Pre-Patched Software

Jianing Guo       Jun Yuan       Rob Johnson
Stony Brook University

http://www.splat.cs.sunysb.edu/
Bugs in Deployed Software

- The problem with patches:
  - Slow and error-prone to develop
  - Long “window of vulnerability” that exposes users to a possible “zero day exploit”

- The problem with run-time checks
  - High overhead
  - Compatibility issues

- Pre-Patched Software
  - Uses latent run-time checks
  - Low run-time overhead
  - Rapid response to new vulnerabilities
  - Backwards compatible
Zero Day Exploit Problem

Deploying

Testing...

User Testing...

Oy! Too Slow & too late!
Pre-Patched Software

Security Checks

Deploying...

Check

Whew! Good thing that I have the check on.

Testing...

User Testing...

Check

Whew! Good thing that I have the check on.
Benefits

- Provides immediate response to vulnerabilities
- Prevents “zero day exploit”
- Users don’t pay a visible overhead until it becomes necessary
- Shipping instrumented binaries allows users to test in advance
Prototype: Memsafe

- Checks against bounds violations
- Based on Jones & Kelly’s [Jones 97] approach to C bounds checking
- Implemented using CIL [Necula 02] platform
void foo() {
    int arr[5];
    B_arr = Register(arr);
    int n = 1;
    B_arr = LookUp(arr + 0);
    Check(arr+0, B_arr);
    arr[0] = n;
    n = bar(n, arr, B_arr);
}

int bar(int n, int*a, bounds B_a) {
    int i, s = 0;
    Check(a+0, B_a);
    s = a[0];
    for( i = 0; i <= n; i++ ) {
        a++;
    }
    return s;
}

- Register only necessary variables
- Caching bounds info
- Bounds passing across functions.
- Support manipulation for OOB ptrs

Not a Problem
Memsafe Optimizations

- Bounds caching
- Bounds passing
- Loop optimization
- Static check elimination
Run-time Check Activation

- Selectively turn on checks – reduces patch overhead
- Instrumentation dependency -- enables metadata maintenance
- Fast path/Slow path – saves time on branch checking

Not memsafe specific
Selective Check Activation

void foo () {
    int arr[5];
    B_arr = Register(arr);
    int n = 1;
    Check(arr+0, B_arr);
    arr[0] = n;
    n = bar (arr, B-a);
}
int bar (int n, int*a, bounds B-a )
{
    int i, s = 0;
    Check(a+0, B-a);
    s = a[0];
    for( i = 0; i <= n; i++ ) {
        a++;
    }
    return s;
}

Checks can be activated independently based on the bit map.
void foo () {
    int arr[5];
    int n = 1;
    arr[0] = n;
    n = bar (arr, 3);
}

int bar (int n, int* a, bounds const B-a )
{
    int i, s = 0;
    s = a[0];
    for( i = 0; i <= n; i++ ) {
        a++;
    }
    return s;
}

Check(arr+0, B_arr);

B_arr = Register(arr);

Check(a+0, B-a);

B_arr = Register(arr);

Dependencies

- Dependency within a single function
- Dependency across functions

How do we determine the bounds for the activated check?
How to reduce the number of checks performed at run time?

Slow Path

```c
int arr[5];
B_arr = Register(arr);
int n = 1;
Check(arr+0, B_arr);
arr[0] = n;
n = bar (arr, B-a);
}
```

Fast Path

```c
int arr[5];
int n = 1;
arr[0] = n;
n = bar (arr, B-a);
```
Performance Evaluation

Three scenarios:
- All checks off (common case)
- One check on (occasional case)
- All checks on (only for testing)

Benchmark programs:
- Gzip and Gunzip
- Olden Benchmark [Rogers 95, Carlisle 95]
Results

- All checks on: About 12 times slower
- All checks off: Under 10% overhead

Ratio to GCC compiled version

Olden Benchmark

Gzip
Results

---

**One Check On:**
- About 33% overhead

Performance may vary depending on check locations.

Overhead is negligible in comparison to all checks on.

Olden Benchmark

---

8/16/2009

Security, Programming Languages, and Theory Lab
Limitations

- Not as efficient & complete as patches
- Depends on compiler auto-generation
- Only applicable to low level security bugs
Conclusion

- Pre-patched software provides immediate response to vulnerabilities
- Latent run-time checks incur low overhead while providing full coverage
- Pre-patched software makes code transformations usable by reducing overheads to a fraction
Q&A

Pre-Patched Software

Jianing Guo  Jun Yuan  Rob Johnson
Stony Brook University

http://www.splat.cs.sunysb.edu/