galois

Continuous Static Analysis

Mike Dodds - Galois, Inc.

FMATS, Cambridge UK
24th September 2018

Who Galois are

Research and development lab of ~90 people

Locations

- Portland, OR
- Arlington, VA
- Dayton, OH

© 2017 Galois, Inc.

What Galois do

PL research meets real-world applications

Programming languages, analysis, verification, security, cryptography

Our tools

- Symbolic execution
- Model checking
- Interactive theorem provers
- Functional programming (esp. Haskell)

© 2017 Galois, Inc.

Galois model

Big research projects from US govt sources, e.g. DARPA

Commercial research projects e.g. Amazon, Facebook, others

Lots of collaborations with academic partners

Spin-out companies commercializing research tech (Tangram, MuseDev)

© 2017 Galois, Inc.

galois
galois.com/careers/

This talk

Galois background

Idea: continuous static analysis / verification

Example: Amazon TLS verification with SAW

Galois MuseDev continuous analysis spin-out

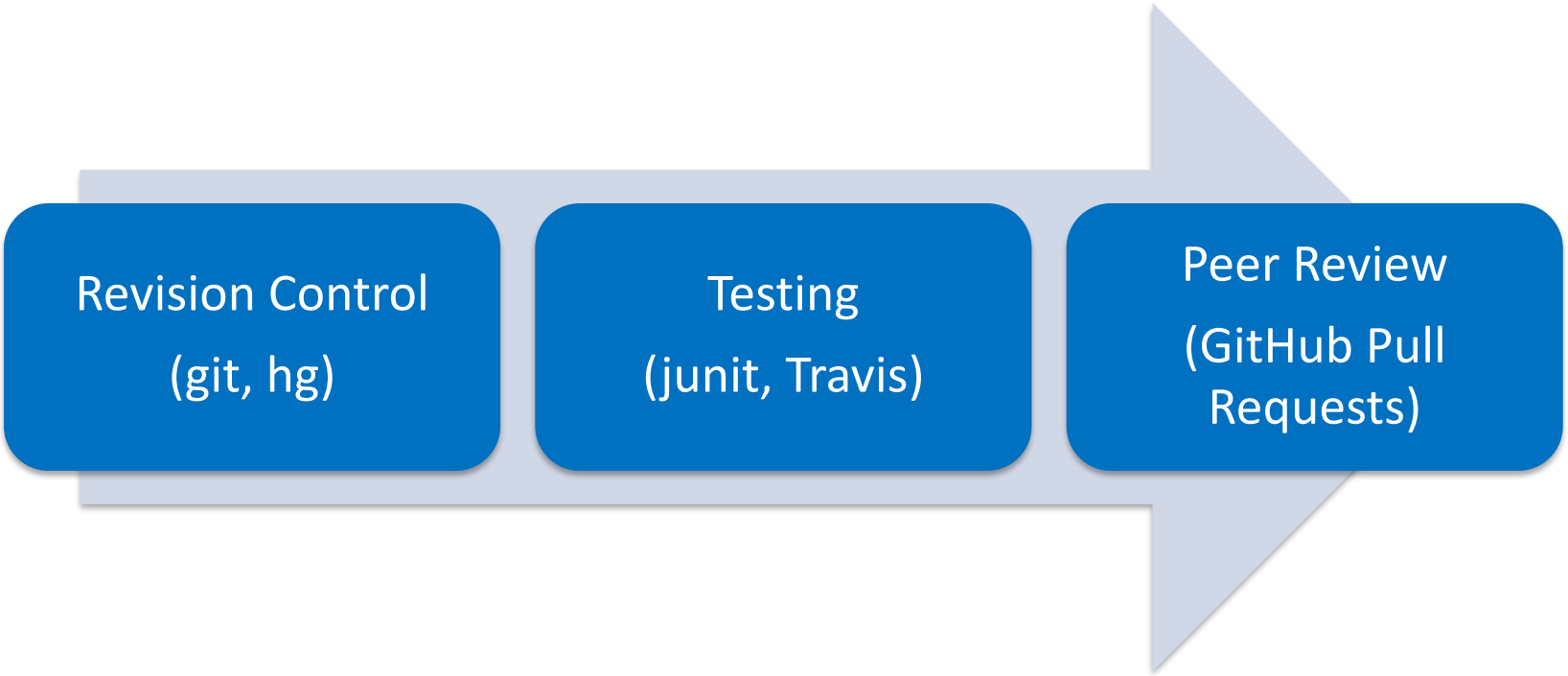
Just to note:

Galois MuseDev ≠ DARPA MUSE

It’s usually not this confusing

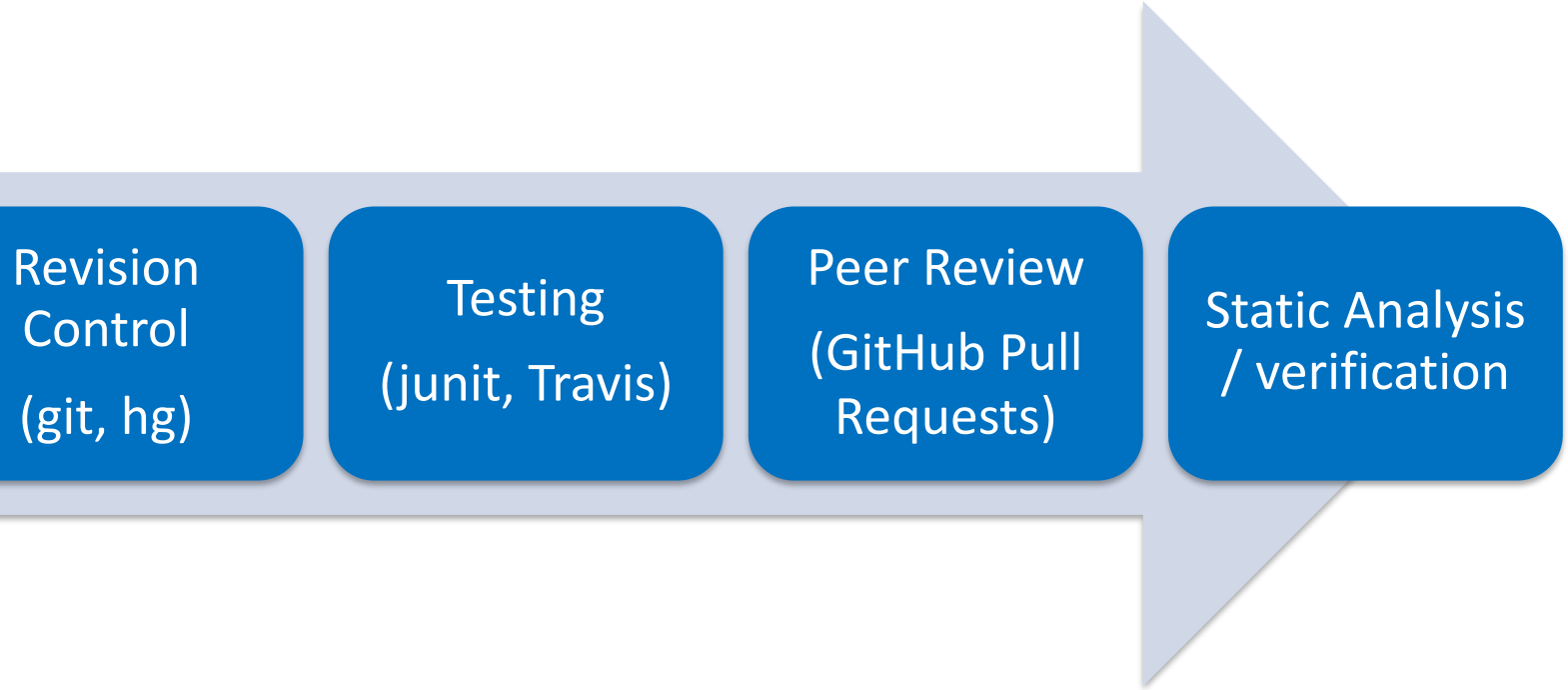
Idea: Continuous Static Analysis

Code quality strategies in industry



© 2018 Galois, Inc.

Code quality strategies in industry



© 2018 Galois, Inc.

Problems with static tools

False positives, or uninteresting bugs:

"This part of the code has been well-tested"

Too many bugs reported:

"We can't work through 1000 bug reports"

Bugs reported too late:

"We already caught this bug through QA"

Tools don't scale to industry projects

© 2017 Galois, Inc.

Continuous analysis / verification

Idea developed in FB, Google, Amazon

- Run analysis tools when code changes
- Integrate into compilation or continuous integration
- Report results immediately or at code review

Advantages:

- Reduce defects that need to be caught in QA / production
- Improve false positive rate – new code is most likely to be buggy
- Report bugs to code reviewers – it's their job to care about bugs
- Enable scalability

"Move fast and don't break things"

© 2017 Galois, Inc.

2018: continuous analysis used by

Google

- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Example tool: ErrorProne

Facebook

- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Example tool: Infer

Amazon

- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Example tool: SAW

galois

© 2018 Galois, Inc.

Example: Amazon s2n verification

Correctness of core components in Amazon's s2n TLS library.

s2n verification team

© 2018 Galois, Inc.

TLS: Transport Layer Security

TLS (newer version of SSL) provides us most of the

Confidentiality
Data-Integrity
Authentication

guarantees that we enjoy on the internet today.

© 2017 Galois, Inc.

Amazon s2n: A TLS Implementation

- Inspired by TLS vulnerabilities discovered by researchers in other implementations.
- Written with security and performance as primary goals.
- Drops some arguably insecure/less secure features.
  - Result: Much smaller, clearer, more auditable code.
  - OpenSSL TLS is 70k lines of C code.
  - s2n is only 6k.
- Used in production at Amazon Web Services.

© 2017 Galois, Inc.

HMAC: A Component of TLS

- keyed-Hash Message Authentication Code
- Provides a signature for a message that confirms:
  - Authenticity: the message was signed by the expected sender
  - Integrity: the message has not been modified

derived from key

message

$$\text{HMAC}(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))$$

arbitrary hash function

constants from NIST

- Code is still complex
  - 521 lines of C code

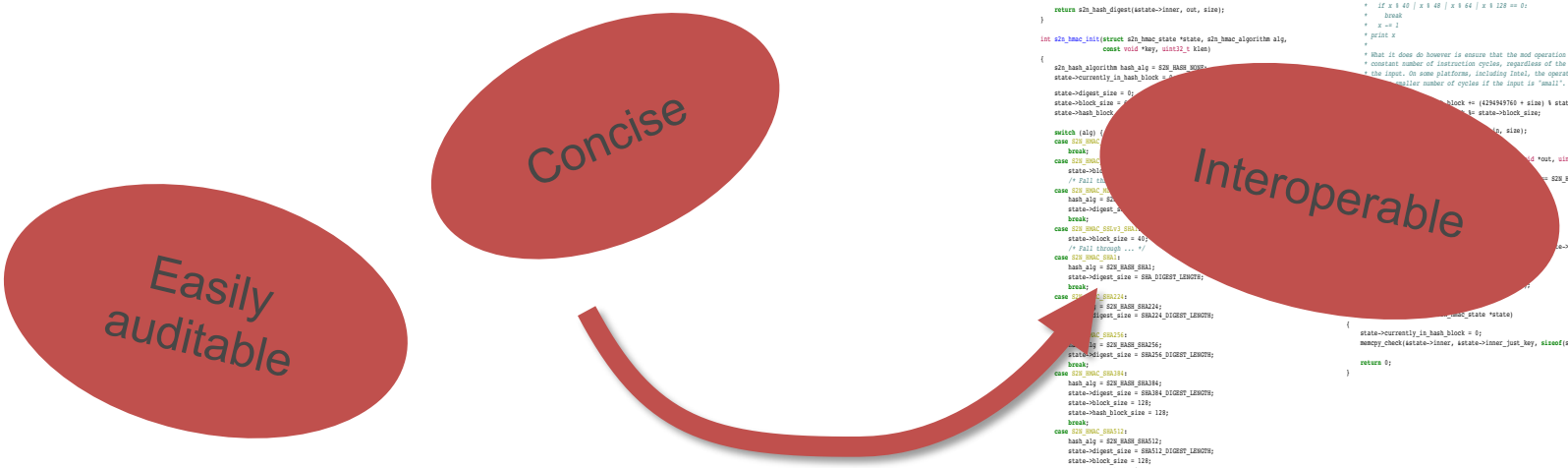
© 2017 Galois, Inc.

HMAC Specification

C HMAC

HMAC(K, m) =

H((K ⊕ opad) ll H((K ⊕ ipad) ll m))



Goal: bridge this gap

© 2017 Galois, Inc.

s2n verification vs HACL*

HACL*:

- Aim: end-to-end verification of the whole implementation
- Generated C code from high-level F* implementation
- Integrate synthesized code into projects

Galois s2n work:

- Aim: verify high-risk portions of code to raise assurance
- Apply to Amazon's hand-optimized C code
- Proofs rerun as C code changes

© 2017 Galois, Inc.

Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

© 2017 Galois, Inc.

Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

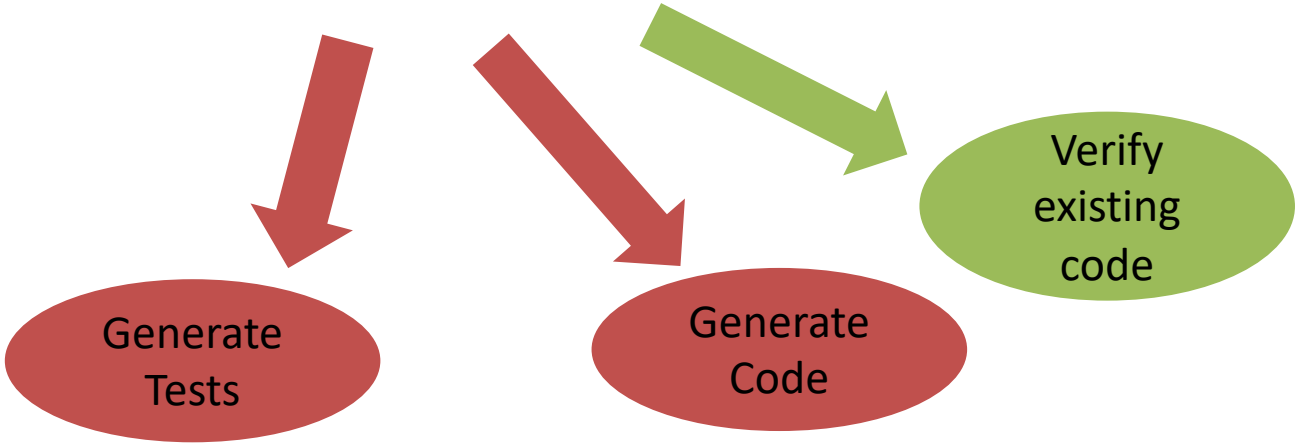
© 2017 Galois, Inc.

Summary of Approach

$$\text{HMAC}(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))$$

Step 1: Capture this specification in a formal language (we used Cryptol - https://cryptol.net/)

```
hmac k message =
   H((k ^ opad) # H((k ^ ipad) # message))
```



© 2017 Galois, Inc.

HMAC Specification

C HMAC

```
hmac k message =

    H((k ^ opad) # H((k ^ ipad) # message))
```

Easily auditable

Concise

Fast

Interoperable

Goal: bridge this gap

© 2017 Galois, Inc.

Summary of Approach

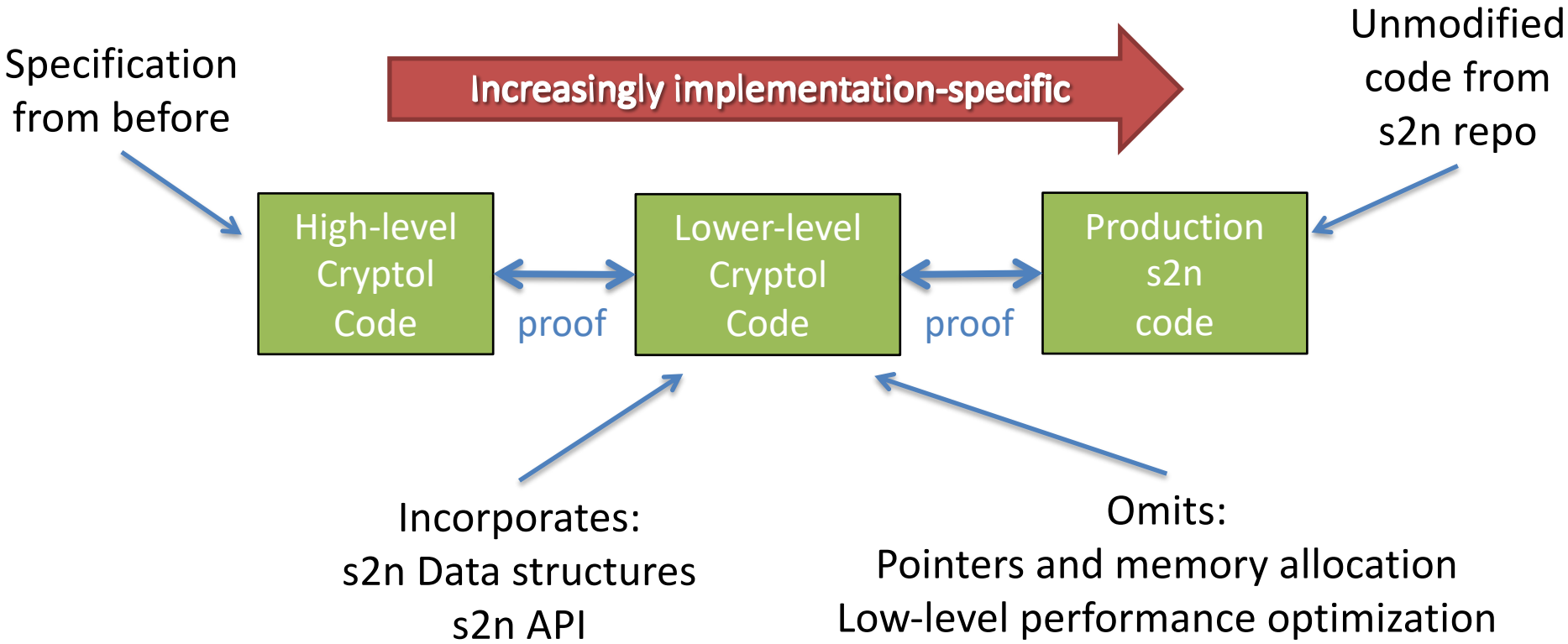
1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

© 2017 Galois, Inc.

Bridging the gap

Solution: Layers of Abstraction



© 2017 Galois, Inc.

Summary of Approach

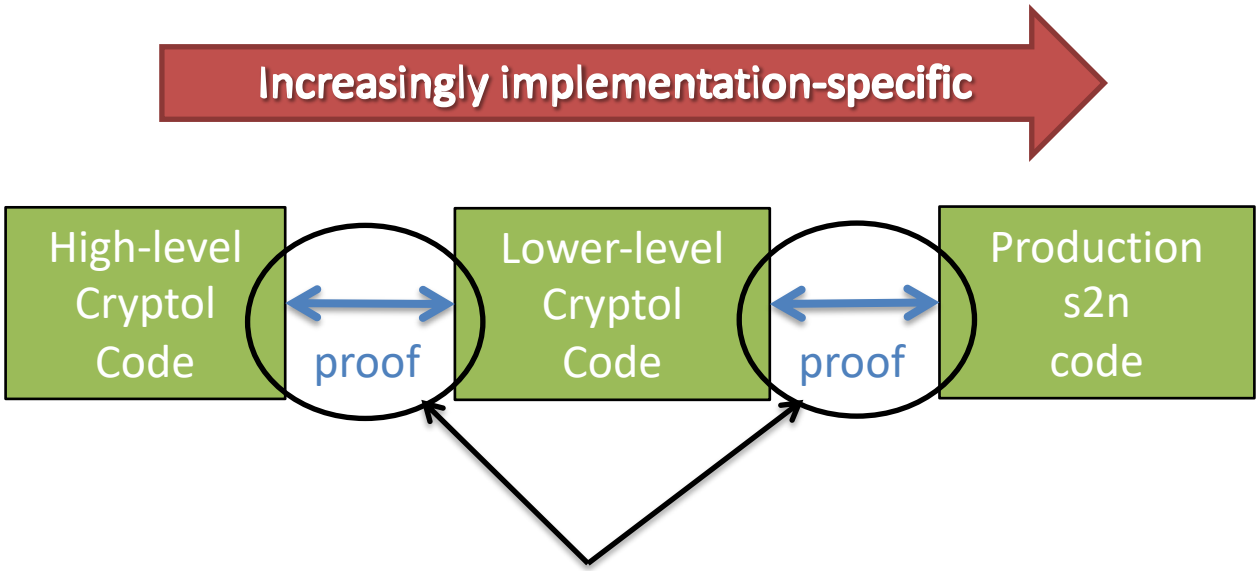
1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between Specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

© 2017 Galois, Inc.

Bridging the gap

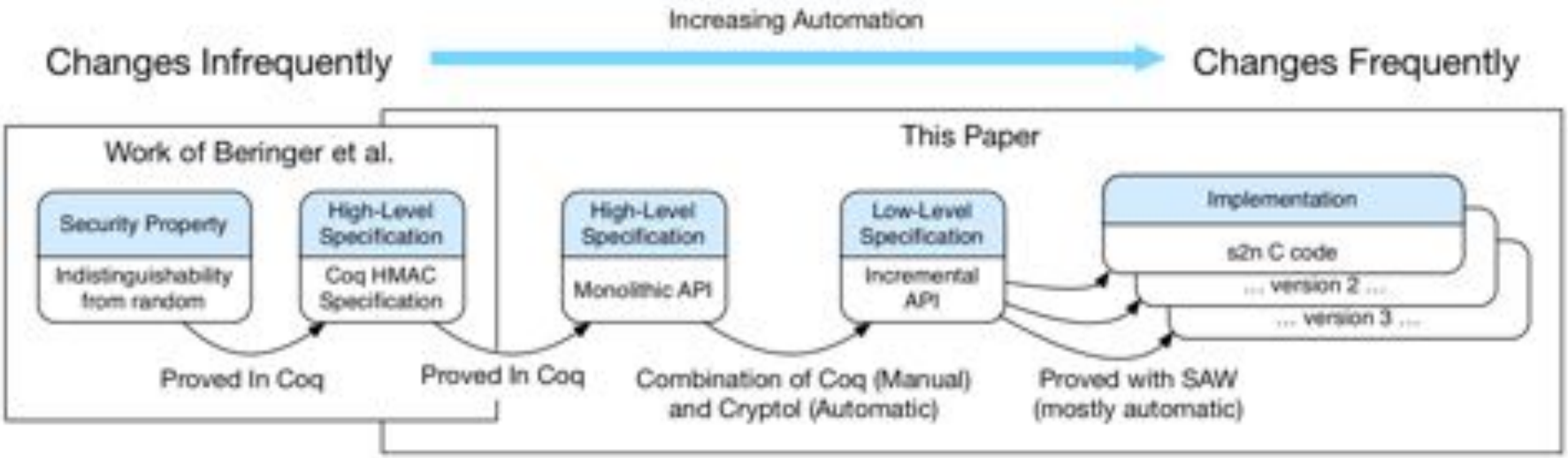
Solution: Layers of Abstraction



Automatically Constructed by SAW
(Software Analysis Workbench)
via translation to SMT and application of constraint solvers

© 2017 Galois, Inc.

Verified HMAC pipeline



© 2017 Galois, Inc.

Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between Specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

© 2017 Galois, Inc.

Continuous Integration

- Proofs run automatically on code changes
  - Proof failure is a build failure
- Proof is independent of exact C code, depends only on:
  - Interfaces (arguments and struct layouts)
  - Function call structure
- Proof is easily adapted:
  - Function body changes → likely no proof changes
  - Interface changes → similarly-sized proof changes
  - Call structure changes → tiny proof changes

© 2017 Galois, Inc.

Travis CI

Travis CI Blog Status Help Sign in with GitHub

awslabs / s2n build passing

Current Branches Build History Pull Requests Build #953 More options

✓ master Merge pull request #517 from xonatius/allocator_overrides

Added guards around allocator overrides

Commit 02ade5e

Compare 9eb9b99..02ade5e

Branch master

Matthew Baldwin authored GitHub committed

#953 passed

Ran for 1 hr 13 min 8 sec

Total time 4 hrs 54 min 44 sec

20 days ago

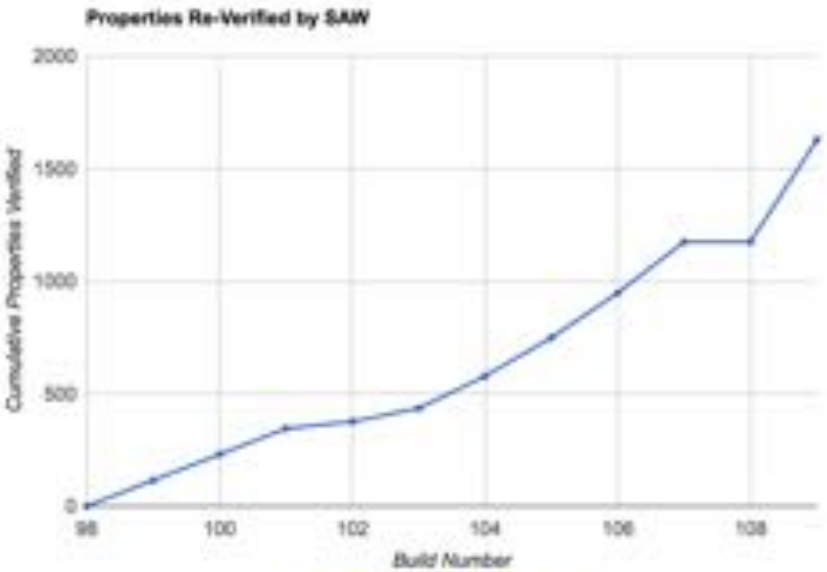
Build Jobs

✓ # 953.1 Xcode: xcode8 C TESTS=ctverif 4 min 59 sec

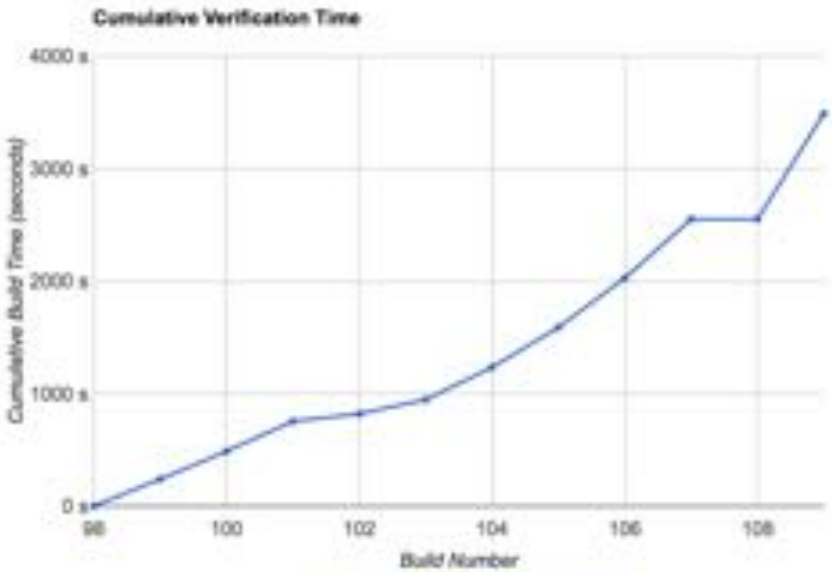
© 2017 Galois, Inc.

Proof Metrics

We have run 12 builds on branch demo and verified 1629 properties. To gain equivalent assurance through test cases we would need to run 2.4e130 tests.



Change Axis to Build Date



Change Axis to Build Date

Run Summary:

For all builds we ran 1629 verifications.

Name	Function	Build	Size	Equivalent Tests	Complexity*	Time	Succ
hmac_c_state_correct size 0		109	0	1		0	✓
hmac_c_state_correct size 1		109	1	2		0	✓
hmac_c_state_correct size 128		109	128	3.40e38		0	✓
MD5, key size = 64, msg size = 1	s2n_hmac_update	109	65	3.68e19	56,865	3.451	✓
MD5, key size = 64, msg size = 1	s2n_hmac_digest	109	65	3.68e19	2,972	0.859	✓

Number of SAW verifications this week : 0
Number of SAW verifications all time : 1629
Number of failed SAW checks this week : 0
Number of failed SAW checks all time : 0
Average successful SAW runtime this week : No recorded times
Average successful SAW runtime all time : 2.1 seconds
Average LOC covered by SAW reasoning this week : 103
Average LOC covered by SAW reasoning all time : 103
Average LOC of specification covered by SAW reasoning this week : 787

© 2017 Galois, Inc.

Example: Protocol Correctness

Correct implementation of authentication and key exchange.

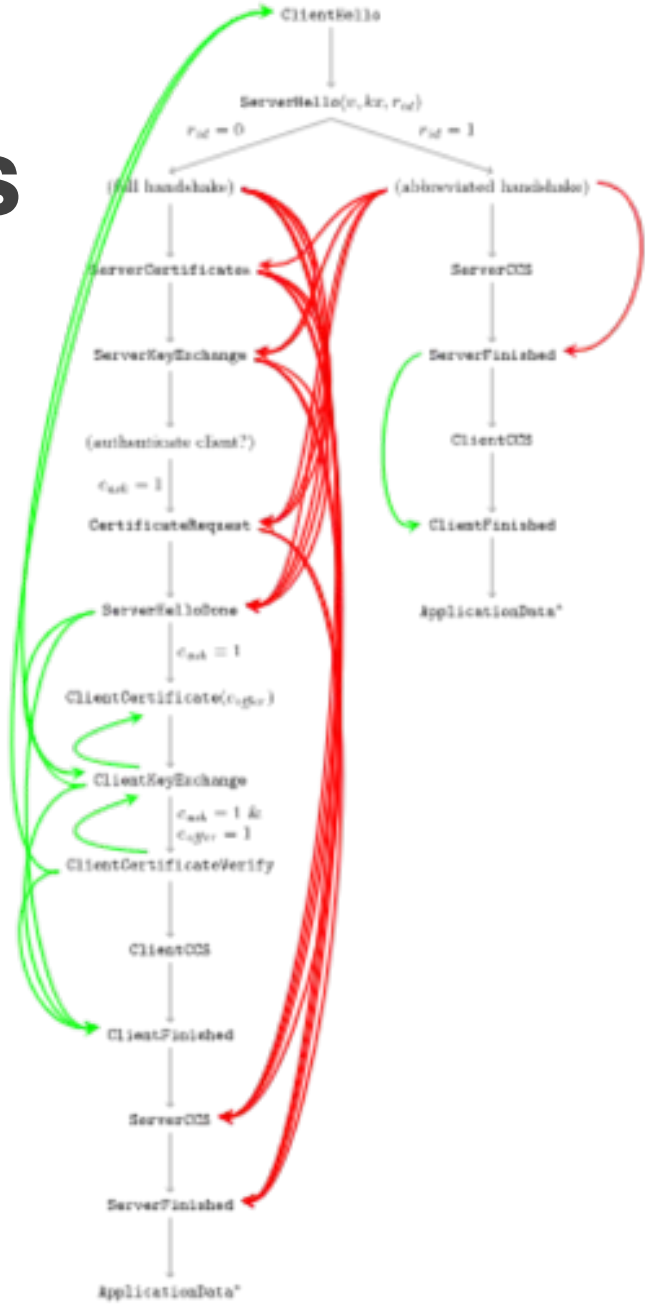
Protocol Correctness

- Previous work targeted correctness of underlying crypto.
- The protocol level is also security critical (and sometimes wrong)

© 2017 Galois, Inc.

State Machine Attacks

- Unexpected message ordering in other implementations causes authentication steps to be bypassed.



From https://www.mitls.org/pages/attacks/SMACK

© 2017 Galois, Inc.

Protocol State Machine

C Implementation

```c
static int s2n_sslv3_mac_init(struct s2n_hmac_state *state,
                              s2n_hmac_algorithm alg, const void *key,
                              uint32_t klen)
{
    s2n_hash_algorithm hash_alg = S2N_HASH_NONE;

    if (alg == S2N_HMAC_SSLv3_MD5) {
        hash_alg = S2N_HASH_MD5;
    }
    if (alg == S2N_HMAC_SSLv3_SHA1) {
        hash_alg = S2N_HASH_SHA1;
    }

    for (int i = 0; i < state->block_size; i++) {
        state->xor_pad[i] = 0x36;
    }

    GUARD(s2n_hash_init(&state->inner_just_key, hash_alg));
    GUARD(s2n_hash_update(&state->inner_just_key, key, klen));
    GUARD(s2n_hash_update(&state->inner_just_key, state->xor_pad,
                          state->block_size));

    for (int i = 0; i < state->block_size; i++) {
        state->xor_pad[i] = 0x5c;
    }

    GUARD(s2n_hash_init(&state->outer, hash_alg));
    GUARD(s2n_hash_update(&state->outer, key, klen));
    GUARD(s2n_hash_update(&state->outer, state->xor_pad, state->block_size));

    /* Copy inner_just_key to inner */
    return s2n_hmac_reset(state);
}

static int s2n_sslv3_mac_digest(struct s2n_hmac_state *state, void *out,
                                uint32_t size)
{
    for (int i = 0; i < state->block_size; i++) {
        state->xor_pad[i] = 0x5c;
    }

    GUARD(s2n_hash_digest(&state->inner, state->digest_pad,
                          state->digest_size));
    memcpy_check(&state->inner, &state->outer, sizeof(state->inner));
    GUARD(s2n_hash_update(&state->inner, state->digest_pad,
                          state->digest_size));

    return s2n_hash_digest(&state->inner, out, size);
}

int s2n_hmac_init(struct s2n_hmac_state *state, s2n_hmac_algorithm alg,
                  const void *key, uint32_t klen)
{
    s2n_hash_algorithm hash_alg = S2N_HASH_NONE;
    state->currently_in_hash_block = 0;

    state->digest_size = 0;
    state->block_size = 64;
    state->hash_block_size = 64;

    switch (alg) {
    case S2N_HMAC_NONE:
        break;
    case S2N_HMAC_SSLv3_MD5:
        state->block_size = 48;
        /* Fall through ... */
    case S2N_HMAC_MD5:
        hash_alg = S2N_HASH_MD5;
        state->digest_size = MD5_DIGEST_LENGTH;
        break;
    case S2N_HMAC_SSLv3_SHA1:
        state->block_size = 40;
        /* Fall through ... */
    case S2N_HMAC_SHA1:
        hash_alg = S2N_HASH_SHA1;
        state->digest_size = SHA_DIGEST_LENGTH;
        break;
    case S2N_HMAC_SHA224:
        hash_alg = S2N_HASH_SHA224;
        state->digest_size = SHA224_DIGEST_LENGTH;
        break;
    case S2N_HMAC_SHA256:
        hash_alg = S2N_HASH_SHA256;
        state->digest_size = SHA256_DIGEST_LENGTH;
        break;
    case S2N_HMAC_SHA384:
        hash_alg = S2N_HASH_SHA384;
        state->digest_size = SHA384_DIGEST_LENGTH;
        state->block_size = 128;
        state->hash_block_size = 128;
        break;
    case S2N_HMAC_SHA512:
        hash_alg = S2N_HASH_SHA512;
        state->digest_size = SHA512_DIGEST_LENGTH;
        state->block_size = 128;
        state->hash_block_size = 128;
        break;
    default:
        S2N_ERROR(S2N_ERR_HMAC_INVALID_ALGORITHM);
    }

    gte_check(sizeof(state->xor_pad), state->block_size);
    gte_check(sizeof(state->digest_pad), state->digest_size);

    state->alg = alg;

    if (alg == S2N_HMAC_SSLv3_SHA1 || alg == S2N_HMAC_SSLv3_MD5) {
        return s2n_sslv3_mac_init(state, alg, key, klen);
    }

    GUARD(s2n_hash_init(&state->inner_just_key, hash_alg));
    GUARD(s2n_hash_init(&state->outer, hash_alg));

    uint32_t copied = klen;
    if (klen > state->block_size) {
        GUARD(s2n_hash_update(&state->outer, key, klen));
        GUARD(s2n_hash_digest(&state->outer, state->digest_pad,
                              state->digest_size));

        memcpy_check(state->xor_pad, state->digest_pad, state->digest_size);
        copied = state->digest_size;
    } else {
        memcpy_check(state->xor_pad, key, klen);
    }

    for (int i = 0; i < copied; i++) {
        state->xor_pad[i] ^= 0x36;
    }
    for (int i = copied; i < state->block_size; i++) {
        state->xor_pad[i] = 0x36;
    }

    GUARD(s2n_hash_update(&state->inner_just_key, state->xor_pad,
                          state->block_size));

    /* 0x36 xor 0x5c == 0x6a */
    for (int i = 0; i < state->block_size; i++) {
        state->xor_pad[i] ^= 0x6a;
    }

    return s2n_hmac_reset(state);
}

int s2n_hmac_update(struct s2n_hmac_state *state, const void *in, uint32_t size)
{
    /* Keep track of how much of the current hash block is full
     *
     * Why the 4294949760 constant in this code? 4294949760 is the
     * highest 32-bit value that is congruent to 0 modulo all of our
     * HMAC block sizes, that is also at least 16k smaller than 2^32. It
     * therefore has no effect on the mathematical result, and no valid
     * record size can cause it to overflow.
     *
     * The value was found with the following python code:
     *
     * x = (2 ** 32) - (2 ** 14)
     * while True:
     *   if x % 40 | x % 48 | x % 64 | x % 128 == 0:
     *       break
     *   x -= 1
     * print x
     *
     * What it does do however is ensure that the mod operation takes a
     * constant number of instruction cycles, regardless of the size of
     * the input. On some platforms, including Intel, the operation can
     * take a smaller number of cycles if the input is "small".
     */
    state->currently_in_hash_block += (4294949760 + size) % state->hash_block_size;
    state->currently_in_hash_block %= state->block_size;

    return s2n_hash_update(&state->inner, in, size);
}

int s2n_hmac_digest(struct s2n_hmac_state *state, void *out, uint32_t size)
{
    if (state->alg == S2N_HMAC_SSLv3_SHA1 || state->alg == S2N_HMAC_SSLv3_MD5) {
        return s2n_sslv3_mac_digest(state, out, size);
    }

    GUARD(s2n_hash_digest(&state->inner, state->digest_pad,
                          state->digest_size));
    GUARD(s2n_hash_reset(&state->outer));
    GUARD(s2n_hash_update(&state->outer, state->xor_pad, state->block_size));
    GUARD(s2n_hash_update(&state->outer, state->digest_pad,
                          state->digest_size));

    return s2n_hash_digest(&state->outer, out, size);
}

int s2n_hmac_reset(struct s2n_hmac_state *state)
{
    state->currently_in_hash_block = 0;
    memcpy_check(&state->inner, &state->inner_just_key, sizeof(state->inner));

    return 0;
}
```

SAW / Cryptol proof bridging the gap

© 2017 Galois, Inc.

Other crypto work

- **Other SAW / Cryptol verification projects:**
  - DRBG: Deterministic Random Bit Generator: The main source of cryptographic randomness (see paper at CAV'18)
  - AES / SHA256 / ECDSA verification
  - Synthesizing verified hardware crypto
- **Working on Facebook Fizz TLS1.3 library**

© 2017 Galois, Inc.

MuseDev

A Galois spin-out company focusing on continuous analysis

As a reminder:

Galois MuseDev ≠ DARPA MUSE

Although Suresh is involved with both projects

Lots of amazing tools available for free!

ErrorProne

Prevent
error patterns

Infer

Catch
deep defects

SAW

Prove correctness of
security primitives

AFL

Detect and rank
errors

and others

Analyzing billions of lines of code every day at

Google **Amazon** **Facebook**

© 2017 Galois, Inc.

Problem: tools need integration

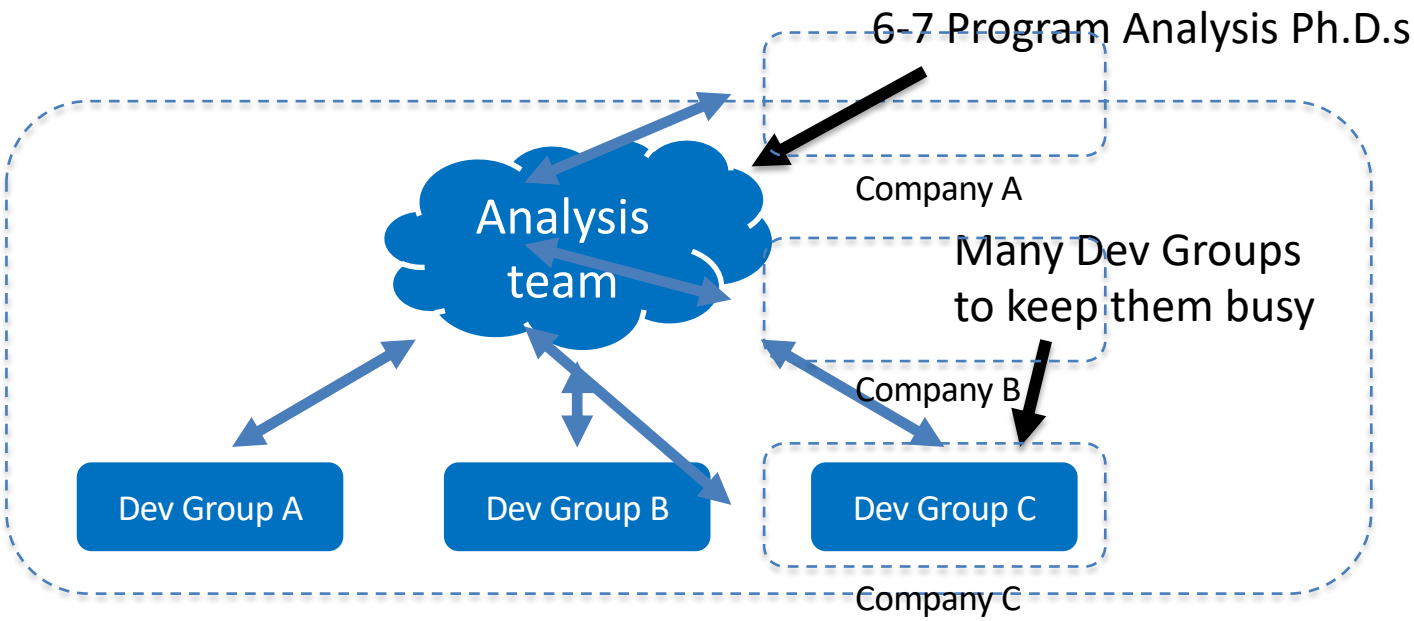
Integration into devops toolchain is a major challenge:

1. Build system *(eg. Basel, Make, Buck, Gradle)*
2. CI system *(eg. TravisCI, Jenkins)*
3. Version control *(eg. GitHub, BitBucket, GitLab)*
4. Code review *(eg. GitHub Pull Requests)*
5. Data analysis and developer feedback *(custom solutions)*

Integrations are typically closed source – tied to internal architecture.

© 2017 Galois, Inc.

The Facebook/Google/Amazon model



© 2017 Galois, Inc. Proprietary Information

MuseDev, our platform for continuous analysis

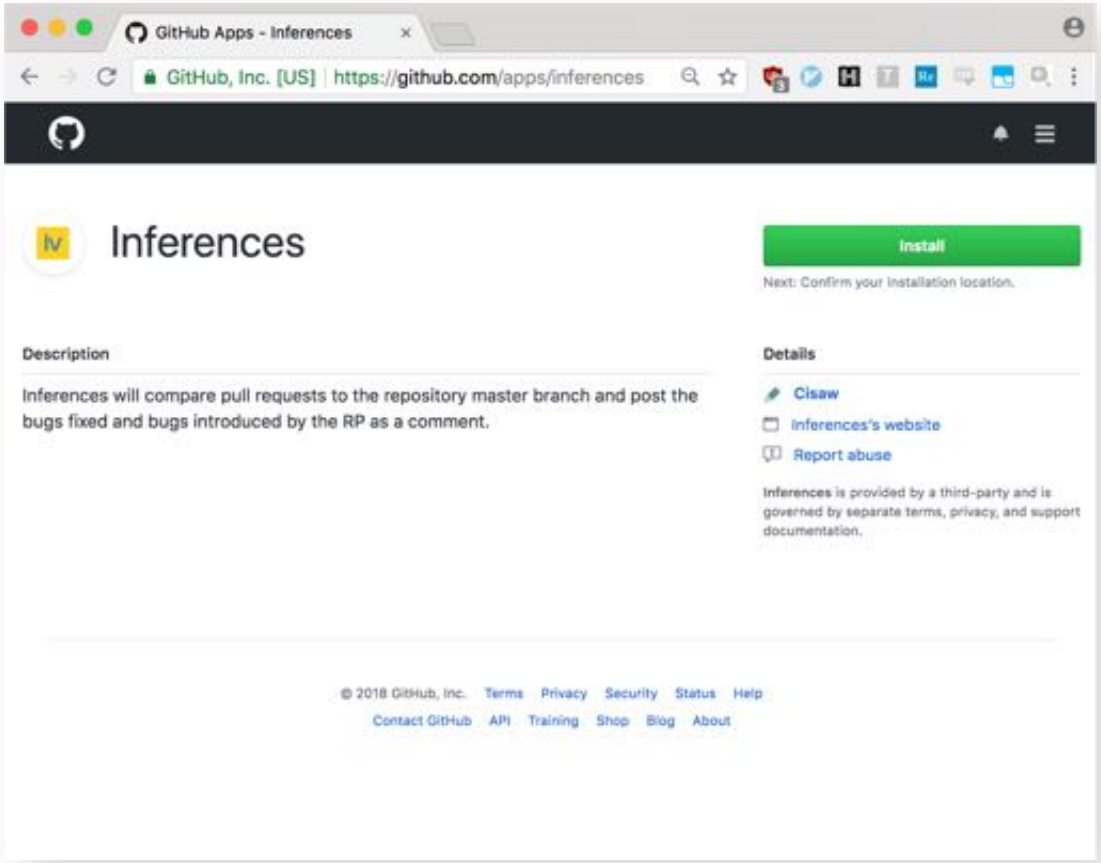
- Integrate into multiple devops toolchains
- Combine data from
  - Multiple static tools (currently: ErrorProne, Infer)
  - Repository history (GitHub, BitBucket)
  - Developer feedback
- Present information at code review, as patch comments (GitHub PR, Bitbucket)

"Static analysis as a service"

© 2017 Galois, Inc.

Tool walkthrough

Example MuseDev workflow integrating Facebook Infer into GitHub.



© 2017 Galois, Inc.

Tool walkthrough

TomMD / libacvp
forked from cisco/libacvp

Unwatch

Code | Pull requests 1 | Projects 0 | Wiki | Insights | Settings

Mirror of cisco libacvp#62 #4

Open TomMD wants to merge 4 commits into `pr62-dst` from `pr62-src`

Conversation 3 | Commits 4 | Checks 0 | Files changed 10

TomMD commented 2 days ago — Owner

Mirror of cisco libacvp#62

- This hasn't been fully tested with the NIST server since I haven't been able to successfully retrieve a vector set
- Adds the library side for ECDSA
- Has a stub for the ECDSA handler in the app code

elliemdaw added some commits on Apr 9

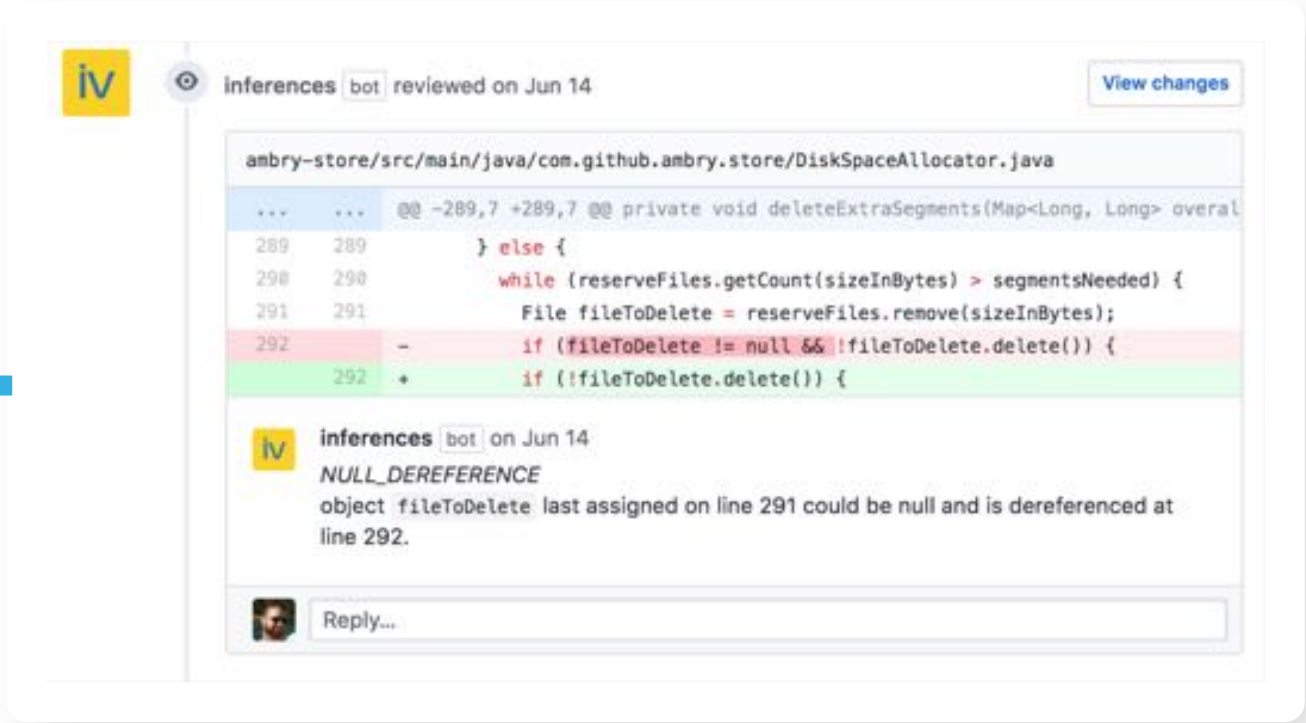
Adds ECDSA support to example app	c68f238
Adds new file and updates Makefiles	a5432be
Adds ECDSA library support	7198e67
Adds better logging and updates minor PR changes	2d4fa9e

Step 1: create a pull request in a monitored repository.

© 2017 Galois, Inc.

Tool walkthrough

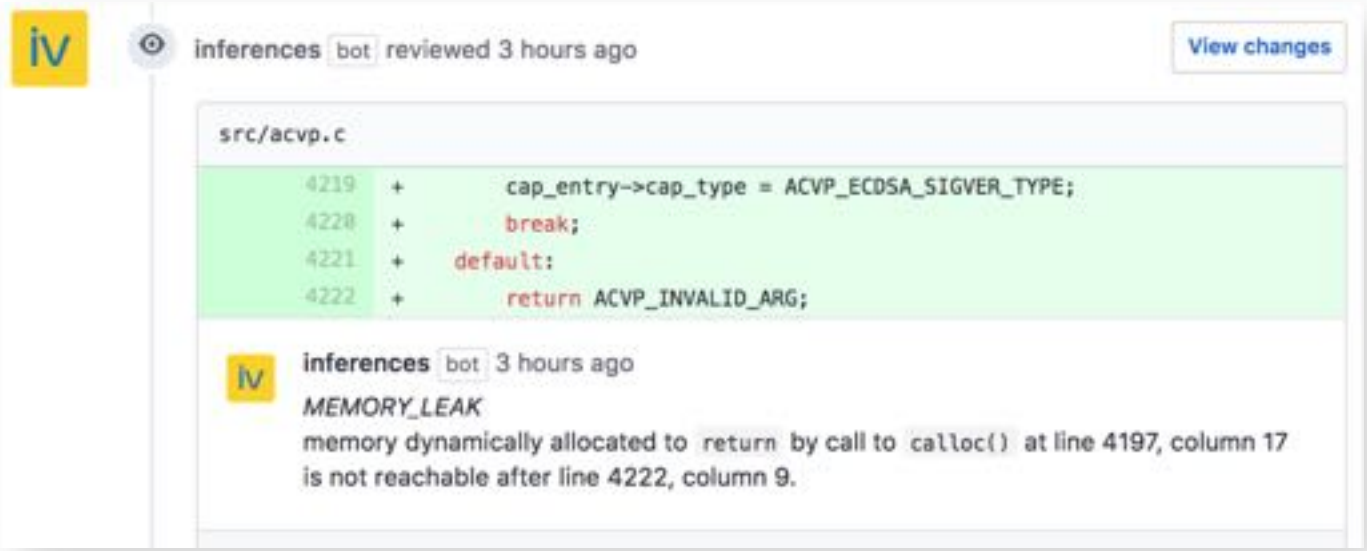
Step 2: MuseDev runs static tools on the patch. Errors are annotated on the pull request



© 2017 Galois, Inc.

Tool walkthrough

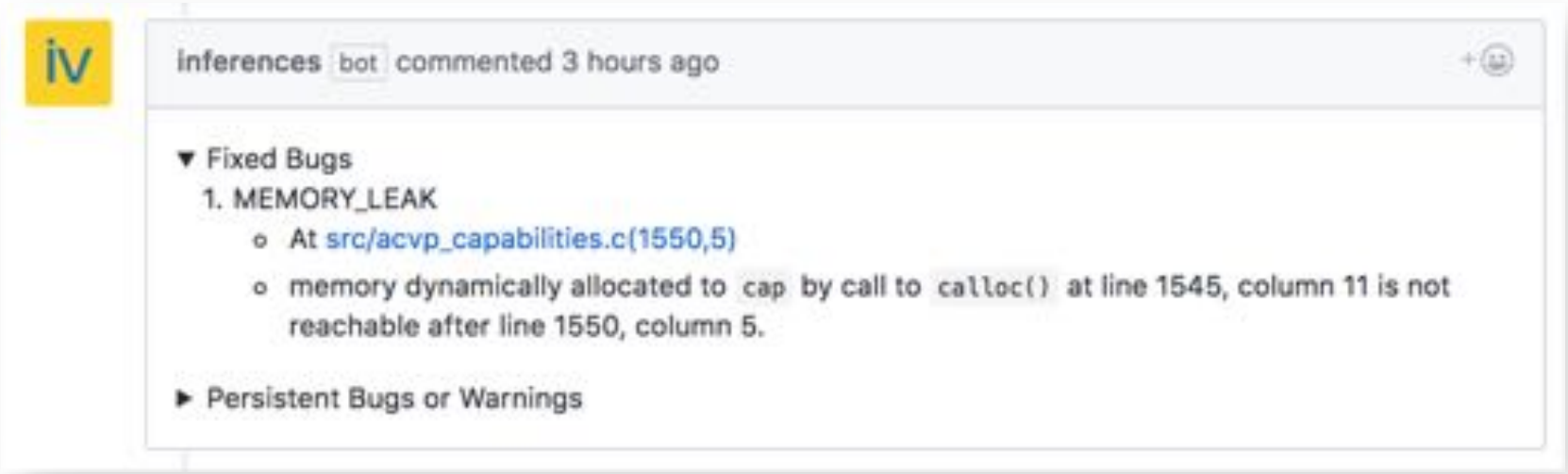
MuseDev reports a memory error derived from Infer



© 2017 Galois, Inc.

Tool walkthrough

MuseDev keeps track of how bug persist in the codebase



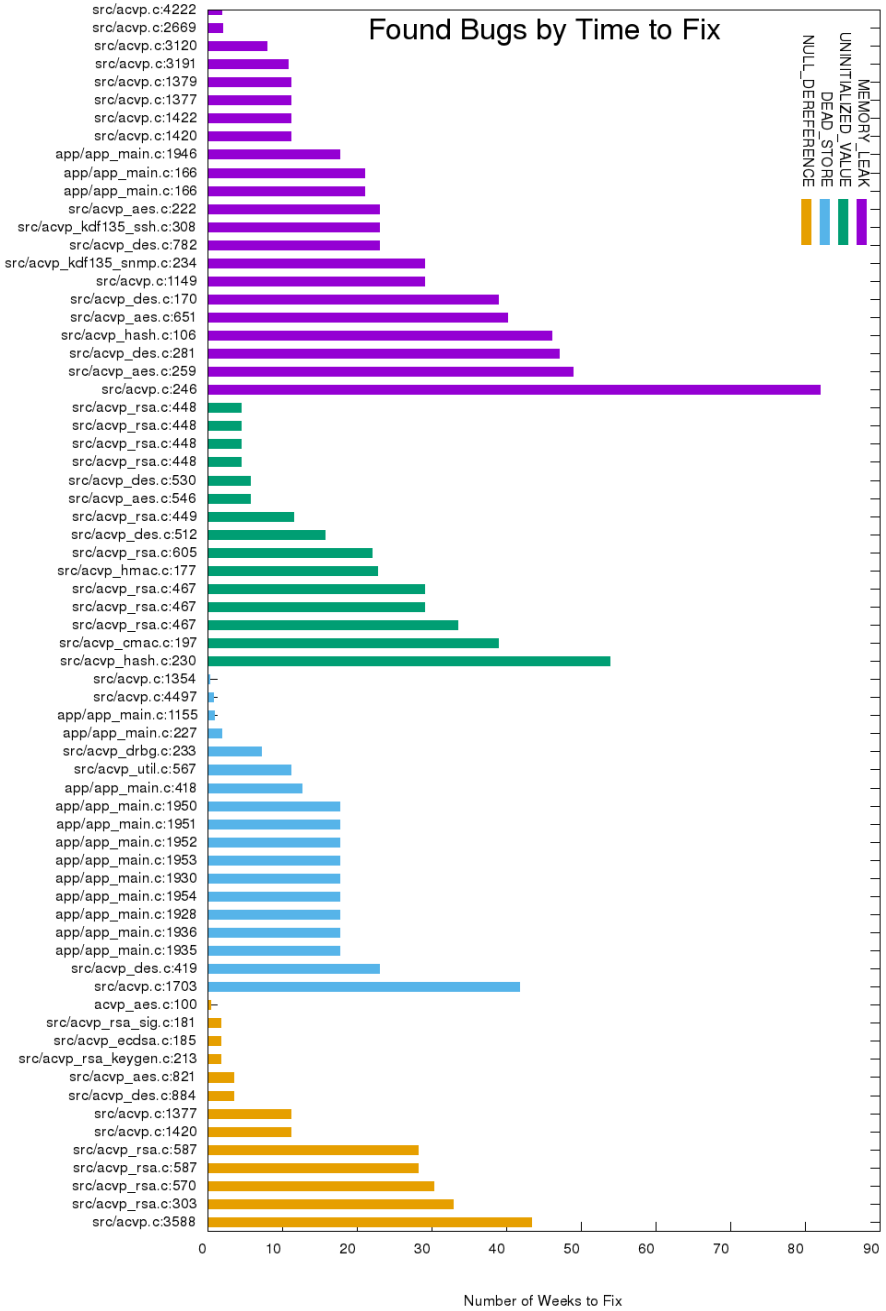
© 2017 Galois, Inc.

Tool walkthrough

MuseDev can monitor properties of repositories over time

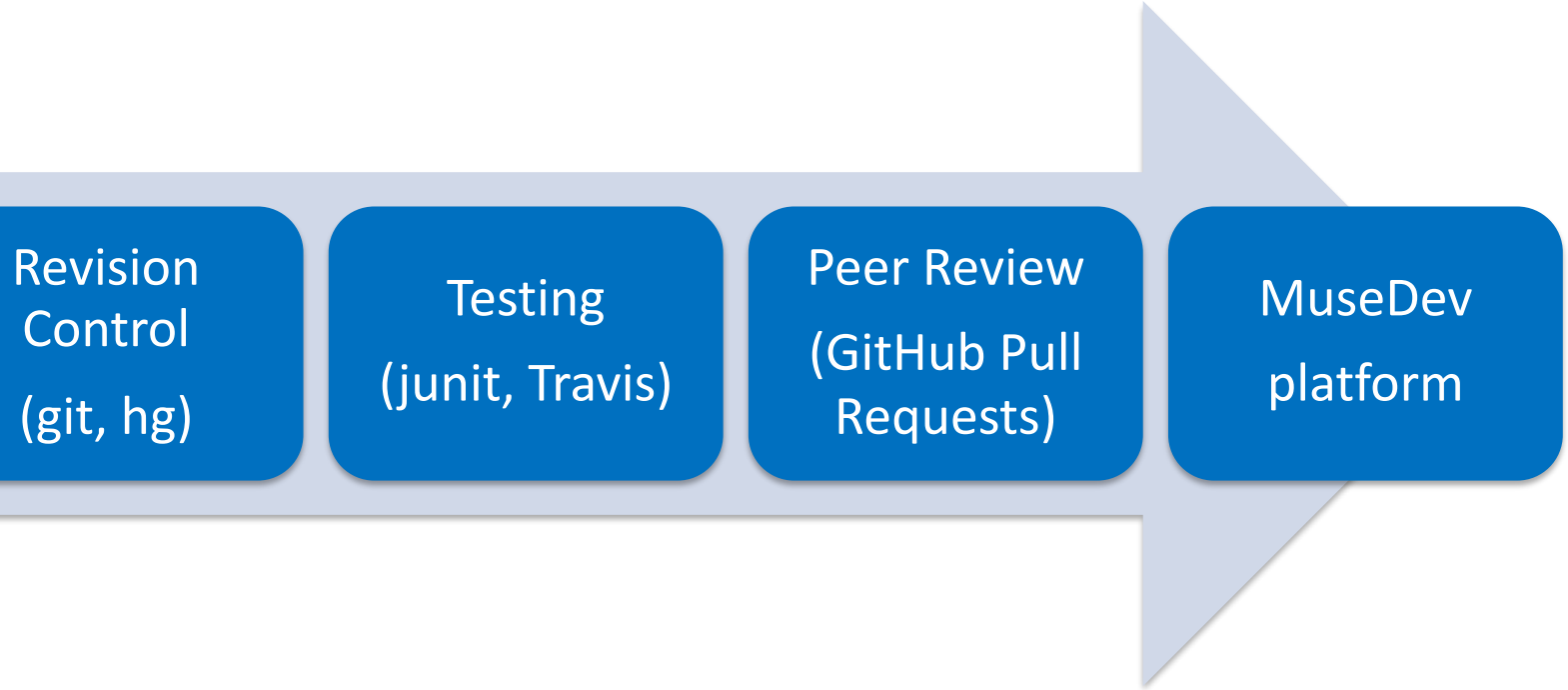
Eg. investigate when bugs are introduced and when they are eventually removed.

Example bug history from libacvp Github repository, generated with Infer



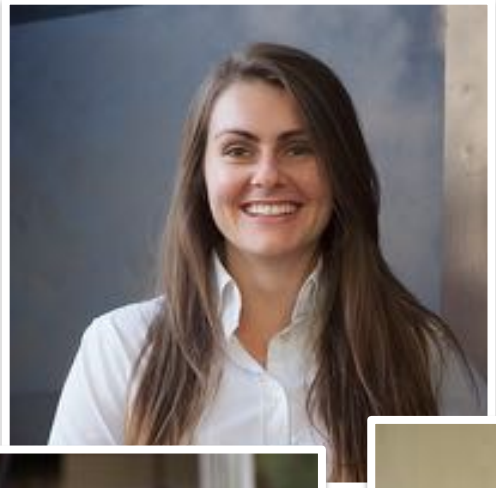
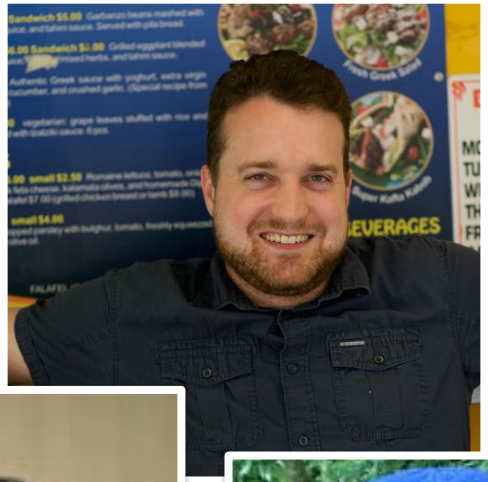
© 2017 Galois, Inc.

MuseDev aim: make static analysis / verification a standard approach to code quality



© 2018 Galois, Inc.

Galois MuseDev team

© 2017 Galois, Inc.

Summary

Continuous integration is making static analysis feasible for industry-scale code.

Lots of great tools that fit different niches, from critical crypto, to large-scale bug-finding

MuseDev is a Galois spin-out company aimed at making it easy to integrate static tools in industry

galois
galois.com/careers/