WiSA Demo Session

Hao Wang, Hong Lin

BREW – A Binary REWriting Infrastructure

> Hao

Hong Lin, Mihai Christodorescu, Jonathon Giffin, Hao Wang

Dynamic Buffer Overrun Detection

> Hong

Mihai Christodorescu

Malicious Code Obfuscation and Detection

> Mihai

Vinod Ganapathy

Statically Detecting Buffer Overruns

> Vinod

Gogul Balakrishnan

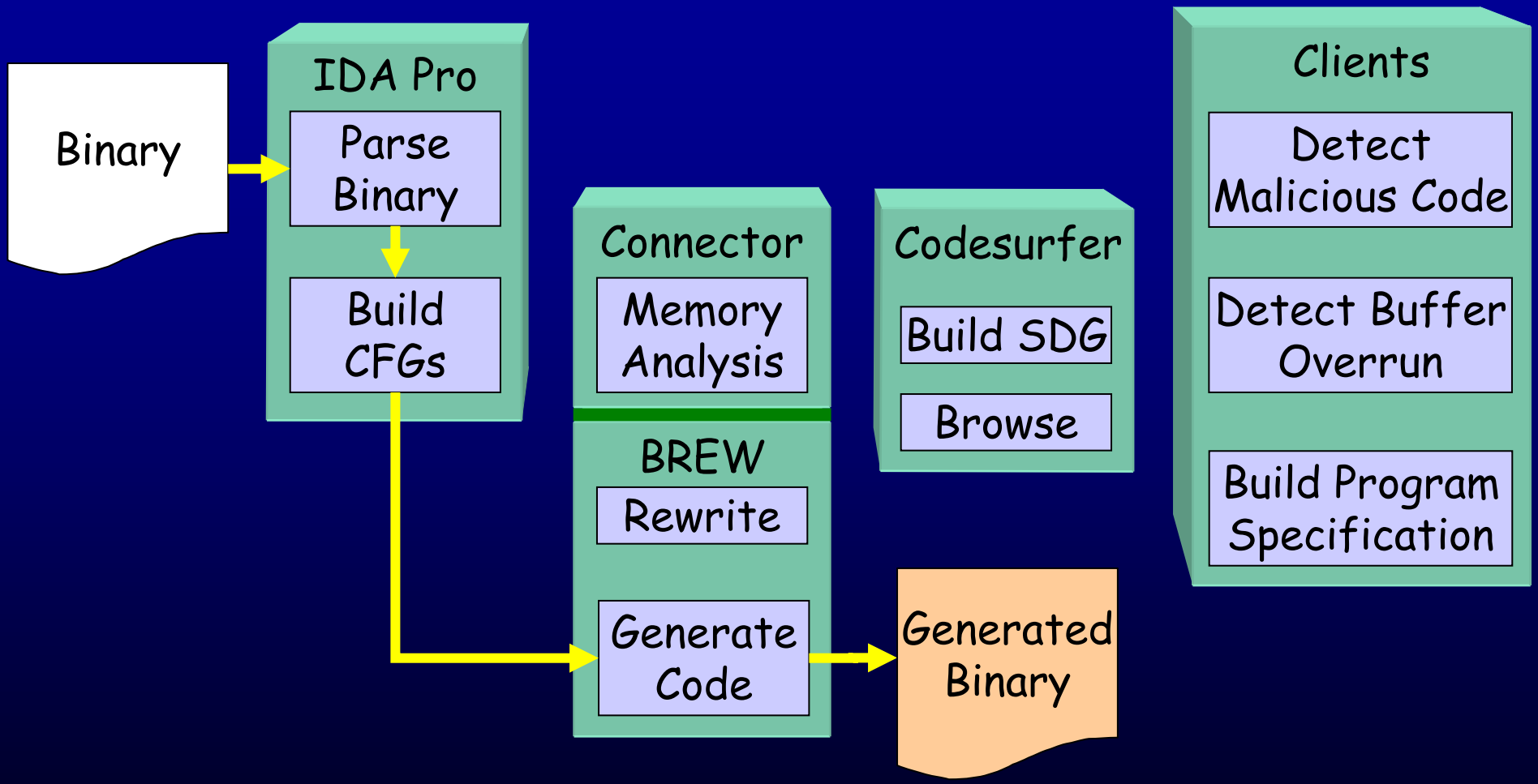
Enhanced CodeSurfer/x86

> Gogul

BREW—A Binary REWriting Infrastructure (Demo)

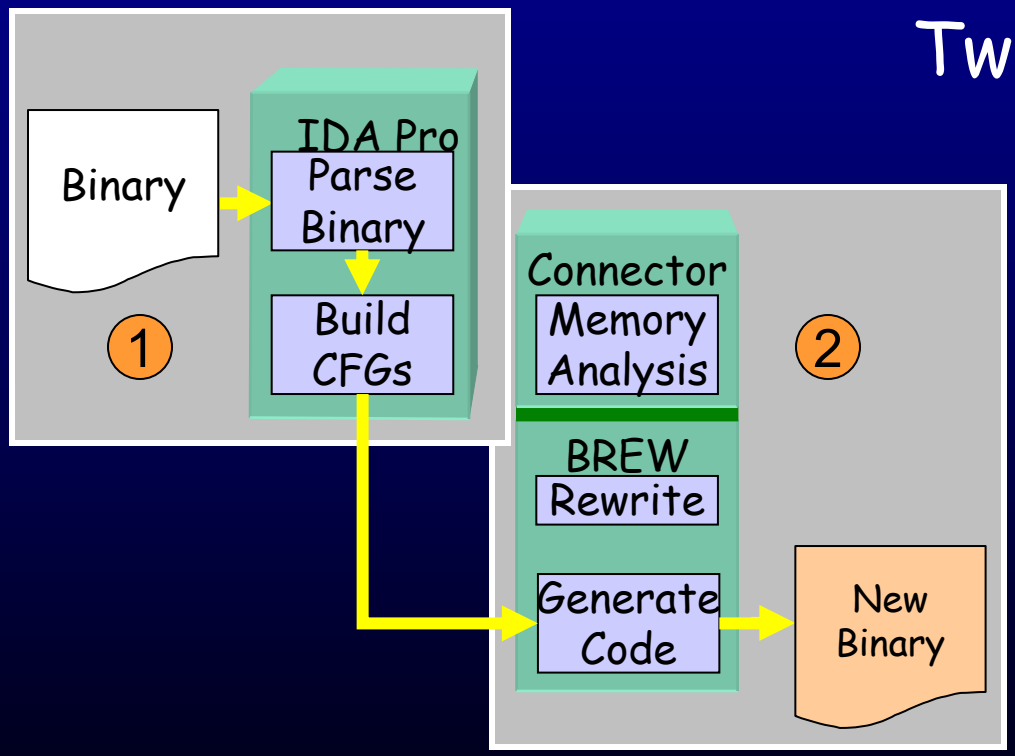
Hong Lin and Hao Wang
(honglin, hbwang)@cs.wisc.edu
WiSA
University of Wisconsin - Madison

Code Generation



Code Generation

Goal: Binary-to-Binary rewriting



Two-step process:

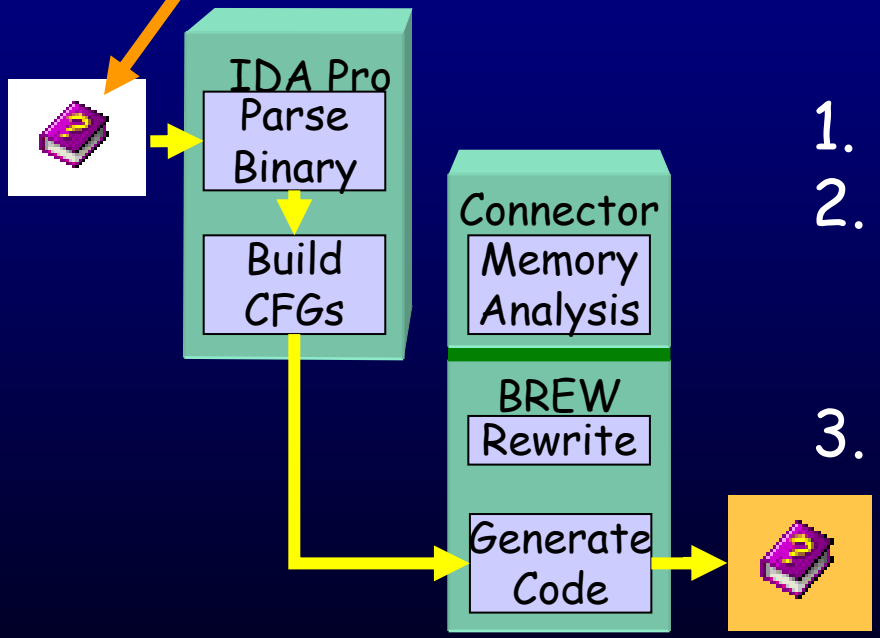
1. Disassemble original binary

Intermediate Data Structures

2. Reassemble new binary

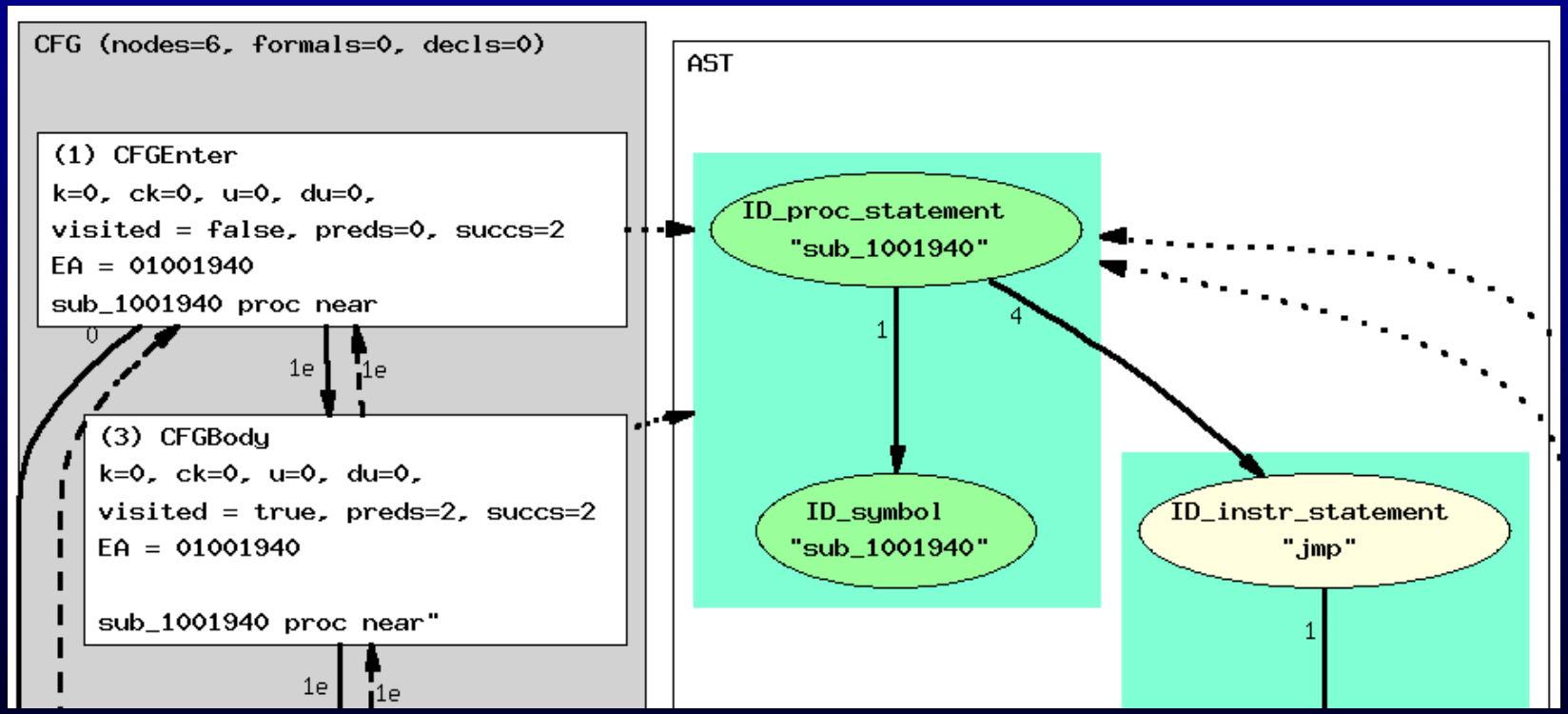
Demo Outline

Target program: Winhelp



1. Show original winhlp32.exe
2. Disassemble and reassemble new winhlp32.exe
3. Show new winhlp32.exe

Intermediate Data Structures



Technical Challenges

- How to distinguish between code and data
- Identify indirect jumps and function pointers
- How to deal with encrypted binaries
- Relocation issues

Related Work

- Dynamic rewriting
  - Paradyn/DynInst
  - Detour
- Static rewriting
  - ATOM—only works on Alpha
  - EEL—only works on Sparc
  - Vulcan—relies on Program Database (PDB)
  - UQBT—Translate from one architecture to another

Status

- Currently only support x86
  - Investigating other platforms
- Code generator prototype is working
- Rewriting API is being implemented
- Check out web page for latest status
  - http://www.cs.wisc.edu/wisa

WiSA Demo Session

Hao Wang, Hong Lin
BREW – A Binary REWriting Infrastructure

> Hao

Hong Lin, Mihai Christodorescu, Jonathon Giffin, Hao Wang
Dynamic Buffer Overrun Detection

> Hong

Mihai Christodorescu
Malicious Code Obfuscation and Detection

> Mihai

Vinod Ganapathy
Statically Detecting Buffer Overruns

> Vinod

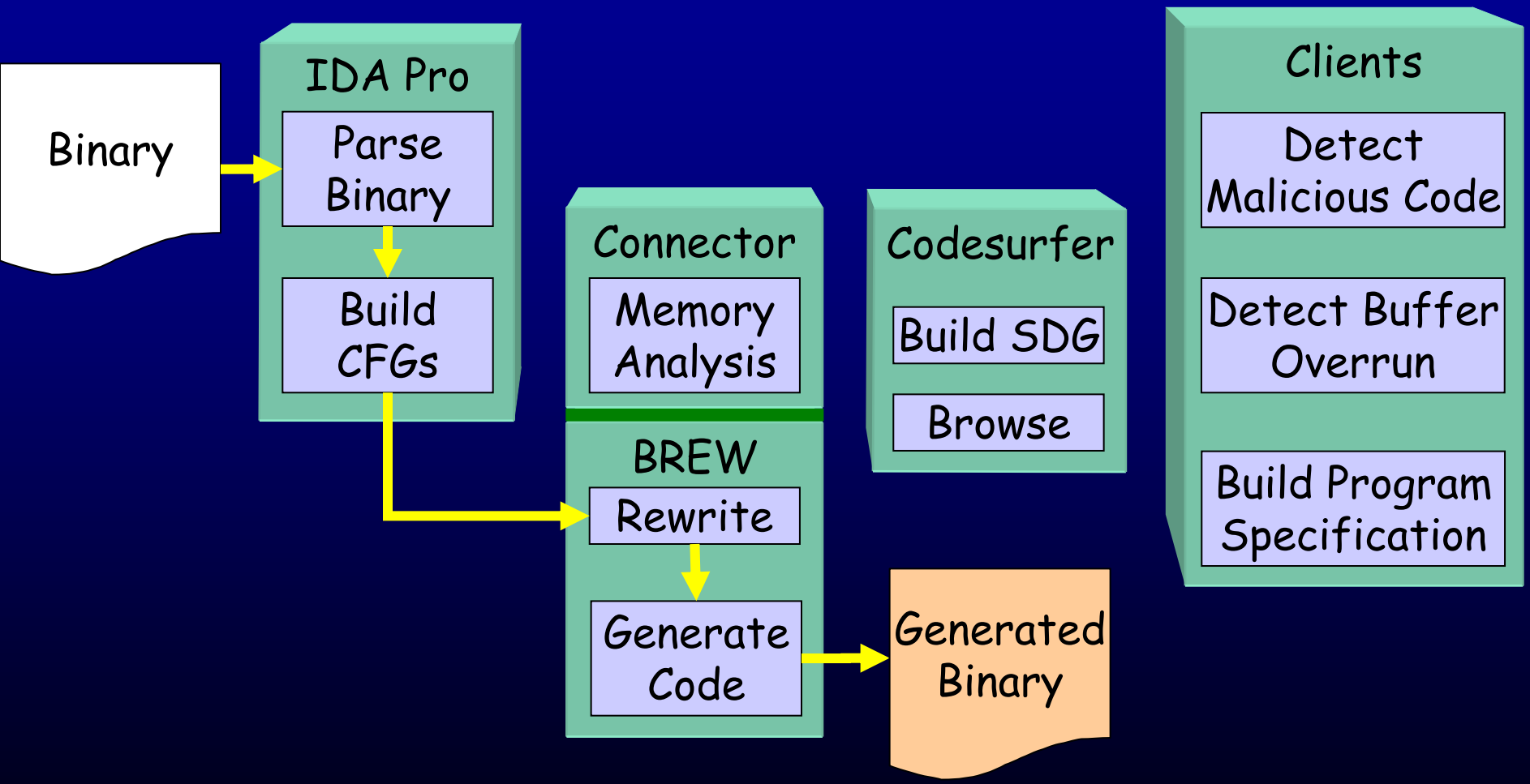
Gogul Balakrishnan
Enhanced CodeSurfer/x86

> Gogul

Dynamic Buffer Overrun Detection

Hong Lin
WiSA
University of Wisconsin – Madison

Code Instrumentation



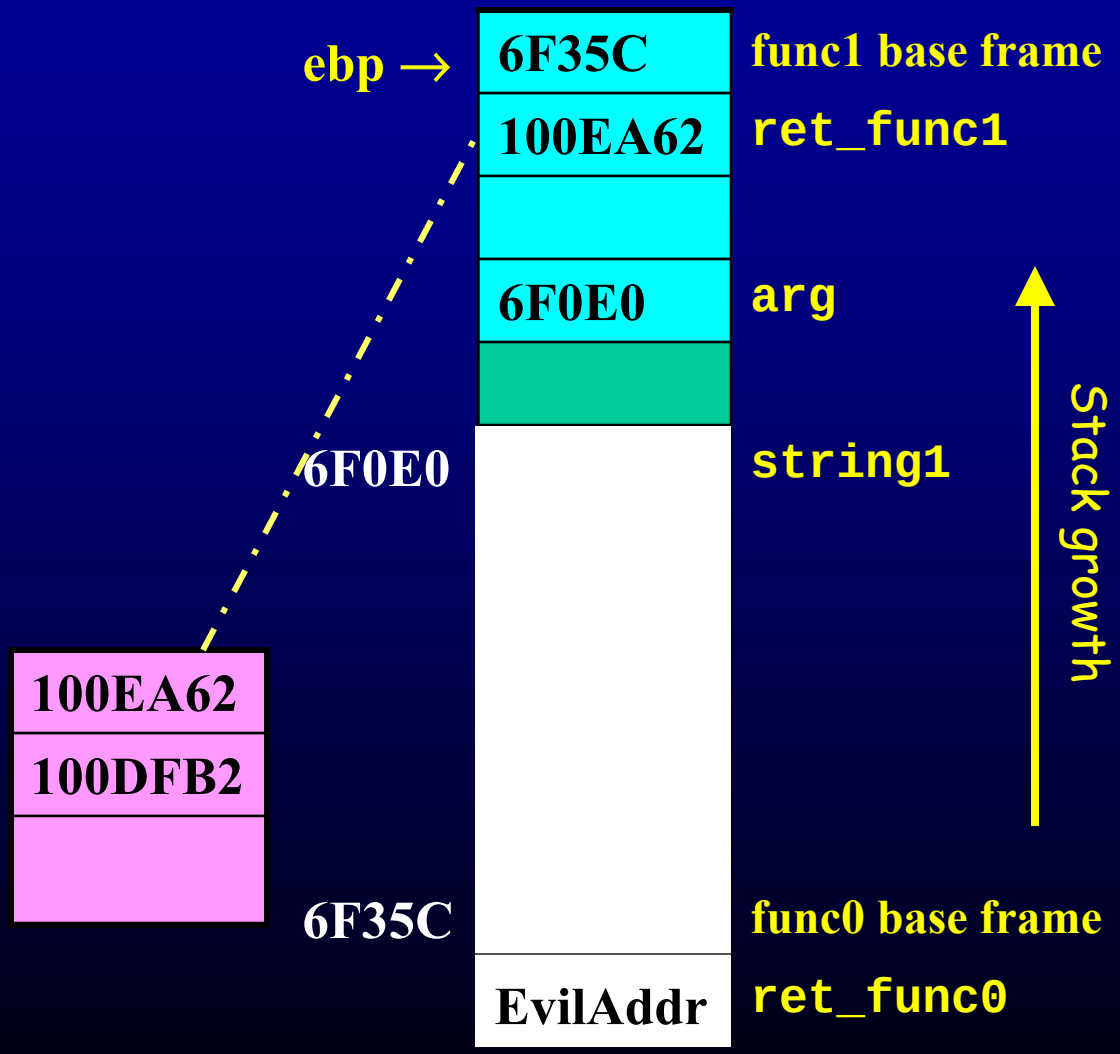
Demo Outline

- Winhlp32.exe
  - normal operation
  - buffer overrun exploit
- Instrumented winhlp32.exe with dynamic buffer overrun detection
  - maintains original behavior with valid input
  - detects overrun with malicious input

Demo

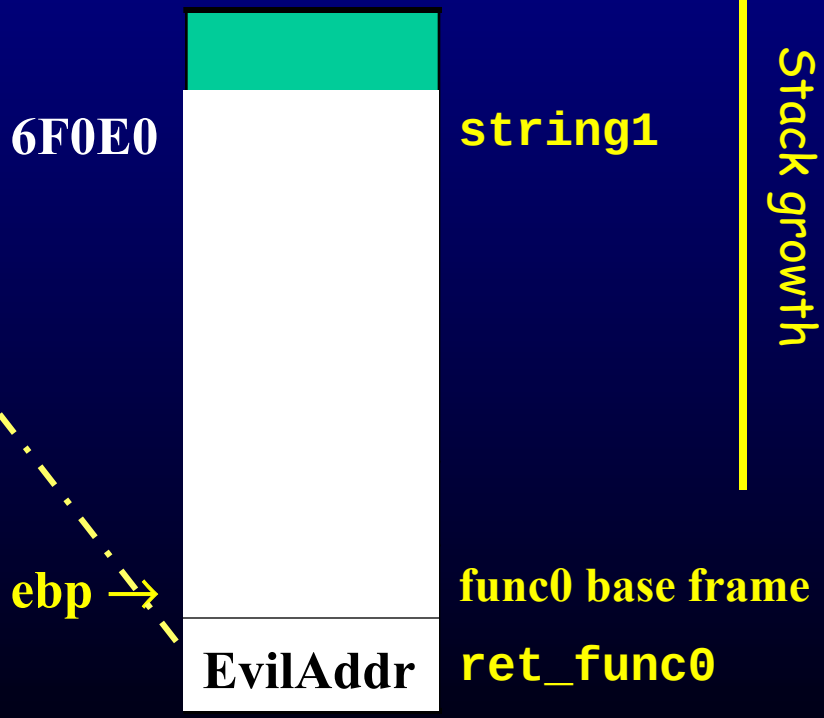
Winhlp32 Vulnerability

```
func0 proc near
    call func1
    mov eax, 09h
endp

func1 proc near
    string1 = byte ptr -104h
    …
    push [ebp+string]
    call func2
endp
func2 proc near
    arg = dword ptr 0Ch
    …
    push [ebp+arg]
    call lstrcpyA
endp
```



Winhlp32 Vulnerability

```
func0 proc near
    call func1
    mov eax, 09h
endp

func1 proc near
    string1 = byte ptr -104h
    …
    push [ebp+string]
    call func2
endp
func2 proc near
    arg = dword p
    …
    push [ebp+arg]
    call lstrcpyA
endp
```



Return Address Defense (RAD)

- Idea:
  save a redundant copy of return address on a different stack [Chiueh & Hsu '01]
- Implementation
  - hardware
  - source code
  - binary

Using BREW

```
snippet* entry_code = get_snippet();
snippet* exit_code = get_snippet();
char* save_instr[]={“push dword ptr[esp]",
                    “call _save_ret_addr"}
char* check_instr[]={“push dword ptr[esp]",
                     “call _verify_ret_addr“}

entry_code->insert_instruction(save_instr, 2);
exit_code->insert_instruction(check_instr, 2);
```

```
for (int i = 0; i < cfgTable->NrOfCFGs(); i++) {
   nCFG* cfg = cfgTable->GetCFG(i);
   add_entry_code(cfg, entry_code, true);
   add_exit_code(cfg, exit_code, true);
}
```

```
incorporate_obj (“ret_stack.obj”)
```

Generated Assembly

```
sub_100EA2C   proc near
   push dword ptr [esp]
   call _save_ret_addr
   push ebp
   mov  ebp, esp
   sub  esp, 027ch
   ...
loc_100ECA7:
   pop  edi
   pop  esi
   pop  ebx
   push dword ptr [esp]
   call _verify_ret_addr
   leave
   retn 08h
sub_100EA2C   endp
```

Demo

Binary Implementation at Stony Brook

[Prasad & Chiueh '03]

Idea:

- Disassemble binary, identify function boundary
- Patch and edit binary image to implement RAD

Implementation:

- Based on an existing disassembler
- A specific tool that can insert instructions at function entry and exit

References

- Buffer Overrun Occurs When You Start Winhlp32.exe Under NTSD
  http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B293338
- Winhlp32.exe Remote Buffer Overrun
  http://archives.neohapsis.com/archives/ntbugtraq/2002-q3/0051.html
- Tzi-cker Chiueh, Fu-Hau Hsu: *RAD: A Compile-Time Solution to Buffer Overflow Attacks*. ICDCS 2001: 409-417
- Manish Prasad and Tzi-cker Chiueh, *A Binary Rewriting Defense Against Stack-based Buffer Overflow Attacks,* in the Proceedings of USENIX Annual Technical Conference, San Antonio, TX, June 2003

WiSA Demo Session

Hao Wang, Hong Lin
BREW – A Binary REWriting Infrastructure

> Hao

Hong Lin, Mihai Christodorescu, Jonathon Giffin, Hao Wang
Dynamic Buffer Overrun Detection

> Hong

Mihai Christodorescu
Malicious Code Obfuscation and Detection

> Mihai

Vinod Ganapathy
Statically Detecting Buffer Overruns

> Vinod

Gogul Balakrishnan
Enhanced CodeSurfer/x86

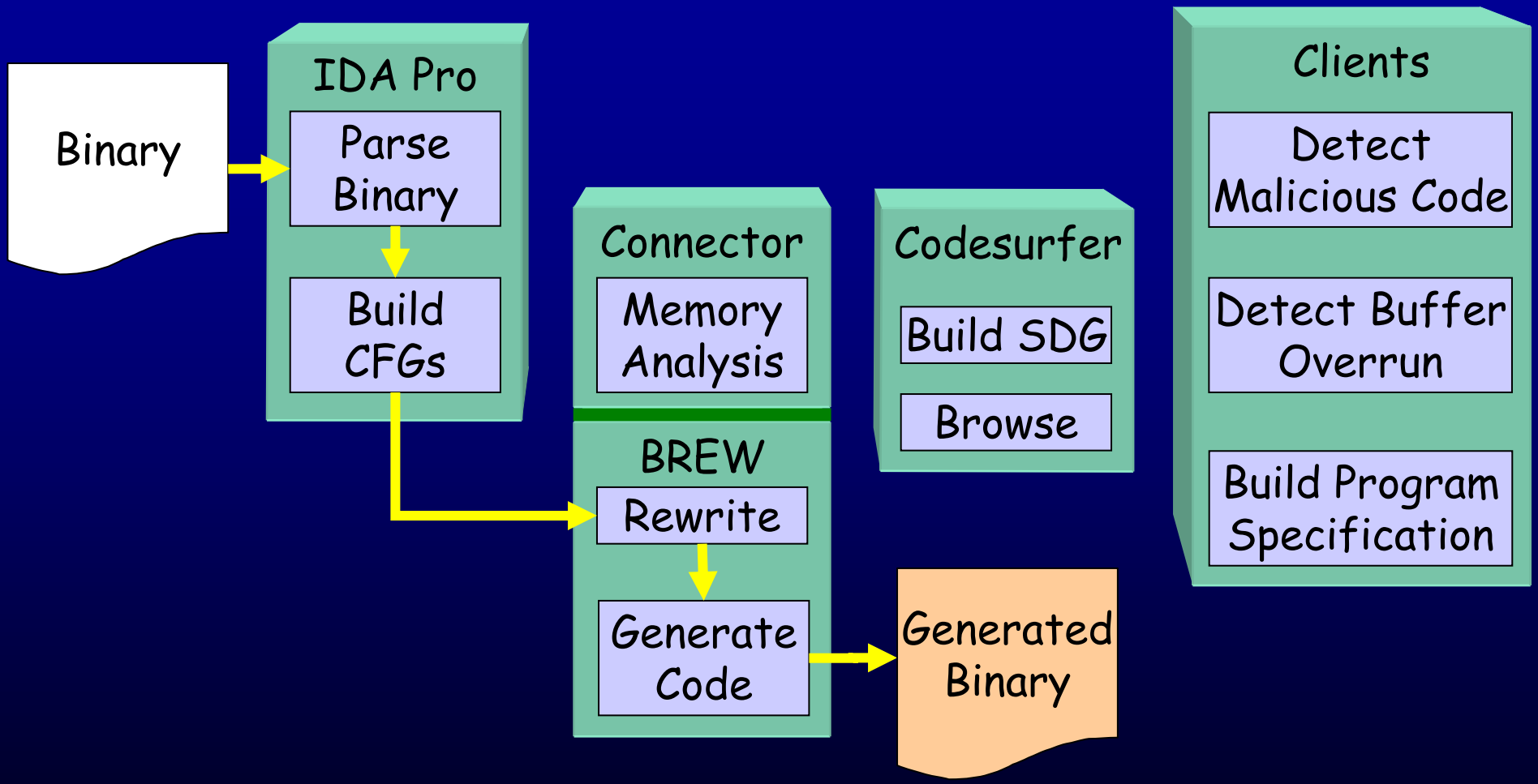
> Gogul

Malicious Code Obfuscation and Detection

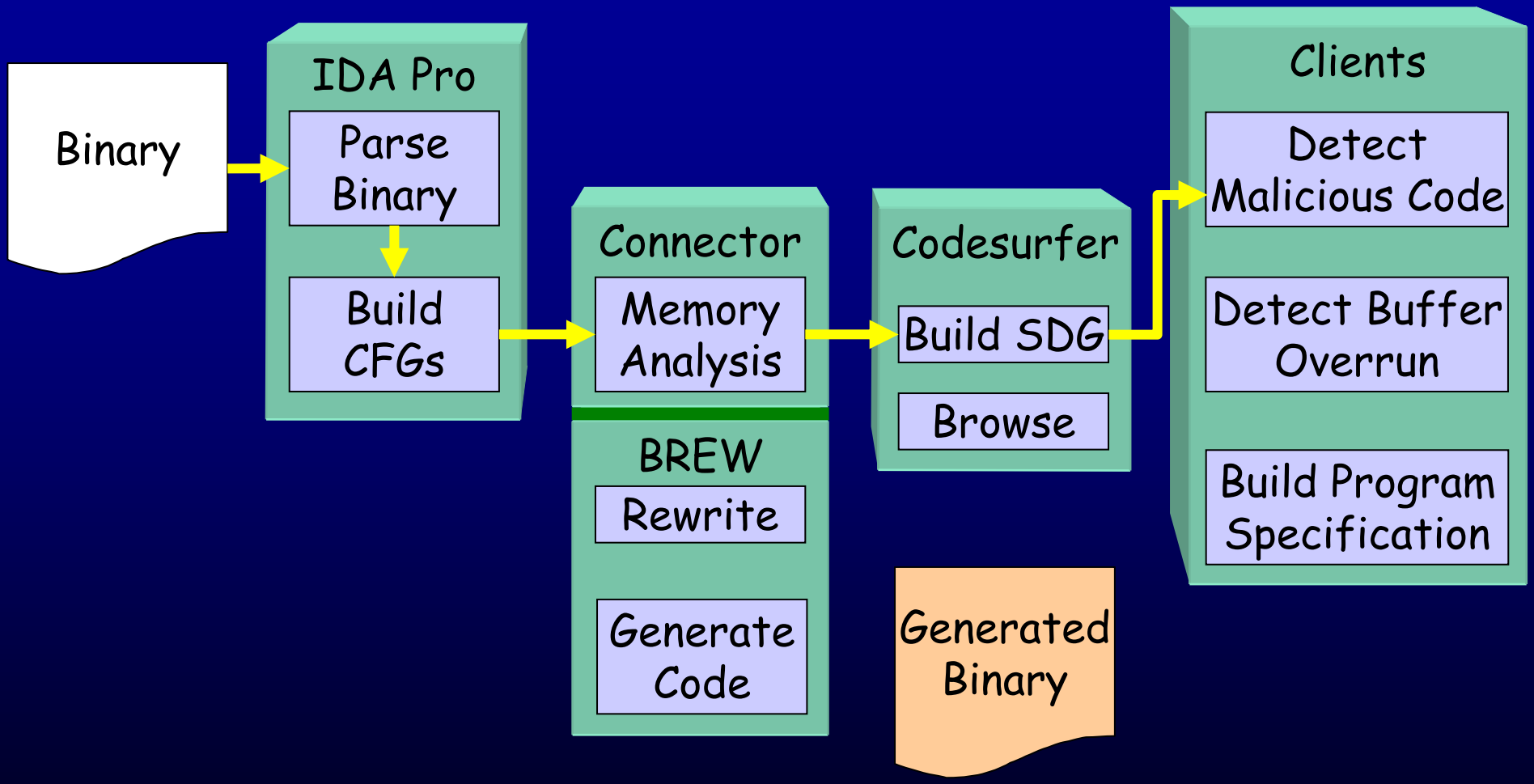
Mihai Christodorescu
WiSA
University of Wisconsin, Madison

Virus Obfuscation

through Binary Code Rewriting



Malicious Code Detection



Building on the Infrastructure

Virus obfuscation

1. Disassemble
2. Rewrite
3. Generate new version

Virus detection

1. Disassemble
2. Analyze
3. Check against malicious code specification

Virus Obfuscation

Change data:

- New strings, new d
- Encode / encrypt

Change control:

- Insert garbage
- Encode / encr
- Reorder code
- Add new features
- …

Types of garbage code:

- NOPs

```
instr_1
nop
nop
instr_2
```

- Simple sequences

```
instr_1
inc eax
dec eax
instr_2
```

- Complex sequences
  - Save/restore state
  - Construct/destruct data structures

Obfuscation

Virus Code
(from FOsfOrO):

```
mov dword ptr [ebp+infected], 4
lea eax,[ebp+offset DiReCtOrY0]
push eax
push 260
call [ebp+GetCurrentDirectoryA]
or eax,eax
```

Morphed Virus Code
(from FOsfOrO):

```
mov dword ptr [ebp+infected], 4
nop
lea eax,[ebp+offset DiReCtOrY0]
nop
push eax
nop
push 260
nop
call [ebp+GetCurrentDirectoryA]
or eax,eax
```

[demo]

Binary Rewriter API

- Easy to obfuscate:

```
for each procedure P
{
   for each instruction I in P
   {
      insert( before I, “nop” code snippet )
   }
}
generate new binary()
```

Obfuscation & Detection

- Not all obfuscations are created equal:

  [demo]

- Enhanced virus detection:

  [demo]

Visual Basic Malware

- Preview of VB obfuscator:

  [demo]

- VB malware detector:
  - In progress

WiSA Demo Session

Hao Wang, Hong Lin

BREW – A Binary REWriting Infrastructure

> Hao

Hong Lin, Mihai Christodorescu, Jonathon Giffin, Hao Wang

Dynamic Buffer Overrun Detection

> Hong

Mihai Christodorescu

Malicious Code Obfuscation and Detection

> Mihai

Vinod Ganapathy

Statically Detecting Buffer Overruns

> Vinod

Gogul Balakrishnan

Enhanced CodeSurfer/x86

> Gogul

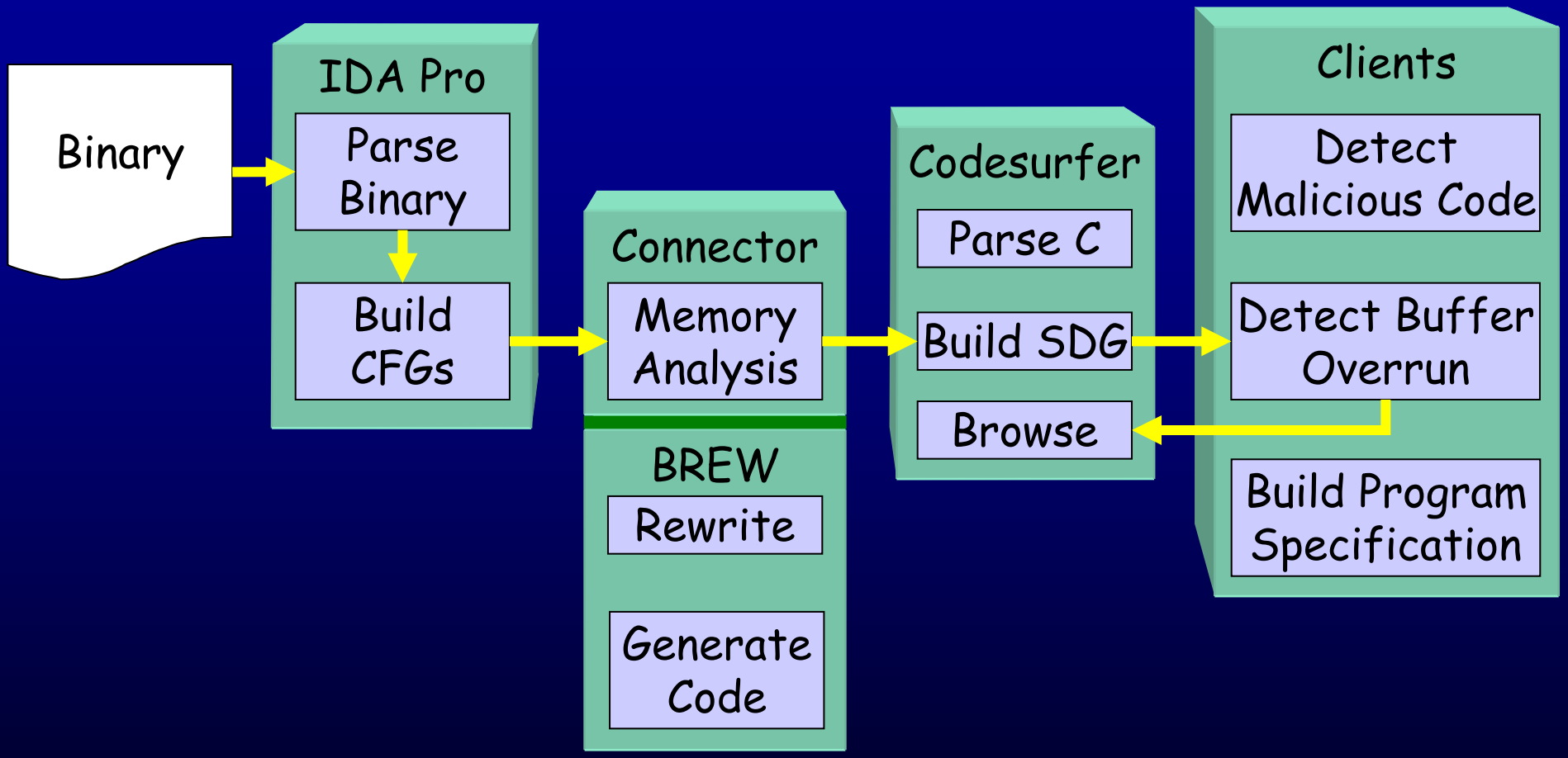
Statically detecting buffer overruns

Vinod Ganapathy

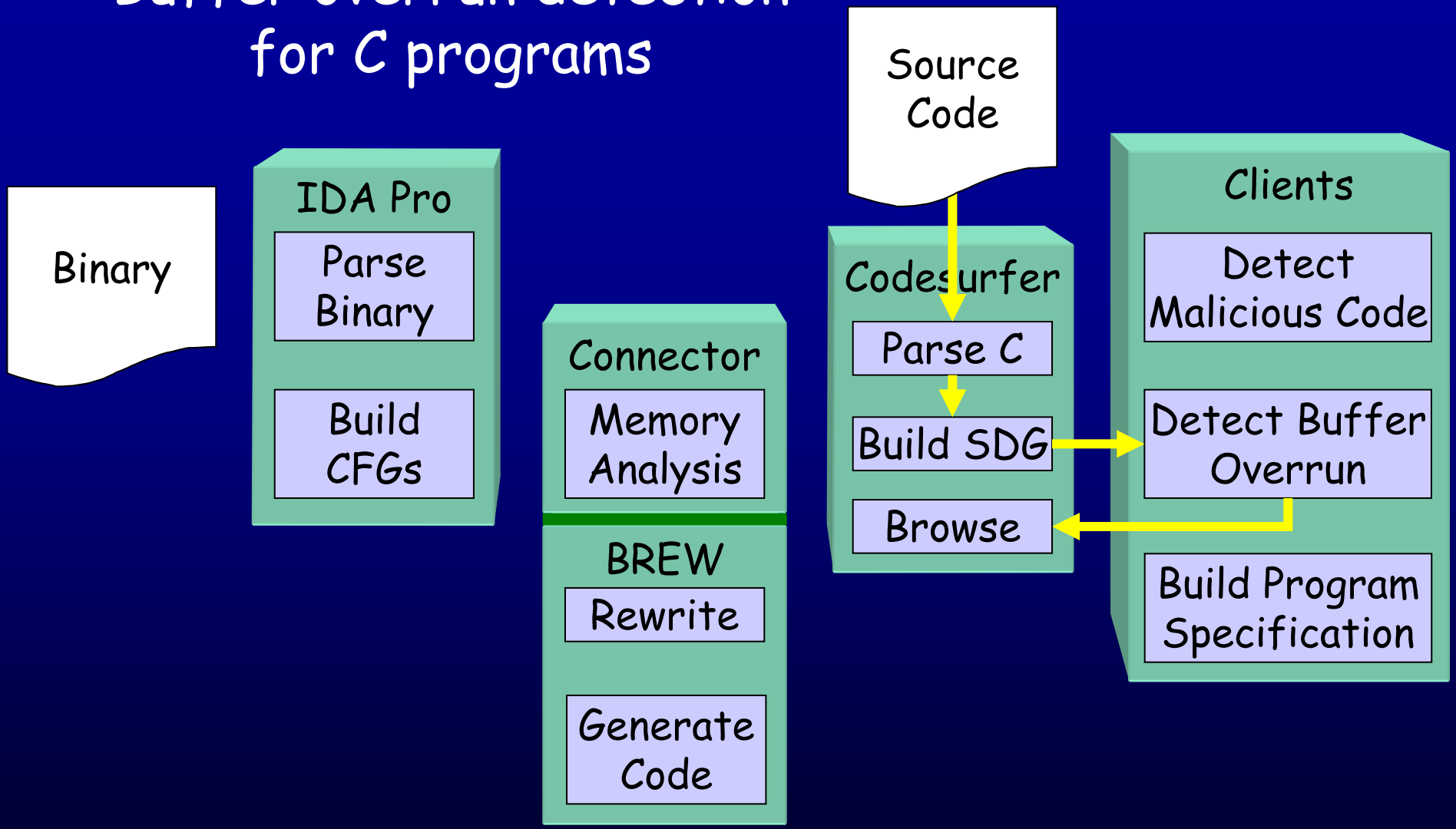
Overview

- Existing Infrastructure
  - *Static* detection of overruns on *source code*
- Goal
  - Static detection of overruns on *binaries*
  - Many ideas from source code analysis carry over

Buffer overrun detection for object code



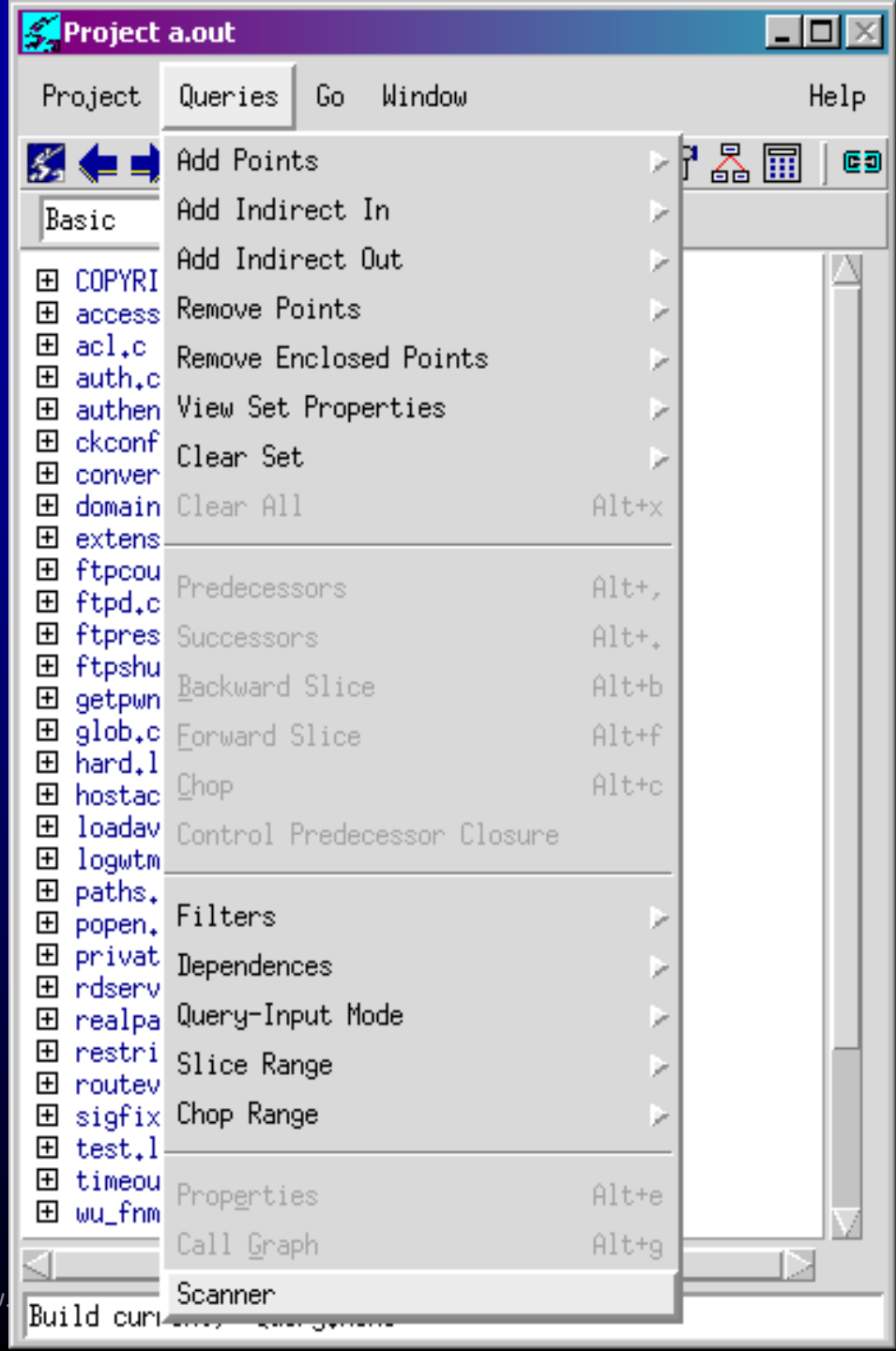
Buffer overrun detection for C programs



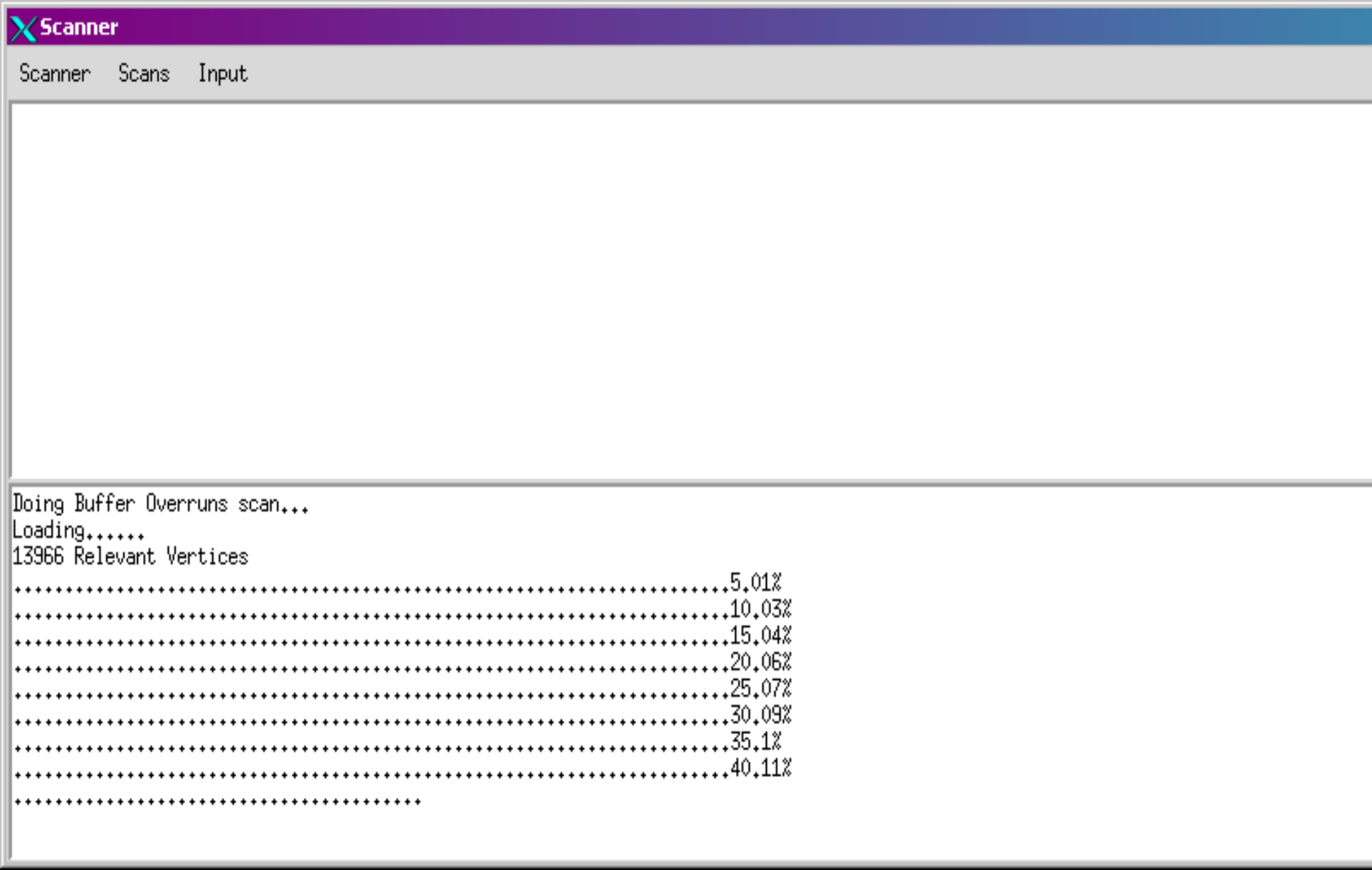
Results

- Analyzed wuftpd-2.6.2
  - Popular file transfer server
  - Most recent version
  - We found 4 ***new*** overruns
- Demo (Codesurfer + B.O. Scanner)

Build the program using Codesurfer. Perform a buffer-overruns scan



Buffer Overruns Scan in progress



Warnings on WuFtpD-2.6.2

Scanner

Scanner Scans Input

```
⊟ Buffer Overruns (178)
  ⊞ (3) ftpd.c :: vreply :: buf   Allocated: 8192..8192   Used: -0..8196
  ⊞ (3) ftprestart.c :: main''''' :: altmsgpath   Allocated: 4095..4095   Used: -0..16383
  ⊟ (3) ftprestart.c :: main''''' :: root   Allocated: 4095..4095   Used: -0..8192
      [call-site] strcpy()
      [declaration] (Local) root
  ⊟ (3) rdservers.c :: read_servers_line :: accesspath   Allocated: 1024..4095   Used: -0..8192
      [call-site] strcpy()
      [call-site] read_servers_line()
      [call-site] read_servers_line()
      [call-site] read_servers_line()
      [call-site] read_servers_line()
  ⊞ (4) access.c :: acl_deny :: msgpathbuf   Allocated: 4095..4095   Used: -0..+INF
```

```
[call-site] strcpy()

Source Code:
rdservers.c:103:  strcpy(accesspath, acp);

Constraints:
(Priority  3) accesspath!len CONTAINS + acp!len  WHICH EQUALS [-0..8192]
        acp!len IS IN [-0..8192]
```

Source code corresponding to the warning

rdservers.c

File Edit Functions Queries Go Window

```
    ecp = acp;

    while (*ecp && (!isspace(*ecp)) && *ecp != '\n')
        ++ecp;

    *ecp = '\0';

    if ((hp = gethostbyname(hcp)) != NULL) {
        struct in_addr in;
        memmove(&in, hp->h_addr, sizeof(in));
        strcpy(hostaddress, inet_ntoa(in));
    }
    else
        strcpy(hostaddress, hcp);

    strcpy(accesspath, acp);

    return (1);
    }
    return (0);
}
#endif
```

Can now employ other queries
e.g. Slicing, Data Predecessor

rdservers.c

File Edit Functions Queries Go Window

```
while (fgets(buffer, BUFSIZ, svrfp) != NULL) {

/* Find first non-whitespace character */
for (bcp = buffer; ((*bcp == '\t') || (*bcp == ' ')); bcp++);

/* Get rid of comments */
if ((ecp = strchr(buffer, '#')) != NULL)
    *ecp = '\0';

/* Skip empty lines */
if ((bcp == ecp) || (*bcp == '\n'))
    continue;

/* separate parts */

hcp = bcp;
for (acp = hcp;
     (*acp && !isspace(*acp)); acp++);

/* better have something in access path or skip the line */
if (!*acp)
    continue;

*acp++ = '\0';

while (*acp && isspace(*acp))
    acp++;

/* again better have something in access path or skip the line */
if (!*acp)
    continue;

ecp = acp;

while (*ecp && (!isspace(*ecp)) && *ecp != '\n')
    ++ecp;

*ecp = '\0';

if ((hp = gethostbyname(hcp)) != NULL) {
    struct in_addr in;
    memmove(&in, hp->h_addr, sizeof(in));
    strcpy(hostaddress, inet_ntoa(in));
}
else
    strcpy(hostaddress, hcp);

strcpy(accesspath, acp);
```

WiSA Demo Session

Hao Wang, Hong Lin

BREW – A Binary REWriting Infrastructure > Hao

Hong Lin, Mihai Christodorescu, Jonathon Giffin, Hao Wang

Dynamic Buffer Overrun Detection > Hong

Mihai Christodorescu

Malicious Code Obfuscation and Detection > Mihai

Vinod Ganapathy

Statically Detecting Buffer Overruns > Vinod

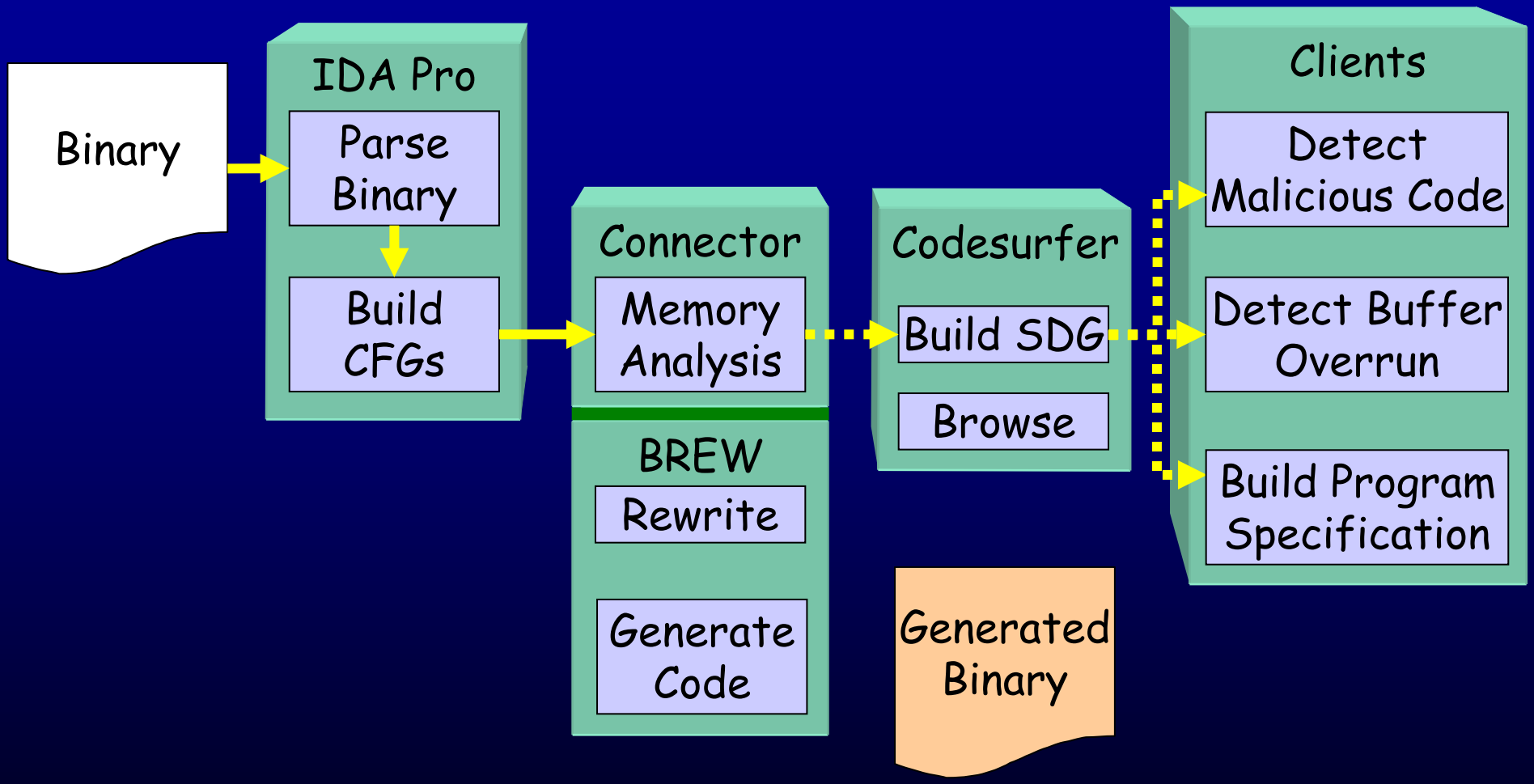
Gogul Balakrishnan

Enhanced CodeSurfer/x86 > Gogul

Enhanced CodeSurfer/x86

Gogul Balakrishnan

Value-set Analysis



Existing CodeSurfer/x86

- Existing CodeSurfer/x86
  - Ignores pointer-arithmetic
  - Unsafe used, killed and c-killed sets
    - $\Rightarrow$ Data dependences missing

Enhanced CodeSurfer/x86

- Value-set analysis (VSA)
  - combined pointer and numeric analysis
  - takes care of pointer-arithmetic
  - safe used, killed, c-killed sets
  - generates reports
    - color-code program listing
    - sample reports
      - possible buffer overruns
      - conditions when sets are not safe
      - etc.,

Example Program

```
#include <stdio.h>

int main() {
   int buf[20],a[6],sum=0;
   int i;
   /*Initialize buf*/
   for(i=0;i<25;++i) {
        buf[i]=i;
   }
   /*Read from a */
   for(int j=0;j<6;++i) {
        sum = sum + a[i];
   }
   return 0;
}
```

Buffer overrun

```
  sub     esp, 68h
  xor     eax, eax
  xor     ecx, ecx
  lea     edx, [esp]
loc_B:
  mov     [edx], ecx
  inc     ecx
  add     edx, 4
  cmp     ecx, 19h
  jl      short loc_B

  push    esi
  lea     ecx, [esp+84]
  mov     edx, 6
loc_20:
  mov     esi, [ecx]
  add     ecx, 4
  add     eax, esi
  dec     edx
  jnz     short loc_20
  pop     esi
  add     esp, 68h
  retn
```

Example Program

[CodeSurfer/x86 without VSA]

[CodeSurfer/x86 with VSA]

```
  sub     esp, 68h
  xor     eax, eax
  xor     ecx, ecx
  lea     edx, [esp]
loc_B:
  mov     [edx], ecx
  inc     ecx
  add     edx, 4
  cmp     ecx, 19h
  jl      short loc_B

  push    esi
  lea     ecx, [esp+84]
  mov     edx, 6
loc_20:
  mov     esi, [ecx]
  add     ecx, 4
  add     eax, esi
  dec     edx
  jnz     short loc_20
  pop     esi
  add     esp, 68h
  retn
```

WiSA Demo Session

Hao Wang
BREW – A Binary REWriting Infrastructure > Hao

Hong Lin
Dynamic Buffer Overrun Detection > Hong

Mihai Christodorescu
Malicious Code Obfuscation and Detection > Mihai

Vinod Ganapathy
Statically Detecting Buffer Overruns > Vinod

Gogul Balakrishnan
Enhanced CodeSurfer/x86 > Gogul