Hardware-based Control Flow Monitoring to Prevent Malicious Control Flow Redirection

Nidhi Shah
PhD Candidate

Linda Wills
Associate Professor

School of Electrical and Computer Engineering
Georgia Institute of Technology
Outline

- Introduction
  - Motivation for hardware-based approach
  - Targeted security problems
  - Expected benefits of dynamic hardware-based approach
- Hardware-Based Control Flow Monitoring
  - Conceptual Architecture Model
  - Leveraging Indirect Branch Prediction Mechanism
- Preliminary Results
- Open Questions
Current Approaches

- Flawfinder, ITS4, and RATS: Source Code Inspection
- Stackguard, ProPolice: Compiler Extension
- Libsafe: Library Call Monitoring
- Converting Static Array memory allocation to Dynamic Heap memory Allocation
Why Revisiting Code is Not Enough

- **Source Code Unavailability**
  - Legacy Application
  - Third Party Integration

- **If Source Code is Available**
  - Requires Diverse Programming Language Support
  - Operating Environment Support
  - Tractability of Static Approaches: Swamped with Analysis Data

- **Zero Day Attack**: With only static analysis support protection is not possible

- **Missing Software Updates, Simultaneous Existence of Multiple Versions**
Control Flow Redirection

- Stack Smashing
- Heap Exploits
- Format String Exploits

Exploits that allow attacker to overwrite any random bytes in the memory!!
Benefits of Hardware-based Approach

- Code that is executing needs to be analyzed
  - Extraneous code is not exposed so can be filtered
  - Fast execution
- All applications supported by hardware are supported
  - No language dependency
  - No operating system dependency
- Malicious behavior is caught immediately
  - Supports Zero Day Exploits
  - No need to update the system
  - Multiple versions of software can exist in the system with equal security support
Outline

- Introduction
  - Motivation for hardware-based approach
  - Targeted security problems
  - Expected benefits of dynamic approach
- Hardware-Based Control Flow Monitoring
  - Conceptual Architecture Model
  - Leveraging Indirect Branch Prediction Mechanism
- Preliminary Results
- Open Questions
Hardware Based Control Flow Monitoring

```c
if ( i > 10 )
    a = b;
else
    a = c;
strcpy(a,b);
return;
```
Hardware Based Control Flow Monitoring

- Only branches that compute their target address (Indirect Branches) can be exploited

- Examples:
  - Function pointers (e.g., C++ vtables, callbacks for dynamic binding)
  - Return addresses
  - Global Offset Table for dynamic library linking
  - DTORS section for storing constructor/destructor address

- We model indirect branch targets and monitor them for change in behavior
Conceptual Architecture View

- Branch Target Tag Cache
  - Branch Address
  - Branch Target Address
  - Branch Miss
  - Target Mismatch

- Indirect Branch

- Make new entry
- Exception

- Fetch
- Dispatch
- Execute
- Commit
Branch Prediction

Branch Resolution

Branch Address

<table>
<thead>
<tr>
<th>Branch Address</th>
<th>Branch Taken/not Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>400c</td>
<td>taken</td>
</tr>
</tbody>
</table>

is Branch?

Fetch

Dispatch

Execute

Commit

4010: jmp[r3]
400c: brtrue __address
4008: add r3, r3
4004: cmp r1, r2

Update Branch Prediction If Miss Prediction

Branch Address, Branch target address
Branch Prediction

Branch Resolution

is Indirect Branch?

yes

4010: jmp[r3]

Branch Address

Branch Taken/not Taken

4008 taken

Target Address

Branch Address

4010

2992

Update Branch Prediction

If Miss Prediction

Branch Address, Branch target address

4010: jmp[r3]

Fetch

Dispatch

Execute

Commit
Relative Branch Target Address Range Distribution-Spec2000 Benchmarks

16MB ~ $2^{24}$
Relative Target Address Range Distribution-Spec2000 Benchmarks

Target Address

Benchmarks

16MB ~ $2^{24}$
Tag Comparator to Detect Malicious Control Flow Redirection

Branch Address (32)

Target Address (32)

Tag (8)

Match ?

No

Exception

Yes

continue

Index (n)

Tag Cache (8x2^n)

Index function
Extension to Indirect Branch Prediction for Malicious Control Flow Redirection

Indirect Branch Prediction Hardware

Branch Address (32)
Path Register (32)

Target Cache (32x2^n)

Index (n)

Target

Additional Logic for Malicious Control Flow

Target Address (32)

32.....24

Tag (8)

Match ?
No
Exception
Yes
continue

11/07/2005  Georgia Institute of Technology
Outline

- Introduction
  - Motivation for hardware-based approach
  - Targeted security problems
  - Expected benefits of dynamic approach
- Hardware-Based Control Flow Monitoring
  - Conceptual Architecture Model
  - Leveraging Indirect Branch Prediction Mechanism
- Preliminary Results
- Open Questions
Simulation Environment

- Simics® x86 Simulator from VirtueTech
- Linux Operating System
- Test Cases
  - WU-FTP : Format String
  - Home Brewed EchoServer : String Buffer Overflow
Wu-FTP Vulnerability

void vreply(long flags, int n, char *fmt, va_list ap)
{
    char buf[BUFSIZE];
    flags &= USE_REPLY_NOTFMT | USE_REPLY_LONG;
    if(n)
        sprintf(buf, "%03d%c" n,flags & USE_REPLY_LONG? ' ': ' ');
    snprintf(buf+(n ? 4 : 0), n ? Sizeof(buf) - 4:
    sizeof(buf), "%s", fmt);
    else
        vsnprintf(buf+(n ? 4 : 0), n? sizeof(buf) - 4 : sizeof(buf), fmt, ap);
    if (debuf)
        syslog(LOG_DEBUG, "< ---- %s", buf);
        printf("%s \n\n", buf);
      .... .... ...
}

Step 1: objdump -R /usr/sbin/in.ftpd (Reverse engineer program binary to see GOT table from in assembly format)

083b9213 R_386_JUMP_SLOT printf

Step 2: anonymous connect to wu-ftp server
Step 3: send malicious string to print parameters on the stack and determine where the string that attacker (here we) send is located
Step 4: send malicious string to overwrite printf address

Suppose the string that we send start at parameter 272 on the stack then send string containing:
1) Number of characters equal to address of the malicious code followed by,
2) Address of the printf function entry in GOT table e.g.
   "%4000$s\x13\x92\x3b\08%272$s"
Preliminary Results

- Echoserver: Buffer overflow exploited to overwrite a function pointer on the function stack: successfully caught
- Wu-FTP Format String Vulnerability Exploited to Overwrite a Global Offset Table (GOT) Entry
  - GOT is used for Dynamic library linking, widely used software technique for third party share
Open Issues

- Extension to RISC architecture
- Support for applications that are passive in nature
Acknowledgement

- EASL Group, Georgia Institute of Technology
- National Science Foundation
  - Grant CCR-0092552 & CCR-0209179