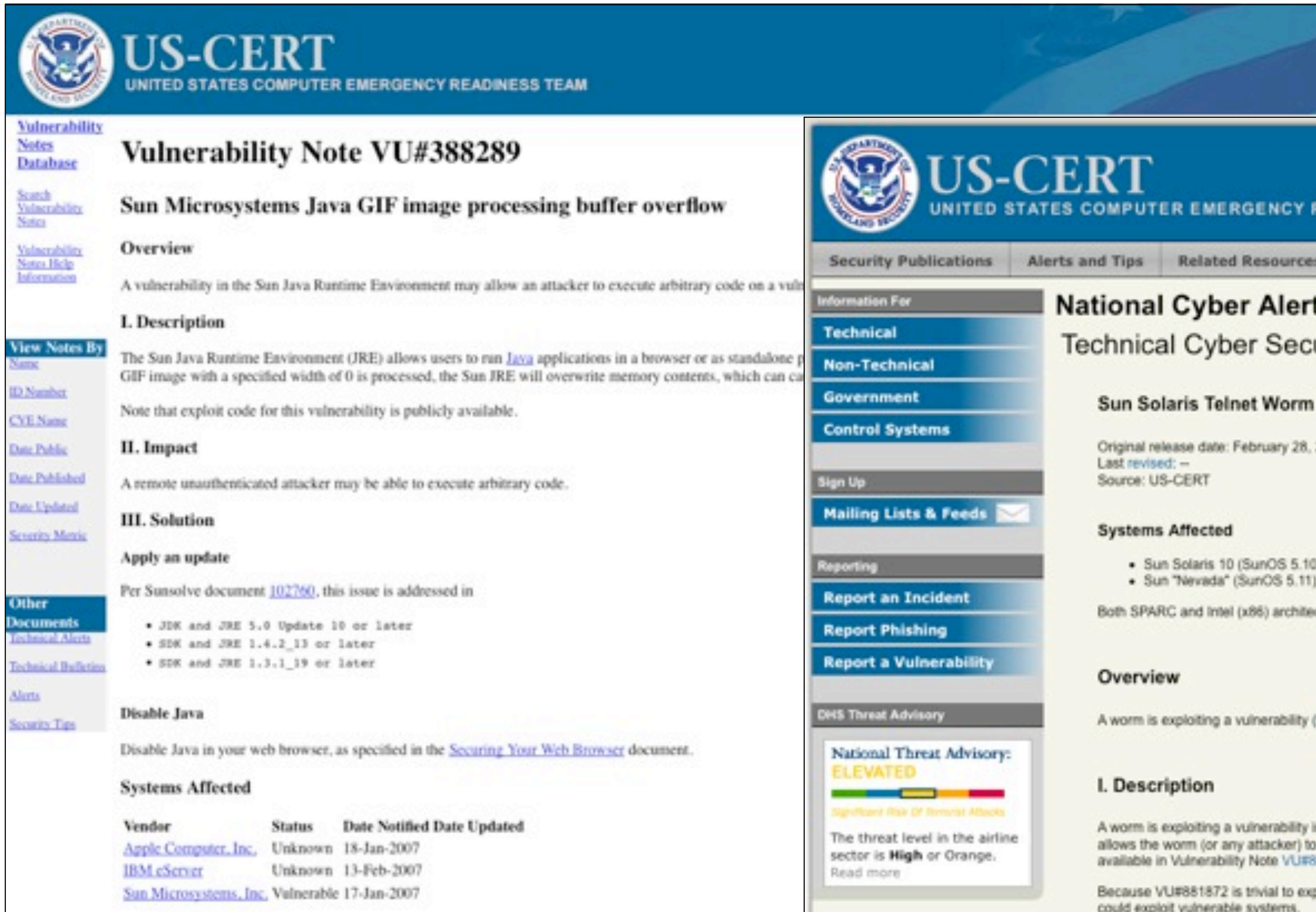
The Parfait Bug-Checker

Cristina Cifuentes, Nathan Keynes,
Lian Li

Sun Microsystems Laboratories
Brisbane, Australia

2 October, 2009

Bugs are Part of Life



US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Vulnerability Notes Database
Search Vulnerability Notes
Vulnerability Notes Help Information
View Notes By
Name
ID Number
CVE Name
Date Public
Date Published
Date Updated
Severity Metric
Other Documents
Technical Alerts
Technical Bulletins
Alerts
Security Tips

Vulnerability Note VU#388289

Sun Microsystems Java GIF image processing buffer overflow

Overview

A vulnerability in the Sun Java Runtime Environment may allow an attacker to execute arbitrary code on a vuln

I. Description

The Sun Java Runtime Environment (JRE) allows users to run Java applications in a browser or as standalone p
GIF image with a specified width of 0 is processed, the Sun JRE will overwrite memory contents, which can ca

Note that exploit code for this vulnerability is publicly available.

II. Impact

A remote unauthenticated attacker may be able to execute arbitrary code.

III. Solution

Apply an update

Per Sunsolve document 102760, this issue is addressed in

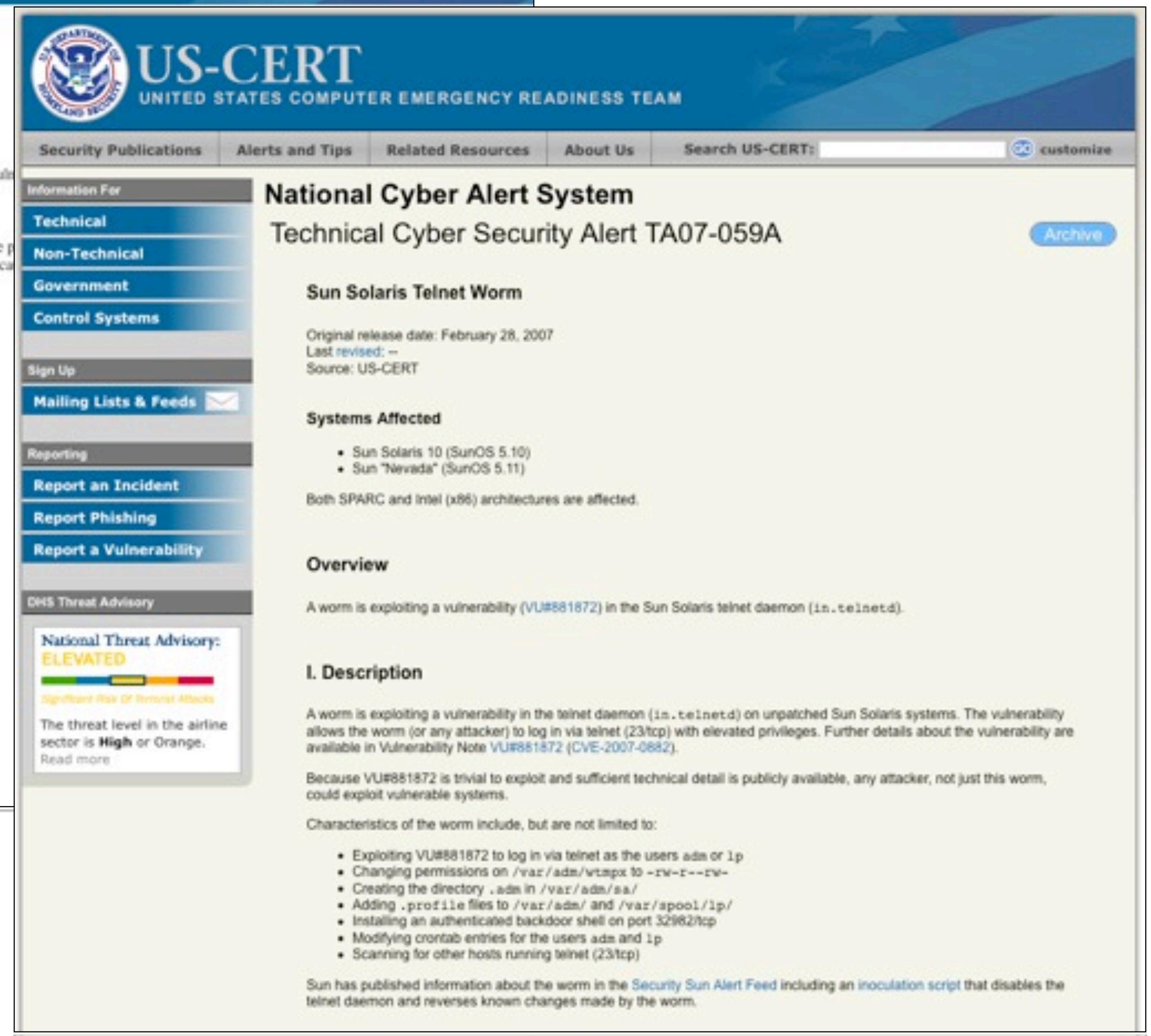
- `JDK and JRE 5.0 Update 10 or later`
- `SDK and JRE 1.4.2_13 or later`
- `SDK and JRE 1.3.1_19 or later`

Disable Java

Disable Java in your web browser, as specified in the Securing Your Web Browser document.

Systems Affected

Vendor	Status	Date Notified	Date Updated
Apple Computer, Inc.	Unknown	18-Jan-2007	
IBM eServer	Unknown	13-Feb-2007	
Sun Microsystems, Inc.	Vulnerable	17-Jan-2007	



US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Security Publications | Alerts and Tips | Related Resources | About Us | Search US-CERT: | customize

Information For
Technical
Non-Technical
Government
Control Systems
Sign Up
Mailing Lists & Feeds
Reporting
Report an Incident
Report Phishing
Report a Vulnerability
DHS Threat Advisory
National Threat Advisory: ELEVATED
The threat level in the airline sector is **High** or Orange.
Read more

National Cyber Alert System

Technical Cyber Security Alert TA07-059A

Archive

Sun Solaris Telnet Worm

Original release date: February 28, 2007
Last revised: --
Source: US-CERT

Systems Affected

- Sun Solaris 10 (SunOS 5.10)
- Sun "Nevada" (SunOS 5.11)

Both SPARC and Intel (x86) architectures are affected.

Overview

A worm is exploiting a vulnerability (VU#881872) in the Sun Solaris telnet daemon (`in.telnetd`).

I. Description

A worm is exploiting a vulnerability in the telnet daemon (`in.telnetd`) on unpatched Sun Solaris systems. The vulnerability allows the worm (or any attacker) to log in via telnet (23/tcp) with elevated privileges. Further details about the vulnerability are available in Vulnerability Note VU#881872 (CVE-2007-0882).

Because VU#881872 is trivial to exploit and sufficient technical detail is publicly available, any attacker, not just this worm, could exploit vulnerable systems.

Characteristics of the worm include, but are not limited to:

- Exploiting VU#881872 to log in via telnet as the users `adm` or `lp`
- Changing permissions on `/var/adm/wtmpx` to `-rw-r--rw-`
- Creating the directory `.adm` in `/var/adm/sa/`
- Adding `.profile` files to `/var/adm/` and `/var/spool/lp/`
- Installing an authenticated backdoor shell on port 32982/tcp
- Modifying crontab entries for the users `adm` and `lp`
- Scanning for other hosts running telnet (23/tcp)

Sun has published information about the worm in the Security Sun Alert Feed including an inoculation script that disables the telnet daemon and reverses known changes made by the worm.

Various Bug-Checking Tools Available in the Market

PolySpace

Digital WRL ESC

Microsoft Research Slam

VERACODE

GRAMMATECH Codesonar

FORTIFY

coverity

Prefast Microsoft

Klocwork

clang

findbugs

blast

jlint

jpf

splint

uno

Commercial

Open Source

Why Aren't These Tools Used at Sun?

- Long-running times over MLOC
  - Up to 1 week over ~6 MLOC
- Large false positive rate in practice
  - 30-50%
- High cost for commercial tools given the above
  - Proportional to # LOC
  - Tied to specific software to be checked
  - Maintenance fee on a yearly basis

Sample Source Code Sizes at Sun

- Vast majority is C/C++ system code
  - ...
  - JDK™ platform
    - 900 KLOC (VM and native libs)
  - ...
  - OpenSolaris™ operating system
    - OS/Networking (ON) consolidation: 10 MLOC
    - Full distro: >20 MLOC

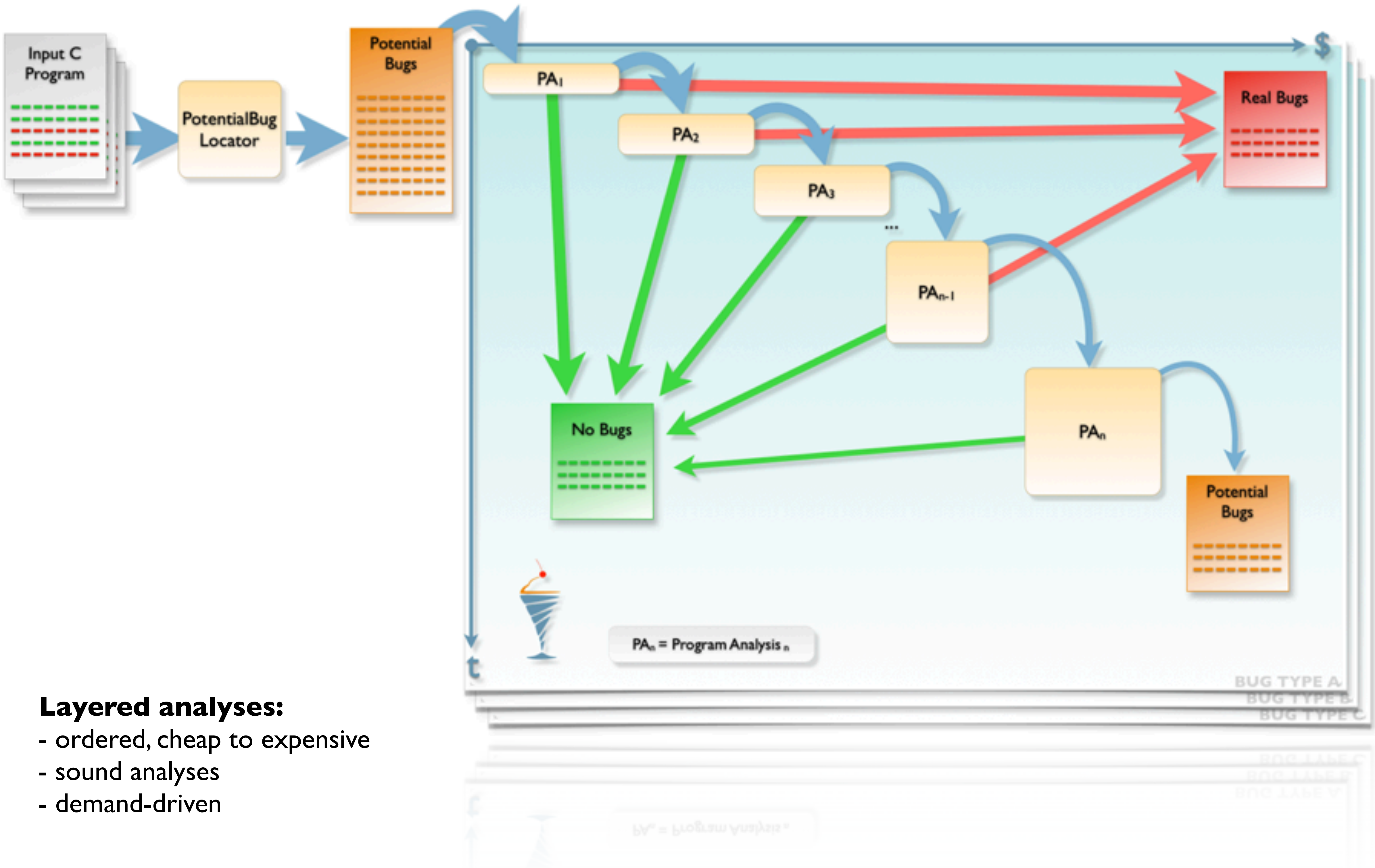
The Parfait Design

Key Features of the Parfait Design

- Scalability achieved by
  - Layered approach
  - Demand-driven analyses
  - Multiple ways to parallelize framework
    - per bug-type basis, per analysis, per “executable”-file basis

- Precision achieved by
  - Multiple lists of bugs
  - Bugs moved from PotentialBugs to RealBugs list conservatively

The Parfait Framework



Layered analyses:

- ordered, cheap to expensive
- sound analyses
- demand-driven

Layers of Analysis by Example

Finding buffer overflow
3 layers

Sample Program

```
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;

  if (argc != 3){
    printf("Usage: name length data\n");
    exit(-1);
  }

  for (i = 1; i <= n; i++) {
    buf[i] = 'A';
  }
  buf[n] = '\0';

  n = atoi(argv[1]);
  buf2 = (char*)malloc(n);
  for (i = 0; i <= n; i++) {
    buf2[i] = argv[2][i];
  }

 return 0;
}
```

Sample Program

```c
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;

  if (argc != 3){
    printf("Usage: name length data\n");
    exit(-1);
  }

  for (i = 1; i <= n; i++) {
    buf[i] = 'A';
  }
  buf[n] = '\0';

  n = atoi(argv[1]);
  buf2 = (char*)malloc(n);
  for (i = 0; i <= n; i++) {
    buf2[i] = argv[2][i];
  }

  return 0;
}
```

Sample Program

Layer 1 - Constant Propagation & Index Checks

```
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;
...
  for (i = 1; i <= n; i++) {
    buf[i] = 'A';
  }
  buf[n] = '\0';
```

Sample Program

Layer 1 - Constant Propagation & Index Checks

```
 0 #include <stdlib.h>
 1 #define BUFF_SIZE 100
 2
 3 int main (int argc, char *argv[])
 4 { char buf[BUFF_SIZE], *buf2;
 5   int n = BUFF_SIZE, i;
...
12   for (i = 1; i <= n; i++) {
13     buf[i] = 'A';
14   }
15   buf[n] = '\0';
```

```
4 char buf[100];
5  buf[100] = '\0';
```

Sample Program

Layer 2 - Partial Evaluation

```
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;

...
  for (i = 1; i <= n; i++) {
    buf[i] = 'A';
  }
  buf[n] = '\0';
```

Sample Program

Layer 2 - Partial Evaluation

```
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;

...
  for (i = 1; i <= n; i++) {
    buf[i] = 'A';
  }
  buf[n] = '\0';
```

```
  for (i = 1; i <= 100; i++) {
    if (i < 0 || i > 99)
      return (true);
  }
  return (false);
```

Sample Program

Layer 3 - Symbolic Analysis

```
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;
...
  n = atoi(argv[1]);
  buf2 = (char*)malloc(n);
  for (i = 0; i <= n; i++) {
    buf2[i] = argv[2][i];
  }

 return 0;
}
```

Sample Program

Layer 3 - Symbolic Analysis

```
0 #include <stdlib.h>
1 #define BUFF_SIZE 100
2
3 int main (int argc, char *argv[])
4 { char buf[BUFF_SIZE], *buf2;
5   int n = BUFF_SIZE, i;
...
17   n = atoi(argv[1]);
18   buf2 = (char*)malloc(n);
19   for (i = 0; i <= n; i++) {
20     buf2[i] = argv[2][i];
21   }
22
23  return 0;
24 }
```

```
3 int main (int argc, char *argv[])
4 { char buf[BUFF_SIZE], *buf2;
5   int n = 100;

17    n = atoi(argv[1]);// S[n]={N,N}
18    buf2 = (char*)malloc(n);
19    i = 0;            // S[i]={0,0}
      while(i <= n) {
20      buf2[i] = argv[2][i];
                       // S[i]={0,N}
21      i++;           // S[i]={1,N+1}
      }
```

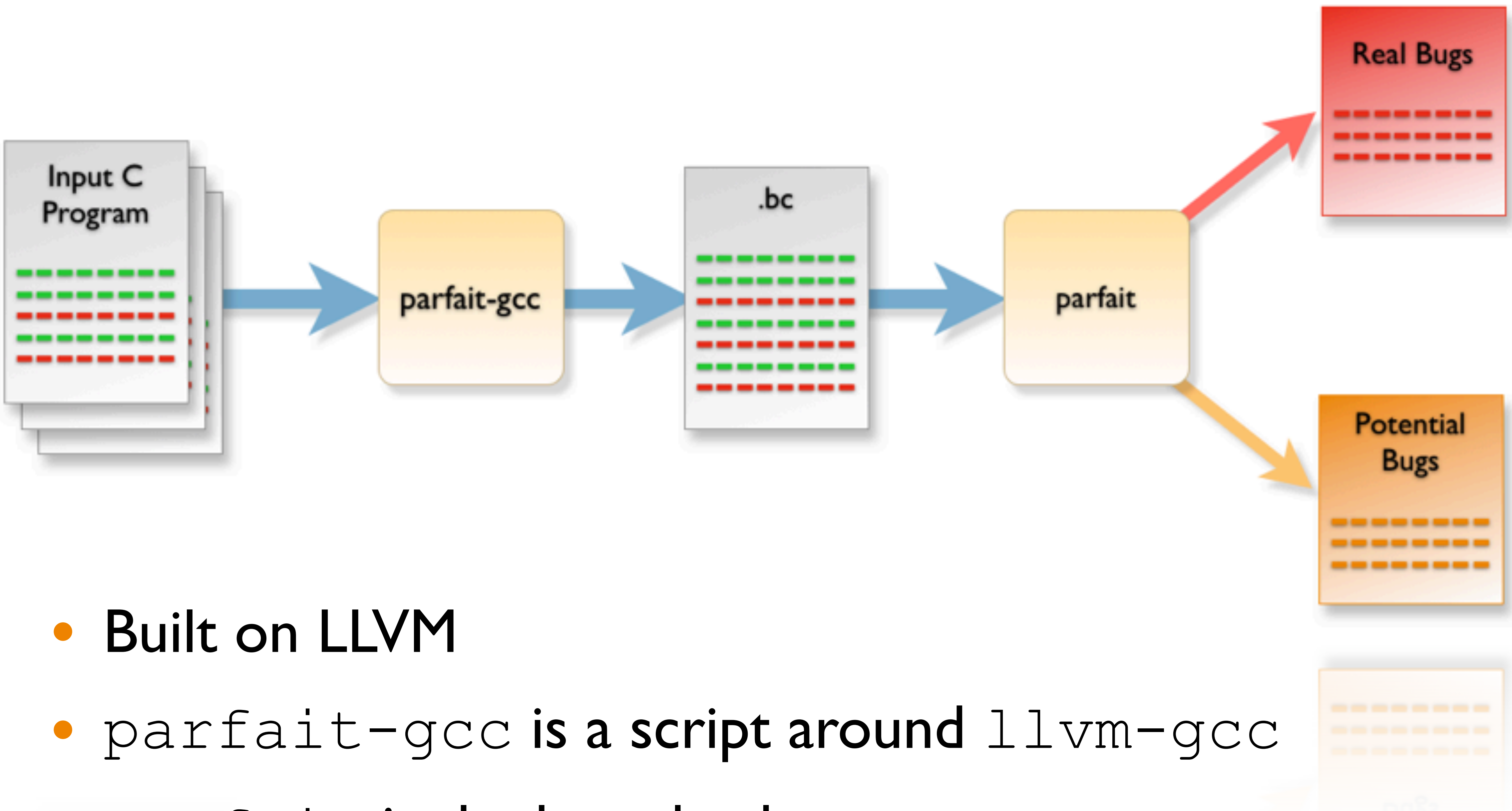
Sample Program

Final Analysed Program by Parfait

```
#include <stdlib.h>
#define BUFF_SIZE 100

int main (int argc, char *argv[])
{ char buf[BUFF_SIZE], *buf2;
  int n = BUFF_SIZE, i;

  if (argc != 3){
    printf("Usage: name length data\n");
    exit(-1);
  }

  for (i = 1; i <= n; i++) {
    buf[i] = 'A';
  }
  buf[n] = '\0';

  n = atoi(argv[1]);
  buf2 = (char*)malloc(n);
  for (i = 0; i <= n; i++) {
    buf2[i] = argv[2][i];
  }

 return 0;
}
```

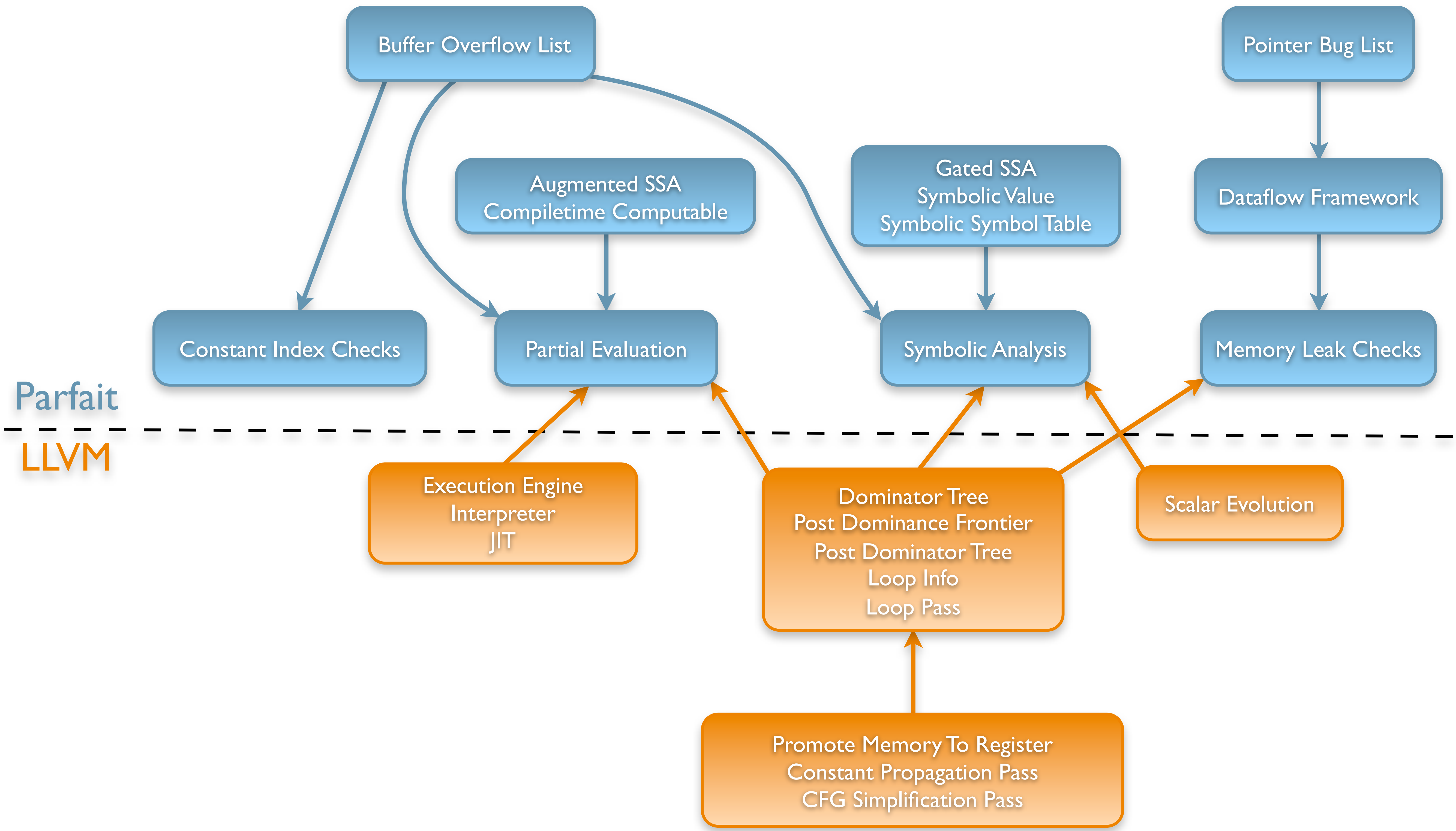
The Parfait Implementation

The Parfait Implementation



- Built on LLVM
- `parfait-gcc` is a script around `llvm-gcc`
- `parfait` is the bug-checker proper

Pass Dependencies in Parfait



Results

Reference dataset: OpenSolaris™ ON b93
Latest dataset: Solaris ON b121
Parfait 0.2.3.584 (18 Sep 09)

Performance Results over Solaris ON b121*

AMD Opteron 2.8 GHz, 2 GB memory

Build	Time (mins)
Normal build (Sun™ Studio, gcc)	364
Parfait build (Sun Studio, gcc, parfait-gcc)	534
Parfait analysis (parfait)	21

* 10 million non-commented lines of C/C++ source code (uts, cmd, lib, common, closed)

Accuracy Results over OpenSolaris b93*

True Positives and False Positives

Bug Type	Parfait reports	TP (%)	FP (%)
Buffer overrun	488	93%	7%
Memory leak	464	92%	8%
Format string type mismatch	1,009	96%	4%

* 7 million non-commented lines of C/C++ source code (uts, cmd, lib, common, closed)

Results with Open Source Kernels

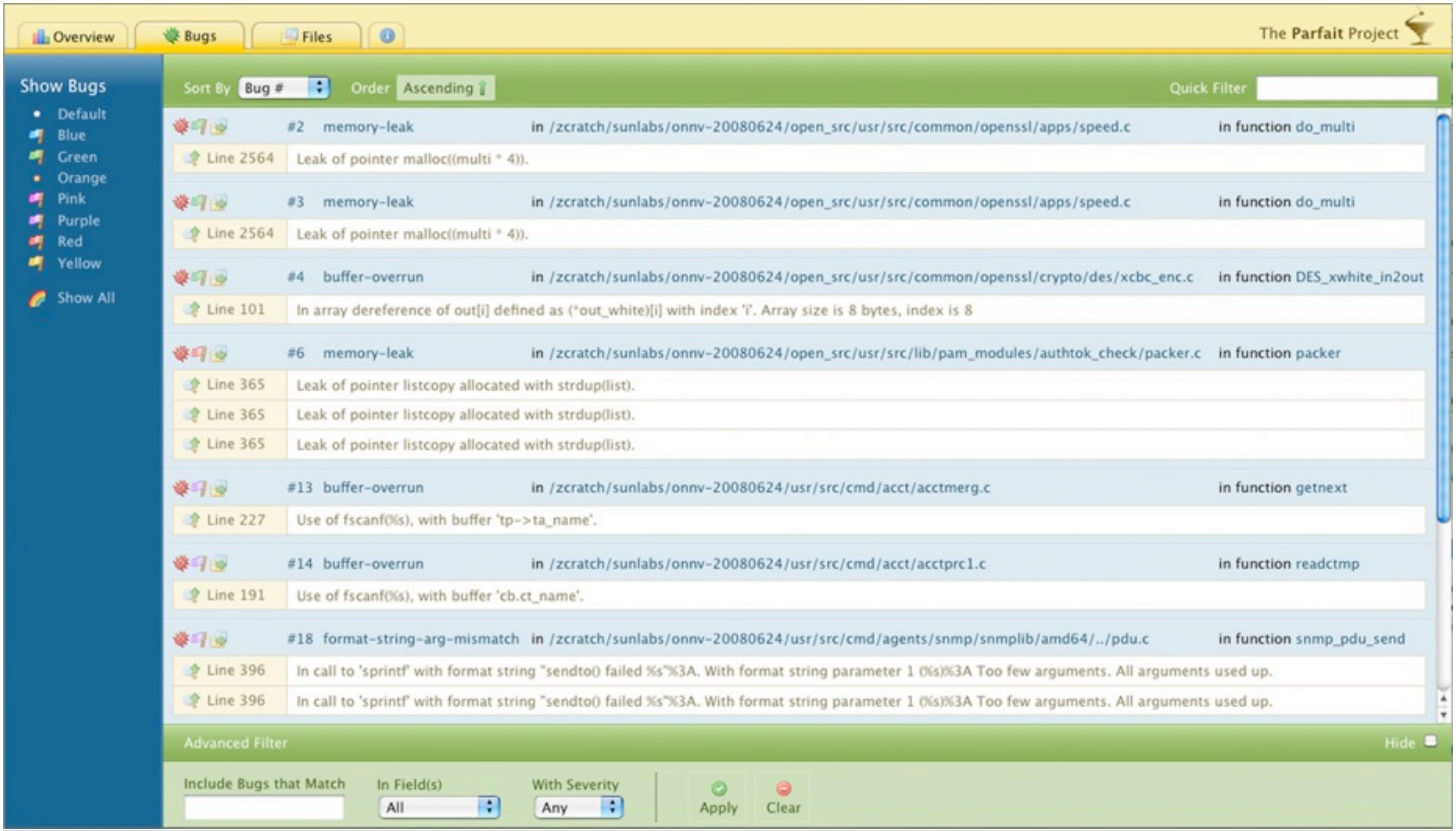
OpenSolaris, Linux, OpenBSD

Kernel	Time (min)	Part	LOC	Buffer overflow	Bug density	Status
OpenSolaris UTS b105	5	Core	2.1M	15	0.0069	Being fixed
		Device drivers	1.2M	67	0.054	Being fixed
Linux 2.6.29*	13	Core	1.6M	12	0.0073	Fixed
		Device drivers	4.1M	85	0.020	Submitted
OpenBSD 4.4	2	Core	0.5M	3	0.0060	Fixed
		Device drivers	0.8M	26	0.029	Fixed

* Linux has the benefit of 2 separate scans already made by Coverity over the Linux code base

The Parfait User Interface

Web-based GUI



Web-based GUI

- GUI tested with
  - Firefox 3, 3.5
  - Safari 4.0
  - Chrome 1, 2, 3
  - Internet Explorer 8, 7
- GUI tested on
  - Solaris, Mac OS X, Linux and Windows
- Usability testing conducted
  - University students
  - Sun engineers

LLVM Evaluation

Benefits

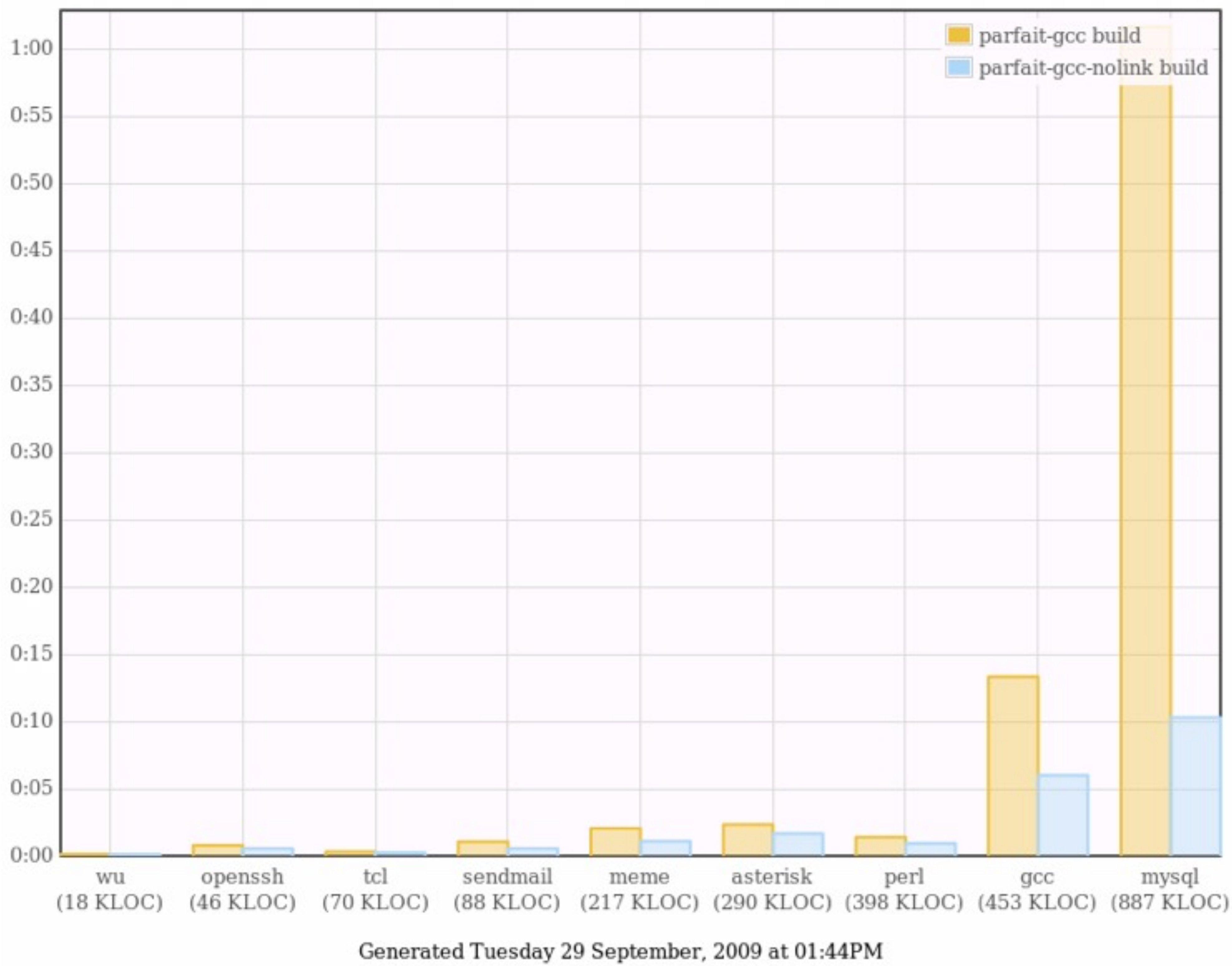
- Modern compilation infrastructure
  - modular
  - uses SSA representation
  - extensible
- Cross platform
- Portable IR
- Well documented
- Ease of prototyping

Challenges

- Lack of union type information in IR
- Lack of backwards compatibility
- Limited support for debug information
- Memory consumption issues
- Performance issues with llvm-ld
- Reliance on "newer" versions of gcc (4.x)
  - much legacy code doesn't compile with gcc 4.x
- Some non-extensible implementations

Challenges

llvm-ld overhead is not linear



Summary

Summary

- Parfait is a new C/C++ bug-checking tool
  - scalable and precise
  - starting to be widely used internally
  - external requests
- Extensible framework
  - Our emphasis
    - Buffer overflow, pointer/memory-related errors, format string
  - Our collaborators
    - Concurrency bugs, automated testing, OO-specific bugs
- Has found real bugs in
  - Solaris, OpenBSD, Linux, JDK, ...

The Parfait Team

parfait-dev@sun.com

http://research.sun.com/projects/parfait