Contemporary Automatic Program Analysis

Julian Cohen

HockeyInJune@gmail.com

@HockeyInJune

Julian Cohen | http://hake.co/

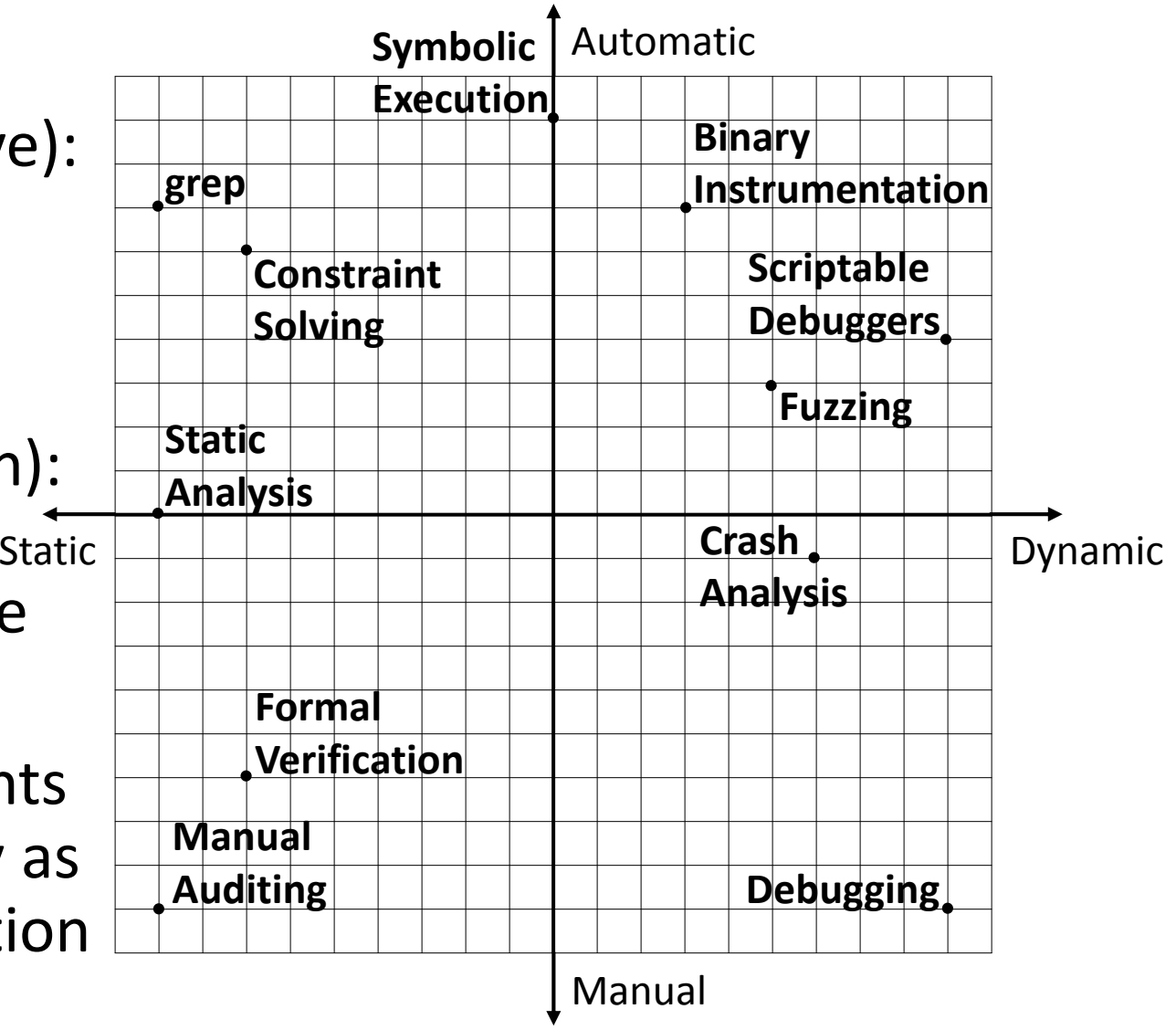
- Application Security | Large Organization
- NYU Polytechnic School of Engineering | http://engineering.nyu.edu/
- Cyber Security Awareness Week | https://csaw.isis.poly.edu/
- ISIS Laboratory | http://www.isis.poly.edu/
- Ghost in the Shellcode | http://ghostintheshellcode.com/
- Moderator | /r/netsec | /r/vrd
- NYSEC | https://twitter.com/nysecsec

Cyber Security Awareness Week

- https://csaw.isis.poly.edu/
- @CSAW_NYUPoly | https://www.facebook.com/NYUPolyCSAW
- Student-run @ NYU Polytechnic School of Engineering
- Six Competitions: CTF | HSF | ESC | Research | Policy | Quiz
- Cyber Security Career Fair
- THREADS: Security Automation
- Downtown Brooklyn, New York

Automatic Program Analysis

- au·to·mat·ic /ˌôtəˈmatik/ (adjective): (of a device or process) working by itself with little or no direct human control
- pro·gram /ˈprōˌgram,-grəm/ (noun): a sequence of instructions that a computer can interpret and execute
- a·nal·y·sis /əˈnaləsis/ (*noun*): detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation

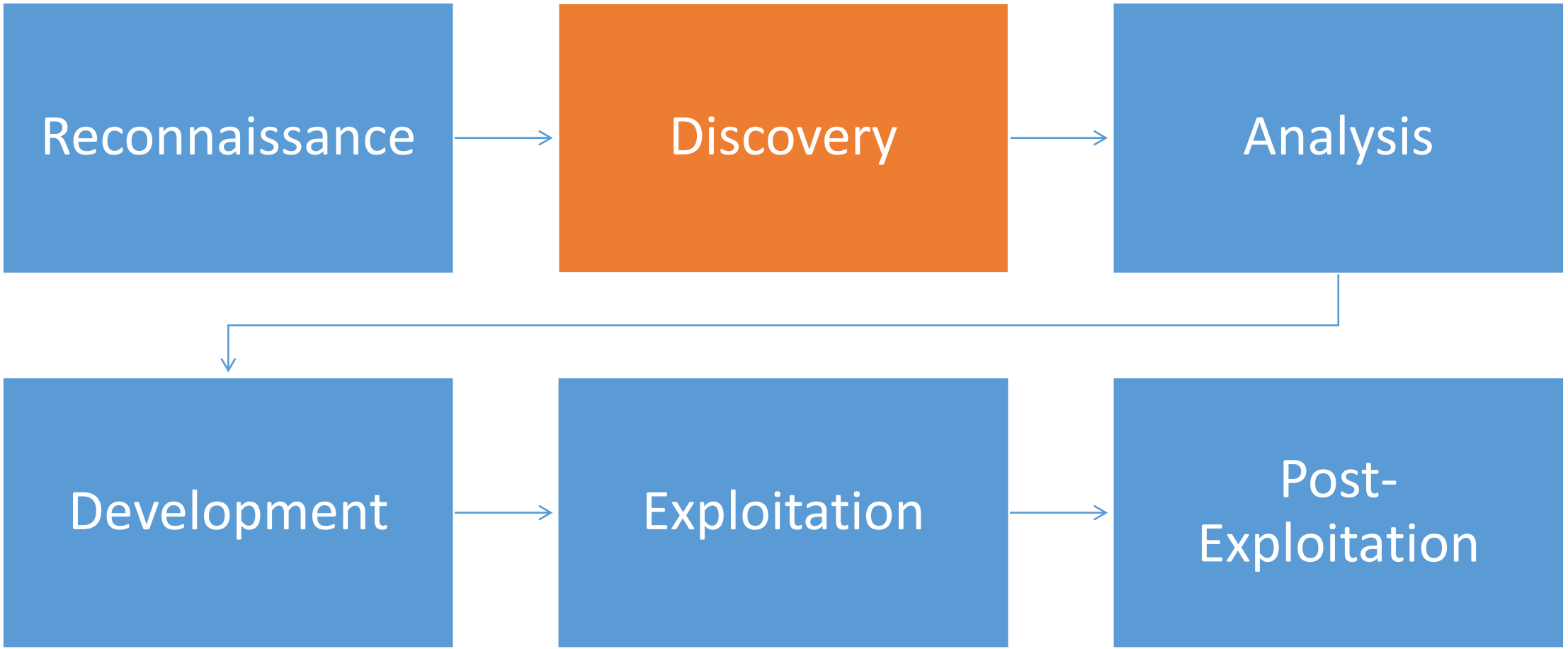


Ambiguity Disclosure

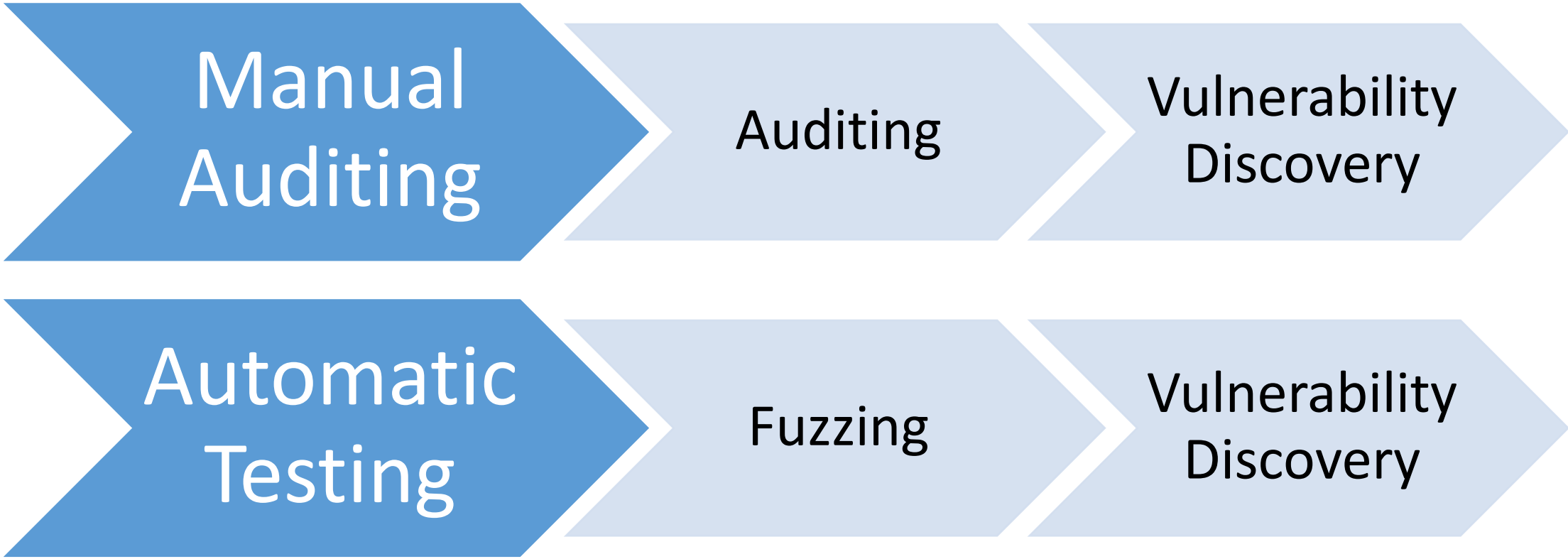
For better and for worse, I will be deliberately ambiguous at times

Use your imagination

Vulnerability Research



Vulnerability Discovery



How can we use program analysis to discover the highest impact vulnerabilities at the lowest cost?

Metacharacter Injection Vulnerabilities

- Cross-Site Scripting

```
<h2>Welcome back, <script>alert(1);</script>!</h2>
```

- SQL Injection

```
SELECT * FROM users WHERE user="admin" AND pass="1" OR "1"="1"
```

- Command Injection

```
cp /tmp/e1zXr /var/www/assets/img-`rm -rf --no-preserve-root /`;
```

githubgrep.py (87 lines)

- githubgrep is a very simple script to automate Github code search

- User provides functions, input, and security keywords
- Creates code search queries based on those keywords
- Outputs how many vulnerabilities each search yields with a link

- Code search only allows a certain number of modifiers per query
- Low false-positive rate, taking into account sanitizer keywords

https://github.com/HockeyInJune/Contemporary-Automatic-Program-Analysis

githubgrep.py

```
./githubgrep.py -language php
-functions
"exec,passthru,shell_exec,system,popen"
-sources "\$_REQUEST,\$_GET,\$_POST"
-sanitizers "escapeshellcmd,escapeshellarg"
```

```
TOTAL NUMBER OF VULNERABILITIES FOUND: 1540130
```

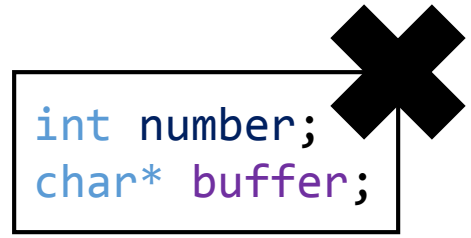
githubgrep.py

```
./githubgrep.py -language java
-functions "write,print,println,out"
-sources
"getHeader,getHeaders,getQueryString,
getRequestURI,getRequestURL,
getParameter,getParameterValues"
-sanitizers "encode,sanitize,validate"
```

```
TOTAL NUMBER OF VULNERABILITIES FOUND:
755536
```

Type Confusion Vulnerabilities

```
union {
      int number;
      char *buffer;
} u;
u.number = recv();
strncpy(u.buffer, input, sizeof(u.buffer));
```



grep (1 line)

```
hij@vm:~/WebKit/Source/WebCore/css/$ grep -A 4 union CSSParser.cpp
    union {
        double fValue;
        int iValue;
        CSSParserString string;
        CSSParserFunction *function;
```

- Very high false-positive rate, effective only when you know exactly what to look for

http://blog.leafsr.com/2012/06/27/webkit-css-type-confusion/

Implicit Type Conversion Vulnerabilities

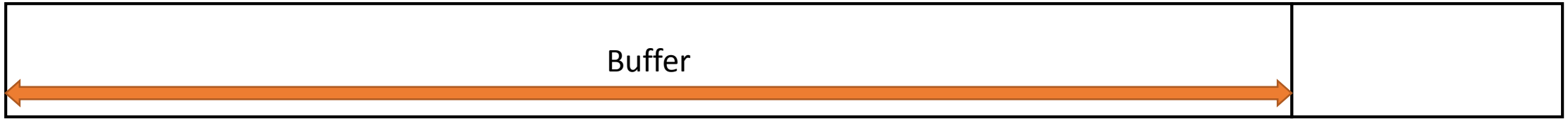
```
char buffer[128];
int size = recv();

if (size < 100) {
     strncpy(buffer, input, size);
}
```

gcc -Wconversion (1 line)

```
hij@vm:~$ gcc -Wconversion implicit.c
warning: conversion to ‘size_t’ from ‘int’ may
change the sign of the result [-Wsign-conversion]
   strncpy(buffer, input, size);
   ^
```

Buffer Overflow Vulnerabilities




```
char buffer[128];
strcpy(buffer, input);
```

RATS: Rough Auditing Tool for Security

- RATS uses simple static analysis to find potentially vulnerable code
- C, C++, Perl, PHP, Python, and Ruby

- Parses and tokenizes source code
- Checks variables, identifiers, and comments against rules

- High false-positive rate, but every finding is usually worth looking into

https://code.google.com/p/rough-auditing-tool-for-security/

RATS: Rough Auditing Tool for Security

```
hij@vm:~/rough-auditing-tool-for-security$ ./rats ../mozjpeg
Entries in c database: 334

../mozjpeg/wrjpgcom.c:456: High: strcpy
Check to be sure that argument 2 passed to this function call
will not copy more data than can be handled, resulting in a
buffer overflow.

../mozjpeg/wrjpgcom.c:466: High: strcat
Check to be sure that argument 2 passed to this function call
will not copy more data than can be handled, resulting in a
buffer overflow.
```

```
} else if ( keymatch(arg, "comment", 1) ) {
  comment_arg = argv[++argn];
  if (comment_arg[0] == '"') {
    comment_arg = (char *) malloc( (size_t) MAX_COM_LENGTH );
    strcpy(comment_arg, argv[argn] + 1);
    for (;;) {
      comm_l = (unsigned int) strlen(comment_arg);
      if (comm_l > 0 && comment_arg[comm_l - 1] == '"') {
        comment_arg[comm_l - 1] = '\0';
        break;
      }
      strcat(comment_arg, " ");
      strcat(comment_arg, argv[++argn]);
} } }
```

Valgrind memcheck

- Valgrind uses dynamic binary instrumentation to detect memory errors

- Rewrites code in memory in order instrument memory operations
- Reports any invalid memory operations

- Low false-positive rate, nearly every report is a bug
- But you need test cases to exercise each vulnerability

http://valgrind.org/

Valgrind memcheck

```
valgrind --trace-children=yes ./wrjpgcom -comment "A*70000" testorig.jpg
```

```
 Invalid write of size 1
    at 0x4C2BFFC: strcpy (in valgrind/vgpreload_memcheck-amd64-linux.so)
    by 0x40123E: main (in mozjpeg/.libs/lt-wrjpgcom)
  Address 0x5201e28 is 0 bytes after a block of size 65,000 alloc'd
    at 0x4C2B6CD: malloc (in valgrind/vgpreload_memcheck-amd64-linux.so)
    by 0x4011E5: main (in mozjpeg/.libs/lt-wrjpgcom)
```

Valgrind memcheck

```
valgrind --trace-children=yes ./wrjpgcom -comment "A*60000 A*10000 testorig.jpg
```

```
Invalid write of size 1
   at 0x4C2BCCC: strcat (in valgrind/vgpreload_memcheck-amd64-linux.so)
   by 0x401324: main (in mozjpeg/.libs/lt-wrjpgcom)
 Address 0x5201e28 is 0 bytes after a block of size 65,000 alloc'd
   at 0x4C2B6CD: malloc (in valgrind/vgpreload_memcheck-amd64-linux.so)
   by 0x4011E5: main (in mozjpeg/.libs/lt-wrjpgcom)

Invalid write of size 1
   at 0x4C2BCDF: strcat (in valgrind/vgpreload_memcheck-amd64-linux.so)
   by 0x401324: main (in mozjpeg/.libs/lt-wrjpgcom)
 Address 0x52031b1 is not stack'd, malloc'd or (recently) free'd
```

quicksec (275 lines)

- quicksec is a simple static analysis tool for native code
  - Written by Kevin Chung, a student in my class

- Similar to RATS, it parses and looks for function calls, but in binaries
- It also attempts to determine the vulnerability of its findings

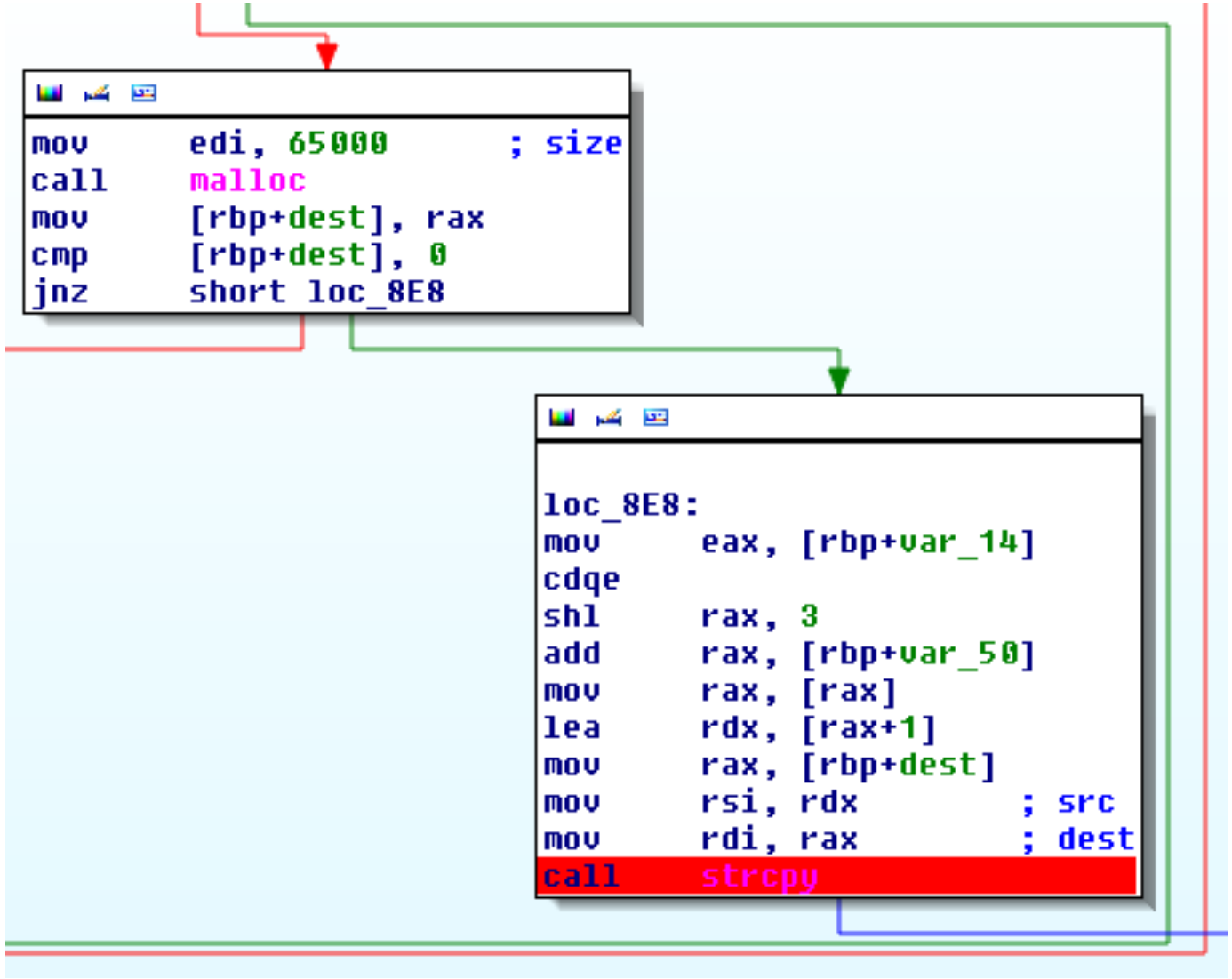
- Low false-positive rate, and every finding is worth looking into

https://github.com/ColdHeat/quicksec

quicksec

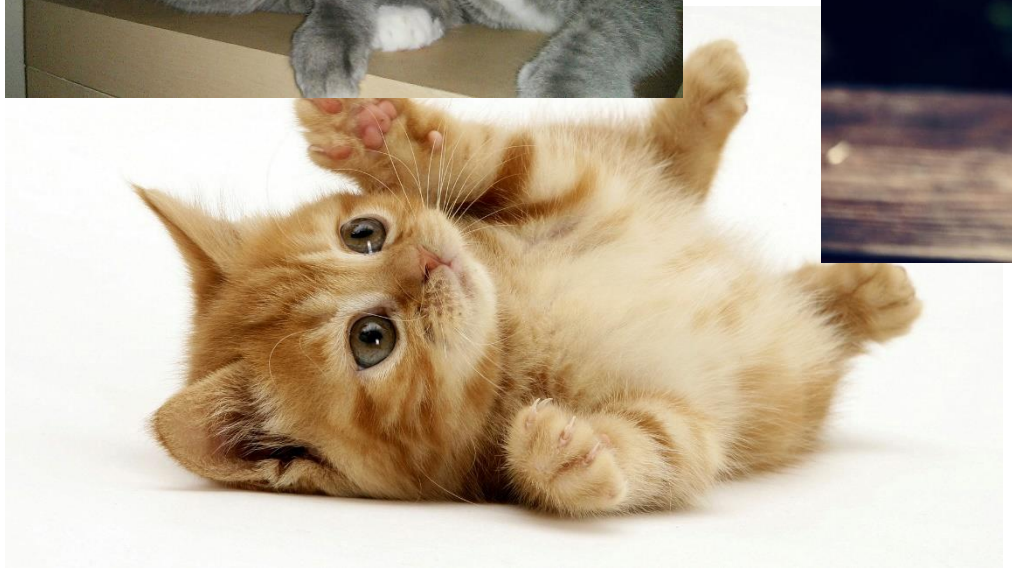
- Output:

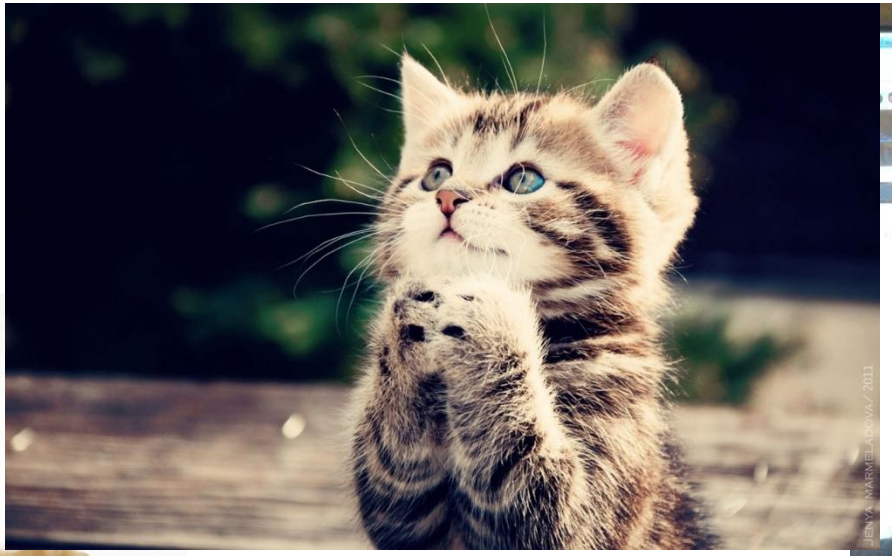
```
gets []
fgets []
strcpy [2310]
recv []
printf []
read []
```

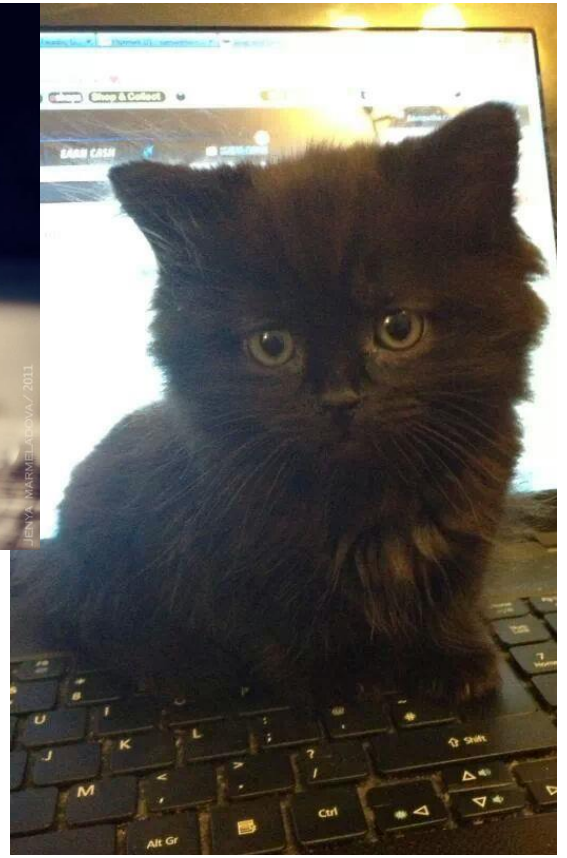


Kitten Interlude

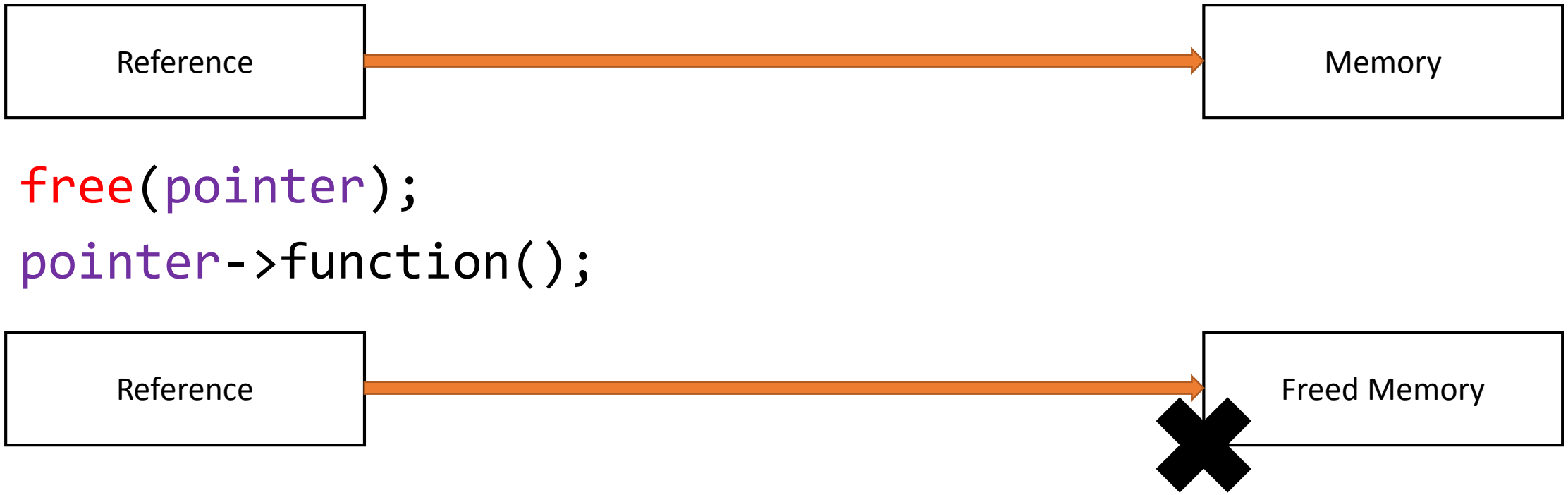
Brought you by Brad Antoniewicz, Joshua Drake, and ancat!





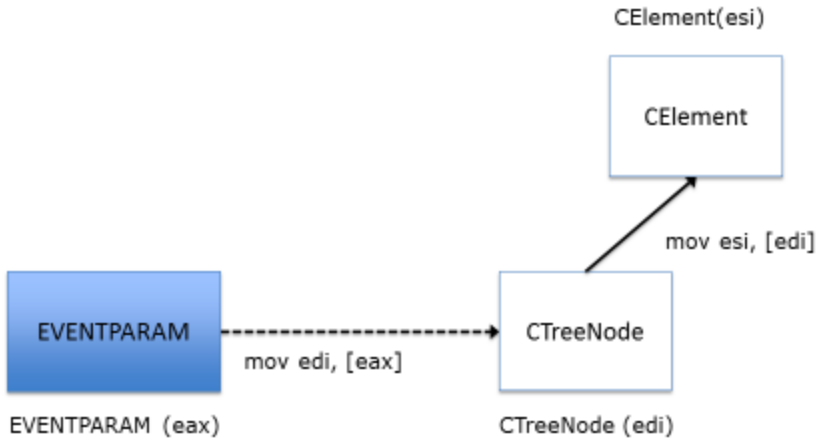


Use-After-Free Vulnerabilities



The Aurora Use-After-Free Vulnerability

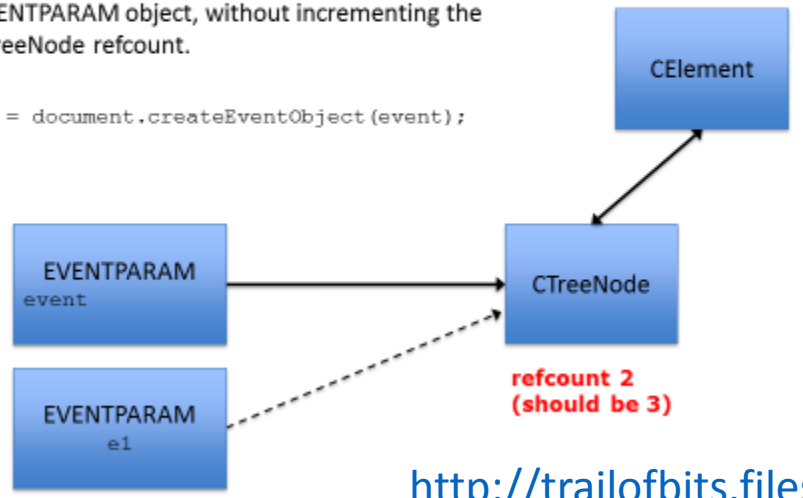
The Aurora Vulnerability



The Aurora Vulnerability

The event handler function creates a copy of the EVENTPARAM object, without incrementing the CTreeNode refcount.

```
el = document.createEventObject(event);
```



EVENTPARAM Copy Cor

What is wrong with this copy constructor?

```
.text:74E4C892 ; public: __thiscall EVENTPARAM::EVENTPARAM(st
.text:74E4C892
.text:74E4C892 arg_0           = dword ptr  8
.text:74E4C892 arg_4           = dword ptr  0Ch
.text:74E4C892
.text:74E4C892                 mov     edi, edi
.text:74E4C894                 push    ebp
.text:74E4C895                 mov     ebp, esp
.text:74E4C897                 push    ebx
.text:74E4C898                 mov     ebx, [ebp+arg_0]
.text:74E4C89B                 push    esi
.text:74E4C89C                 mov     esi, [ebp+arg_4]
.text:74E4C89F                 push    edi
...
.text:74E4C8D5                 push    3Eh
.text:74E4C8D7                 pop     ecx
.text:74E4C8D8                 mov     edi, ebx
.text:74E4C8DA                 rep movsd
```

http://trailofbits.files.wordpress.com/2010/04/practical-rop.pdf

aurorauaf.py (62 lines)

- aurorauaf checks a binary for compiler-generated copy constructors

- Uses IDAPython to find copy constructors and demangle their names
- Visual C++ uses `rep movsd` for shallow-copy copy constructors

- Low false-positive rate, but high false-negative rate
- Copy constructors are very often inlined by Visual C++ compiler

https://github.com/HockeyInJune/Contemporary-Automatic-Program-Analysis

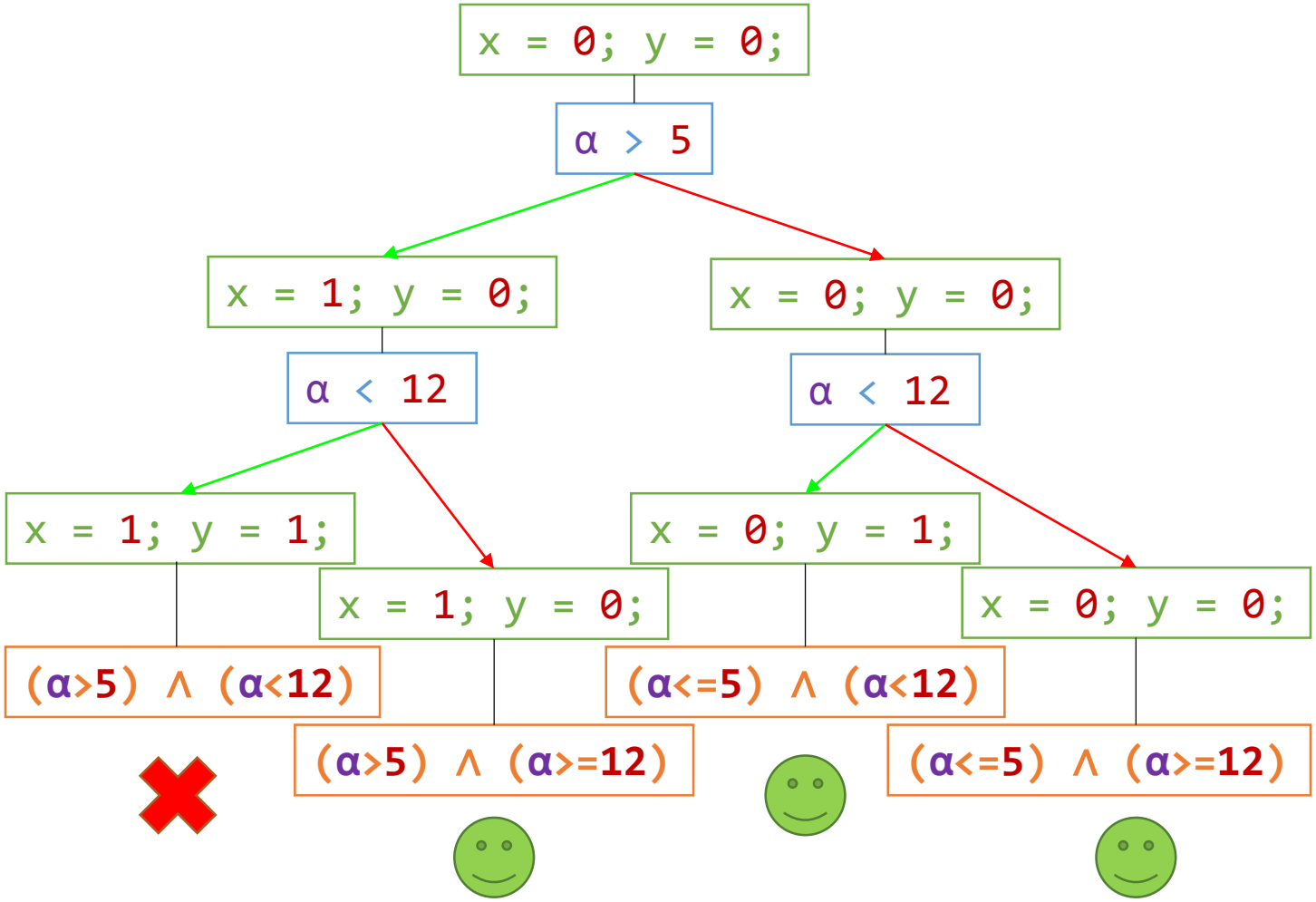
aurorauaf.py

- Output:

```
EVENTPARAM::EVENTPARAM(EVENTPARAM const *) looks
compiler generated!  Check it out at 0x74e4c892

CEditEvent::CEditEvent(CEditEvent const *) looks
compiler generated!  Check it out at 0x750bbfc2
```

Symbolic Execution




```
int a = α; //symbolic
int x = 0, y = 0;
if ( a > 5 ) {
      x = 1;
}
if ( a < 12 ) {
      y = 1;
}
assert( x + y != 2 );
```

http://www.seas.harvard.edu/courses/cs252/2011sp/slides/Lec13-SymExec.pdf

Constraint Solving

- z3, theorem prover, Lisp s-expression interpreter

$$(\alpha > 5) \wedge (\alpha < 12)$$ ✖

```
(declare-const a Int)
(assert (> a 5))
(assert (< a 12))
(check-sat)
(get-model)
```

```
sat
(model
  (define-fun a () Int
    6)
)
```

http://z3.codeplex.com/

LLVM KLEE

- KLEE is an automatic test case generator built on top of LLVM

- Uses symbolic execution to explore potential states of a program
- Constraint solving to generate test cases to increase code coverage

- Excellent at increasing code coverage and generating test cases
- Requires source code or LLVM bitcode

http://klee.github.io/klee/

LLVM KLEE

```
#include <klee/klee.h>
int main(int argc, char** argv) {
      int a; //symbolic
      klee_make_symbolic(&a, sizeof(a), "a");
      int x = 0, y = 0;
      if ( a > 5 ) { x = 1; }
      if ( a < 12 ) { y = 1; }
      if ( ( x + y ) == 2 ) { return -1; }
      else { return 0; }
}
```

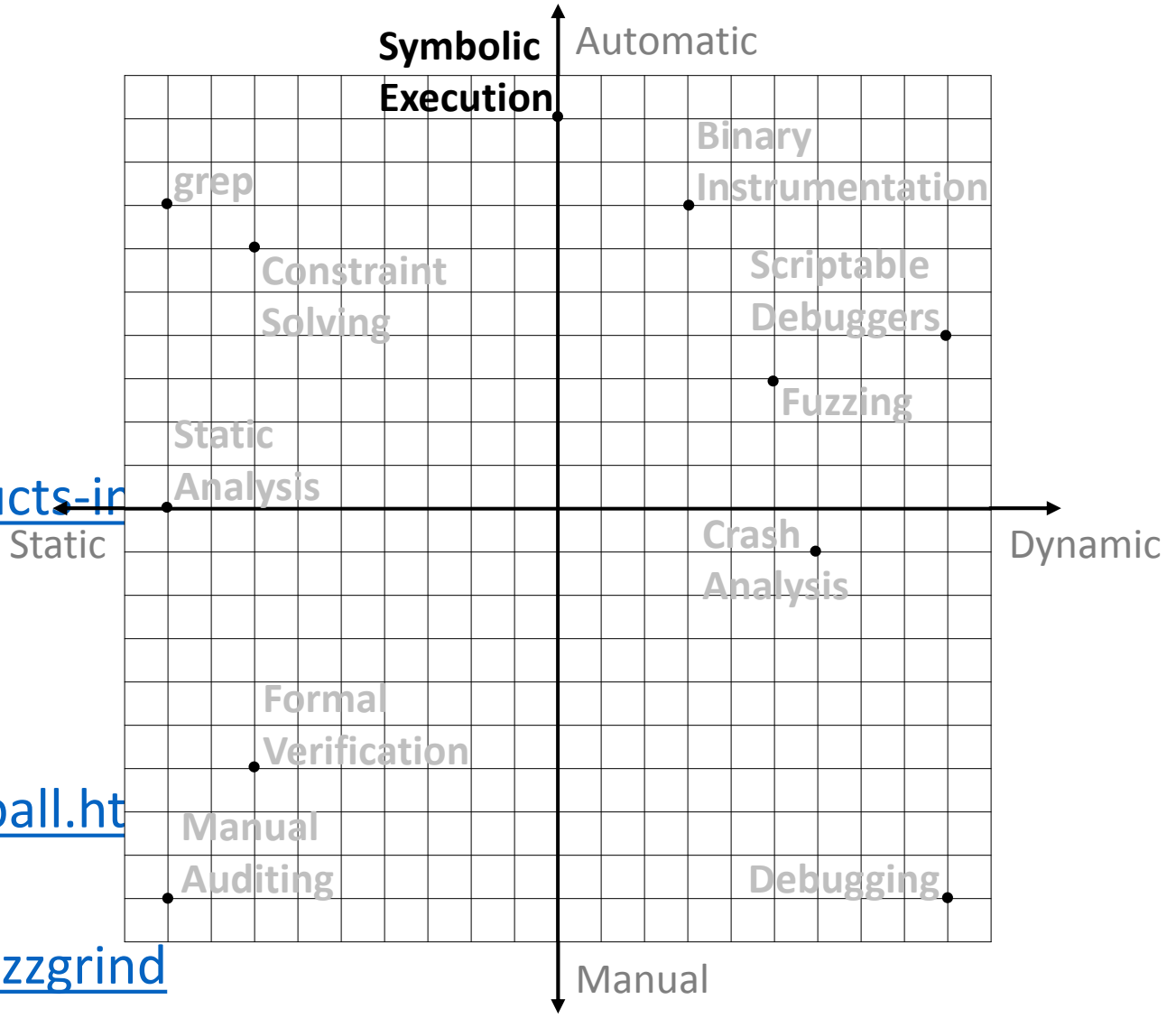
LLVM KLEE

```
u@ec2:~/klee/examples/demo$ klee demo.o
KLEE: output directory is “klee/examples/demo/klee-out-0"
KLEE: done: total instructions = 62
KLEE: done: completed paths = 3
KLEE: done: generated tests = 3
u@ec2:~/klee/examples/demo$ ktest-tool test000001.ktest
object    0: name: 'a'
object    0: size: 4
object    0: data: 0
```

LLVM KLEE Demo

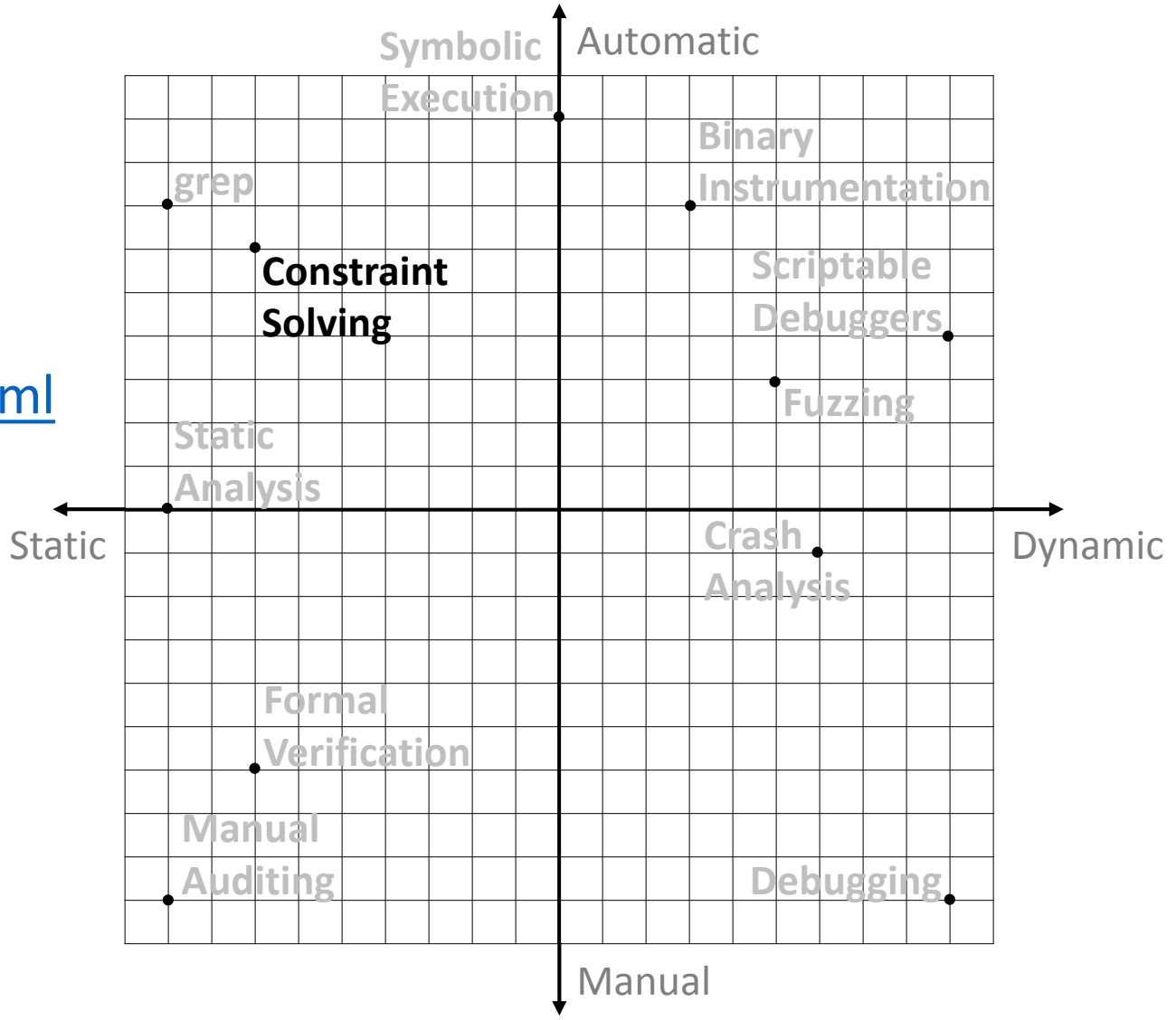
Symbolic Execution Engines

- S2E
  - http://s2e.epfl.ch/
- Clang Static Analyzer
  - http://clang-analyzer.llvm.org/
- Immunity Debugger
  - http://www.immunityinc.com/products-ir
- KLEE
  - http://klee.github.io/klee/
- FuzzBALL
  - http://bitblaze.cs.berkeley.edu/fuzzball.ht
- Fuzzgrind
  - http://esec-lab.sogeti.com/pages/Fuzzgrind



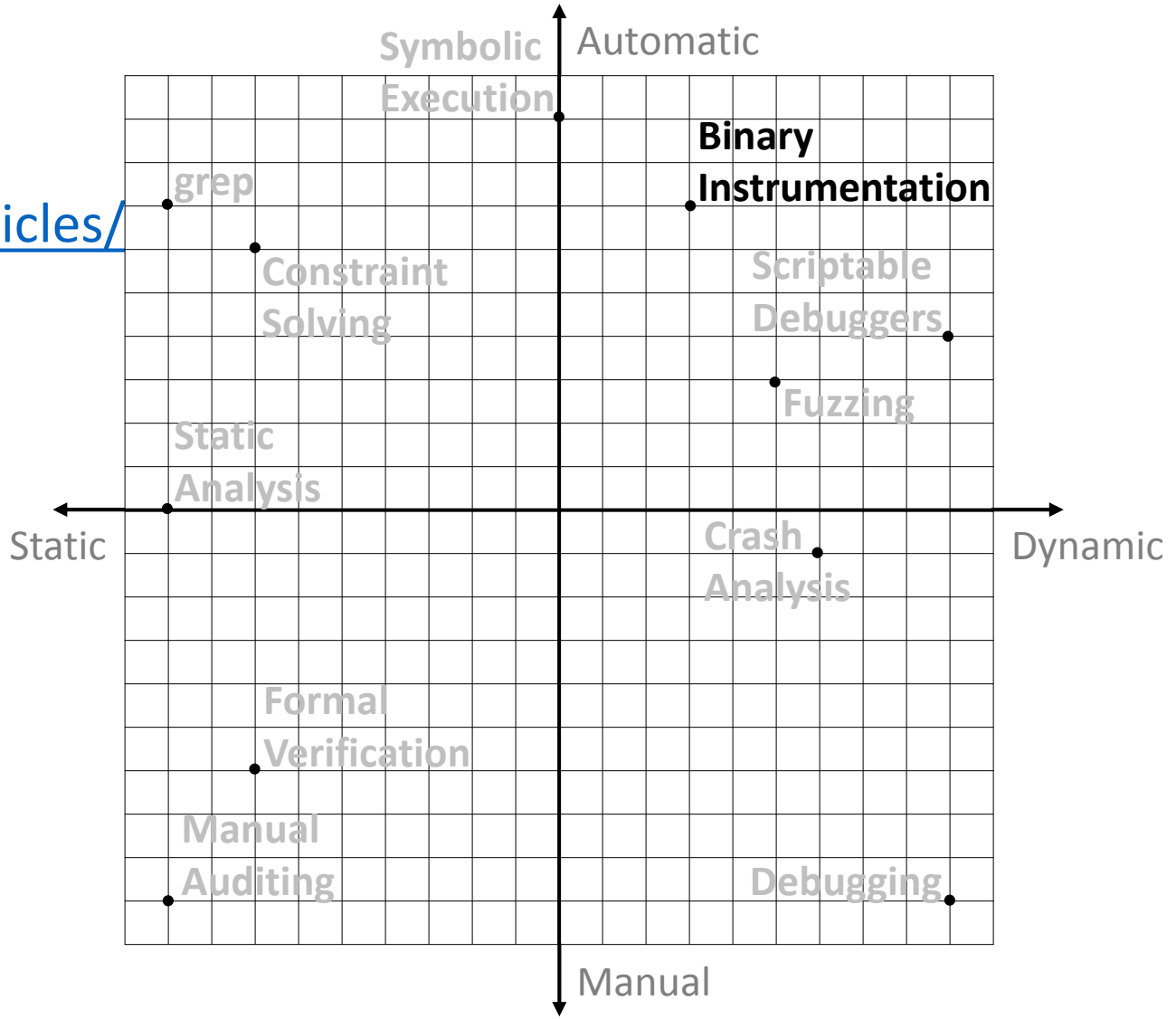
Constraint Solvers

- z3
  - http://z3.codeplex.com/
- Kleaver
  - http://klee.github.io/klee/KQuery.html
- STP
  - http://stp.github.io/stp/
- CVC4
  - http://cvc4.cs.nyu.edu/web/
- Yices 2
  - http://yices.csl.sri.com/



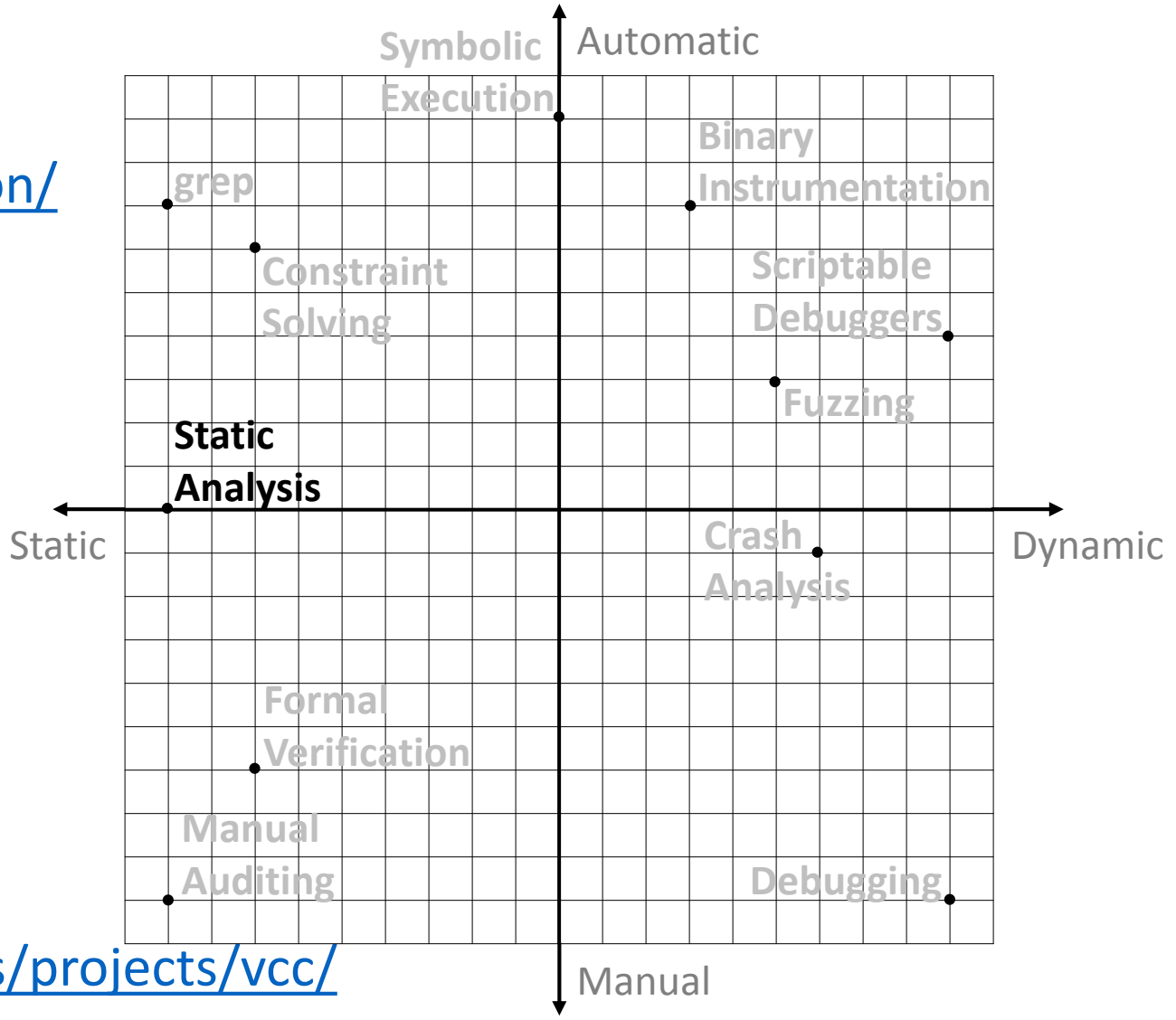
Binary Instrumentation Engines

- PIN
  - https://software.intel.com/en-us/articles/
- DynamoRIO
  - http://dynamorio.org/
- Valgrind
  - http://valgrind.org/



Static Analysis Platforms

- IDA Python
  - https://code.google.com/p/idapython/
- BAP: Binary Analysis Platform
  - http://bap.ece.cmu.edu/
- Insight
  - http://insight.labri.fr/trac
- Jakstab
  - http://www.jakstab.org/
- Stack
  - http://css.csail.mit.edu/stack/
- VCC: A Verifier for Concurrent C
  - http://research.microsoft.com/en-us/projects/vcc/



Conclusions

Program analysis is a set of powerful, low-cost techniques

If you do vulnerability discovery, invest in program analysis

Thanks

Andrew Ruef

Apneet Jolly

Ben Nell

Chris Surage

John Villamil

Julien Vanegue

#bap on Freenode

Students of CS 9223 at NYU Polytechnic School of Engineering

Special Thanks

Alexander Sotirov

Brandon Edwards

Chris Rohlf

Dan Guido

Dino Dai Zovi

Erik Cabetas

Geri Del Priore

John Terrill

Jordan Wiens

Look at all the bugs we discovered!
Do you have any questions about them?