WiSA Demo Session

Hao Wang, Hong Lin
*BREW - A Binary REWring Infrastructure*

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

Mihai Christodorescu
*Malicious Code Obfuscation and Detection*

Vinod Ganapathy
*Statically Detecting Buffer Overruns*

Gogul Balakrishnan
*Enhanced CodeSurfer/x86*
BREW—A Binary REWriting Infrastructure (Demo)

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

Binary

IDA Pro
Parse Binary
Build CFGs

Connector
Memory Analysis
BREW Rewrite
Generate Code

Codesurfer
Build SDG
Browse

Clients
Detect Malicious Code
Detect Buffer Overrun
Build Program Specification

Generated Binary
Goal: Binary-to-Binary rewriting

Two-step process:

1. Disassemble original binary
   - Intermediate Data Structures

2. Reassemble new binary
   - New Binary

Binary → IDA Pro
   - Parse Binary
   - Build CFGs

Connector
   - Memory Analysis

BREW
   - Rewrite

Generate Code
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

CFG (nodes=6, formals=0, decls=0)

(1) CFGEnter
k=0, ck=0, u=0, du=0,
visited = false, preds=0, succs=2
EA = 01001940
sub_1001940 proc near

(3) CFGBody
k=0, ck=0, u=0, du=0,
visited = true, preds=2, succs=2
EA = 01001940
sub_1001940 proc near

AST

ID_proc_statement "sub_1001940"

ID_symbol "sub_1001940"

ID_instr_statement "jmp"
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

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

Mihai Christodorescu
Malicious Code Obfuscation and Detection

Vinod Ganapathy
Statically Detecting Buffer Overruns

Gogul Balakrishnan
Enhanced CodeSurfer/x86
Dynamic Buffer Overrun Detection

Hong Lin
WiSA
University of Wisconsin - Madison
Code Instrumentation

Binary

IDA Pro
- Parse Binary
- Build CFGs

Connector
- Memory Analysis
- BREW
  - Rewrite
- Generate Code

Codesurfer
- Build SDG
- Browse

Clients
- Detect Malicious Code
- Detect Buffer Overrun
- Build Program Specification

Generated Binary
Demo Outline

- **Winhelp32.exe**
  - normal operation
  - buffer overrun exploit

- **Instrumented winhelp32.exe with dynamic buffer overrun detection**
  - maintains original behavior with valid input
  - detects overrun with malicious input
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

July 22, 2003
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

ret_func0

100DFB2
6F0E0
EvilAddr

Stack growth

overrun!
Return Address Defense (RAD)

• Idea:
  save a redundant copy of return address on a different stack [Chiuieh & Hsu ’01]

• Implementation
  - hardware
  - source code
  - binary
Using BREW

```c
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
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


WiSA Demo Session

Hao Wang, Hong Lin
*BREW - A Binary REWriting Infrastructure*

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

Mihai Christodorescu
*Malicious Code Obfuscation and Detection*

Vinod Ganapathy
*Statically Detecting Buffer Overruns*

Gogul Balakrishnan
*Enhanced CodeSurfer/x86*
Malicious Code Obfuscation and Detection

Mihai Christodorescu
WiSA
University of Wisconsin, Madison
Virus Obfuscation
through Binary Code Rewriting

Binary

IDA Pro
Parse Binary
Build CFGs

Connector
Memory Analysis

BREW
Rewrite

Codesurfer
Build SDG
Browse

Clients
Detect Malicious Code
Detect Buffer Overrun
Build Program Specification

Generated Binary

Detect Malicious Code

Build Program Specification
Malicious Code Detection

1. Binary
2. IDA Pro
   - Parse Binary
   - Build CFGs
3. Connector
   - Memory Analysis
4. BREW
   - Rewrite
5. Codesurfer
   - Build SDG
   - Browse
6. Clients
   - Detect Malicious Code
   - Detect Buffer Overrun
   - Build Program Specification
7. Generated Binary
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 data
- Encode / encrypt

Change control:
- Insert garbage
- Encode / encrypt
- 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/destroy data structures
Obfuscation

**Virus Code**  
(from F0sf0r0):

```
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 F0sf0r0):

```
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
```
Binary Rewriter API

• **Easy to obfuscate:**

```plaintext
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*

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

Mihai Christodorescu
*Malicious Code Obfuscation and Detection*

Vinod Ganapathy
*Statically Detecting Buffer Overruns*

Gogul Balakrishnan
*Enhanced CodeSurfer/x86*
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

- Binary
  - IDA Pro
    - Parse Binary
    - Build CFGs
  - Connector
    - Memory Analysis
    - BREW
      - Rewrite
    - Generate Code
  - Codesurfer
    - Parse C
    - Build SDG
    - Browse
  - Clients
    - Detect Malicious Code
    - Detect Buffer Overrun
    - Build Program Specification
Buffer overrun detection for C programs

IDA Pro
- Parse Binary
- Build CFGs

Binary

Connector
- Memory Analysis
- Build CFGs
- BREW
  - Rewrite
  - Generate Code

Codesurfer
- Parse C
- Build SDG
- Browse

Source Code

Clients
- Detect Malicious Code
- Detect Buffer Overrun
- Build Program Specification

July 22, 2003
WiSA http://www.cs.wisc.edu/wisa
Results

• Analyzed wuftpdl-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-overflow scan.
Buffer Overruns Scan in progress

Doing Buffer Overruns scan...
Loading......
13966 Relevant Vertices

5.01%
10.03%
15.04%
20.06%
25.07%
30.09%
35.1%
40.11%
Warnings on WuFtpD-2.6.2

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

copy

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
Enhanced CodeSurfer/x86

Gogul Balakrishnan
Value-set Analysis

Binary

IDA Pro
- Parse Binary
- Build CFGs

Connector
- Memory Analysis
- Build SDG

BREW
- Rewrite
- Browse

Codesurfer
- Detect Malicious Code
- Detect Buffer Overrun

Clients
- Build Program Specification

Generated Binary
Existing CodeSurfer/x86

- Existing CodeSurfer/x86
  - Ignores pointer-arithmetic
  - Unsafe used, killed and c-killed sets
    ⇒ 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;
}
```
Example Program

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

[CodeSurfer/x86 without VSA]

[CodeSurfer/x86 with VSA]
WiSA Demo Session

Hao Wang
*BREW - A Binary REWriting Infrastructure*

Hong Lin
*Dynamic Buffer Overrun Detection*

Mihai Christodorescu
*Malicious Code Obfuscation and Detection*

Vinod Ganapathy
*Statically Detecting Buffer Overruns*

Gogul Balakrishnan
*Enhanced CodeSurfer/x86*