C Source Code Analysis for Memory Safety using Abstract Interpretation

September 17, 2020

Henny Sipma
Kestrel Technology

Paul E. Black
National Institute of Standards and Technology

Kestrel Technology CodeHawk Tool Suite

.class
.jar
.war

Java byte code front end

sound abstraction from Java byte code into CHIF

.c

CIL

C source code front end

sound abstraction from preprocessed CIL code into CHIF

.exe

x86 binary front end

disassembly
abstraction from x86 binary code into CHIF

CodeHawk

Abstract interpretation engine

Iterators

Abstract domains:
constants
intervals
strided intervals
linear equalities
polyhedra
symbolic sets
value sets
taint

CodeHawk C Analyzer

.c

CIL

C source code front end

sound abstraction from preprocessed CIL code into CHIF

CodeHawk

Abstract interpretation engine

Iterators

Abstract domains:
constants
intervals
strided intervals
linear equalities
polyhedra
symbolic sets
value sets
taint

Acknowledgements

The development of the CodeHawk Tool Suite was in part supported by

The development of the CodeHawk C Analyzer was in part supported by

Sound Static Memory Safety Analysis for C

Goal: Mathematically prove absence of memory safety vulnerabilities (covering more than 50 CWEs) for real-world applications

Approach:

- **Specification**: **C99 Standard – specification of undefined behavior**
- Translate into preconditions on instructions and library functions
- Prove that all preconditions are valid

Advantages:

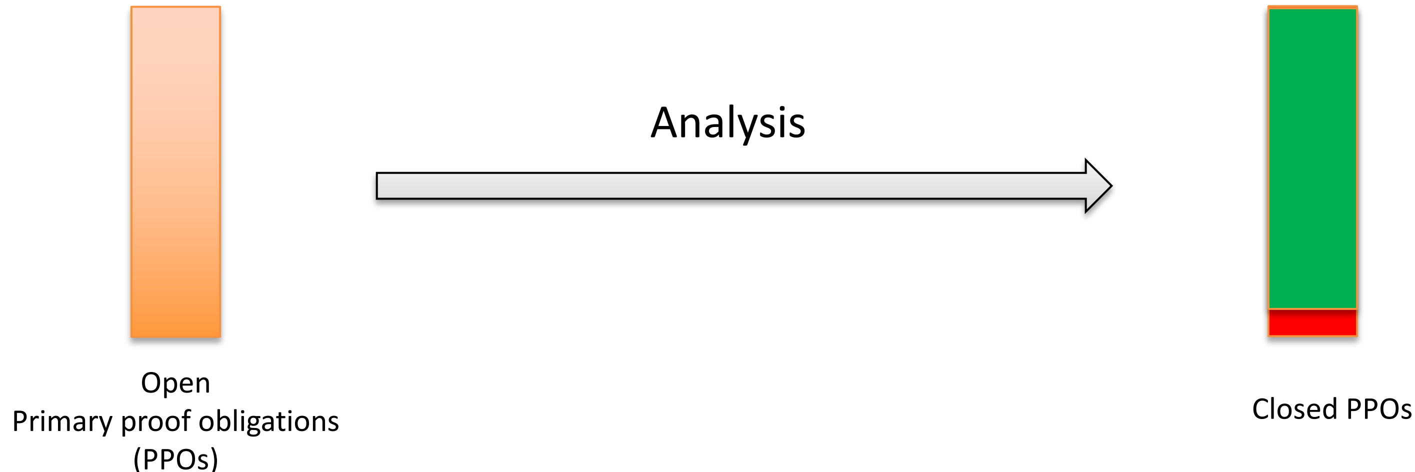
- If successful: full assurance of memory safety
- Exhaustive: no false negatives
- Evidence: results can be independently audited
- **Metrics**:
  1) **Progress**: percentage of proof obligations proven valid (safe: 100%)
  2) **Difficulty**: distribution of proof techniques required

Sound Static Memory Safety Analysis for C Challenges

Not automatic -- May involve significant effort

Approach:

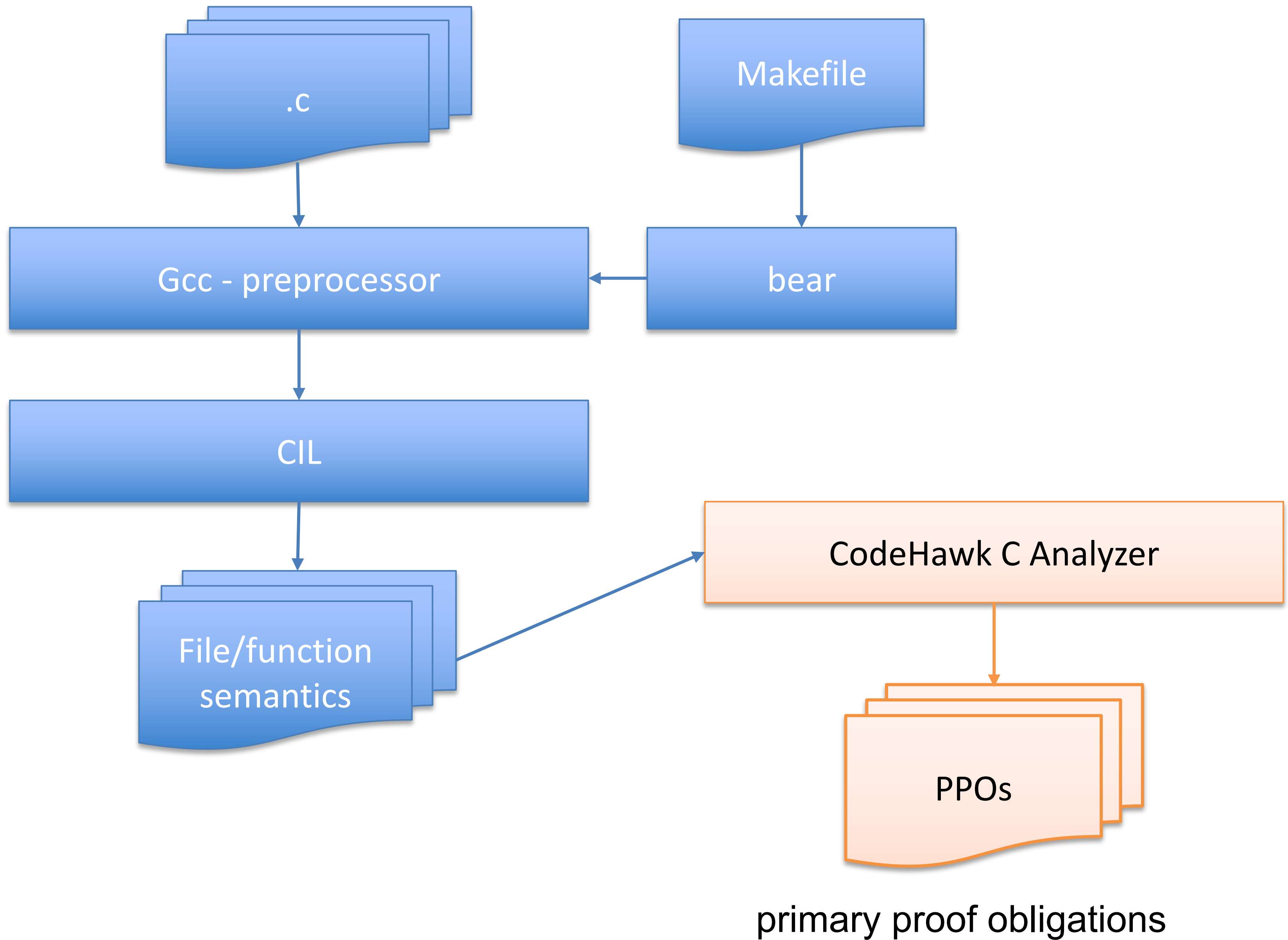
- ✓ **Specification**: C99 Standard – specification of undefined behavior
- ✓ Translate into preconditions on instructions and library functions: primary proof obligations (PPOs)
- ➢ Prove that all primary proof obligations (PPOs) are valid



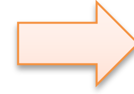
Test Applications

application	LOC
Cairo-1.14.12	227,818
Cleanflight-CLFL-v2.3.2	118,758
Dnsmasq-2.76	29,922
Dovecot-2.0.beta6 (SATE 2010)	208,636
File	14,379
Git-2.17.0	205,636
Hping	11,336
Irssi-0.8.14 (SATE 2009)	61,972
Lighttpd-1.4.18 (SATE 2008)	49,747
Nagios-2.10 (SATE 2008)	47,652
Naim-0.11.8.3.1 (SATE 2008)	25,759
Nginx-1.14.0	103,388
Nginx-1.2.9	102,151
Openssl-1.0.1.f	275,060
Pvm3.4.6 (SATE 2009)	60,029
Wpa_supplicant-2.6	96,554
Total	**1,638,797**

Creating Primary Proof Obligations: Fully Automatic



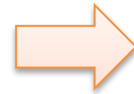
Primary Proof Obligations: How Many?



6,481,212

PPO count

Primary Proof Obligations: How Many?



6,481,212

PPO count

1200000
1,000,000
800,000
600,000
400,000
200,000
0

PPO density (PPOs/LOC)

5
4
3
2
1

cairo
cleanflight
dnsmasq
dovecot
file
git
hping
irssi
lighttpd
nagios
naim
nginx
nginx114
openssl
pvm
wpa_supplicant

Primary Proof Obligations: What are they?

First-order predicates

Some examples:

- allocation-base(p)
- cast(x,t1,t2)
- common-base(p1,p2)
- common-base-type(p1,p2)
- format-string(p)
- global-memory(p)
- index-lower-bound(a)
- index-upper-bound(a,s)
- initialized(v)
- initialized-range(p,s)
- int-overflow(op,a,b,t)
- int-underflow(op,a,b,t)
- lower-bound(p)
- no-overlap(p1,p2)
- non-negative(a)
- not-null(p)
- not-zero(a)
- null(p)
- null-terminated(p)
- pointer-cast(p,t1,t2)
- ptr-lower-bound(op,p,a)
- ptr-upper-bound(op,p,a)
- ptr-upper-bound-deref(op,p,a)
- signed-to-unsigned-cast(a,t1,t2)
- unsigned-to-signed-cast(a,t1,t2)
- upper-bound(p)
- valid-memory(p)
- value-constraint(x)
- width-overflow(a)

Primary Proof Obligations: How do we prove them valid?

Use increasingly sophisticated techniques, based on

A. Individual statement level information

B. Function-local invariants

C. Automatically inferred api conditions

D. Manually constructed contract conditions

Metric 2): Difficulty: distribution of proof techniques required

Primary Proof Obligations: Analysis

Simple Things First

A. Check validity based on individual statement level information

```
int a[10];
...
a[3] = 0;
```

index-lower-bound(3)

index-upper-bound(3,10)

```
strcpy(dst,"string")
```

null-terminated("string")

not-null("string")

lower-bound("string")

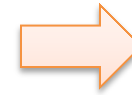
upper-bound("string")

valid-memory("string")

proof obligations

Primary Proof Obligations

Discharge PPOs at the statement level



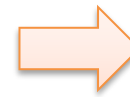
3,917,498

2,563,714

1200000
1000000
800000
600000
400000
200000
0

cairo, cleanflight, dnsmasq, dovecot, file, git, hping, irssi, lighttpd, nagios, naim, nginx, nginx114, openssl, pvm, a_supplicant

Primary Proof Obligations



3,917,498

Discharge PPOs at the statement level
(as a percent of total)

2,563,714

1.0
0.8
0.6
0.4
0.2
0.0

cairo
cleanflight
dnsmasq
dovecot
file
git
hping
irssi
lighttpd
nagios
naim
nginx
nginx114
openssl
pvm
wpa_supplicant

Primary Proof Obligations: Analysis
Generating Invariants

B. Check validity based on invariants generated

```
int a[10];
....
for (int i=0; i < 10; i++) {
  a[i] = 0;
}
```

index-lower-bound(i)
index-upper-bound(i)

i = [0 .. 9]

```
 1. int x;
....
10. x = ....
....
20. x = x + 1;
```

...
initialized (x)
...

x:initialized@10

proof obligations

invariant

Analysis: Generating Local Invariants (Context-insensitive)

- Abstract Interpretation (Cousot, Cousot, 1977)
- Domains:
  - Intervals (Cousot, Cousot)
  - Linear Equalities (Karr, 1976)
  - Value Sets (Reps, 2004)
  - Symbolic Sets
  - Parametric Ranges
- Flow-sensitive, Path-insensitive

Abstract interpretation engine

Iterators

Abstract domains:
constants
intervals
strided intervals
linear equalities
polyhedra
symbolic sets
value sets
taint

Analysis: Generating Local Invariants

.c

Makefile

Gcc - preprocessor

bear

CIL

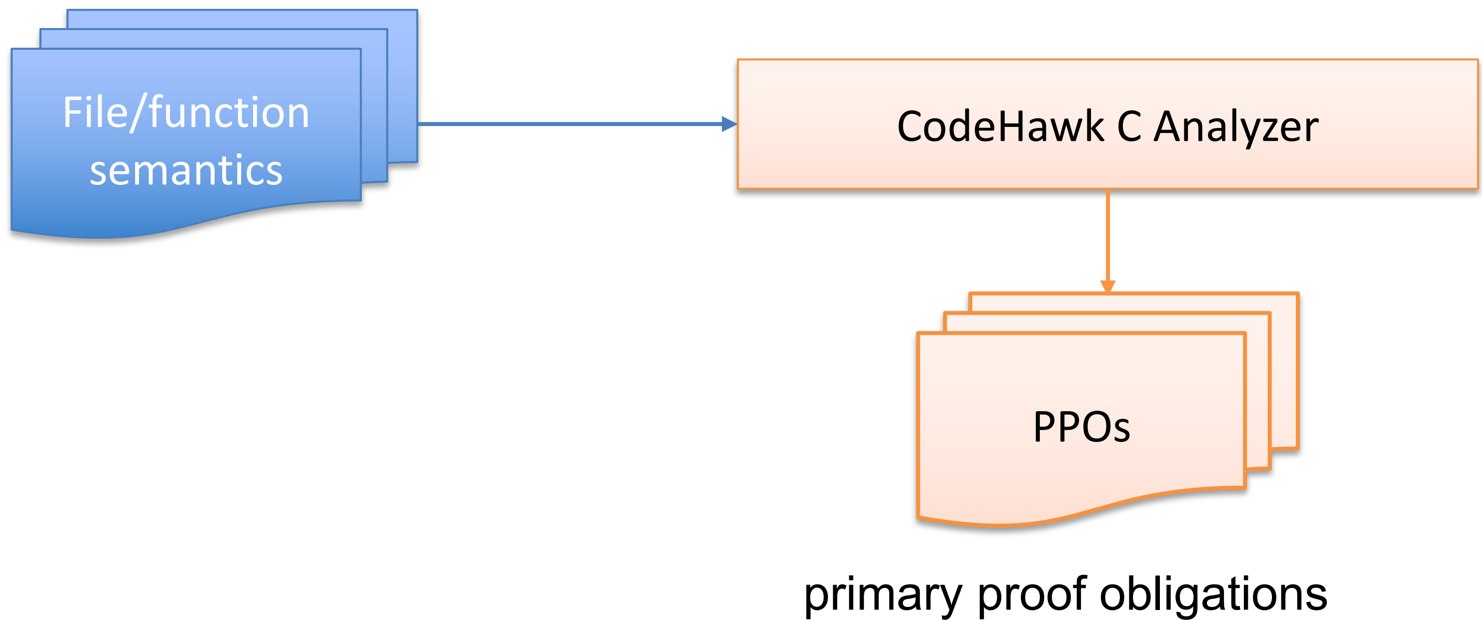
CodeHawk C Analyzer

File/function semantics

PPOs

primary proof obligations

Analysis: Generating Local Invariants



Analysis: Generating Local Invariants

PPOs

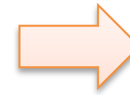
File/function semantics

CodeHawk C Analyzer

invariants

Primary Proof Obligations

Discharge PPOs using local function invariants



1,778,782

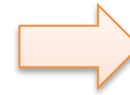
4,702,430

1200000
1000000
800000
600000
400000
200000
0

cairo cleanflight dnsmasq dovecot file git hping irssi lighttpd nagios naim nginx nginx114 openssl pvm wpa_supplicant

Primary Proof Obligations

Discharge PPOs using local function invariants (as a percent of total)



1,778,782

4,702,430

1.0
0.8
0.6
0.4
0.2
0.0

cairo
cleanflight
dnsmasq
dovecot
file
git
hping
irssi
lighttpd
nagios
naim
nginx
nginx114
openssl
pvm
pa_supplicant

Kestrel Technology

Analysis: Delegating Proof Obligations (Context sensitivity)

C. Lift responsibility to api

```
int f (int *p) {
  int *q;
  int x;
  ...
  q = p;
  x = *q + 5;
```

1. not-null(q)
2. valid-memory(q)
3. lower-bound(q)
4. upper-bound(q)
5. initialized(*q)
6. Int-overflow(*q+5)

proof obligations

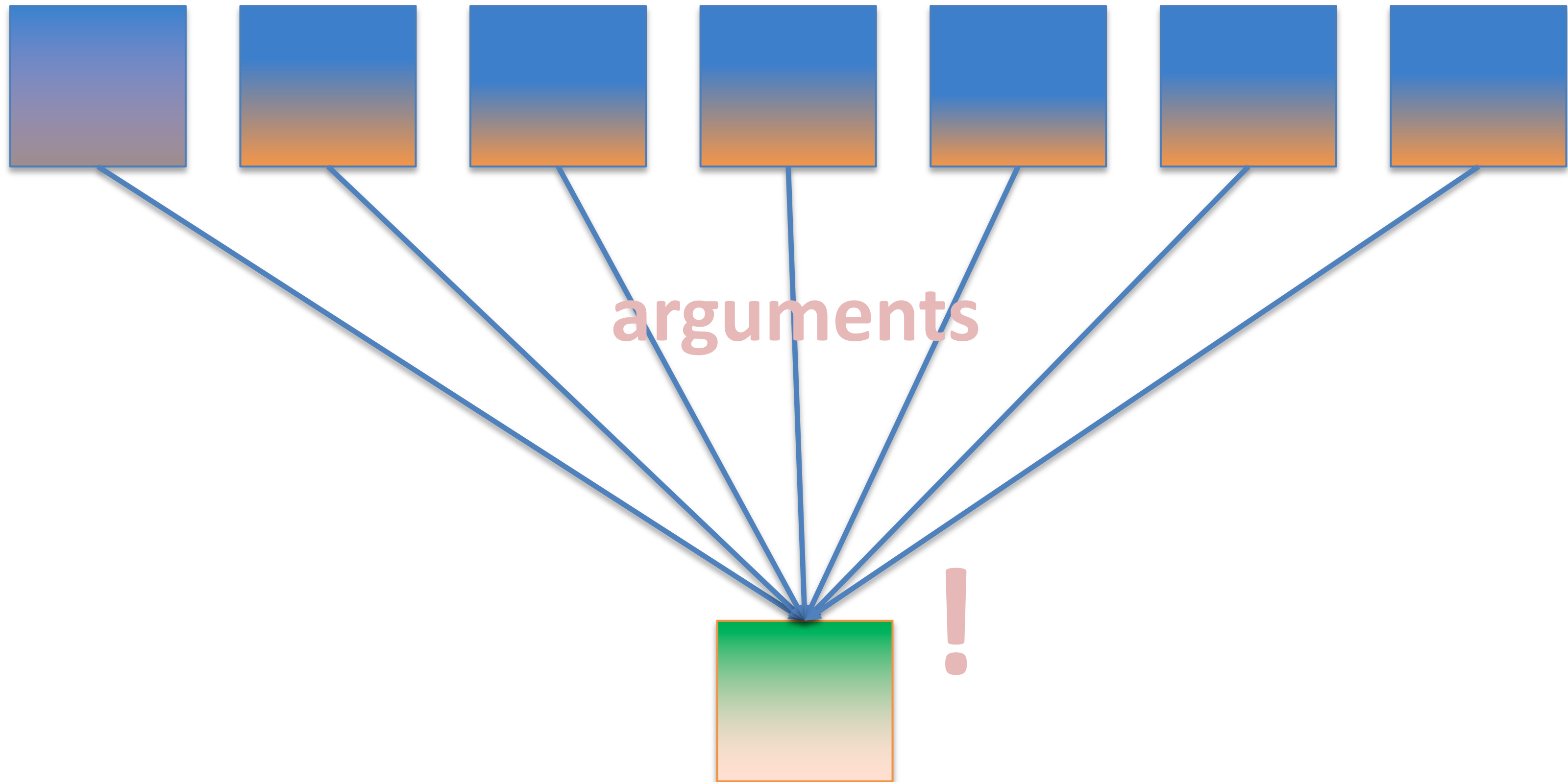
p = q

invariant

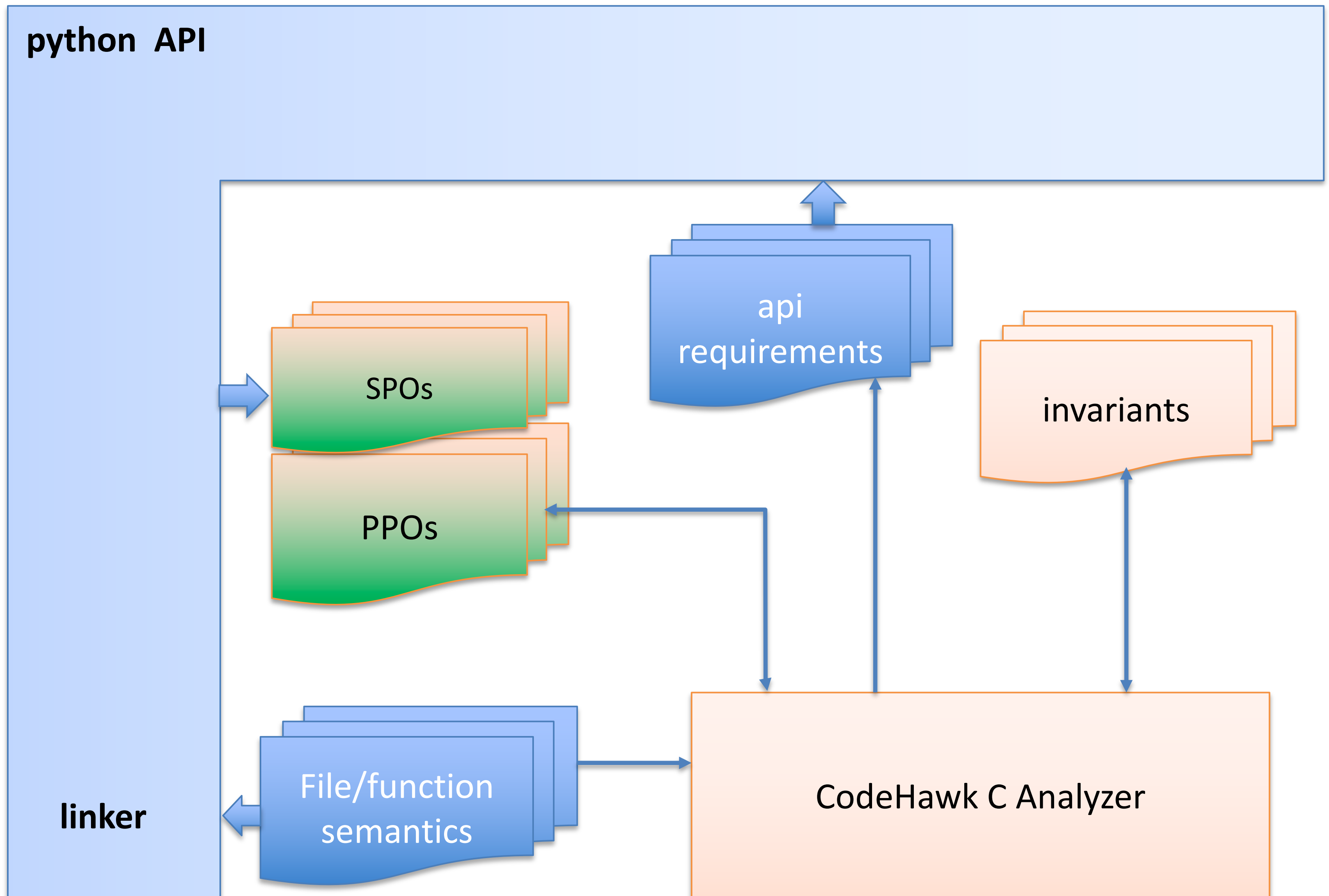
1. not-null(p)
2. valid-memory(p)
3. lower-bound(p)
4. upper-bound(p)
5. initialized(*p)
6. Int-overflow(*p + 5)

API requirements on f

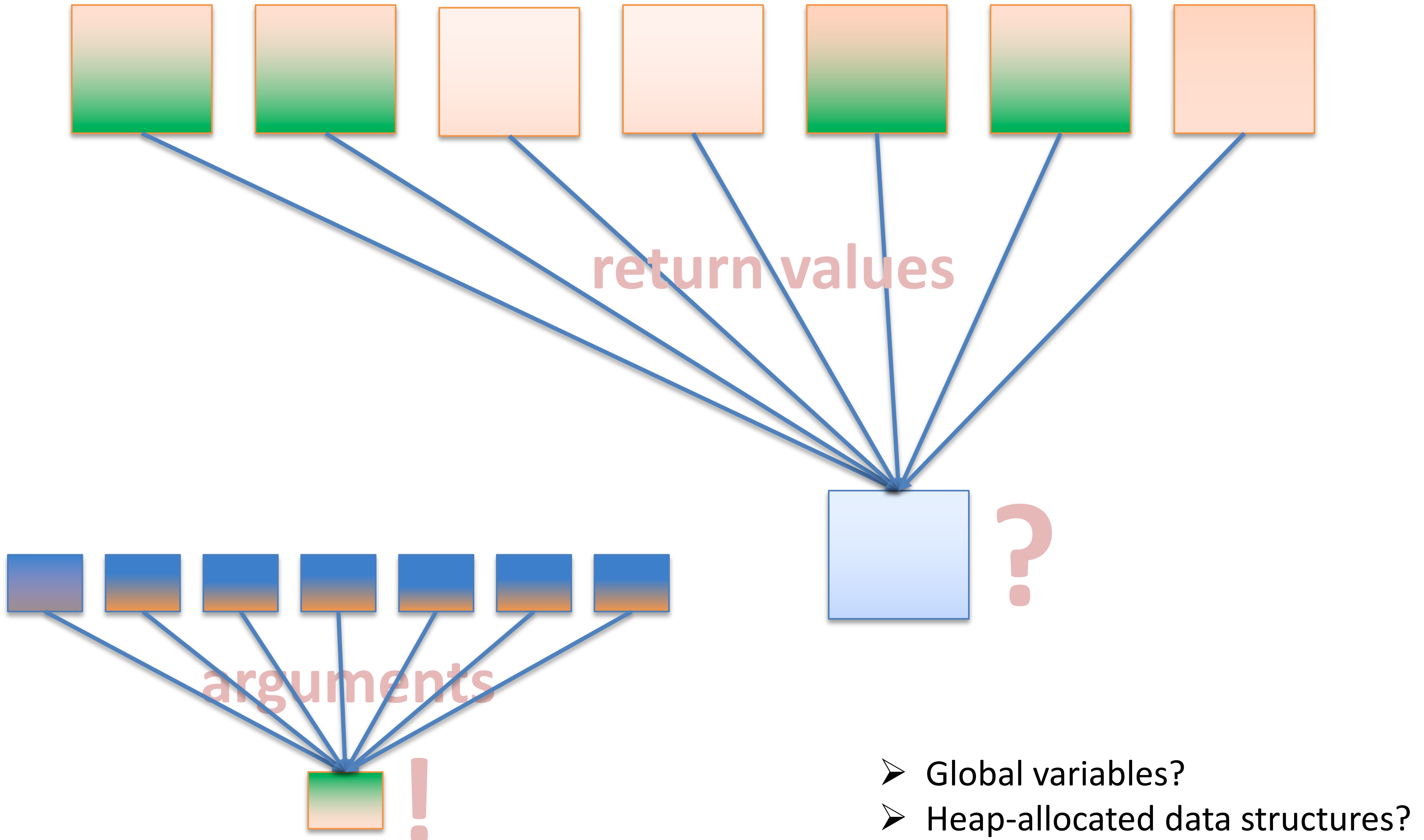
Analysis: Delegating Proof Obligations Impose Preconditions on Callers



Analysis: Delegating Proof Obligations
Create Supporting Proof Obligations

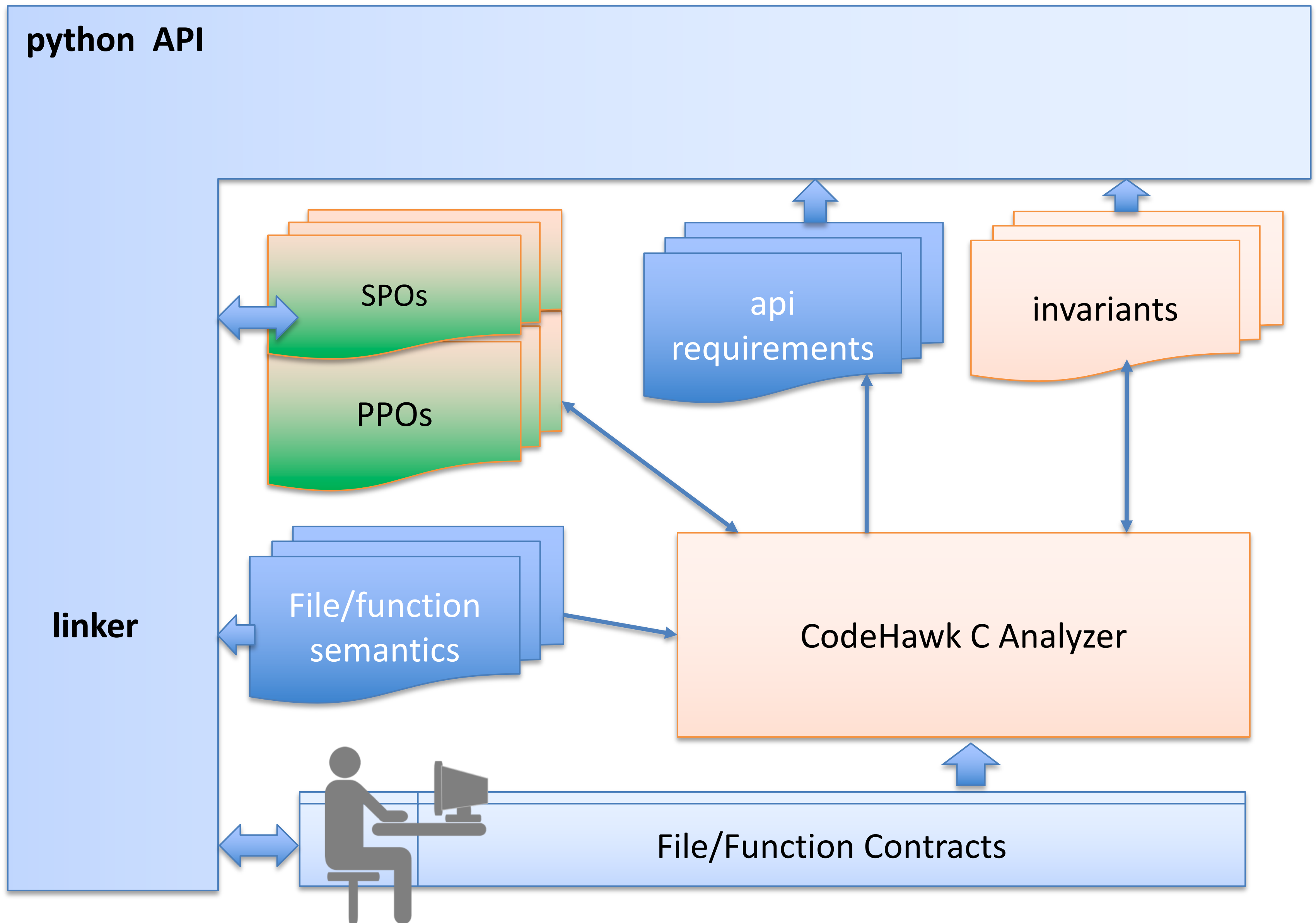


Analysis: Delegating Proof Obligations Impose Postconditions on Callers?



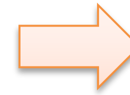
- Global variables?
- Heap-allocated data structures?

Analysis: Delegating Proof Obligations
File/Function Contracts



Primary Proof Obligations

Discharge PPOs using context sensitivity and contracts



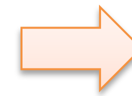
1,193,173

5,288,039

0
200000
400000
600000
800000
1000000
1200000

cairo
cleanflight
dnsmasq
dovecot
file
git
hping
irssi
lighttpd
nagios
naim
nginx
nginx114
openssl
pvm
wpa_supplicant

Primary Proof Obligations



1,193,173

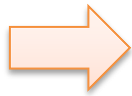
Discharge PPOs using context sensitivity and contracts (as a percent of total)

5,288,039

1.0
0.8
0.6
0.4
0.2
0.0

cairo
cleanflight
dnsmasq
dovecot
file
git
hping
irssi
lighttpd
nagios
naim
nginx
nginx114
openssl
pvm
wpa_supplicant

Primary + Supporting Proof Obligations



1,193,173	+	416,779	=	1,609,952
5,288,039	+	372,264	=	5,660303

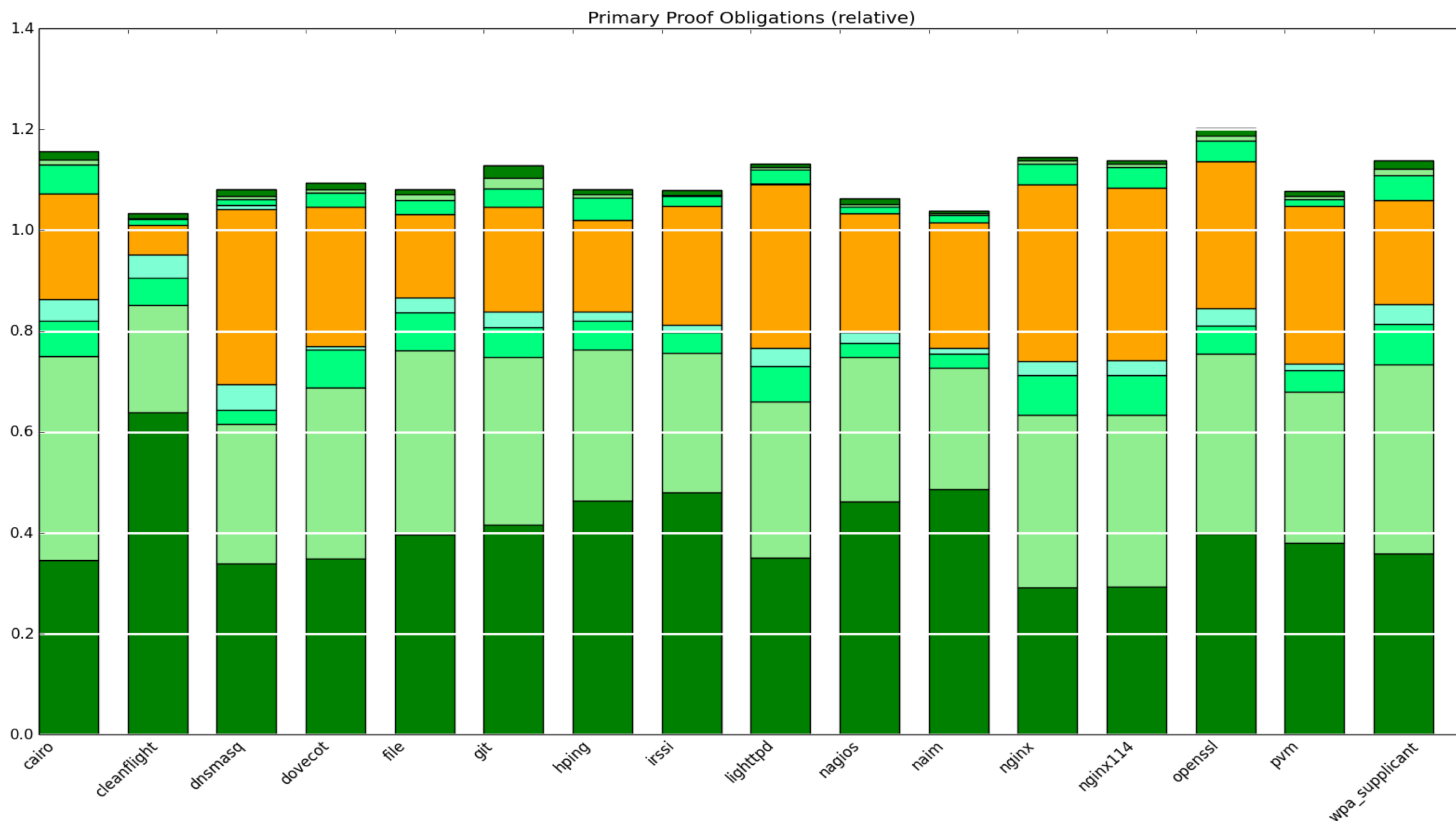
Primary Proof Obligations (relative)

1200000
1000000
800000
600000
400000
200000
0

cairo
cleanflight
dnsmasq
dovecot
file
git
hping
irssi
lighttpd
nagios
naim
nginx
nginx114
openssl
pvm
wpa_supplicant

Primary + Supporting Proof Obligations

1,193,173	+	416,779	=	1,609,952
5,288,039	+	372,264	=	5,660303



False Positives?

Our perspective: Anything that cannot be proven safe needs work:

- Additional user input (in the form of contract conditions), and/or
- Additional analysis capabilities, and/or
- Modifications to the program

Bugs?

A proof obligation is marked ‘violated’ (and closed) if

- A proof obligation is violated for all behaviors (universal), or
- An existential condition is identified that violates a proof obligation
  - Use of return value from malloc, calloc, realloc without null check
  - Use of return value from fopen, getenv, etc., without null check
  - Unchecked user input values
  - Volatile values, random values
- An existential condition outside the realm of reasoning is identified that may violate a proof obligation
  - unchecked return value from strchr, strrchr, strtol, strtoll, etc.

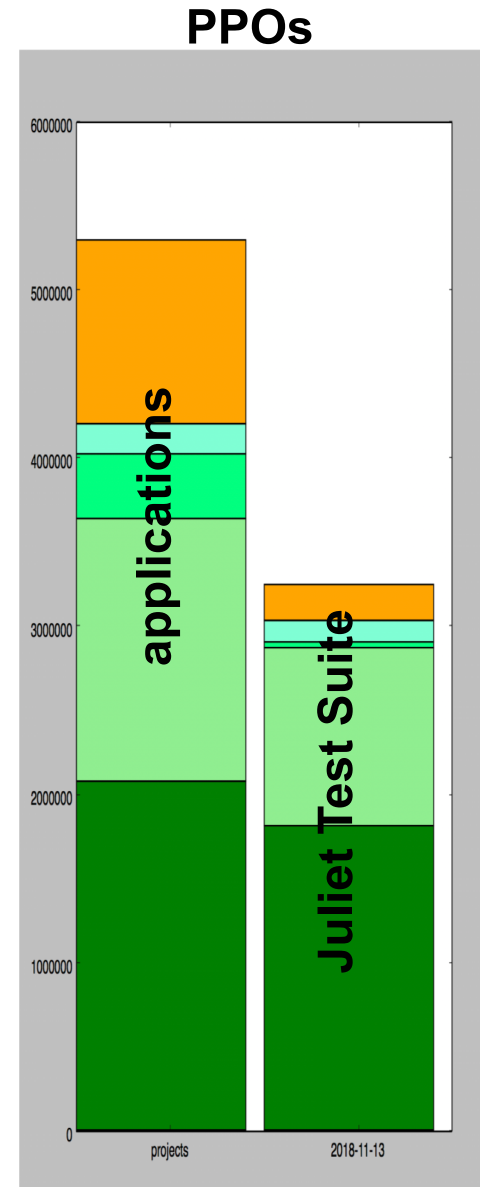
3 potentially serious memory vulnerabilities found in one of the test applications

Juliet Test Suite

- Comprehensive set of tests for wide variety of vulnerabilities
- Developed by CAS (Center for Assured Software)
- Updated and maintained by NIST
- Primary purpose: static analysis tool evaluation
  - Vulnerability coverage
  - Program construct support
- Extremely valuable for tool developers:
  - Tool validation
  - Enumeration of corner cases
  - Regression tests

But ……

not really representative of real-world applications



Juliet Test Suite: Quantitative Comparison with Applications in terms of proof obligation difficulty

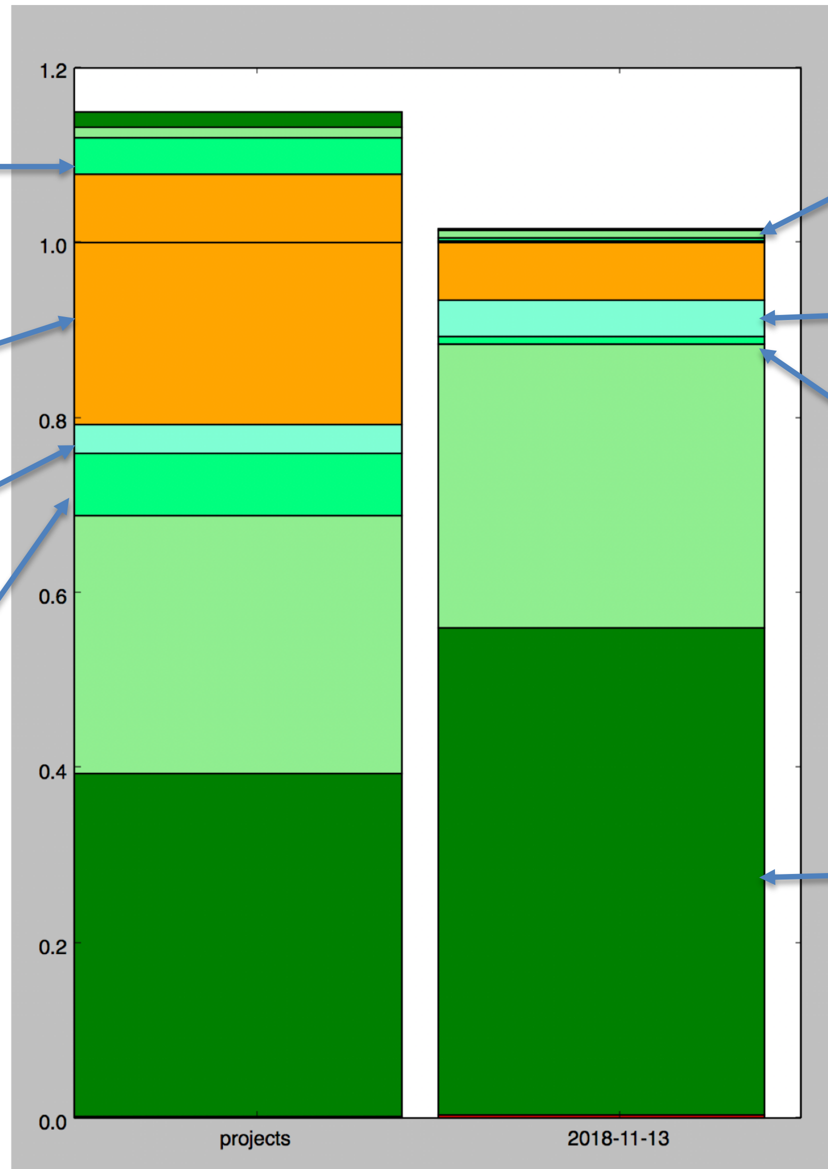
Applications

Much higher level of context sensitivity

Many more “difficult” proof obligations

Fewer contracts: Requires work

Much more API conditions



Juliet Test Suite

Hardly any context sensitivity

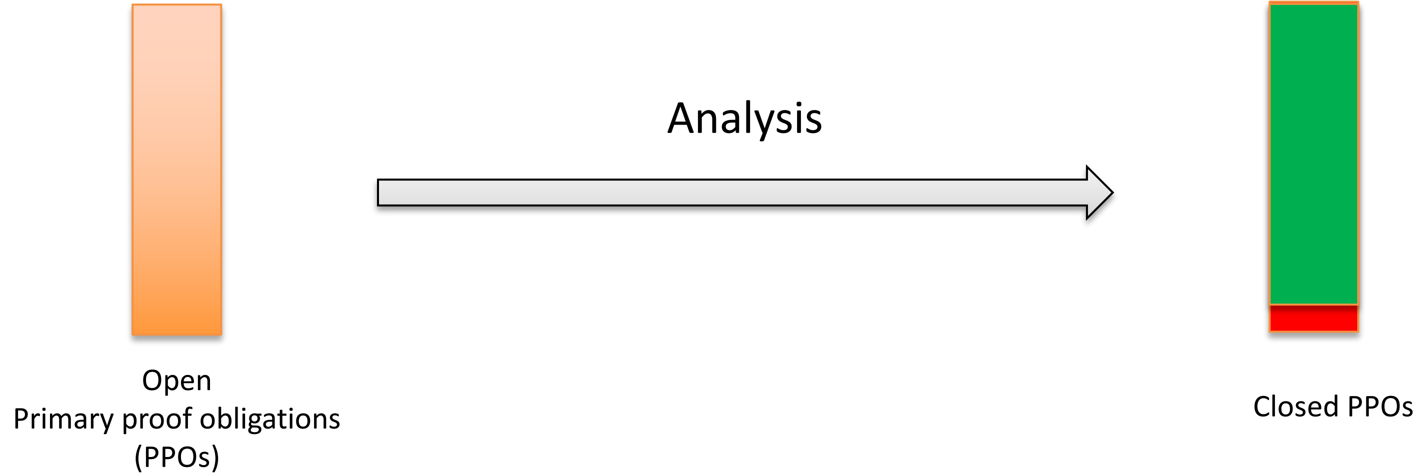
More contracts: From “template”

Hardly any API conditions

Many more “trivial” proof obligations

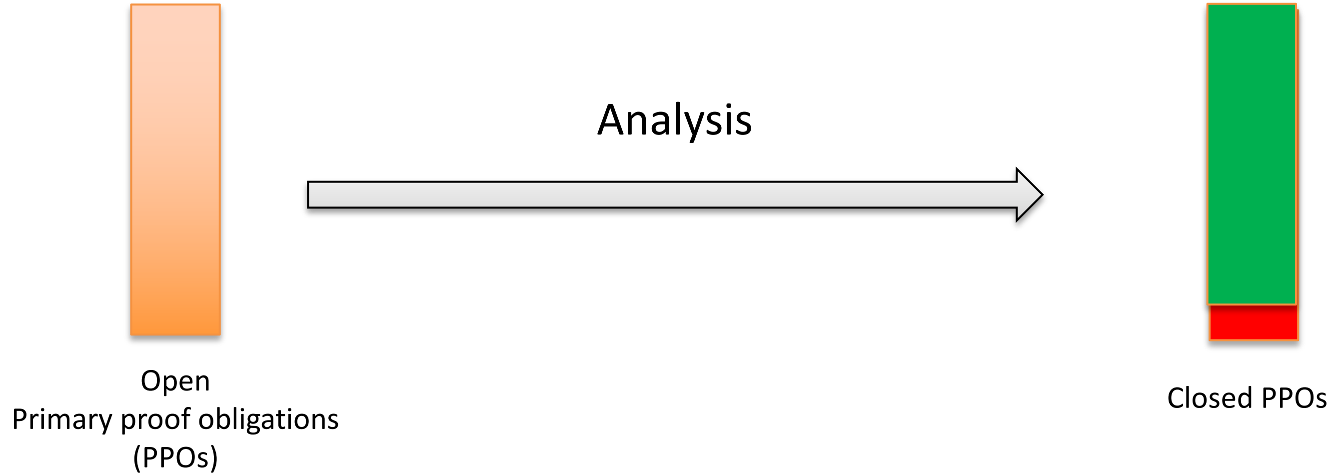
Conclusions

Our goal is:

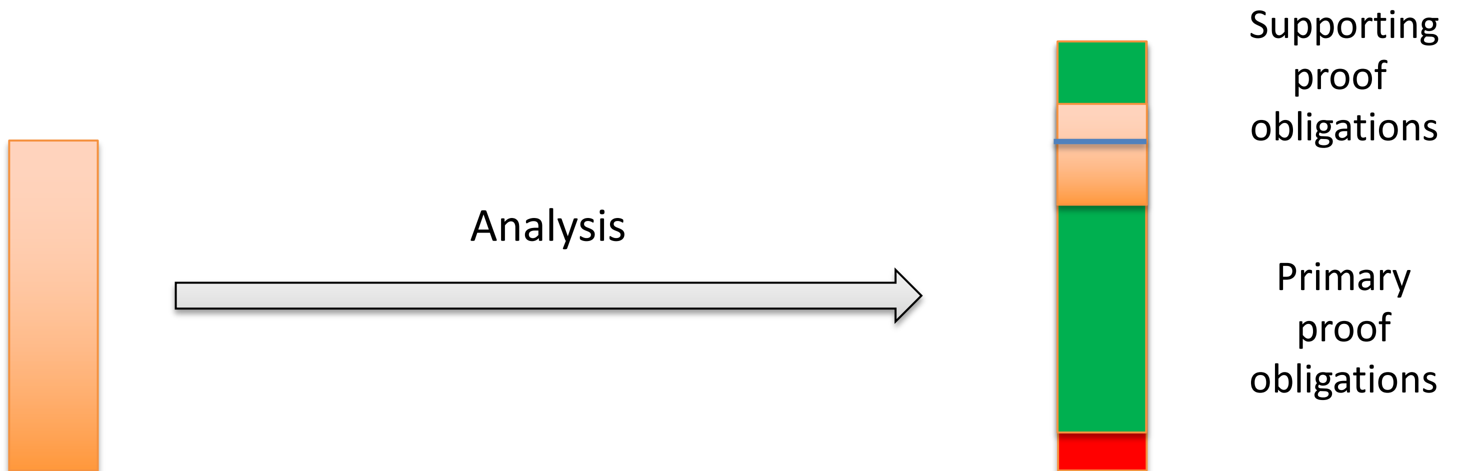


Conclusions

Our goal is:



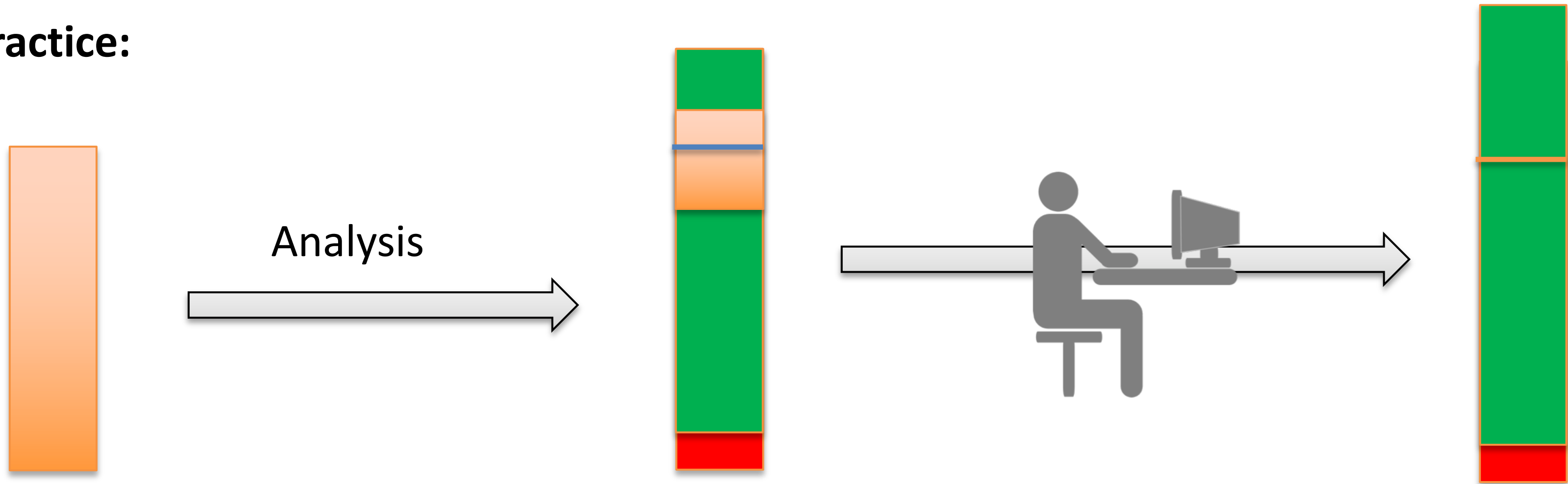
In practice:



Kestrel Technology

Conclusions

In practice:

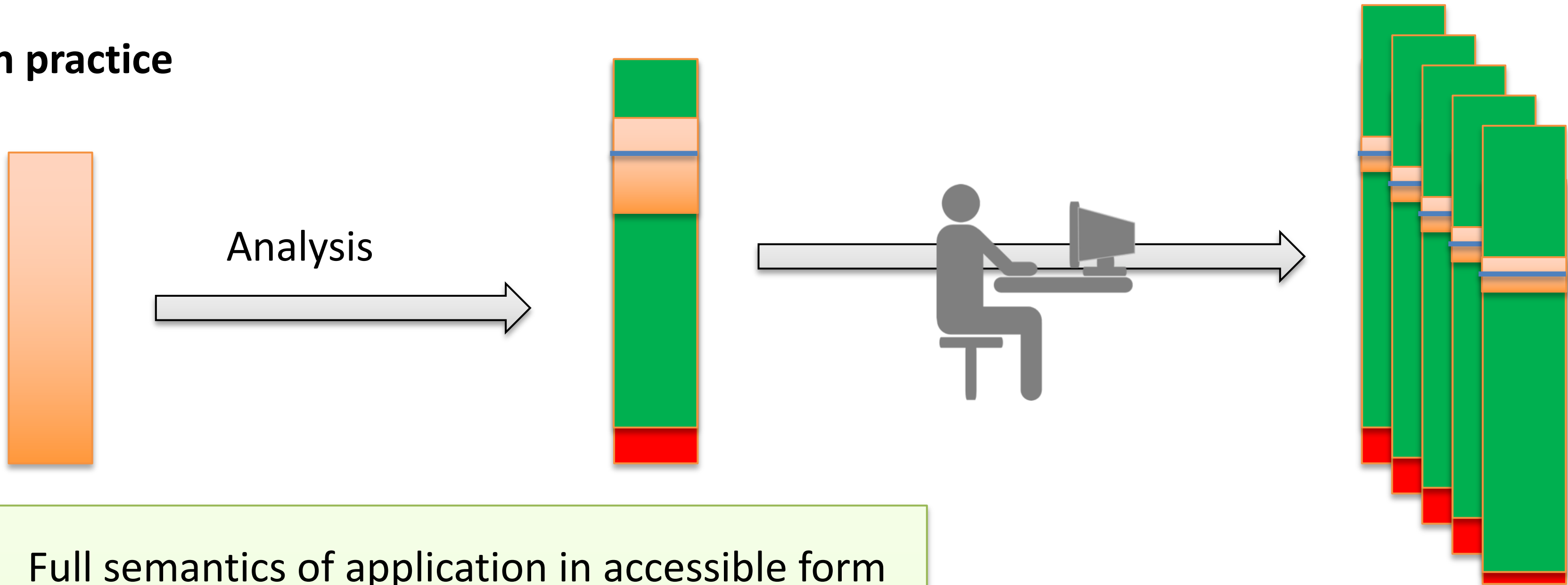


- Full semantics of application in accessible form
- Exhaustive set of proof obligations + evidence
- Function api conditions
- Invariants generated
- Programmable api in python

enables

- **Specialized analyses**
- **Incremental analysis**
- **Every result is subject to verification**
- **Modular analysis (function/file level)**
- **Clear measure of success**

Conclusions



- Full semantics of application in accessible form
- Exhaustive set of proof obligations +evidence
- Function api conditions
- Invariants generated
- Programmable api in python

Most analysis results can be reused across versions;
Assumptions can be rechecked

enables

- Specialized analyses
- Incremental analysis
- Every result is subject to verification
- Modular analysis (function/file level)
- Clear measure of success

Conclusions: What's next?

- Extend with other properties, specified by state machines
- Extend expressiveness of contract specifications
- Continuous improvement of the analyzer, increase automation, C++

……….. and eventually

For every (many) important open-source C applications:

Create an open-source community-owned exhaustive set of proof obligations with (partial) analysis results, full set of assumptions (represented as api requirements and contract conditions) that evolves with new versions created

………… and

Make sound static analysis an integral part of the open-source software development process

Conclusions: What's next?

Open-source: available on GitHub:

https://github.com/static-analysis-engineering

CodeHawk-C

CodeHawk-C-Targets-Juliet

Under MIT License

Give it a try and let us know what you think!

sipma@kestreltechnology.com

paul.black@nist.gov

THANK YOU !

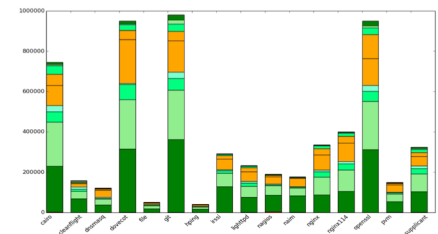
Test Applications

application	LOC
Cairo-1.14.12	227,818
Cleanflight-CLFL-v2.3.2	118,758
Dnsmasq-2.76	29,922
Dovecot-2.0.beta6 (SATE 2010)	208,636
File	14,379
Git-2.17.0	205,636
Hping	11,336
Irssi-0.8.14 (SATE 2009)	61,972
Lighttpd-1.4.18 (SATE 2008)	49,747
Nagios-2.10 (SATE 2008)	47,652
Naim-0.11.8.3.1 (SATE 2008)	25,759
Nginx-1.14.0	103,388
Nginx-1.2.9	102,151
Openssl-1.0.1.f	275,060
Pvm3.4.6 (SATE 2009)	60,029
Wpa_supplicant-2.6	96,554
Total	**1,638,797**

Primary + Supporting Proof Obligations

1,097,471 + 415,211 = 1,512,682

4,191,788 + 378,013 = 4,569,801



CWE's covered

118	Improper access of indexed resource (range error)
119	improper restriction of operations within the bound
120	Buffer copy without checking size of input (classic buffer overflow)
121	Stack-based buffer overflow
122	Heap-based buffer overflow
123	Write-what-where condition
124	Buffer underwrite
125	Out-of-bounds read
126	Buffer over-read
127	Buffer under-read
128	Wrap-around error
129	Improper validation of array index
130	Improper handling of length parameter inconsistency
131	Incorrect calculation of buffer size
135	Incorrect calculation of multi-byte string length
170	Improper null termination

CWE's covered

190	Integer Overflow or wrap-around
191	Integer Underflow or wrap-around
193	Off-by-one error
195	Signed to unsigned conversion error
196	Unsigned to signed conversion error
242	Use of inherently dangerous function (as related to memory safety)
415	Double free
416	Use after free
456	Missing initialization of variable
466	Return of pointer value outside of expected range
467	Use of sizeof() on pointer type
469	Use of pointer subtraction to determine size
476	Null pointer dereference
588	Attempt to access child of non-structure pointer
590	Free of memory not on the heap
785	Use of path manipulation function without maximum-sized buffer

CWE's covered

786	Access of memory location before start of buffer
787	Out-of-bounds write
788	Access of memory location after start of buffer
805	Buffer access with incorrect length value
822	Untrusted pointer dereference
823	Use of out-of-range pointer offset
824	Use of uninitialized pointer
825	Expired pointer dereference
839	Numeric range comparison check without maximum check
843	Access of reource using incompatible type (type confusion)
369	Divide by zero
134	Uncontrolled format string
197	Numeric truncation