Efficiently Detecting Use-after-Free Exploits in Multi-threaded Programs

Master Project Presentation

Vinod Nigade

Supervised and Guided by:
Erik van der Kouwe, Cristiano Giuffrida
Vrije University, Amsterdam

14th September, 2016

Outline

Dangling Pointer

Pointer that points to freed memory

Use-after-Free

- Referencing memory after it has been freed
- Leads to program crash, data corruption, arbitrary code execution
- Highly critical and popular attack vector
- Common coding practice: Set pointer to benign value NULL when object is freed

- Static Analysis
  - Source code or Binary analysis
  - Hard to find: Needs complex inter-procedural and points-to analysis
- Dynamic Analysis
  - Track run-time pointer-object relationship
  - Introduces huge run-time overhead or limited applicability
  - DangNull [Lee et al., 2015] has on an average of 80% run-time overhead, FreeSentry [Younan, 2015] has 25% but with no thread-safety.

Static Instrumentation

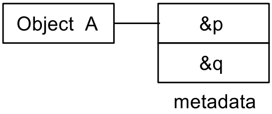
```
int
main(int argc, char *argv[])
{
        char *p, *q;
        p = (char *)malloc(100);  /* Allocated object A */
        |track_ptr|(&p, p);
        q = p + 10;
        |track_ptr|(&q, p + 10));
        free(p);
        |nullify_ptr|(p);
        return 0;
}
```

Background

Run-time Tracking

`track_ptr(ptr,obj)`

- Find metadata given any inbound object address (e.g. `p`, `p + 10`)
- Remove `ptr` from old object metadata /* Optional */
- Store `ptr` info in metadata



`nullify_ptr(obj)`

- Find metadata given object address
- Nullify pointers

```
p = NULL;
q = NULL;
```

- Trees, Hashtables etc. used to store and retrieve pointer-object relationship
- Requirement: fast metadata lookups (support range queries) and allocations with low memory overhead
- MetAlloc [Haller et al., 2016] shadow memory technique with fixed metadata lookup time.
- Supports range queries, variable and fixed compression ratio, uniform metadata tracking

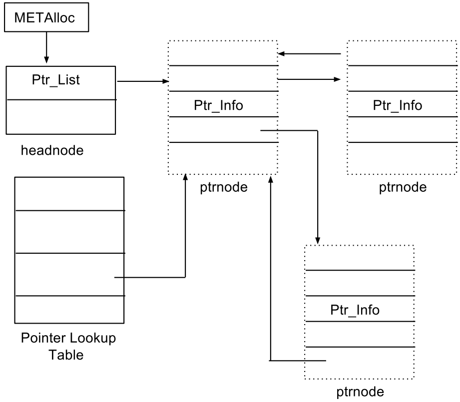


Figure 1: Data Structure design

Problem Statement

- State-of-the-art complete systems incur huge run-time overhead.
- Not all pointers (e.g. Stack and Global) are supported
- Large and complex applications like Web-Servers, Browsers can be multi-threaded

Problem

No efficient system to prevent Use-after-Free exploits in multi-threaded programs

Our Approach

Overview

In a simple Design

Need list of pointers per object.

- Large number of pointer tracking calls
- No need to remove pointer from old object list i.e. Pointer-to-object metadata not needed.
- metadata lookups: 1) Object-to-metadata

Our Approach
Data Structures

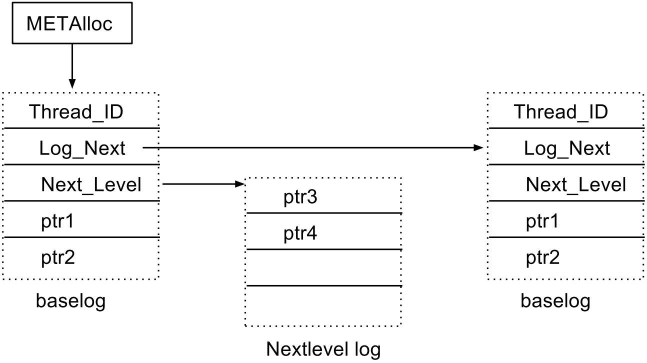


Figure 2: per-Object data structure

Our Approach

Log overflow

- Second level indirect log
- Exponential log resize

N-Lookbehind

- Lookbehind last N entries in the log
- Avoid duplicate pointer logging

Garbage Collection

- Treat log as a circular buffer
- Remove old stale pointers

Pointers subtraction

```
n = p - q        /* p and q are dangling pointers */
```

- Set uppermost bit to 1 i.e. $p = p \mid 0x8000000000000000$

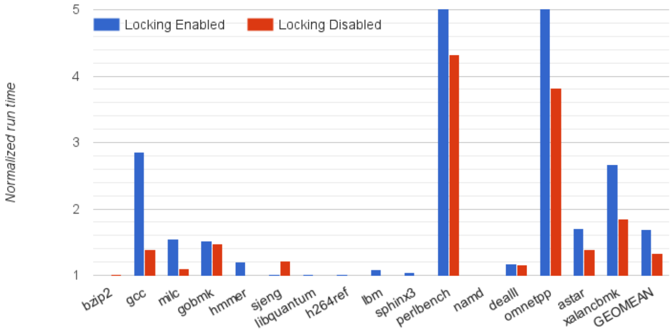
Off-by-one byte

- STL Vector{`start`, `next`, `end`}, `next` and `end` may point out-of-bound by one byte
- Increase object allocation size by one.

Evaluation

Thread-Safety

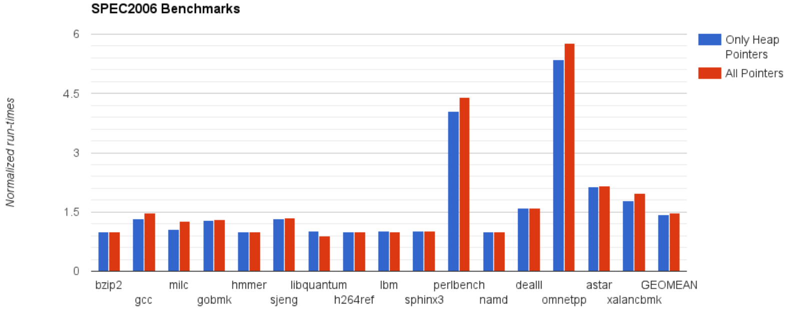
Average run-time overhead: With thread-safety (69.6%) and without (33.2%)



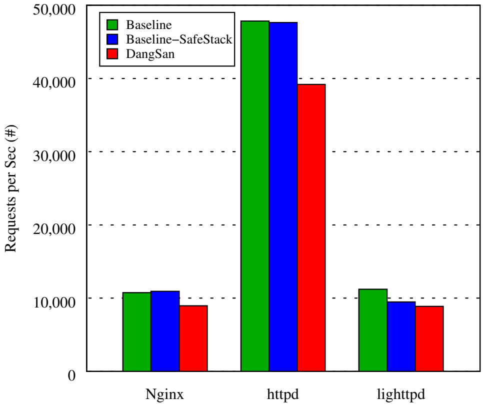
Evaluation

Performance Analysis

- All benchmarks, 43.9% (only heap pointers), 47.6% (all pointers)
- Without perlbench ;), 34.3% (only heap pointers), 37.2% (all pointers)



On an average, Web Servers have 12.8% throughput degradation and negligible service latency



Evaluation

Correctness

CVE 20102939: OpenSSL_1.0.0a 'ssl3_get_key_exchange()' Use-afterFree memory corruption vulnerability

Without protection	**With protection**
46912496417824:error:0407006A:rsa routines: RSA_padding_check_PKCS1_type_1:block type is not 01:rsa_pk1.c:100: 46912496417824:error:04067072:rsa routines:RSA_EAY_PUBLIC_DECRYPT: padding check failed:rsa_eay.c:699: 46912496417824:error:1408D07B:SSL routines:SSL3_GET_KEY_EXCHANGE:bad signature:s3_clnt.c:1570:	src/tcmalloc.cc:290] Attempt to free invalid pointer 0x80000000022ba510 ./runclient: line 9: 20200 Aborted $OPENSSL s_client connect localhost:$SERVER_PORT

Limitations and Future Work

- No stack objects are supported
- Sophisticated and advanced static instrumentation (e.g. avoid tracking for simple pointer arithmetic, i.e. p++)
- Opt-out selective functions from instrumentation.
- Implementation specific: Skip MetAlloc lookups to ignore Stack and Global objects

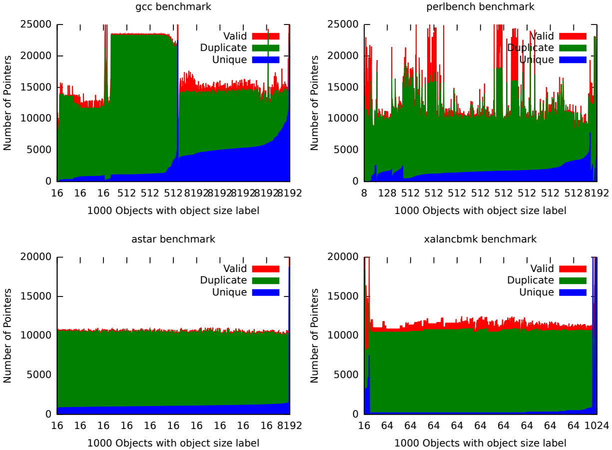
Conclusion

Practical and Complete

Our Use-after-Free detection system is practical and complete compared to state-of-the-art.

- DangNull: 80%, FreeSentry: 25%
- Our Approach: 47.6% (all pointers), 43.9% (heap pointers)
- Without perlbench: 37.2% (all pointers), 34.3% (heap pointers)

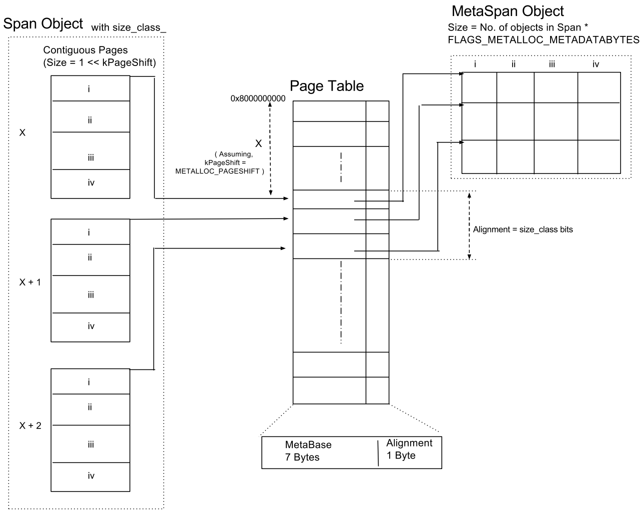
Extra Slides
Pointer Patterns



Duplicate and Stale pointers are much higher

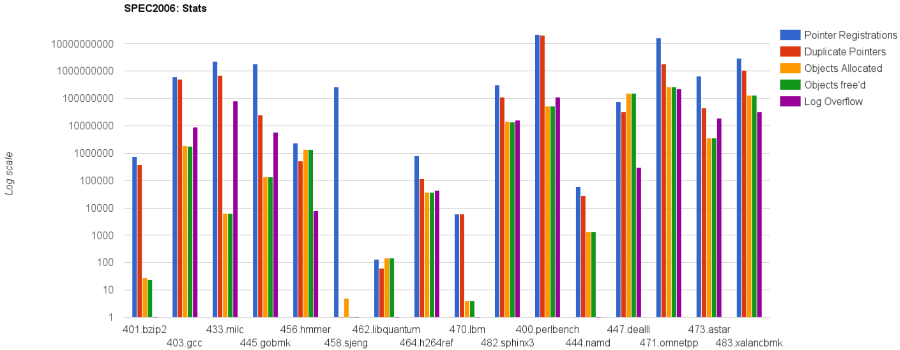
Extra Slides

MetAlloc



Extra Slides

Stats



Haller, I., van der Kouwe, E., Giuffrida, C., and Bos, H. (2016).
Metalloc: Efficient and comprehensive metadata management for software security hardening.
In *Proceedings of the 9th European Workshop on System Security*, EuroSec '16, pages 5:1–5:6, New York, NY, USA. ACM.

Lee, B., Song, C., Jang, Y., Wang, T., Kim, T., Lu, L., and Lee, W. (2015).
Preventing use-after-free with dangling pointers nullification.
In *Proceedings of the 2015 Internet Society Symposium on Network and Distributed Systems Security*.

Younan, Y. (2015).
Freesentry: Protecting against use-after-free vulnerabilities due to dangling pointers.
In *Proceedings of the 2015 Internet Society Symposium on Network and Distributed Systems Security*.