmp1[i][0] < 0 & k < n tmp2[i][0] = mc2[i][0]; tmp2[i][j] is the j-th coefficient of the matrix product  $MC_2^T(MC_1)^{-1} = MC_2^T(MC_1^{-1}M) = MC_2^TM^{-1}$ .
```



- ▶ use GMP library for mathematical integers

```
/*@ assert y-1 == 0; */
mpz_t e_acsl_1, e_acsl_2, e_acsl_3, e_acsl_4;
int e_acsl_5;
mpz_init_set_si(e_acsl_1, y);                                // e_acsl_1 = y
mpz_init_set_si(e_acsl_2, 1);                                // e_acsl_2 = 1
mpz_init(e_acsl_3);
mpz_sub(e_acsl_3, e_acsl_1, e_acsl_2);                      // e_acsl_3 = y-1
mpz_init_set_si(e_acsl_4, 0);                                // e_acsl_4 = 0
e_acsl_5 = mpz_cmp(e_acsl_3, e_acsl_4); // (y-1) == 0
e_acsl_assert(e_acsl_5 == 0);                                // runtime check
mpz_clear(e_acsl_1); mpz_clear(e_acsl_2); // deallocate
mpz_clear(e_acsl_3); mpz_clear(e_acsl_4);
```

- ▶ how to restrict GMPs as most as possible? on-the-fly typing

almost no GMP in practice

[Jakobsson, Kosmatov & Signoles @JFLA'15]

(long m
for i < 0
C1; if (B
tmp2 =
of the

tmp2[i][0] = -1 <= (NBi - 1), else if (tmp1[i][0] > -1 <= (NBi - 1)) tmp2[i][0] = 0 <= (NBi - 1). Then the second pass looks like this first one. Now
tmp1[i][j] = 0; k = 0; k < i; k++ tmp1[i][k] = tmp2[i][k]; The [i][j] coefficient of the matrix product $M_2^{-1} \cdot M_1 P_2$, that is, $(M_2^{-1})^T M_1 P_2$, that is, $M_2^{-1} \cdot (M_1 \cdot P_2) = M_2^{-1} \cdot M_1 \cdot P_2$.
tmp2[i][0] = 1; tmp1[i][0] >= 1; Final rounding: tmp2[i][0] is now represented on 9 bits. If (tmp1[i][0] > 255) m2[0][0] = 255, else if (tmp1[i][0] < -255) m2[0][0] = -255, else m2[0][0] = tmp1[i][0].

