galois

Verifying Real-world Cryptography

Mike Dodds

March 7, 2019

University of Washington

Who Galois are

Research and development lab of ~100 people

Locations

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

What Galois do

Formal methods research meets real-world applications

Programming languages, analysis, verification, security, cryptography

Our tools:

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

Galois clients

Big research projects from US govt

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

Lots of collaborations with academic partners

Galois formal methods priorities

- Tools for real languages / systems *(vs proofs of concept)*
- Highly automated tools *(vs manual proofs)*
- Domain-specific tools / languages *(vs universal tools)*
- Increasing system assurance *(vs absolute correctness)*
- Integrating with SWE workflows *(vs demanding changes)*

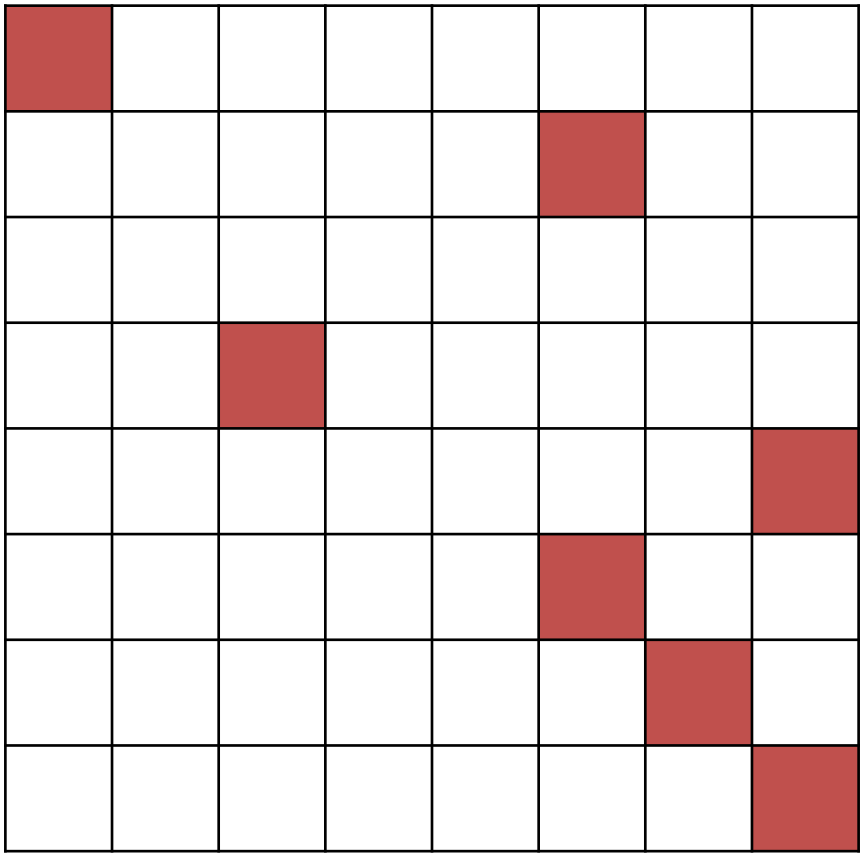
Formal Methods for Security

Computer Security Imbalance

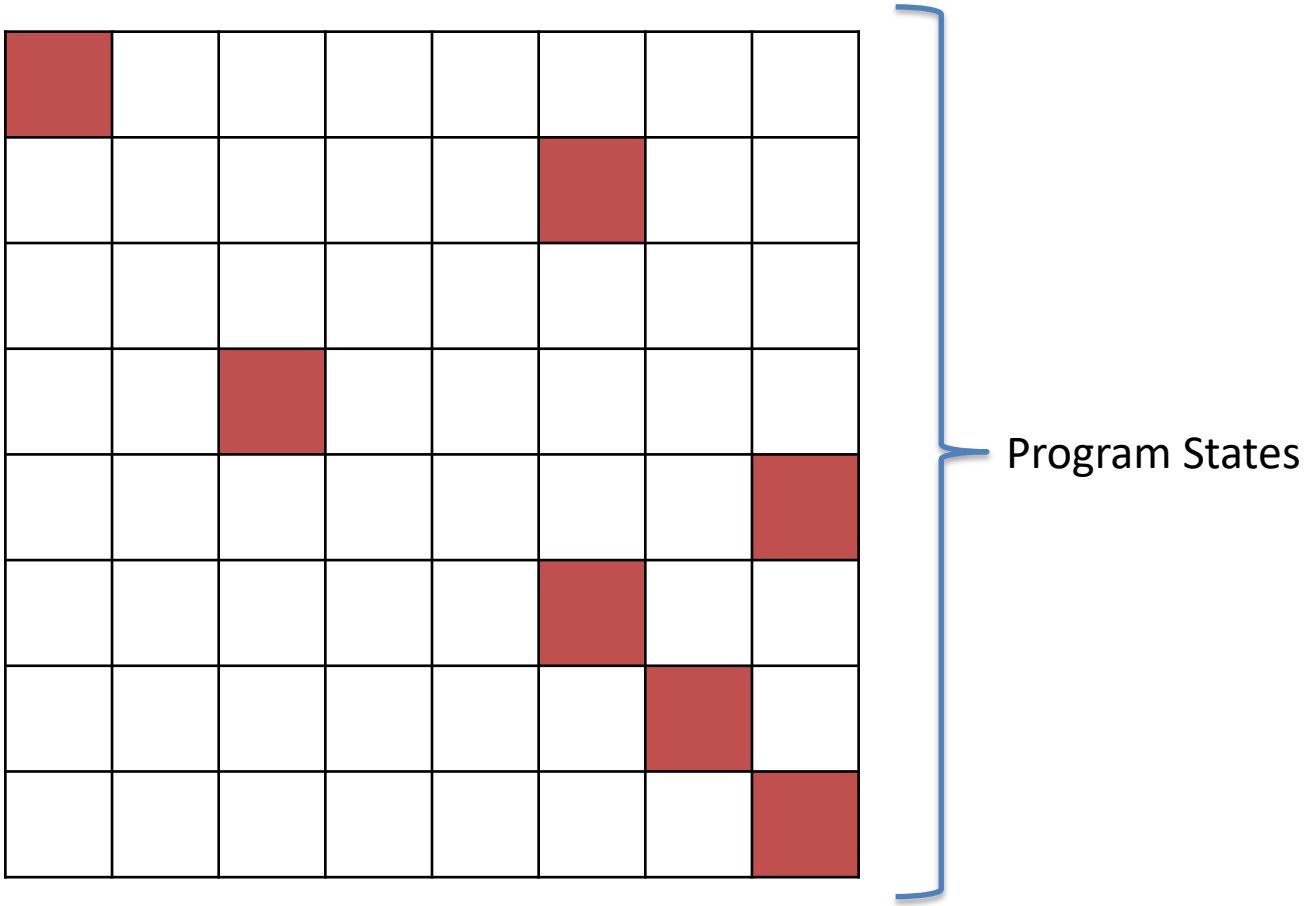
1. Defenders have to prevent all problems.
2. Attackers need only find one entry point.

- Verification aims to enable #1 for critical core components
- Works on small code bases. Useful in practice because
  - Many systems are engineered to keep the security-critical core small (hypervisors, OS kernels, secure channels).
  - Technology advances have decreased the overall level of effort.

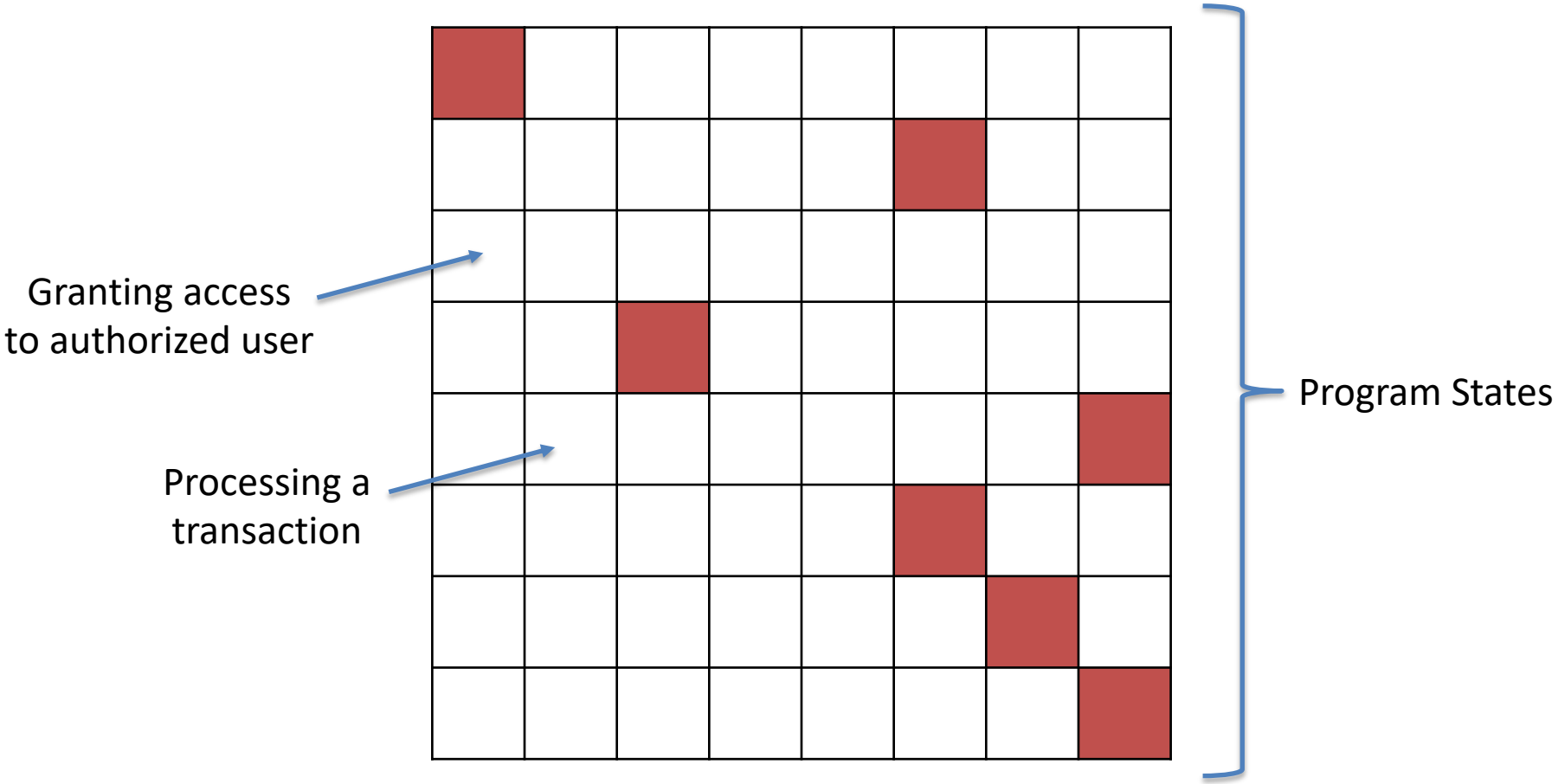
A Grid of Bugs



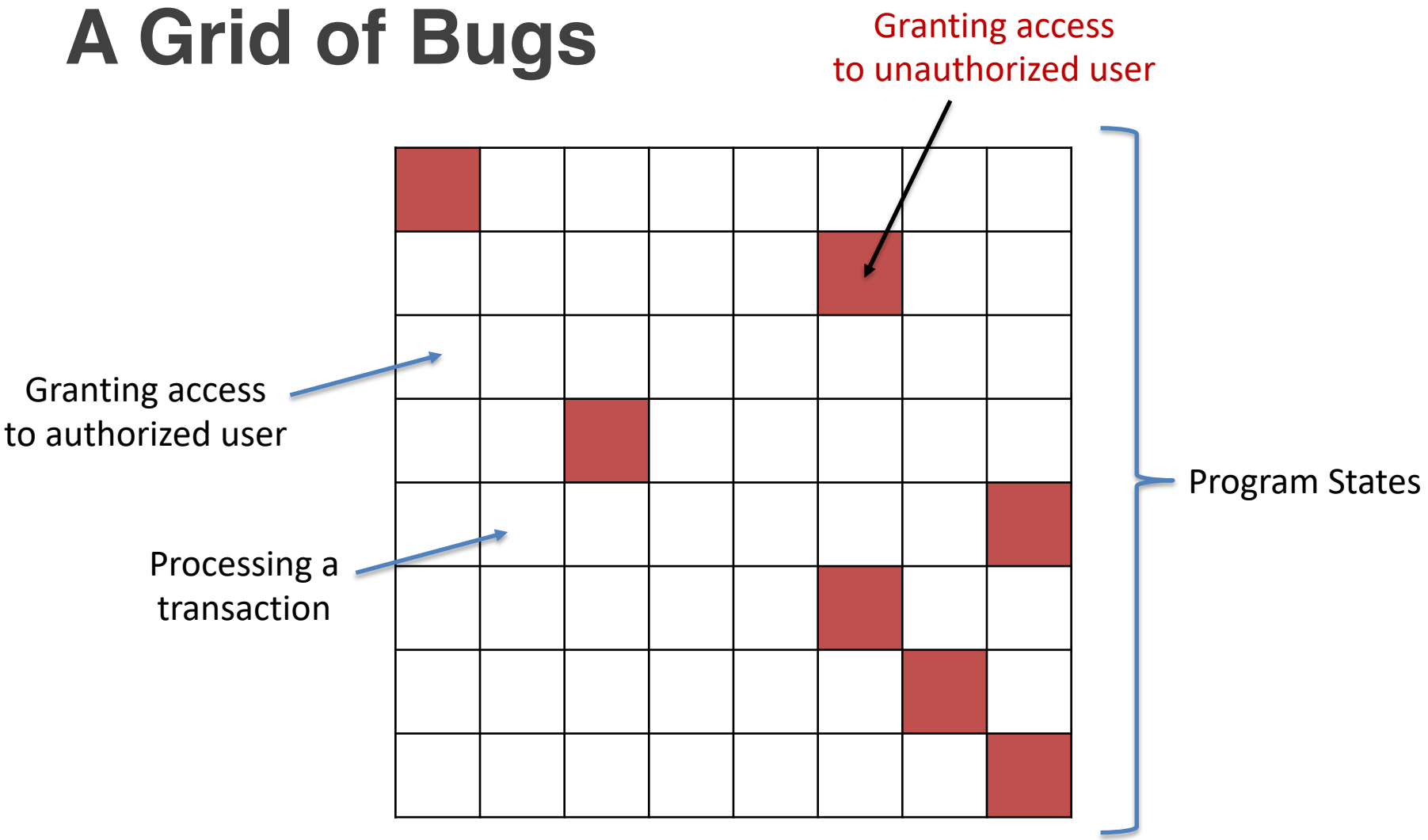
A Grid of Bugs



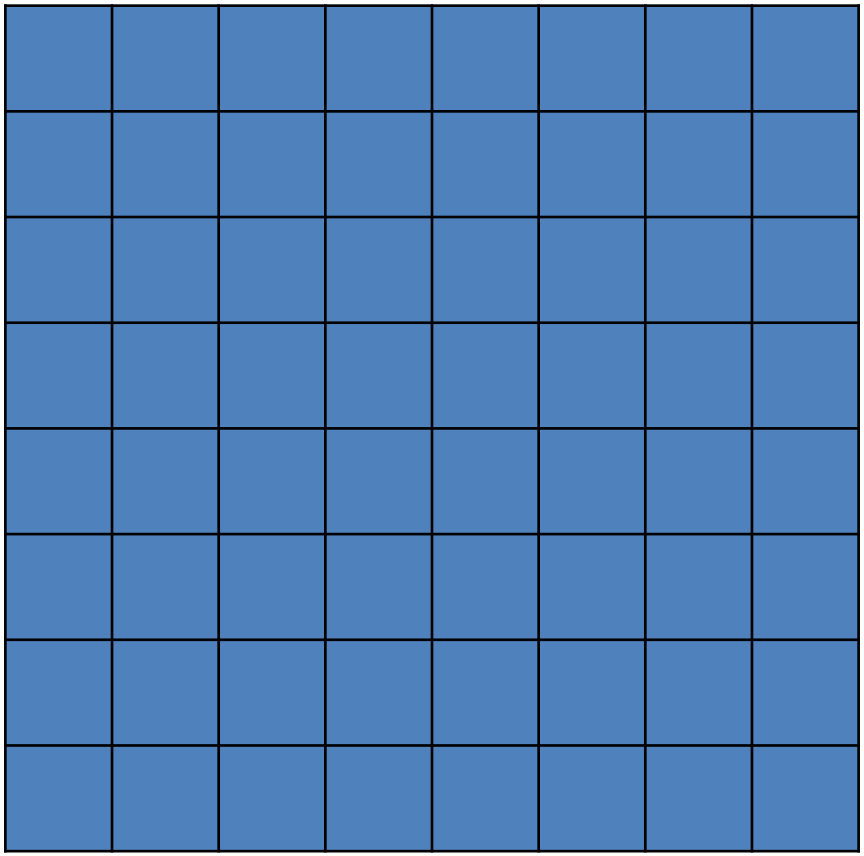
A Grid of Bugs



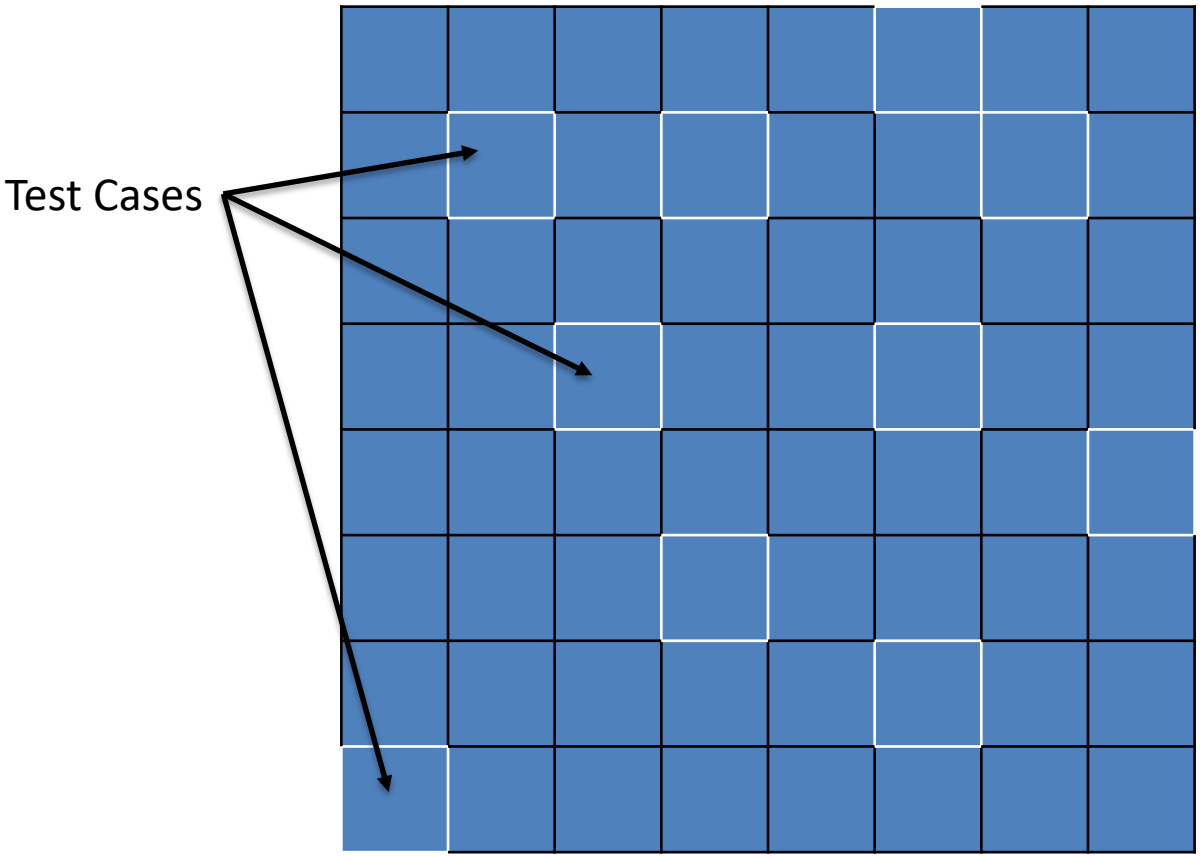
A Grid of Bugs



Software Security As A Game

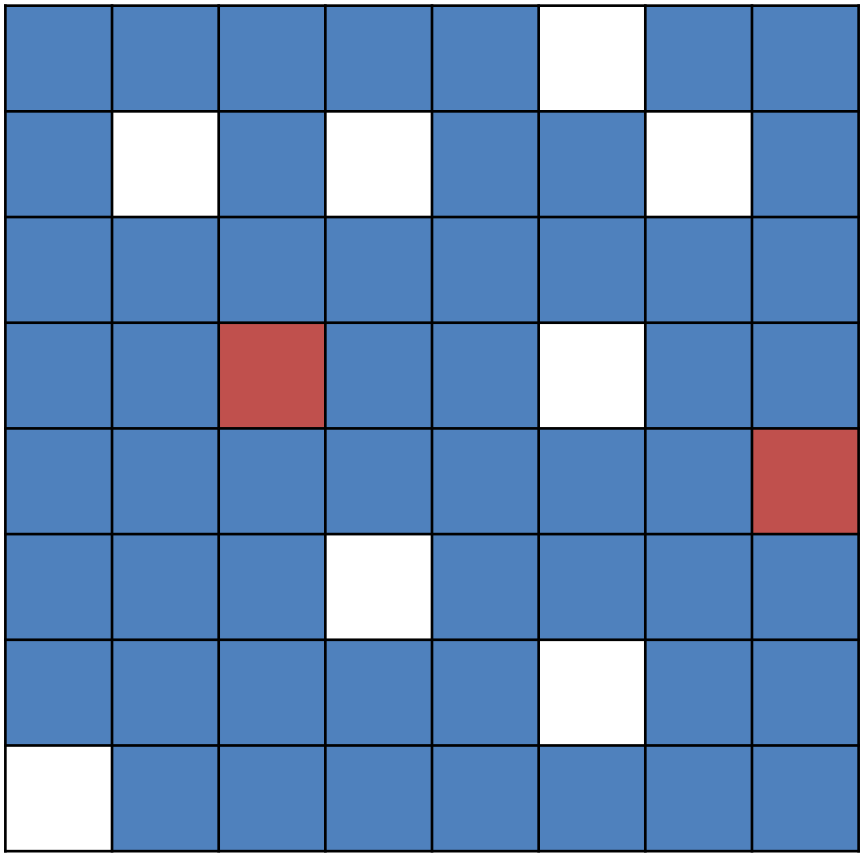


Defender's Turn: Pick 10 Squares

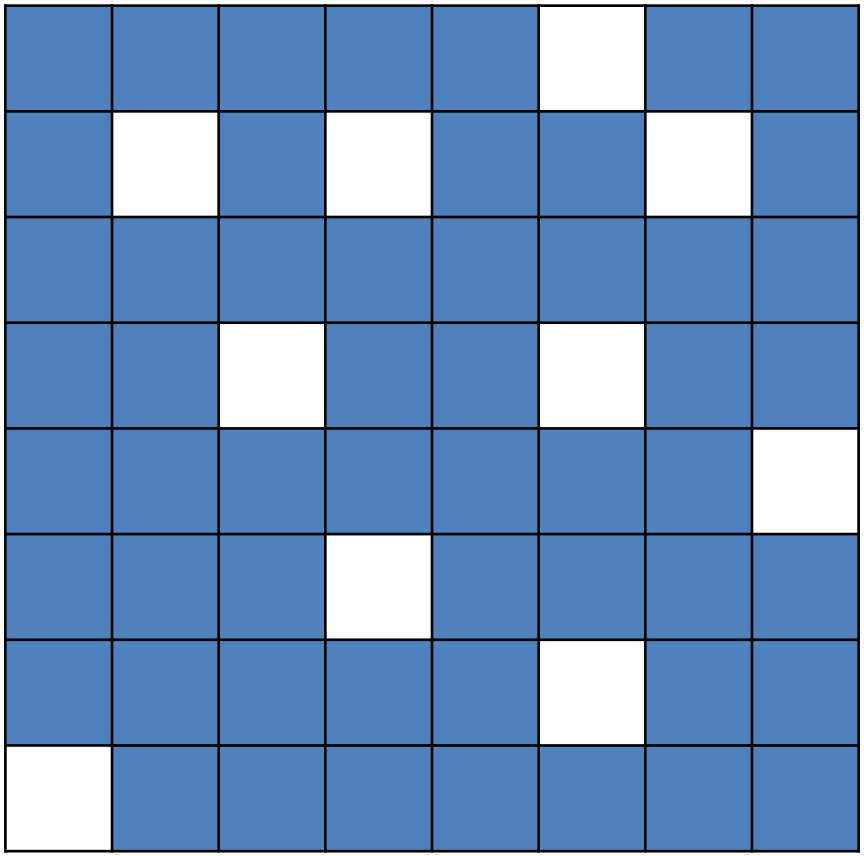


<u>From</u>
Development
Red Teaming
Pen Testing
…

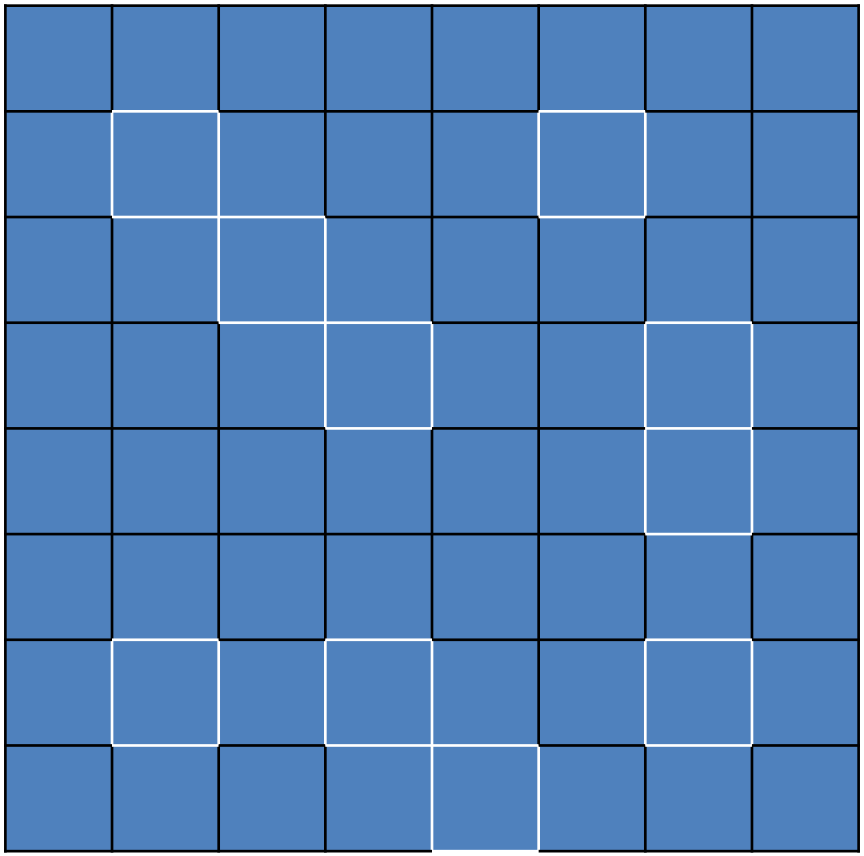
Defender's Turn: Fix Problems



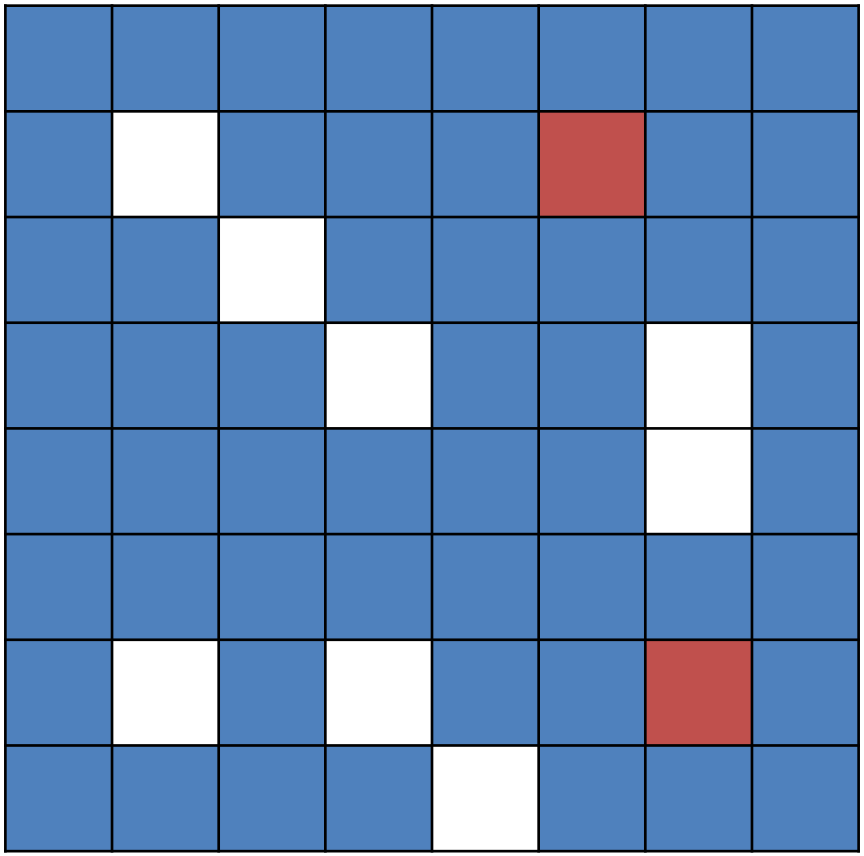
Defender's Turn: Fix Problems



Attacker's Turn: Pick 10 (or 20, or…)



Attacker's Turn: Pick 10 (or 20, or…)



Attacker Advantage

- General per-round odds favor attacker.
  - Find all red squares vs. find any red square

- Attacker generally has more time.
  - Windows XP is 15 years old now.

Verification / Formal Methods

Cover *much* more of the state space by discovering and leveraging underlying *structure*.

Formal Methods: Characterize State

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

i mod 13 = 0

i mod 8 = i div 8

Formal Methods: Characterize State

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

$i \bmod 13 \neq 0$

$i \bmod 8 \neq i \operatorname{div} 8$

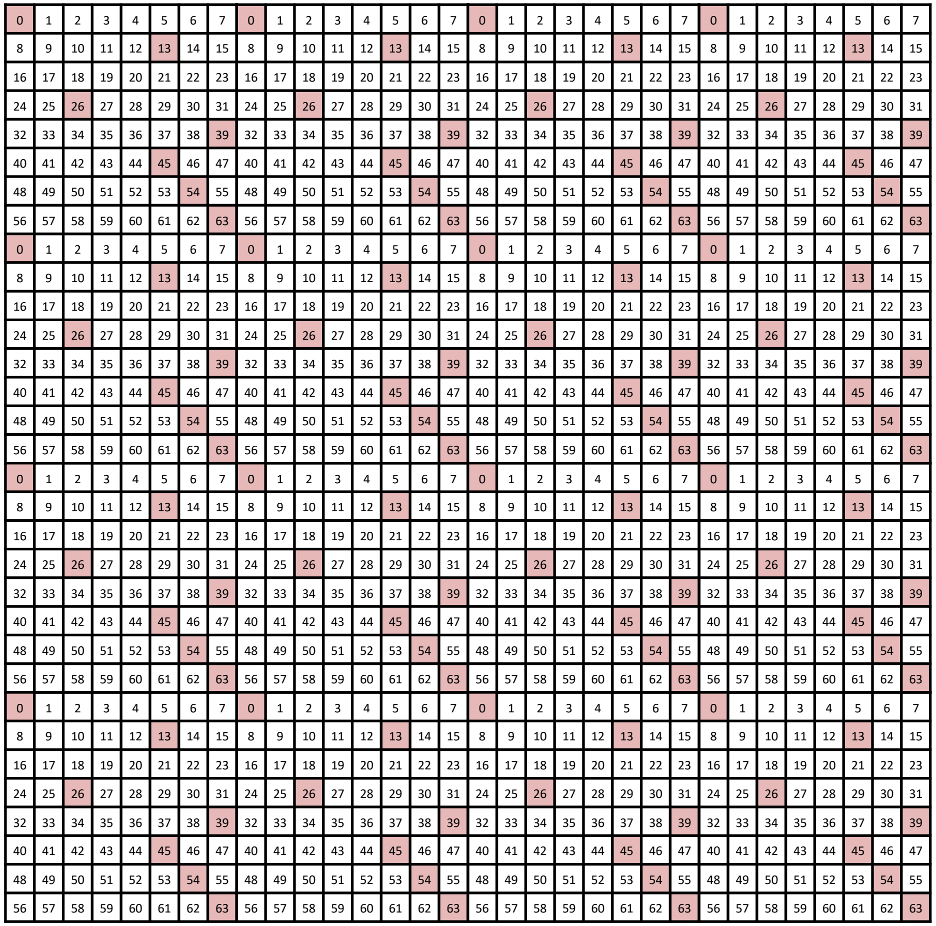
Formal Methods: Characterize State

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63

$i \bmod 13 \neq 0$

$i \bmod 8 \neq i \operatorname{div} 8$

Formal Methods: Characterize State



0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63	56	57	58	59	60	61	62	63

$i \bmod 13 \neq 0$

$i \bmod 8 \neq i \operatorname{div} 8$

Formal Methods

Cover *much* more of the state space by discovering and leveraging underlying *structure*.

In the limit, can prove that code is correct in *all* cases (for the given proof scope).

Method: Characterize the “good” behavior. Show this is the only behavior that can occur.

This is now at a viable cost / benefit point for critical, broadly deployed code.

Galois tools: Cryptol and SAW

Specification language: Cryptol

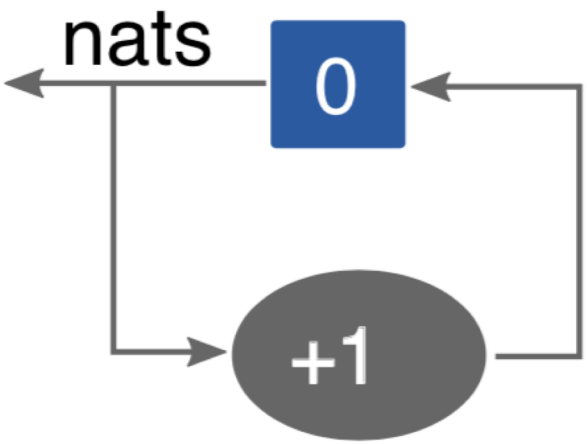
A single, high-level specification for (cryptographic) algorithms

- Cryptol goals
  - Appropriate for cryptography
  - Natural
  - Concise
  - Similar to existing notation
  - Appropriate for execution and verification
- Language features
  - Statically-typed functional language
  - Sized bit vectors (type level naturals)
  - Stream comprehensions (stream diagrams)

A Taste of Cryptol

- Functions and sequences are key notions
- Both can be recursive
- To compute the sequence of all natural numbers

```
nats = [0] # [ n + 1 | n <- nats ]
```



Specifications as code

Cryptol specifications are (Haskell-like) code:

```
hmac h h2 h3 K m =
 h2 (okey # split (h (ikey # m)))
 where
  k0 = kinit h3 K
  okey = [kb ^ 0x5C | kb <- k0]
  ikey = [kb ^ 0x36 | kb <- k0]
```

You can use a Cryptol specification in many ways:

- Execute with an input
- Generate input values
- Compile into code.

Verification tool: SAW

- SAW = Software Analysis Workbench
  - Software: many languages
  - Analysis: many types of analysis, focused on functionality
  - Workbench: flexible interface, supporting many goals
- Intended as a flexible tool for software analysis
- What separates it from other systems?
  - One view: compiler :: imperative code $\rightarrow$ functional code
  - Captures all functional behavior, simplifying later if necessary
  - Uses efficient internal representations tuned to equivalence checking
  - Strong bit vector reasoning support
  - Focus on practicality over novelty
- Open source (BSD3) and available now

SAW verification process

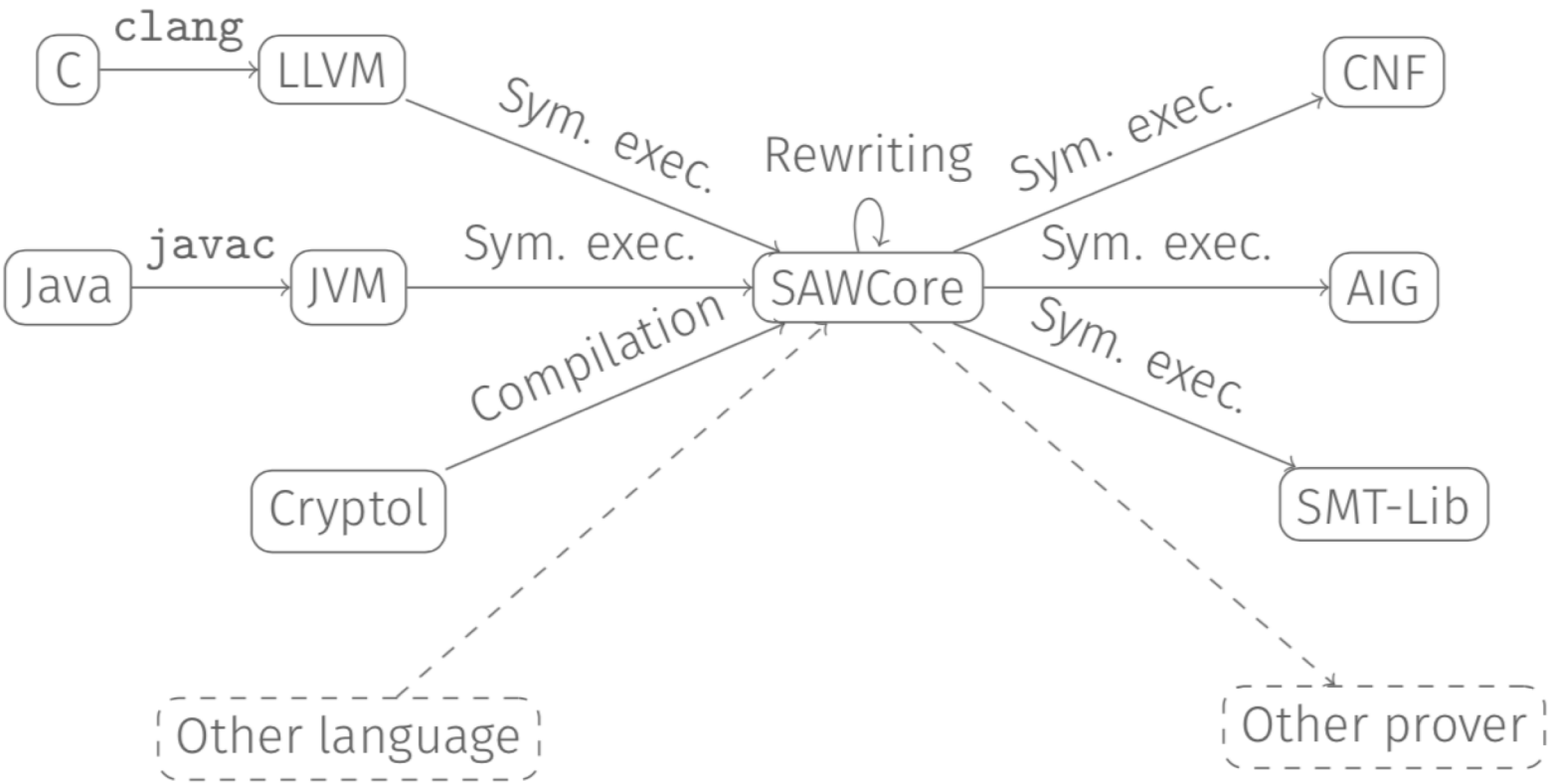
Inputs:

1. Executable specification in Cryptol
2. Target program in C / Java …

Symbolically execute both programs to generate SAWcore terms, a pure intermediate language

Check programs are equivalent, e.g using SMT solvers

SAW architecture



Compositional Verification

Due to its size and complexity, no tool that we know of can verify top-level cryptographic primitives in one go.

To scale up automated tools to larger problems, we need tools for decomposing larger problems into smaller pieces that can be verified individually.

In SAW, we do this by allowing users to verify individual methods independently, and composing the results together in a larger verification effort.

Once a specification is defined, it can be used to simplify later methods.

Example: s2n TLS verification

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

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.

If I go to gmail...

TLS lets me be sure:

1. I'm actually talking to google
2. Nobody (not even my ISP) can read what I'm reading
3. Nobody (not even my ISP) can change the data I'm reading

Also used pervasively for communication between services in the cloud.

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.

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

$$\text{HMAC}(K, m) = H((K0 \oplus opad) \parallel H((K0 \oplus ipad) \parallel m))$$

arbitrary hash function

constants from NIST

- Code is still complex
  - 521 lines of C code

HMAC Specification

C HMAC

HMAC(m,k) =

H((K ⊕ opad) || H((K ⊕ ipad) || m))

Easily auditable

Concise

Fast

Interoperable

Goal: bridge this gap

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.

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.

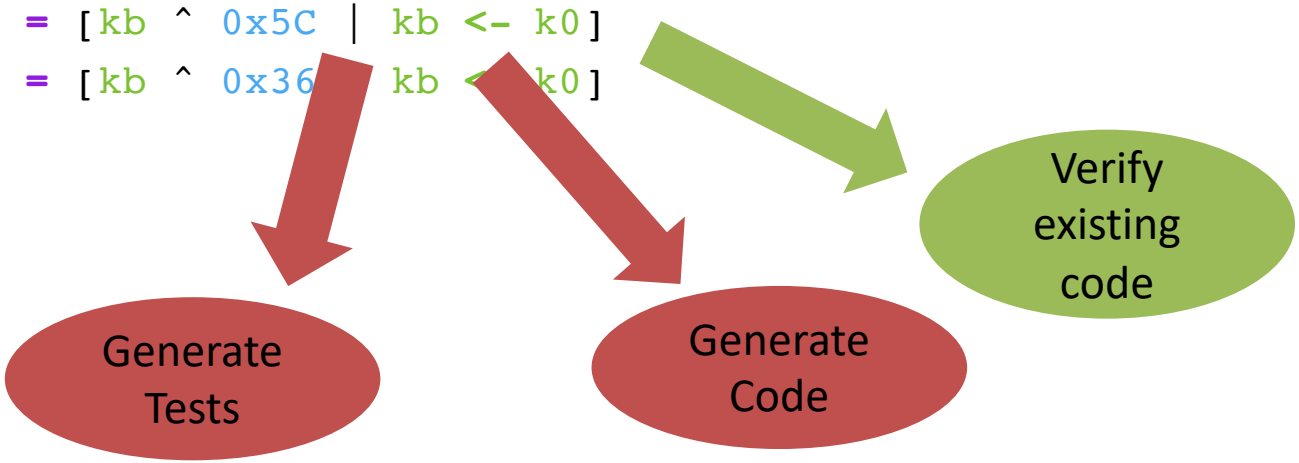
About 2 months of effort.

Summary of Approach

HMAC(K, m) = H((K0 ⊕ opad) ‖ H((K0 ⊕ ipad) ‖ m))

Step 1: Capture this specification in a formal language (we used Cryptol)

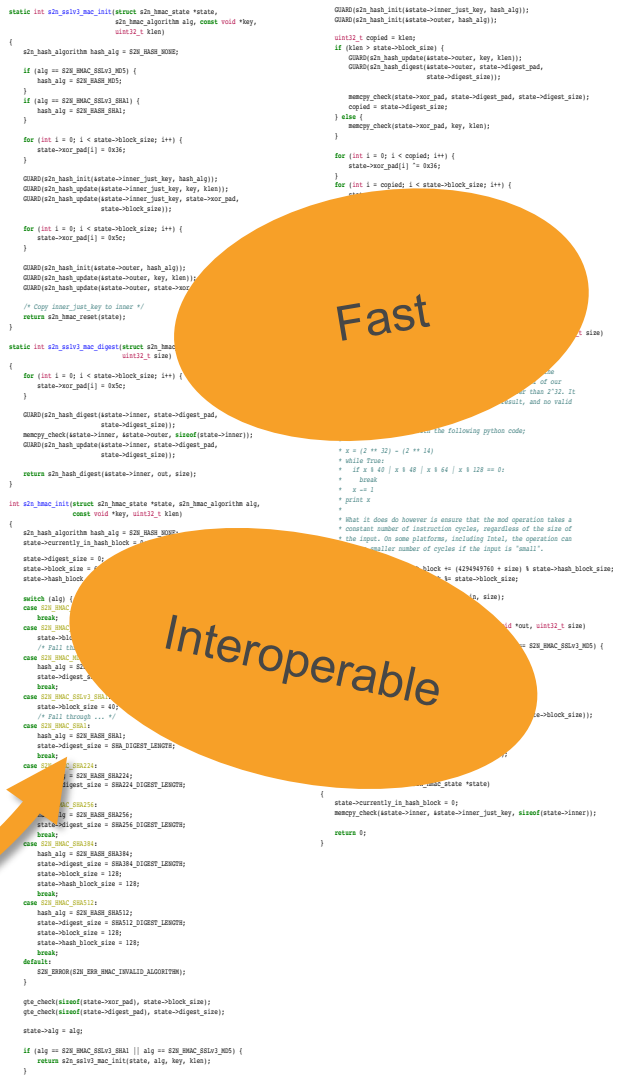
```
hmac h h2 h3 K m =
 h2 (okey # split (h (ikey # m)))
  where
   k0 = kinit h3 K
   okey = [kb ^ 0x5C | kb <- k0]
   ikey = [kb ^ 0x36 | kb <- k0]
```



HMAC Specification

```
hmac h h2 h3 K m =
  h2 (okey # split (h (ikey # m)))
  where
    k0 = kinit h3 K
    okey = [kb ^ 0x5C | kb <- k0]
    ikey = [kb ^ 0x36 | kb <- k0]
```

C HMAC



Fast

Interoperable

Concise

Easily auditable

Goal: bridge this gap

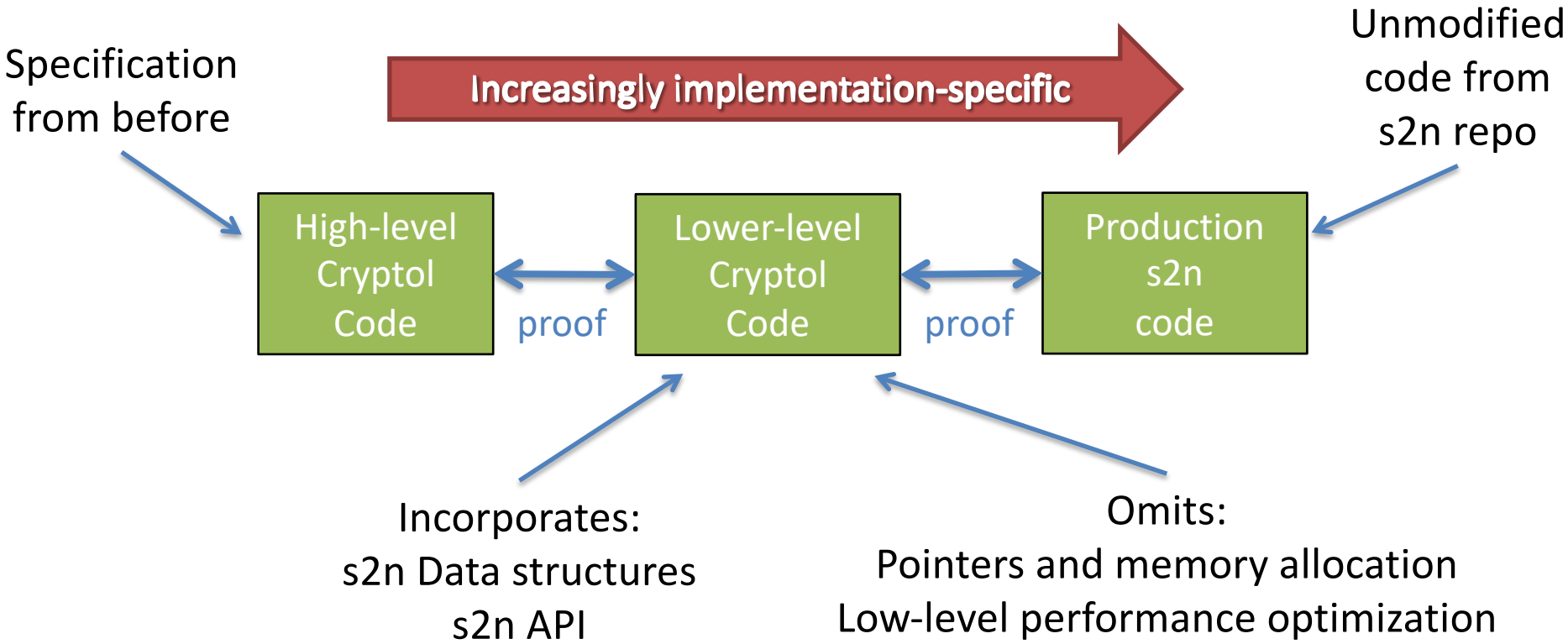
Summary of Approach

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.

Bridging the gap

Solution: Layers of Abstraction



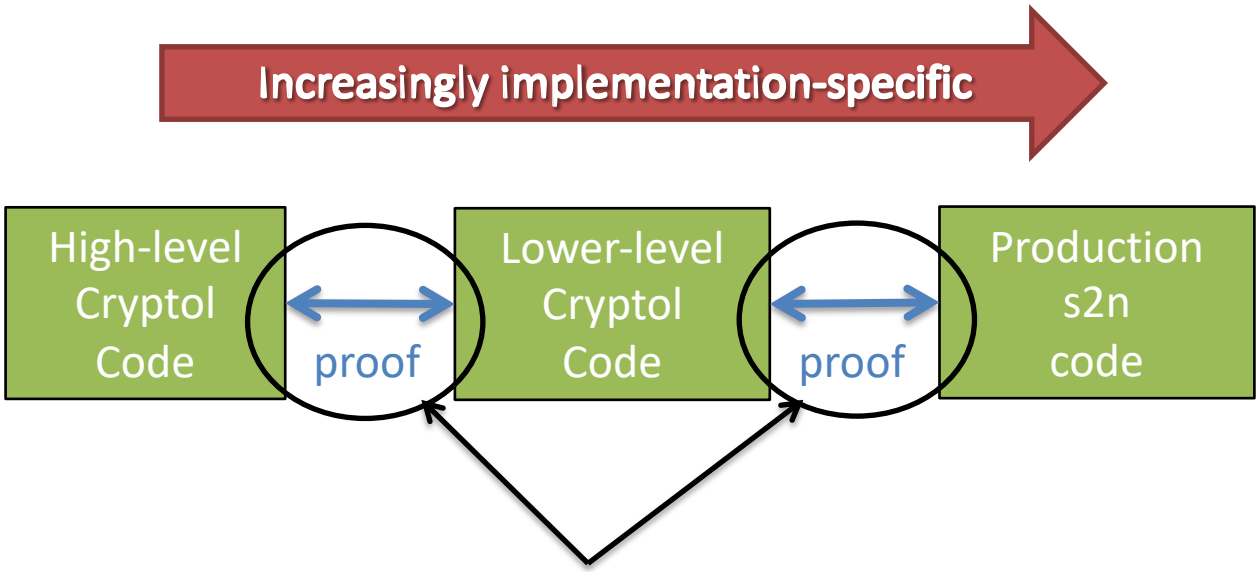
Summary of Approach

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.

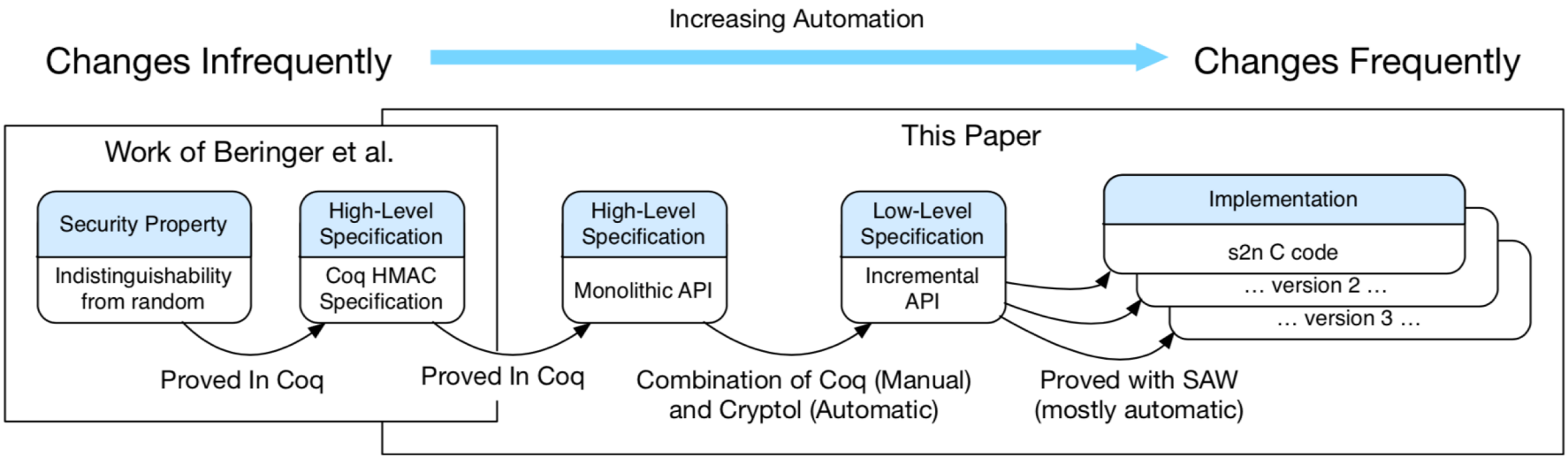
Bridging the gap

Solution: Layers of Abstraction



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

Verified HMAC pipeline



(From our CAV18 paper)

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.

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

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_ #953 passed

Added guards around allocator_overrides

Ran for 1 hr 13 min 8 sec

Total time 4 hrs 54 min 44 sec

Commit 02ade5e

Compare 9eb9b99..02ade5e

Branch master

20 days ago

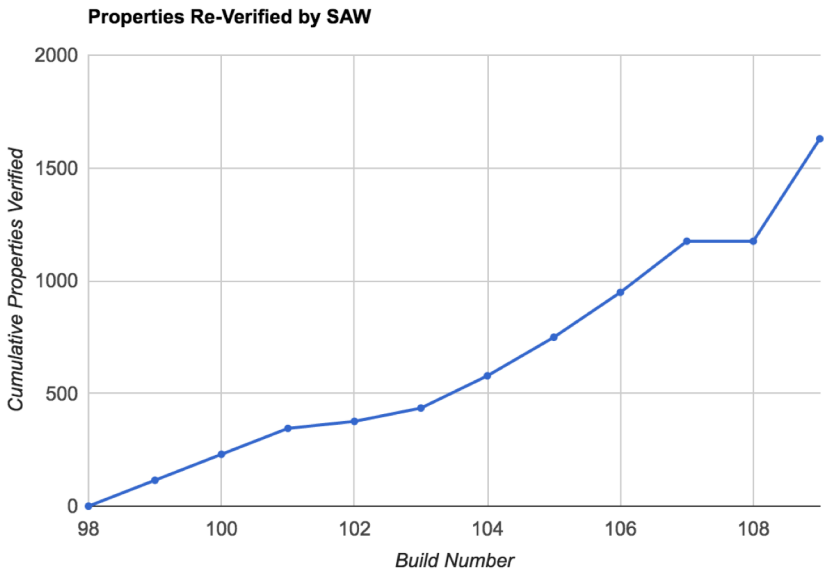
Matthew Baldwin authored GitHub committed

Build Jobs

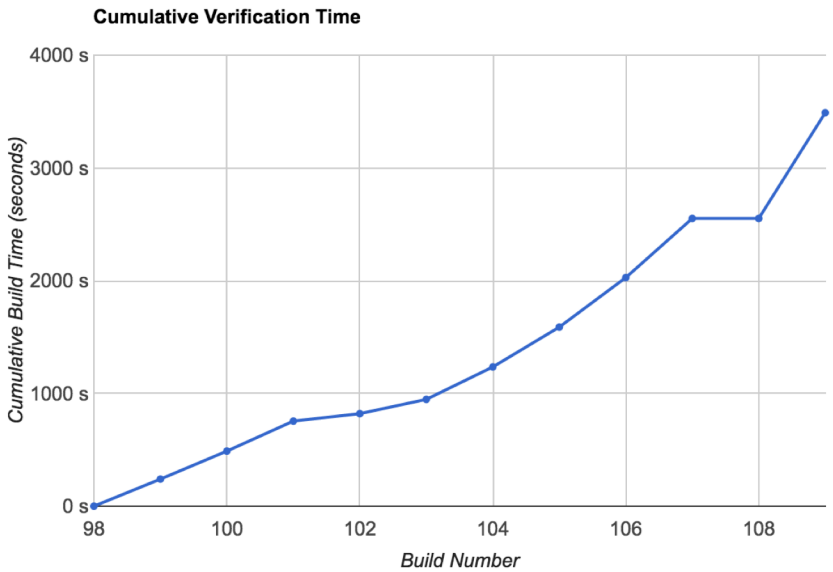
953.1 Xcode: xcode8 C TESTS=ctverif 4 min 59 sec

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	0.659	✓

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

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)

Protocol Problem Example

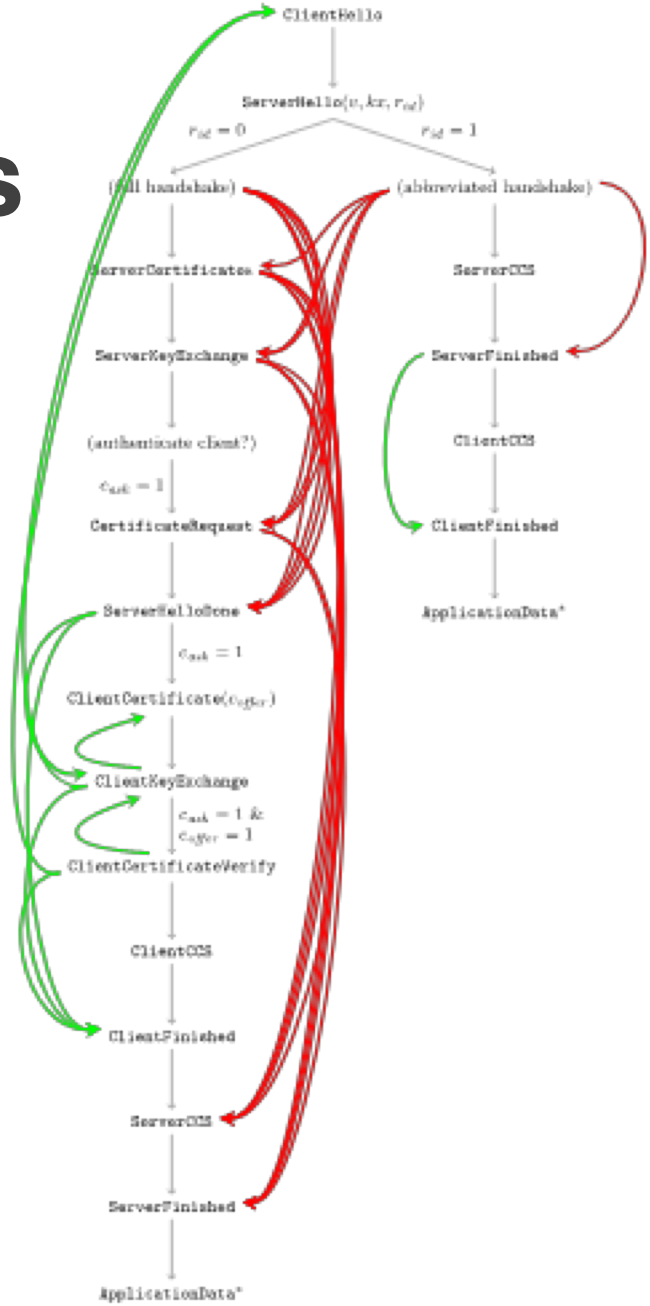
- Early ChangeCipherSpec (Early CCS)

  "If a ChangeCipherSpec message is injected into the connection **after** the **ServerHello**, but **before** the **master secret** has been generated, then ssl3_do_change_cipher_spec **will generate the keys** (2) and the expected Finished hash (3) for the handshake with an *empty* master secret. This means that both are **based only on public information**."

From https://www.imperialviolet.org/2014/06/05/earlyccs.html

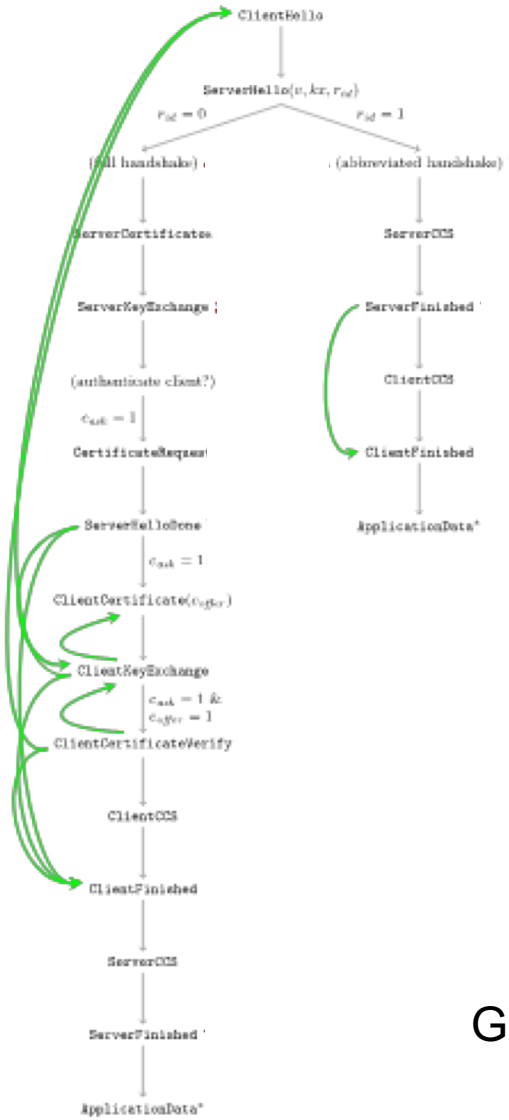
State Machine Attacks

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



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

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 429494760 constant in this code? 429494760 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 += (429494760 + 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;
}
```

Goal: bridge this gap

Fixing the problem

- Write a model of the state machine in Cryptol.
- Verify equivalence using SAW
- Integrate into CI

Other Crypto Work

- Have verified implementations of AES, SHA, ECDSA

NISTCurve.java (line 964):

```
d = (z[0] & LONG_MASK) + of;
    z[0] = (int) d; d >>= 32;
    d = (z[1] & LONG_MASK) - of;
    z[1] = (int) d; d >>= 32;
    d += (z[2] & LONG_MASK);
    z[2] = (int) d; d >>= 32;
    d += (z[3] & LONG_MASK) + of;
```

Other Crypto Work

- Have verified implementations of AES, SHA, ECDSA

Bug only occurs when this addition overflows. (rare since `of` < 5)

NISTCurve.java (line 964):

```
d = (z[0] & LONG_MASK) + of;
    z[0] = (int) d; d >>= 32;
    d += (z[1] & LONG_MASK) - of;
                    ; d >>= 32;
                    ONG_MASK);
                    ; d >>= 32;
    d += (z[3] & LONG_MASK) + of;
```

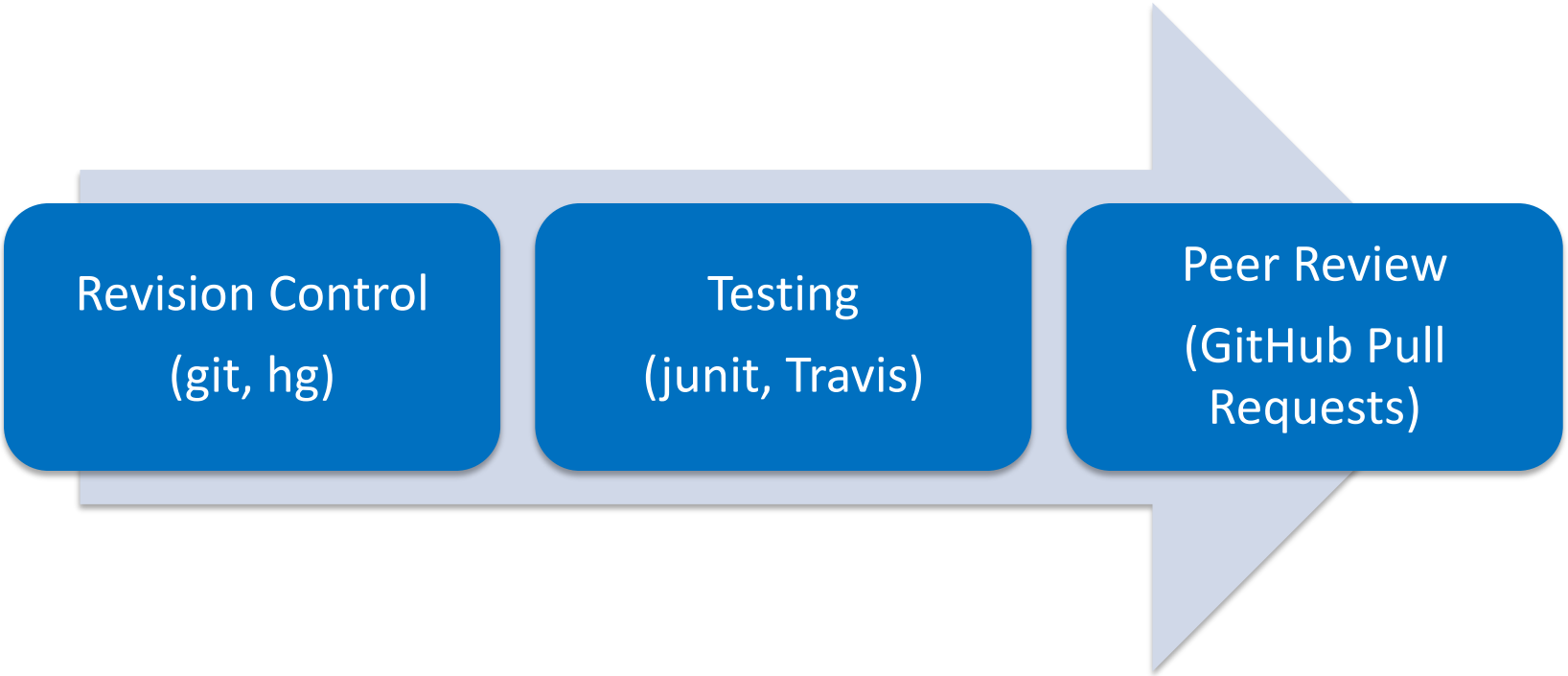
SAW found bug in 20 seconds. Testing found bug after 2 hours (8 billion field reductions later).

Other Crypto Work

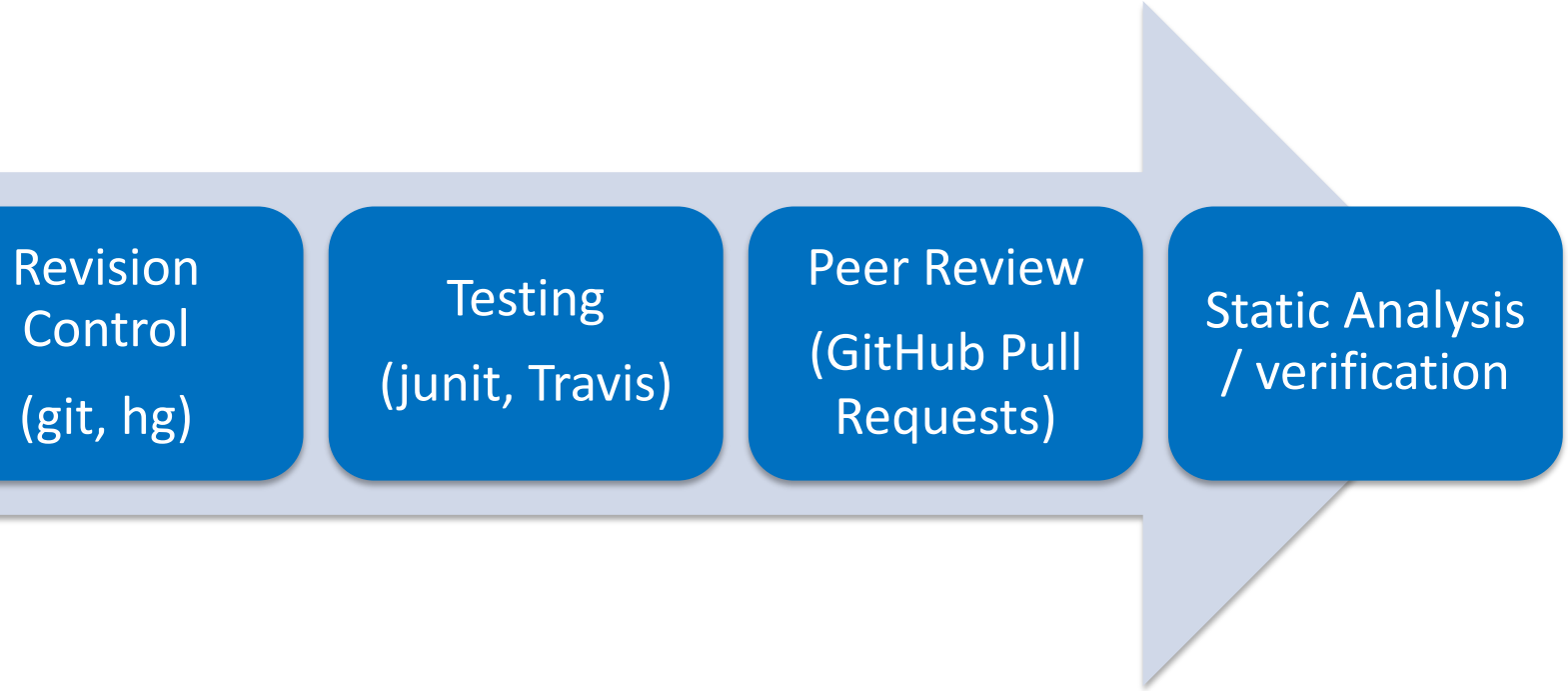
- **Other SAW / Cryptol verification projects:**
  - DRBG: Deterministic Random Bit Generator: The main source of cryptographic randomness (see paper at CAV'18)
  - Synthesizing verified hardware crypto
  - Other AWS projects I'm not able to talk about yet
- **Working on verifying Facebook Fizz TLS1.3 library**

Idea: Continuous Static Analysis

Code quality strategies in industry



Code quality strategies in industry



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

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"*

2019: 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

Summary

- Can prove correct behavior rather than search for errors
- For crypto / authentication / access control: **Behavioral bugs are security bugs**
- Proof can be integrated into development workflow to consistently prevent introduction of errors


galois
galois.com/careers/