CIS 673: Computer Aided Verification

Heartbleed: A Formal Methods Perspective

Aalok Thakkar
University of Pennsylvania

Outline

1. Anatomy

   cause, possible attacks, impact
2. Diagnosis

   combining static and dynamic analyses
3. Prevention

   good habits of program development
4. Treatment

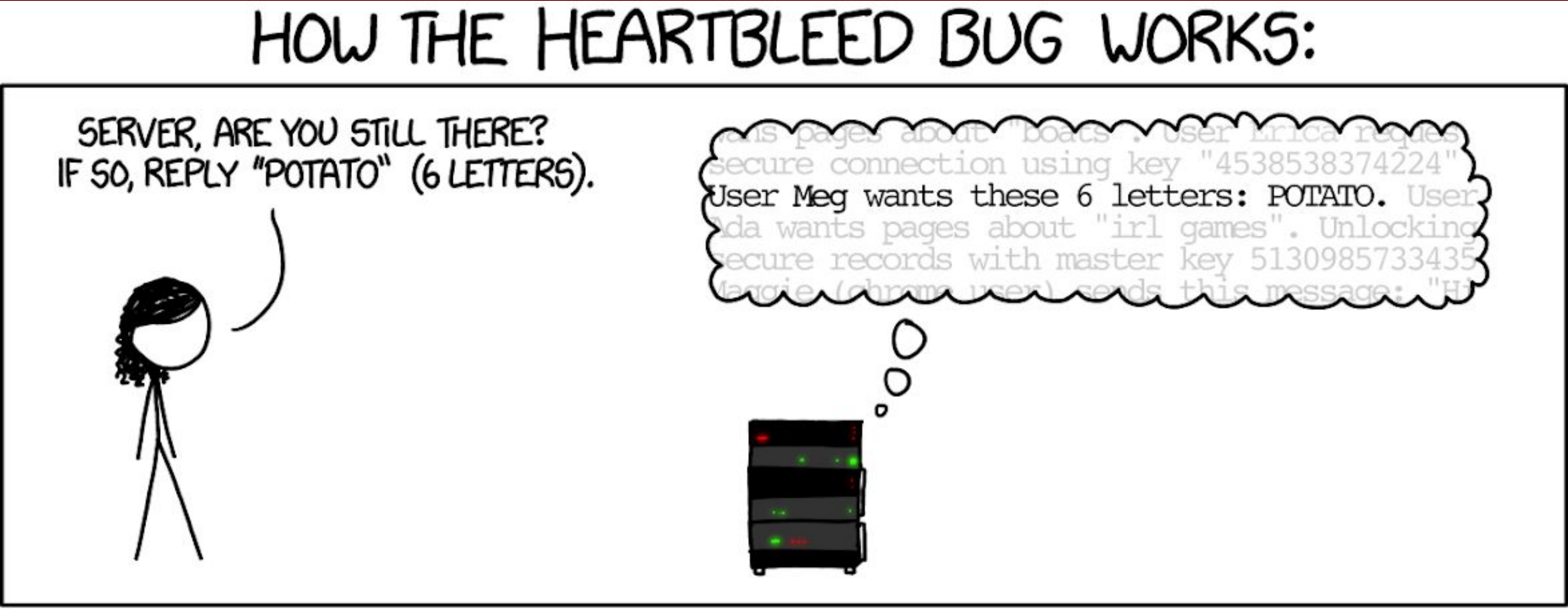
   an application of program synthesis
5. Conclusions

   future work, conclusion

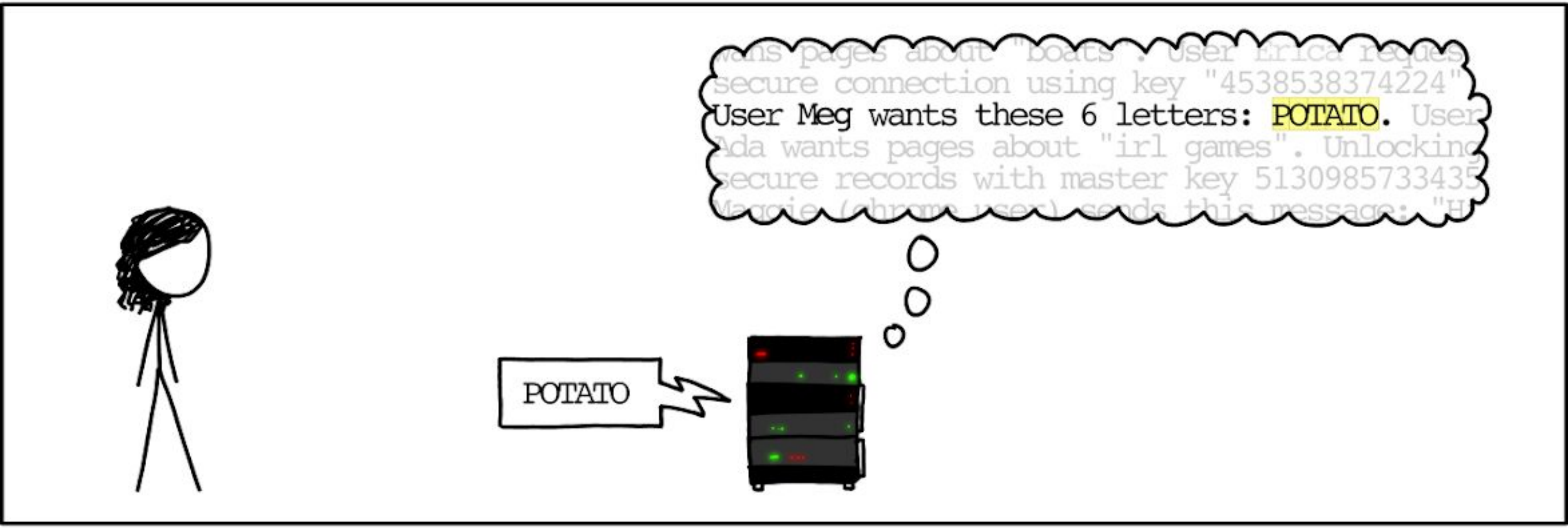
"I cannot believe it. The internet was supposed to be a lawless frontier where all of humanity's desires and vices merge into a royal collective aid held in check by a barely regulated rat's nest of technical abstractions I don't understand. How did that get out of control?"

- The Colbert Report, Comedy Central, April 9, 2014

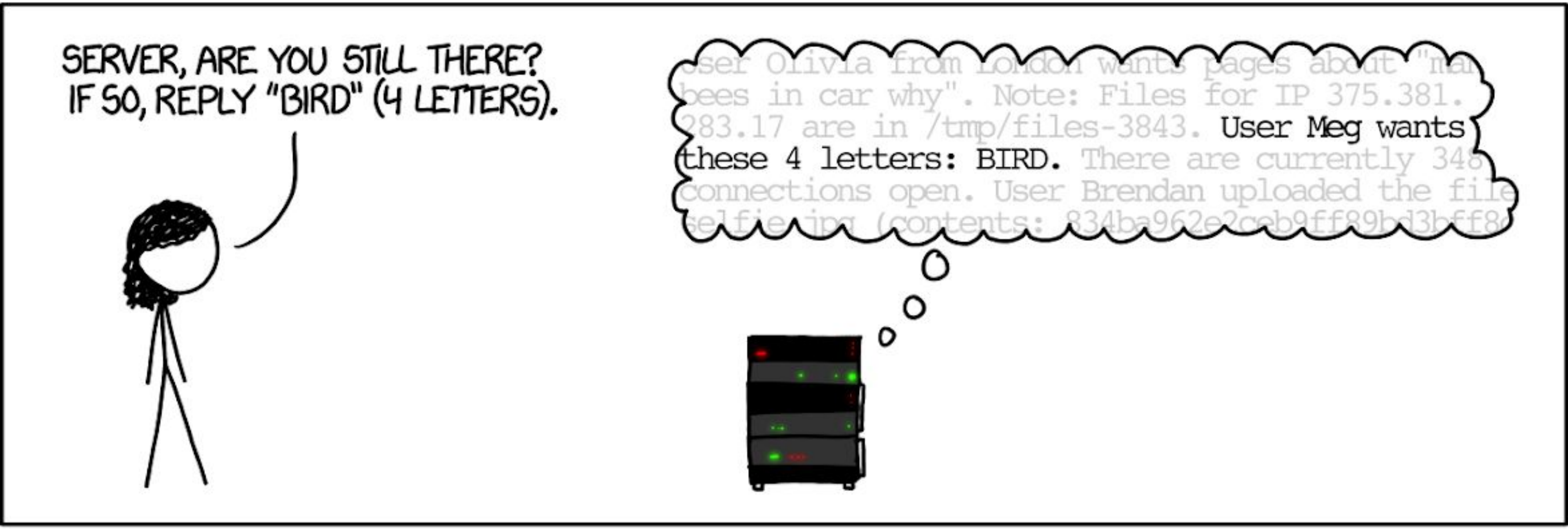
OpenSSL


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
ants pages about "boats". User Erica reques
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H

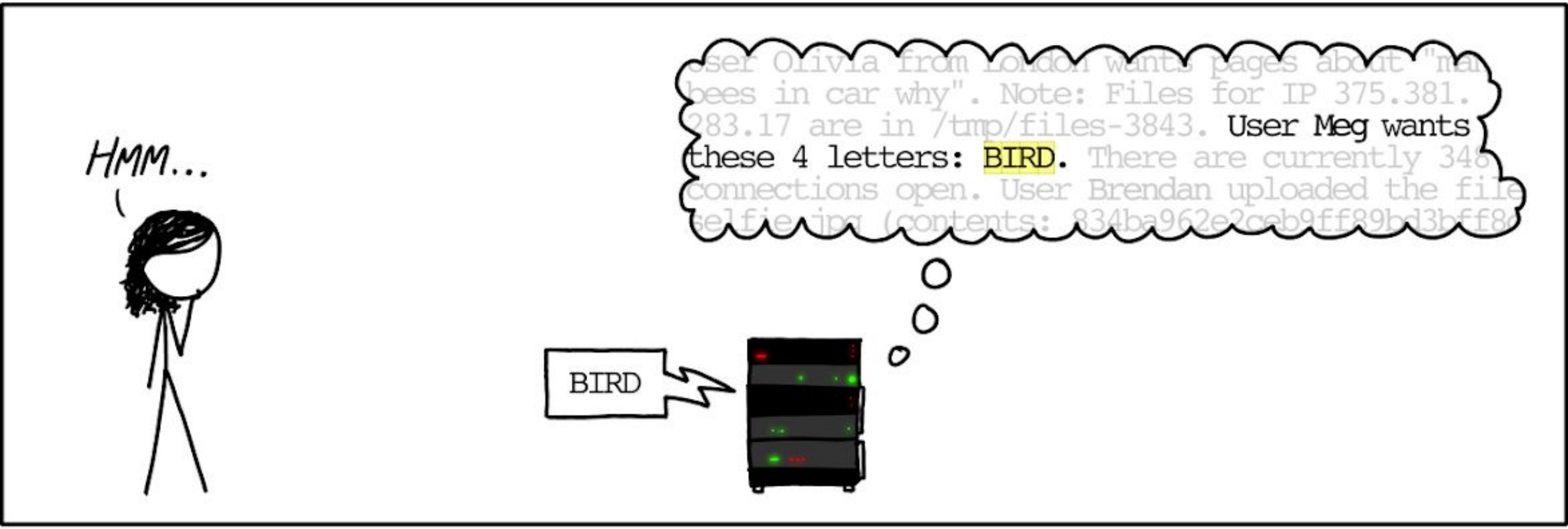
source: xkcd.com/1354/


wants pages about "boats". User Erica reques
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H
POTATO

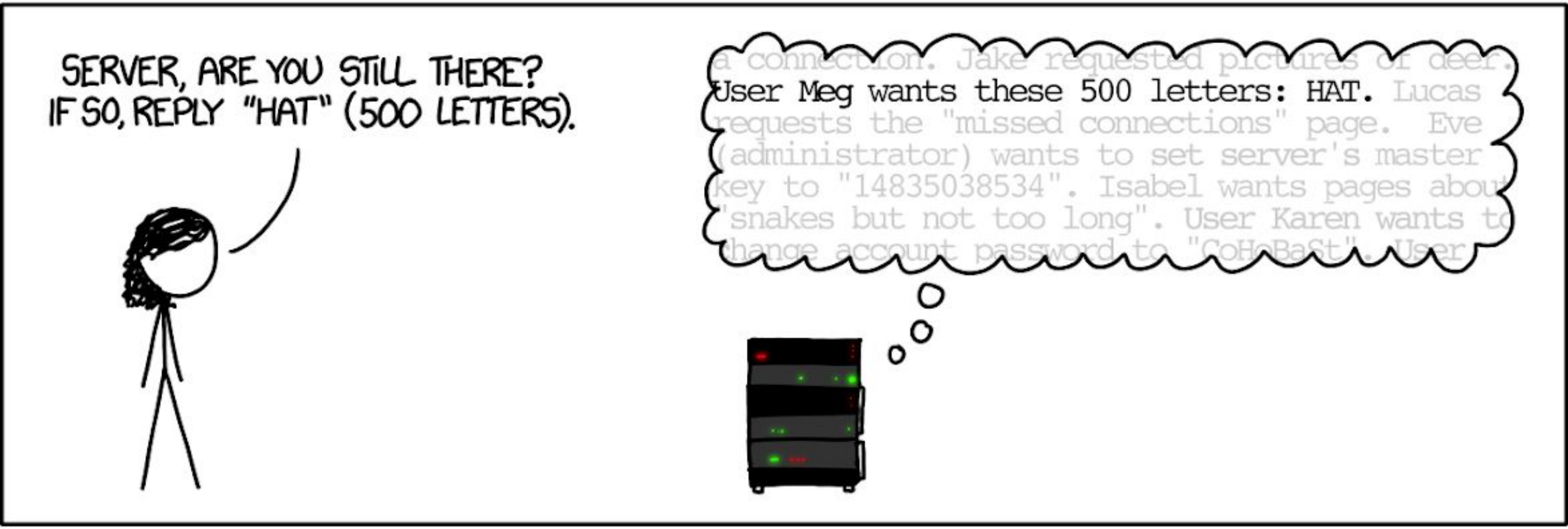
source: xkcd.com/1354/


SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 348
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8

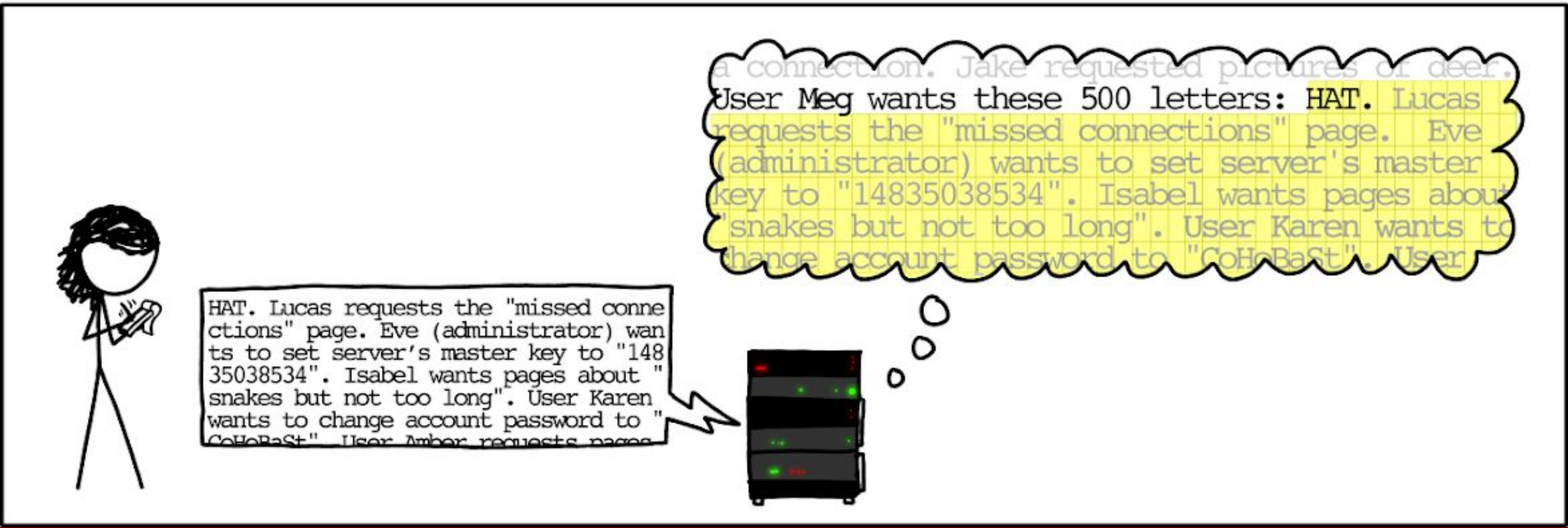
source: xkcd.com/1354/


HMM...
User Olivia from London wants pages about "man bees in car why". Note: Files for IP 375.381.283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 348 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
BIRD

source: xkcd.com/1354/


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User

source: xkcd.com/1354/


a connection. Jake requested pictures of deer.
User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed conne ctions" page. Eve (administrator) wan ts to set server's master key to "148 35038534". Isabel wants pages about " snakes but not too long". User Karen wants to change account password to " CoHoBaSt". User Amber requests pages

source: xkcd.com/1354/

```
if (hbtype == TLS_HB_REQUEST) {
    ...
    memcpy(bp, pl, payload);
    ...
}
```

```
/* Read type and payload length first */
if (1 + 2 + payload + 16 > s->s3->rrec.length):
    return 0; /* discard */
...
if (hbtype == TLS_HB_REQUEST) {
    /* receiver side : */
    /* replies with TLS1_HB_RESPONSE */
}
else if (hbtype == TLS_HB_RESPONSE) {
    /* sender side */
}
```

February 8, 2012: Heartbeat extension is proposed.

March 14, 2012: OpenSSL 1.0.1 is released with Heartbeat extension.

March 21, 2014: Neel Mehta of Google Security discovers Heartbleed vulnerability. Google commits a patch for the flaw and notifies some infrastructure providers under embargo.

March 31, 2014: CloudFlare is informed of Heartbleed and they remove the heartbeat functionality.

April 1, 2014: Google Security notifies OpenSSL team members about the flaw.

April 2, 2014: Finnish IT security testing firm Codenomicon separately discovers the same bug. It notifies the National Cyber Security Centre Finland (NCSC).

April 5, 2014: Codenomicon purchases the Heartbleed.com domain name, and later publishes information about the vulnerability. Multiple companies (including Red Hat, Facebook) start getting leads on Heartbleed and they remove the functionality or add a patch.

April 6, 2014: NCSC asks for a CVE number "on a critical OpenSSL issue." Companies like Red Hat and Facebook get updates on the bug through individuals.

April 7, 2014: A patch for Heartbleed is committed to OpenSSL's Git repository. OpenSSL publishes security advisory and blog posts, tweets, and articles on Heartbleed flood the internet.

April 14, 2014: Guardian reports that a forum with 1.5 million users called Mumsnets is impacted by Heartbleed. A hacker reportedly breached the admin's user account. Canada Revenue Agency announces social insurance numbers of approximately 900 taxpayers were removed by exploiting Heartbleed.

why static analysis fails

the way data is stored and referenced

complexity of following the execution path

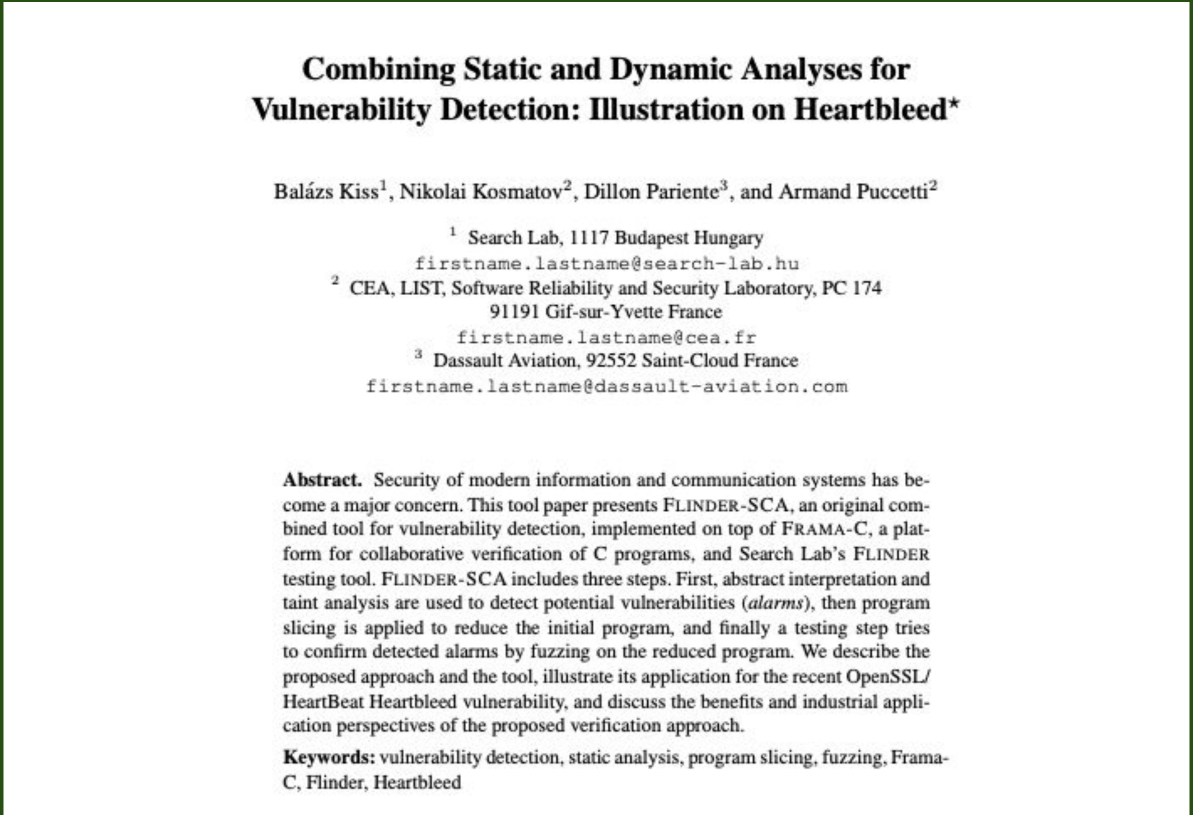
difficulty of identifying the specific parts in the storage structure that are misused

difficulty of determining whether a part has become untainted

why fuzzing fails

choice of memory management used by OpenSSL prevents dynamic testing to detect a memory corruption or over-read problem.

Hardware and Software: Verification and Testing 2015



Combining Static and Dynamic Analyses for Vulnerability Detection: Illustration on Heartbleed*

Balázs Kiss[1], Nikolai Kosmatov[2], Dillon Pariente[3], and Armand Puccetti[2]

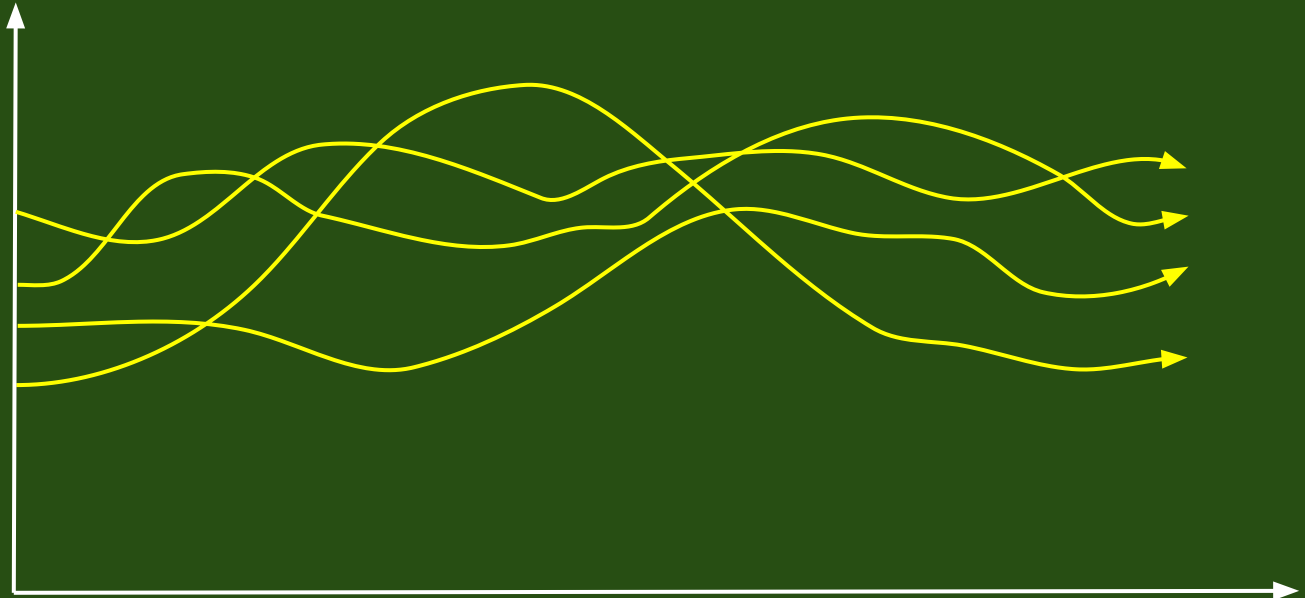
[1] Search Lab, 1117 Budapest Hungary
firstname.lastname@search-lab.hu
[2] CEA, LIST, Software Reliability and Security Laboratory, PC 174
91191 Gif-sur-Yvette France
firstname.lastname@cea.fr
[3] Dassault Aviation, 92552 Saint-Cloud France
firstname.lastname@dassault-aviation.com

Abstract. Security of modern information and communication systems has become a major concern. This tool paper presents FLINDER-SCA, an original combined tool for vulnerability detection, implemented on top of FRAMA-C, a platform for collaborative verification of C programs, and Search Lab's FLINDER testing tool. FLINDER-SCA includes three steps. First, abstract interpretation and taint analysis are used to detect potential vulnerabilities (*alarms*), then program slicing is applied to reduce the initial program, and finally a testing step tries to confirm detected alarms by fuzzing on the reduced program. We describe the proposed approach and the tool, illustrate its application for the recent OpenSSL/HeartBeat Heartbleed vulnerability, and discuss the benefits and industrial application perspectives of the proposed verification approach.

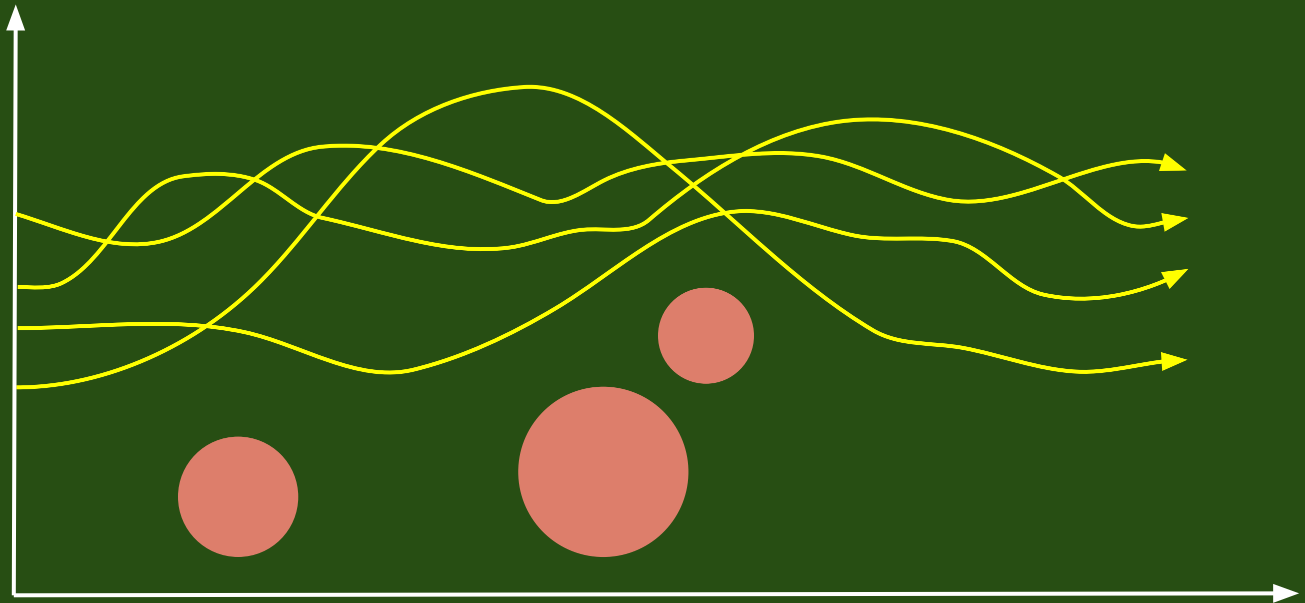
Keywords: vulnerability detection, static analysis, program slicing, fuzzing, Frama-C, Flinder, Heartbleed

abstract interpretation

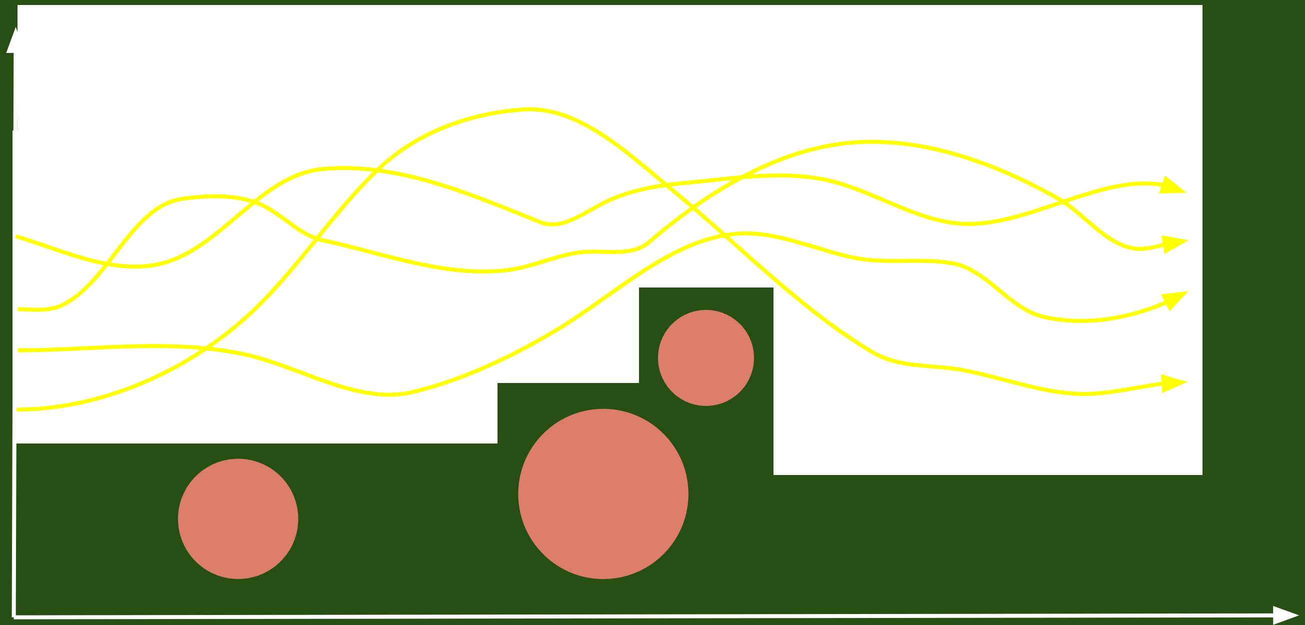
abstract interpretation



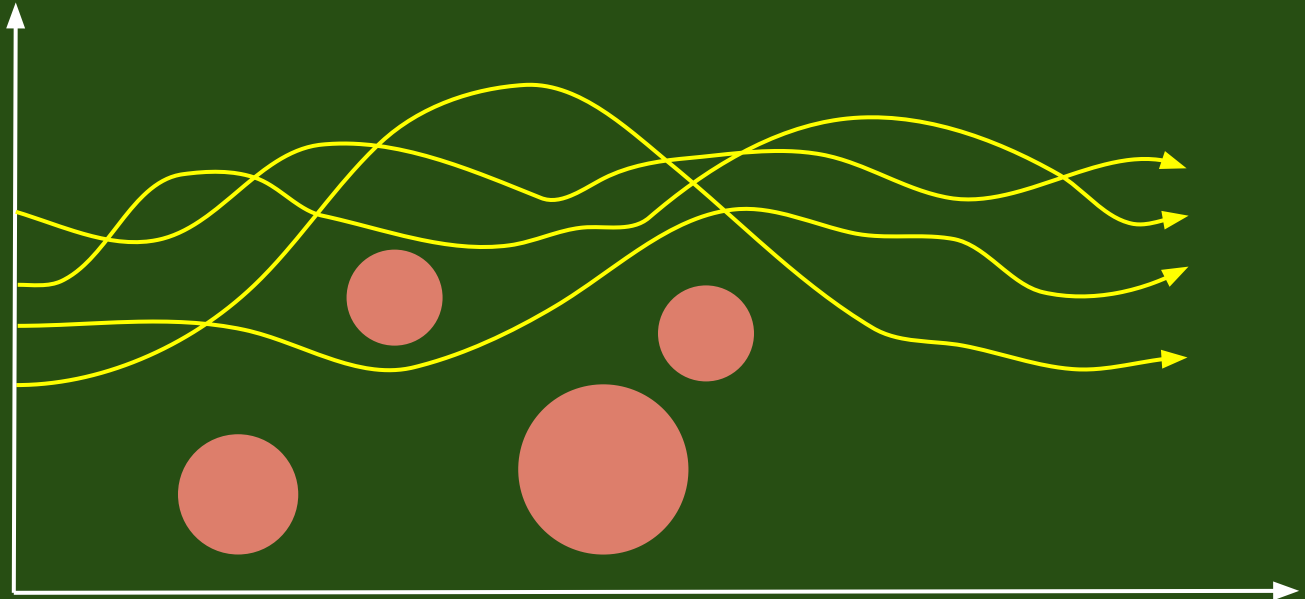
abstract interpretation



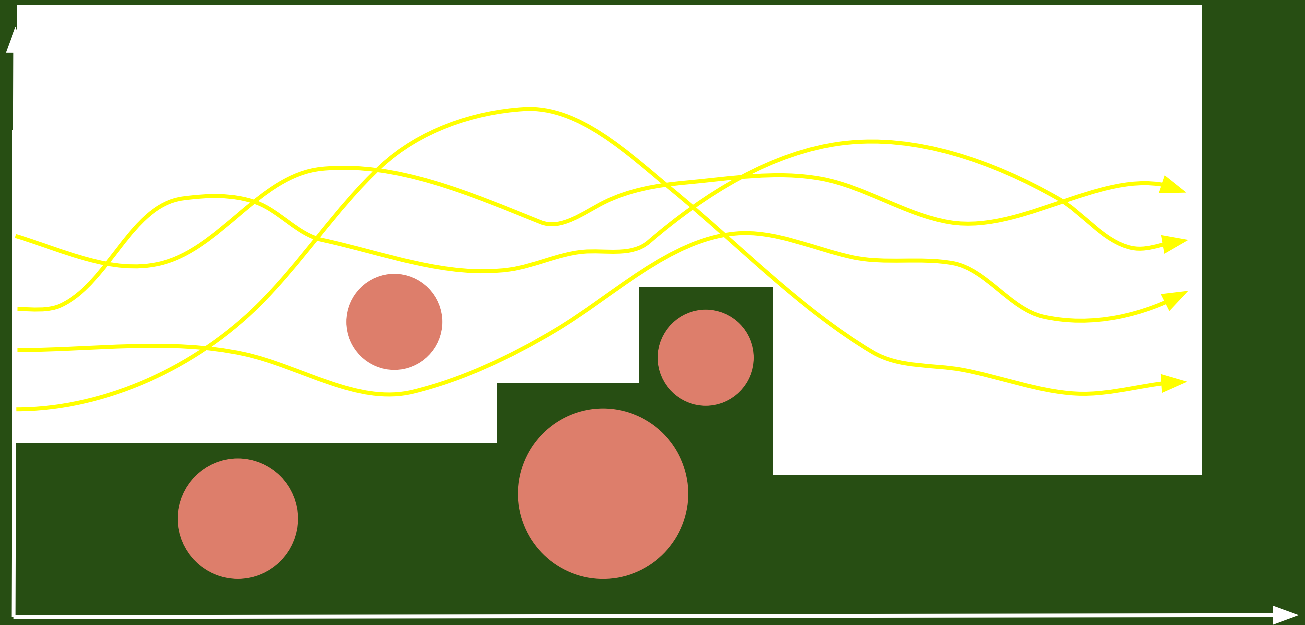
abstract interpretation



abstract interpretation



abstract interpretation



sound.
not complete.

abstract interpretation

```
// assert A1 : mem_access: \valid(bp[0 .. (payload -1)]);
// assert A2 : mem_access: \valid(pl[0 .. (payload -1)]);
...
```

Using Runtime Error Annotation Generation

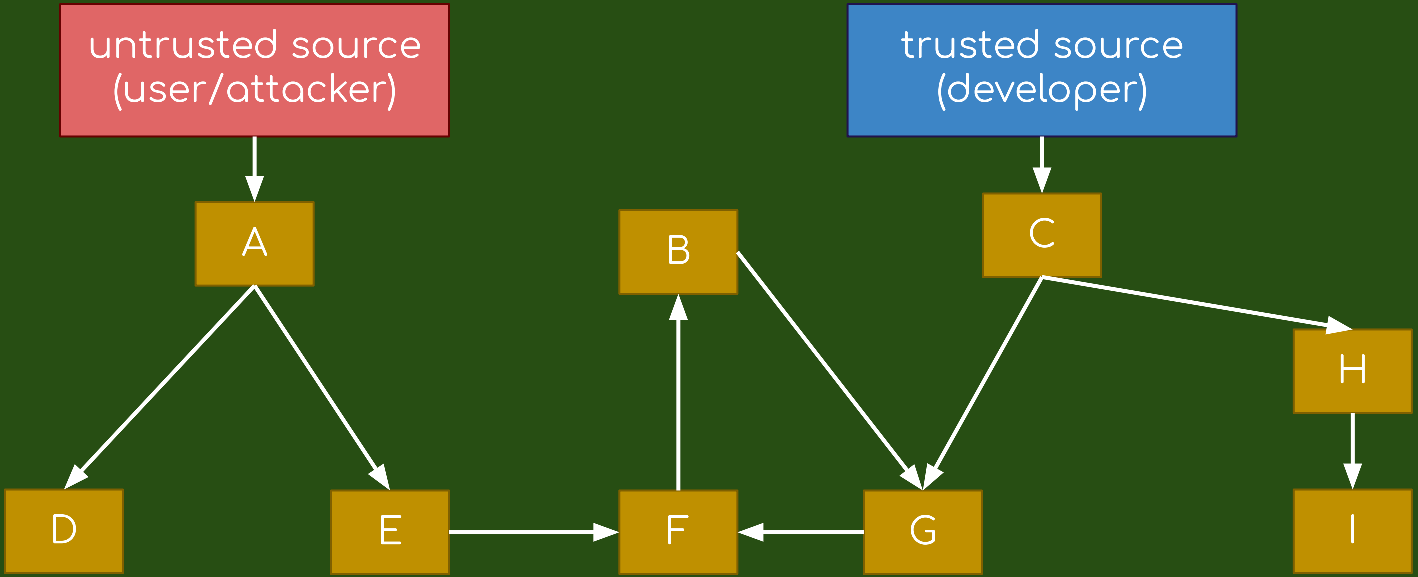
abstract interpretation

```
// assert A1 : mem_access: \valid(bp[0 .. (payload -1)]);
// assert A2 : mem_access: \valid(pl[0 .. (payload -1)]);
...
```

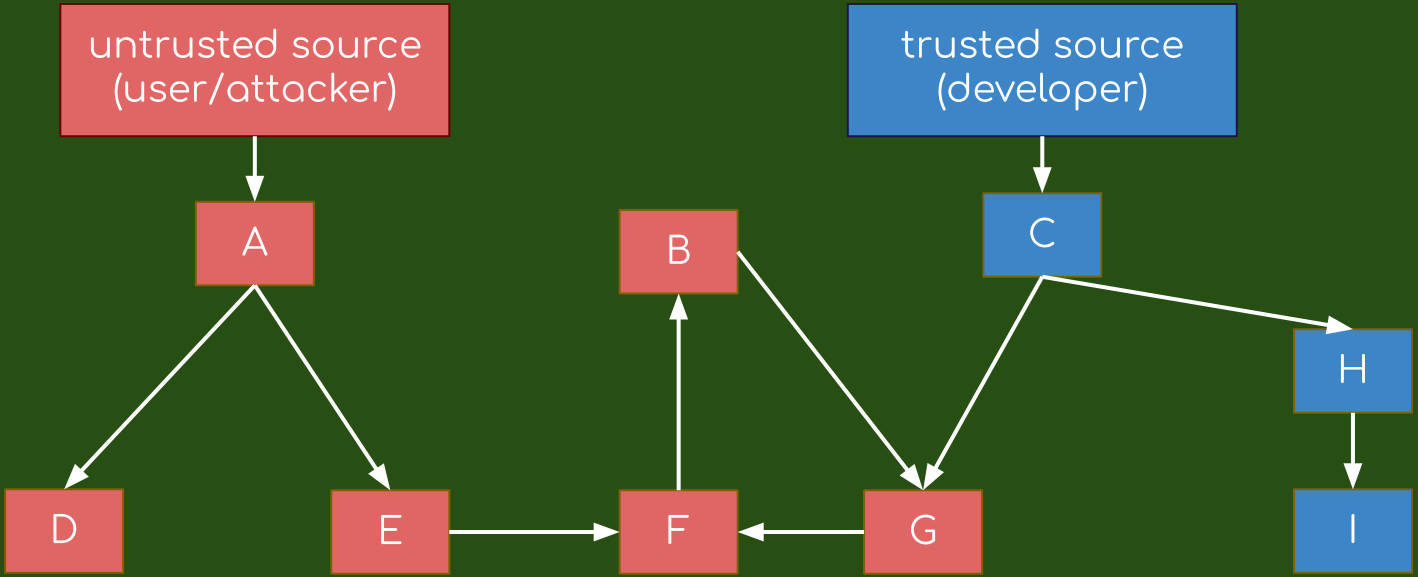
Using Runtime Error Annotation Generation

but, are all the assertions relevant?

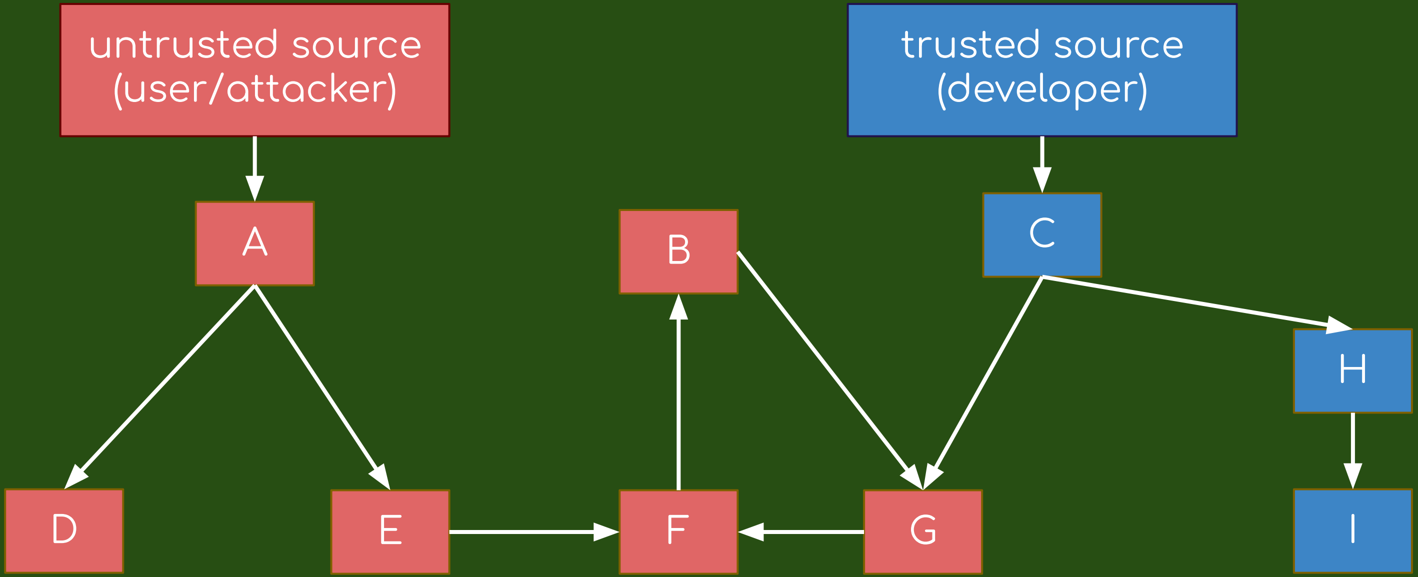
taint analysis



taint analysis



taint analysis



user specifies the potentially taintable inputs (rrec.data) and potentially vulnerable functions (e.g.memcpy).

tool reports that the assertions related to memcpy call handle the taintable input flow

program slicing

program slicing

```
read (n)

i := 1
sum := 0
product := 1

while i <= n do
    sum := sum + i
    product := product * i
    i := i + 1

write (sum)
write (product)
```

program slicing

```
read (n)

i := 1
sum := 0
product := 1

while i <= n do
    sum := sum + i
    product := product * i
    i := i + 1

write (sum)
write (product)
```

program slicing

```
read (n)

i := 1
sum := 0
product := 1

while i <= n do
    sum := sum + i
    product := product * i
    i := i + 1

write (sum)
write (product)
```

program slicing

```
1.   read (n)

2.   i := 1
3.   sum := 0
4.   product := 1

5.   while i <= n do
6.        sum := sum + i
7.        product := product * i
8.        i := i + 1

9.   write (sum)
10.  write (product)
```

program slicing

```
1.   read (n)

2.   i := 1

4.   product := 1

5.   while i <= n do

7.       product := product * i
8.       i := i + 1


10.  write (product)
```

program slicing

```
1.   read (n)

2.   i := 1

4.   product := 1

5.   while i <= n do

7.       product := product * i
8.       i := i + 1


10.  write (product)
```

Original program:
8 defined functions
51 lines of code

After Slicing:
2 defined functions
38 lines of code

fuzz testing

fuzz testing (white box)

Generate list of fuzzing parameters.

Compile the annotated code and feed it to the tool.

Test vectors are generated according to the fuzzing parameters.

Each test vector is sent to the test harness. The test harness detects the alarms.

The tool decides whether the vulnerability is confirmed or not.

how to prevent?

how to prevent?

deep spec

how to prevent?

deep
spec

how to prevent?

Simplify the code and its API such that errors are apparent. Complex code impede formal methods.

Avoid program-specific allocation or memory caching systems. Subdividing memory can thwart analysis.

Use standard licensing to allow more code reviews and contributions.

how to reduce impact?

Overwrite, erase, or destroy sensitive information as quickly as possible

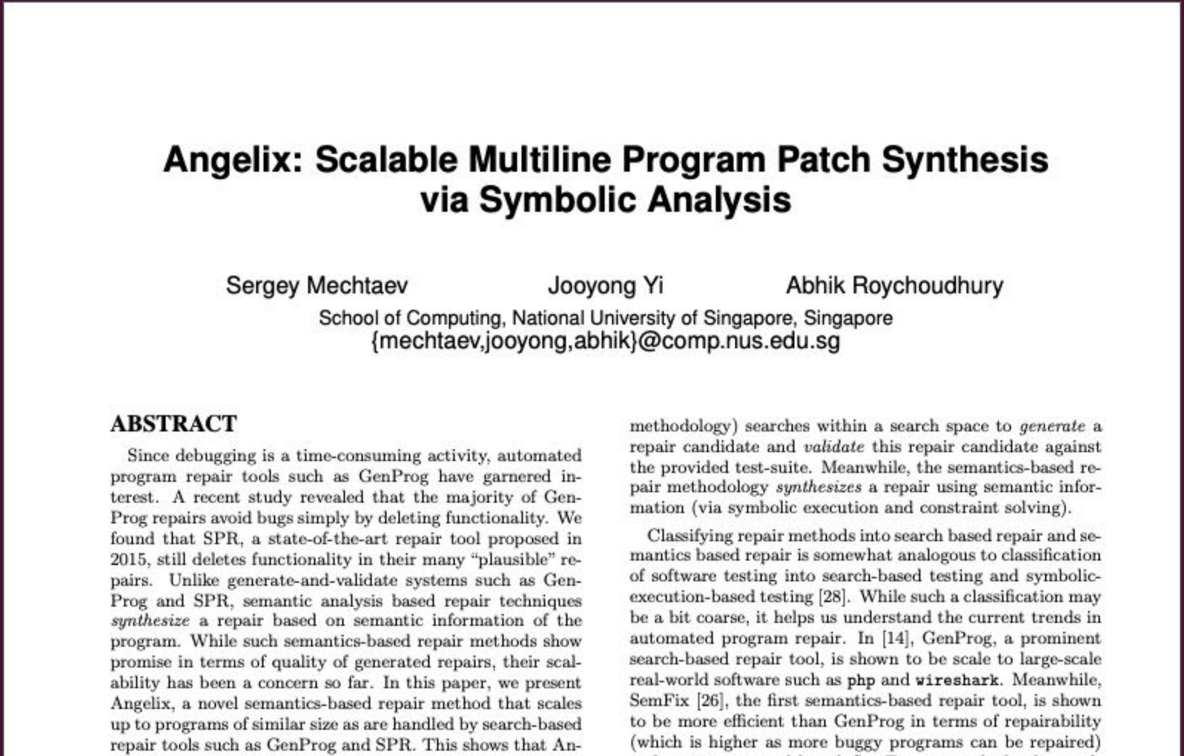
Use perfect forward security (PFS) encryption. That is, non-deterministically generate new random public keys for each session

Separate secrets from the rest of the code

Software updates should be easy and required

Reduce incentives for attackers

International Conference on Software Engineering (2016)



Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis

Sergey Mechtaev Jooyong Yi Abhik Roychoudhury
School of Computing, National University of Singapore, Singapore
{mechtaev,jooyong,abhik}@comp.nus.edu.sg

ABSTRACT

Since debugging is a time-consuming activity, automated program repair tools such as GenProg have garnered interest. A recent study revealed that the majority of GenProg repairs avoid bugs simply by deleting functionality. We found that SPR, a state-of-the-art repair tool proposed in 2015, still deletes functionality in their many "plausible" repairs. Unlike generate-and-validate systems such as GenProg and SPR, semantic analysis based repair techniques *synthesize* a repair based on semantic information of the program. While such semantics-based repair methods show promise in terms of quality of generated repairs, their scalability has been a concern so far. In this paper, we present Angelix, a novel semantics-based repair method that scales up to programs of similar size as are handled by search-based repair tools such as GenProg and SPR. This shows that An-

methodology) searches within a search space to *generate* a repair candidate and *validate* this repair candidate against the provided test-suite. Meanwhile, the semantics-based repair methodology *synthesizes* a repair using semantic information (via symbolic execution and constraint solving).

Classifying repair methods into search based repair and semantics based repair is somewhat analogous to classification of software testing into search-based testing and symbolic-execution-based testing [28]. While such a classification may be a bit coarse, it helps us understand the current trends in automated program repair. In [14], GenProg, a prominent search-based repair tool, is shown to be scale to large-scale real-world software such as `php` and `wireshark`. Meanwhile, SemFix [26], the first semantics-based repair tool, is shown to be more efficient than GenProg in terms of repairability (which is higher as more buggy programs can be repaired)

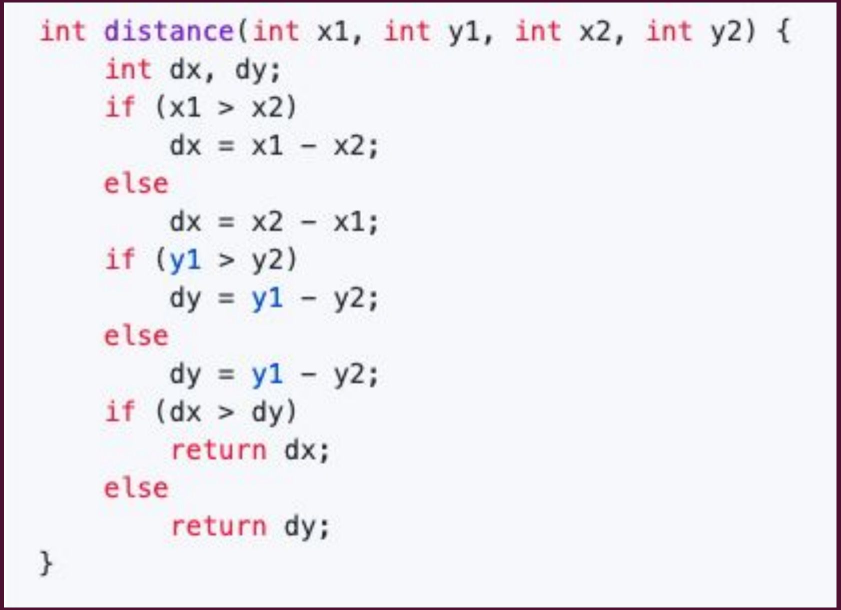
constraint-solving based synthesis

```
for test $t$ in test suite $T$:
   compute repair constraint $\varphi_t$
let ($\varphi = \bigwedge_T \varphi_t$)
synthesize $e$ as a solution for $\varphi$
```

constraint-solving based synthesis

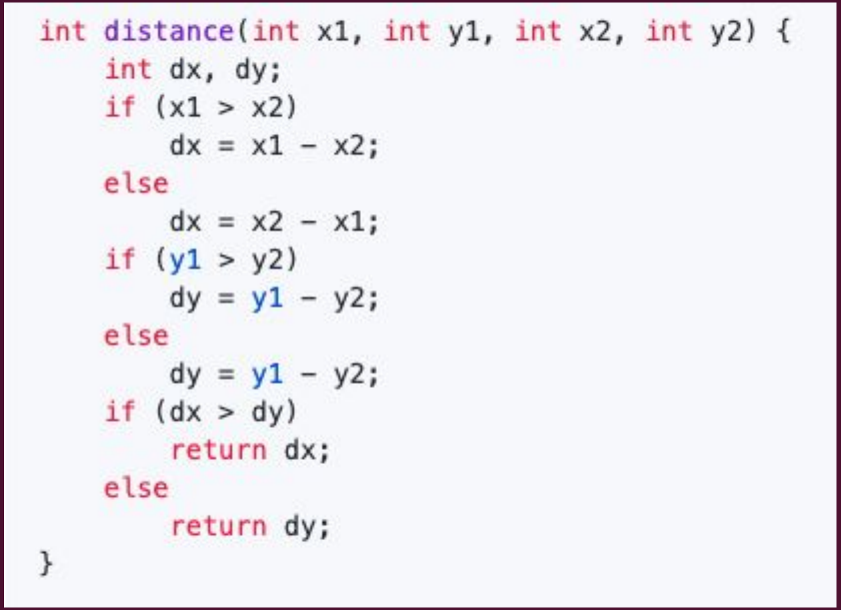
```
for test $t$ in test suite $T$:
    compute repair constraint $\varphi_t$    how?
let ($\varphi = \bigwedge_T \varphi_t$)
synthesize $e$ as a solution for $\varphi$    how?
```

an example



```c
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = y1 - y2;
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

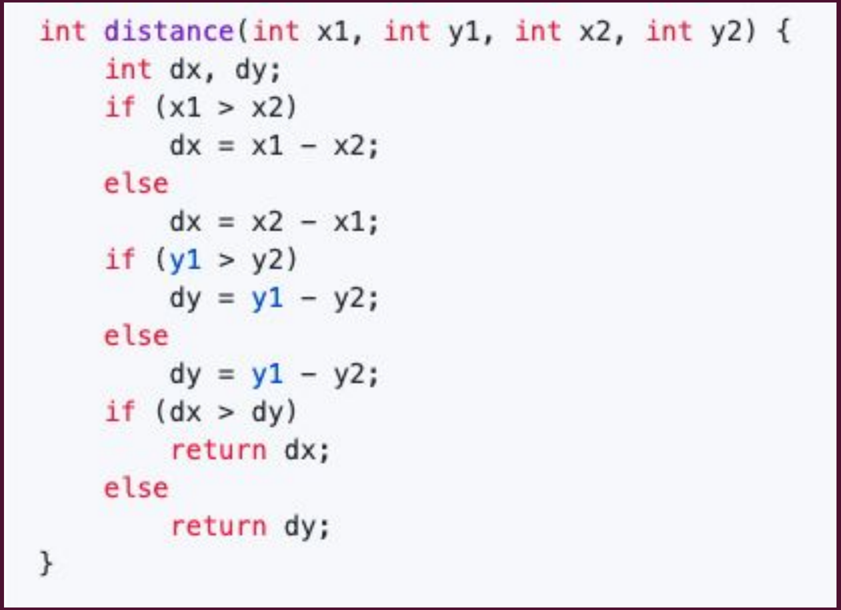
an example



```
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = y1 - y2;
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

distance(0,1,0,5)?

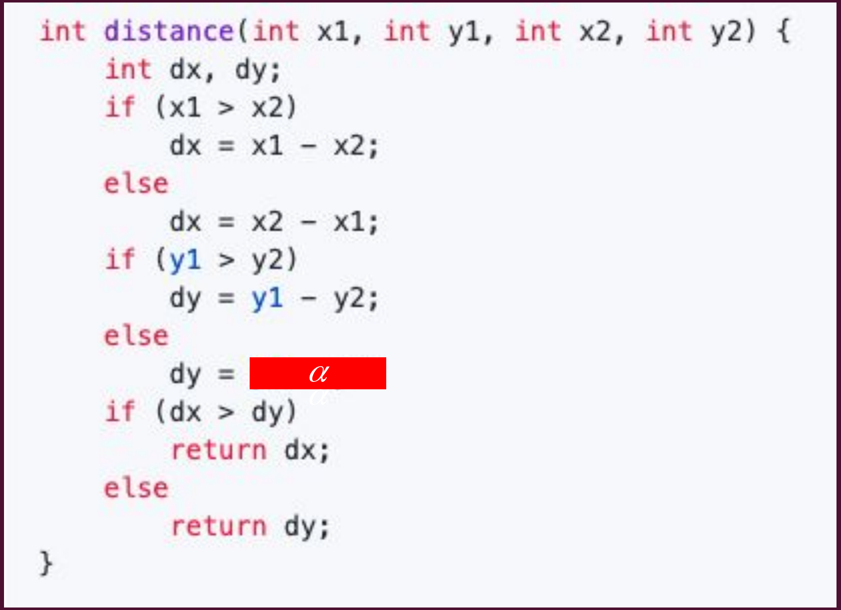
an example



```
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = y1 - y2;
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

x1	y1	x2	y2	expected	observed
0	1	0	5	4	0
1	2	4	2	2	2
3	3	3	0	3	0
0	4	0	4	0	0

an example

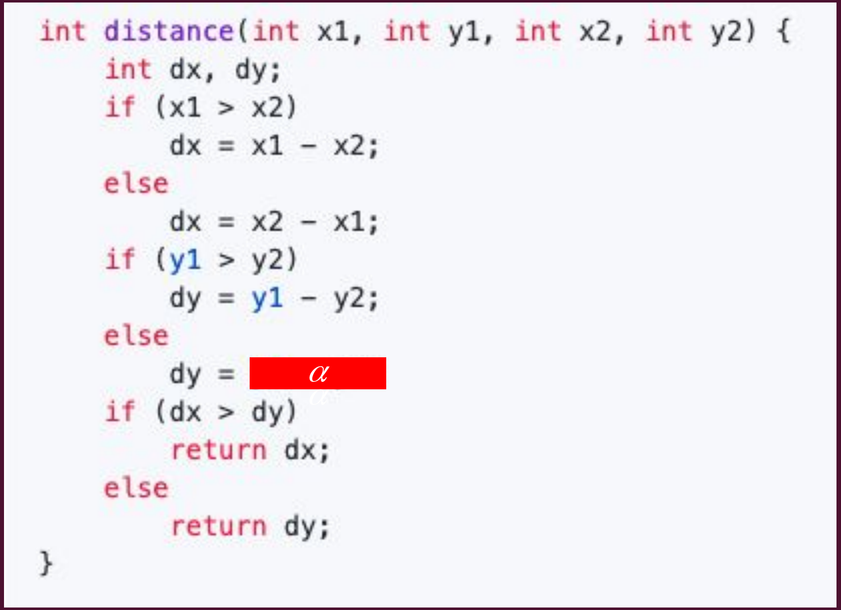


```
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = α
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

x1	y1	x2	y2	expected	observed
0	1	0	5	4	0

pc = (y1 > y2), α

an example

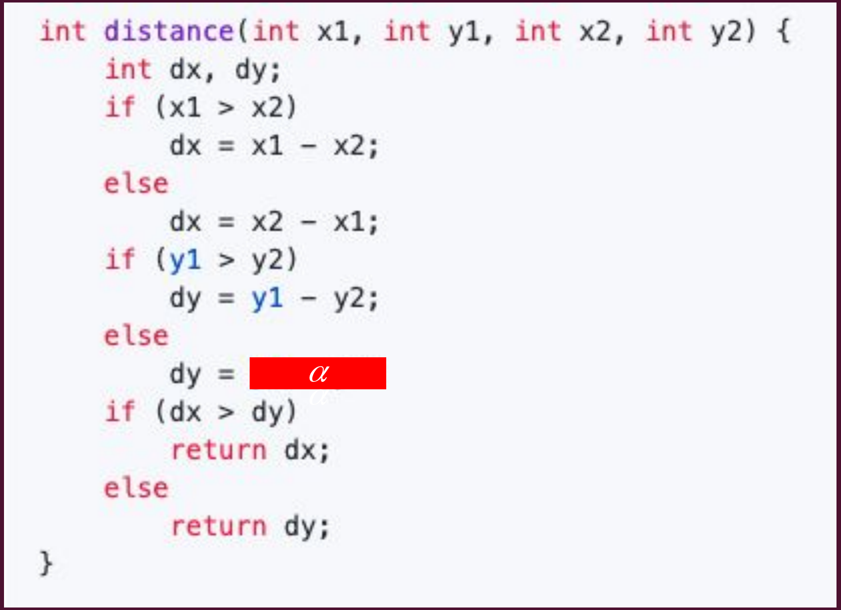


```
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = α
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

x1	y1	x2	y2	expected	observed
0	1	0	5	4	0

$$\begin{tikzcd}
\boxed{pc = (y1 > y2),\ \alpha} \arrow[d] \\
\boxed{\begin{array}{c} pc = (y1 > y2)\ \wedge \\ (dx > \alpha),\ \alpha \end{array}}
\end{tikzcd}$$

an example



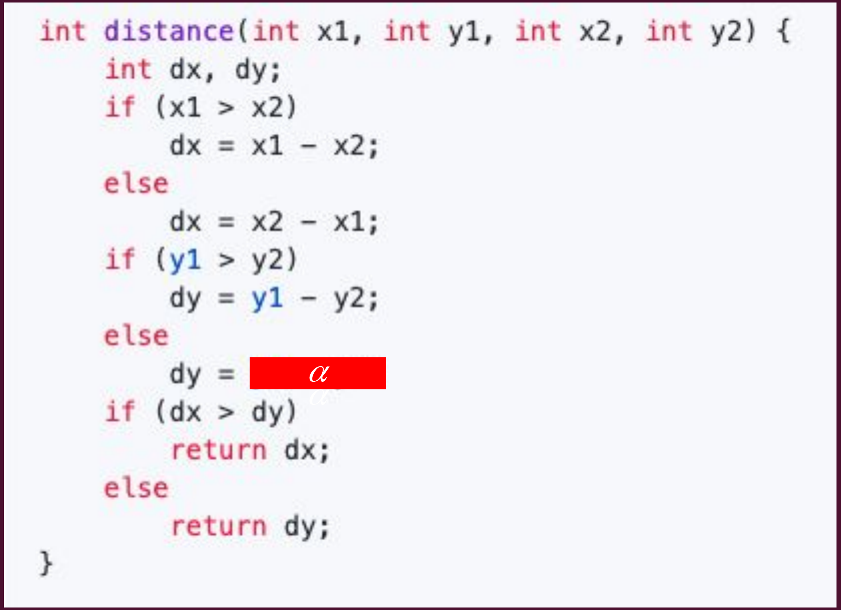
```
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = α
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

x1	y1	x2	y2	expected	observed
0	1	0	5	4	0

$$\begin{tikzcd}
 & \boxed{pc = (y1 \leq y2), \alpha} \arrow[dl] \arrow[dr] & \\
\boxed{\begin{array}{c} pc = (y1 \leq y2) \wedge \\ (dx > \alpha), \alpha \end{array}} & & \boxed{\begin{array}{c} pc = (y1 \leq y2) \wedge \\ (dx \leq \alpha), \alpha \end{array}}
\end{tikzcd}$$

What should α be?

an example



```
int distance(int x1, int y1, int x2, int y2) {
    int dx, dy;
    if (x1 > x2)
        dx = x1 - x2;
    else
        dx = x2 - x1;
    if (y1 > y2)
        dy = y1 - y2;
    else
        dy = α
    if (dx > dy)
        return dx;
    else
        return dy;
}
```

x1	y1	x2	y2	expected	observed
0	1	0	5	4	0
1	2	4	2	2	2
3	3	3	0	3	0
0	4	0	4	0	0

$$(\alpha(0, 1, 0, 5) = 4) \land$$
$$((\alpha(1, 2, 4, 2) < 2) \lor (\alpha(1, 2, 4, 2) = 2)) \land$$
$$(\alpha(3, 3, 3, 0) = 3) \land$$
$$((\alpha(0, 4, 0, 4) < 0) \lor (\alpha(0, 4, 0, 4) = 0))$$

angelic forest generation

angelic value

angelic path

angelic forest

a succinct representation.

independent of the size of the program.

only related to number of expressions considered symbolic

(increment)

patch code generation

input and output of components are associated with location variables

components are connected when the location variable of one component has the same value as the location variable of another

range constraint

consistency constraint

acyclicity constrain

connections constraint

generated patch

```
if (hbtype == TLS_HB_REQUEST
    && payload + 18 < s->s3->rrec.length) {
    /* receiver side : */
        /* replies with TLS1_HB_RESPONSE */
}
```

```
/* Read type and payload length first */
if (1 + 2 + payload + 16 > s->s3->rrec.length):
    return 0; /* discard */
...
if (hbtype == TLS_HB_REQUEST) {
    /* receiver side : */
    /* replies with TLS1_HB_RESPONSE */
}
else if (hbtype == TLS_HB_RESPONSE) {
    /* sender side */
}
```

future work

Combining Static Analysis and Testing

Formal Specification of Security Properties

Safe by Design Programming Languages

Automated Program Repair