Shreds: Fine-grained Execution Units With Private Memory

APR 7TH, 2016

论文下载

1. 文章的目标：为开发人员提供一个机制，可以保护程序中的特殊数据，令这个程序被溢出之后，攻击者也无法获得内存中敏感的数据(如heartbleed泄露服务器私钥)
2. idea:
    - 提出Shred的概念，本身是一个thread执行中的一小段过程。
    - 只有特定的shred有读取敏感数据池s-pool中数据的权限。
    - 保证shred安全的前提下，确保效率

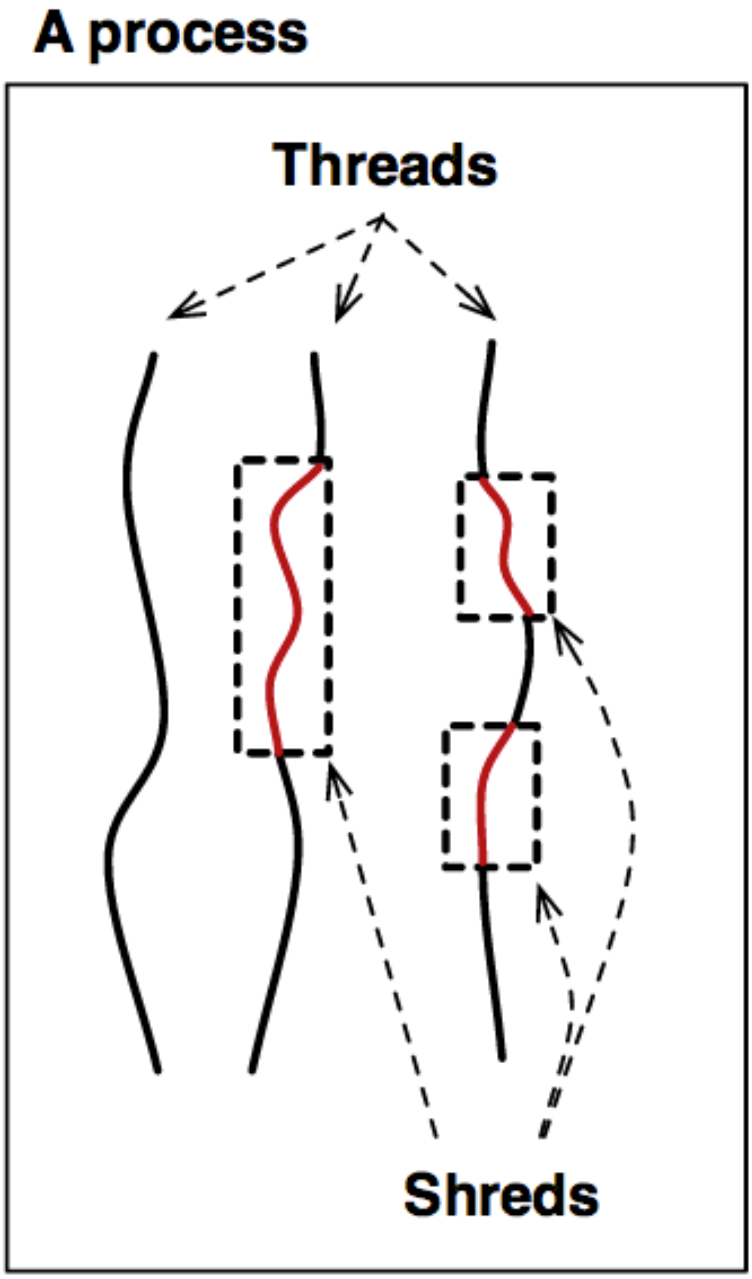
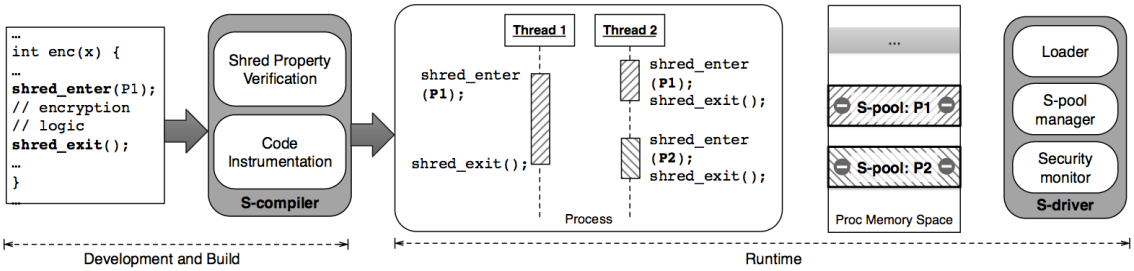


Fig. 1: Shreds, threads, and a process



系统设计与实现

系统目标

- Exclusive access to s-pool:
  - 只有对应的shred可以访问对应的s-pool的内容；
  - 由本文的系统保证.
- Non-leaky entry and exit:
  - 放在s-pool中的数据不应在任何时候出现在其他内存区域中；
  - 要保证开发人员自保证，Compiler检查。
- Untampered execution:
  - Shred内的代码应该是安全的；
  - 由Compiler用CFI保证。

为开发者提供的接口如下：

```
err_t shred_enter(int pool_desc);
err_t shred_exit();
void * spool_alloc(size_t size);
void spool_free(void *ptr);
```

- 系统保证的规则就是shred中alloc的内存只能由该shred访问。

S-Complier：

- Checking shred usage:
  - Compiler会确保每一个shred_enter都会被正常的exit
  - Compiler会做一次数据流分析，确保alloc的区域不会被显式的导出到其他内存区域或从其他内存区域读取

    - 处理了enter到exit的fopen等buffered IO
- Hardening in-shred control flows
    - Compiler会在shred内做CFI，而且是最轻量级的那那种，每个跳转都必须跳转到有效的基本块头
    - 如果shred和非shred都用到一个函数，那么这个函数会被复制两份
- Binding shreds and s-pools
    - 为了防止可能的任何攻击。。。enter和exit在代码中其实是不存在的，每一个shred所在的虚拟地址区域会预先被compiler分析出来，并放入.shred段中

S-Driver

- 需要操作系统层面保证其他线程不能访问shred的s-pool，而普通的MMU的机制只针对进程。
- 但是，ARM以及future Intel CPU有一个叫做memory domain的机制，可以指定每个核心可以访问的one of 16个domain
- 作者就利用这一个机制，限制了每个线程能访问的内存，如下

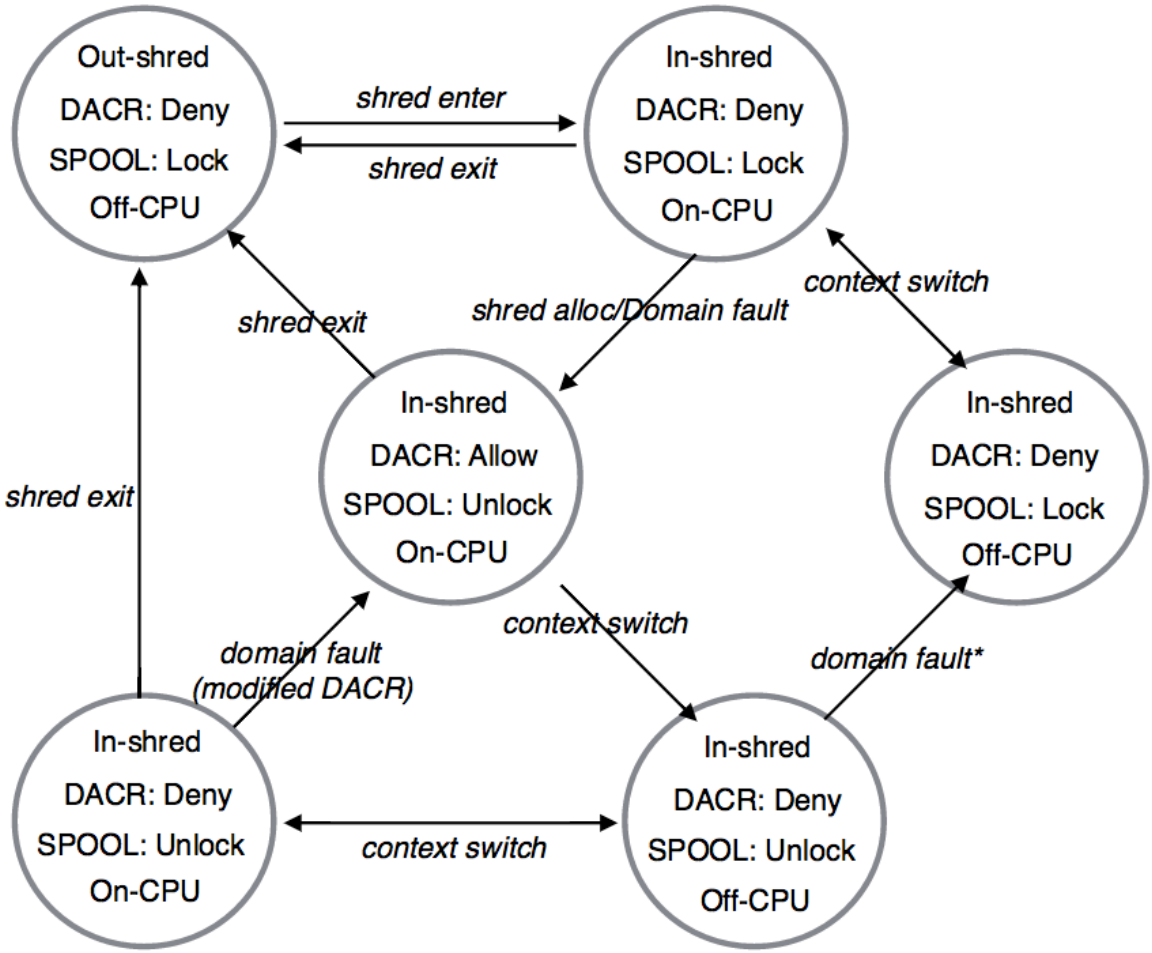


Fig. 4: A shred's transition of states

- S-Driver还有一些其他的工作:
  - 在进入Shred时将栈帧迁移进自己的s-pool中，并且在exit或者处理异常的时候，将栈帧恢复到原本的栈中（这个过程中会修改saved esp等等。。）
  - 对于所有使用shred的程序，运行时会被禁止/proc/mem*的访问和ptrace的调用。。
  - 在shred中被访问的文件会被系统记住，然后其他shred就无法再次访问该文件

作者使用内核模块虚拟一个设备/dev/shred，并用ioctl来调用这些api，这里总觉得不太对。。

实验

- 作者的实验主要三个问题:
  - 整个系统是不是容易部署、兼容性
  - Overhead
- 部署:

TABLE II: 5 open source softwares used in evaluation

	Executable Size(byte)	Category	Protected Data Type	Program Size(KLOC)
curl	227071 curl	http client	password	177
minizip	80572 miniunz 97749 minizip	file compression tool	password	7
openssh	2207588 ssh	remote login tool	credential	130
openssl	3093920 libcrypto.so	crypto library	crypto key	526
lighttpd	85135 mod_auth.so	web server	credential	56

TABLE III: Code changed and time spent in adoption tests

Application	Shred numbers	Code change(SLoC)	Adoption time(min)
curl	2	13	30
minizip	4	23	15
openssh	1	8	20
openssl	3	34	35
lighttpd	2	27	60

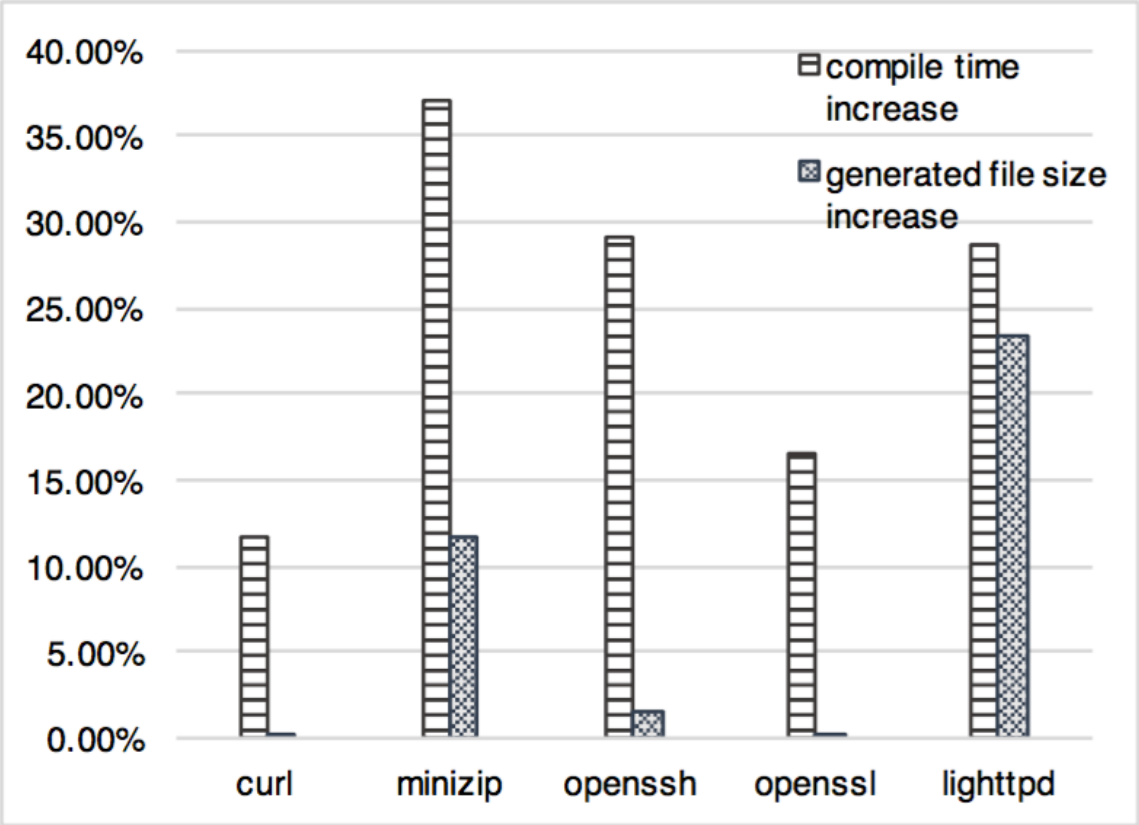
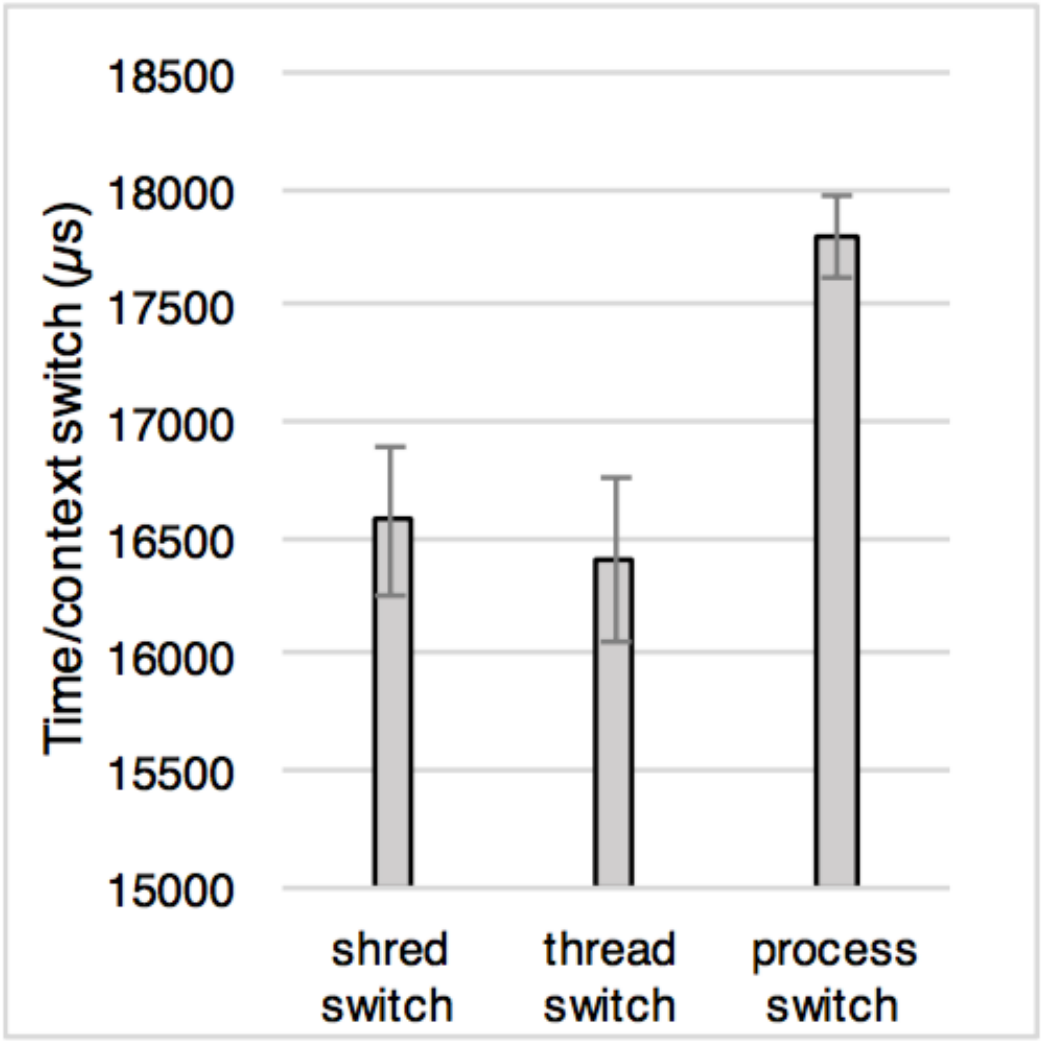


Fig. 5: The time and space overhead incurred by S-compiler during the offline compilation and instrumentation phase

- 性能开销：


18500
18000
17500
17000
16500
16000
15500
15000
Time/context switch (μs)
shred switch
thread switch
process switch

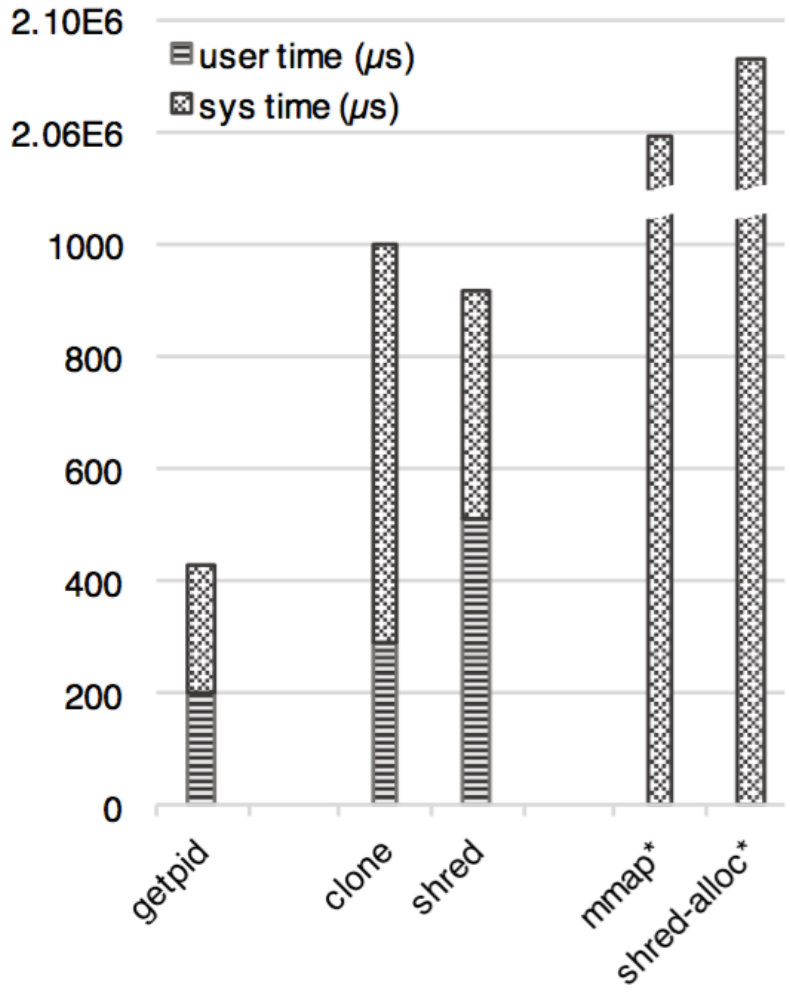


Fig. 7: Invocation time of shred APIs and reference system calls (the right-most two bars are on log scale). It shows that shred entry is faster than thread creation, and s-pool allocation is slightly slower than basic memory mapping.

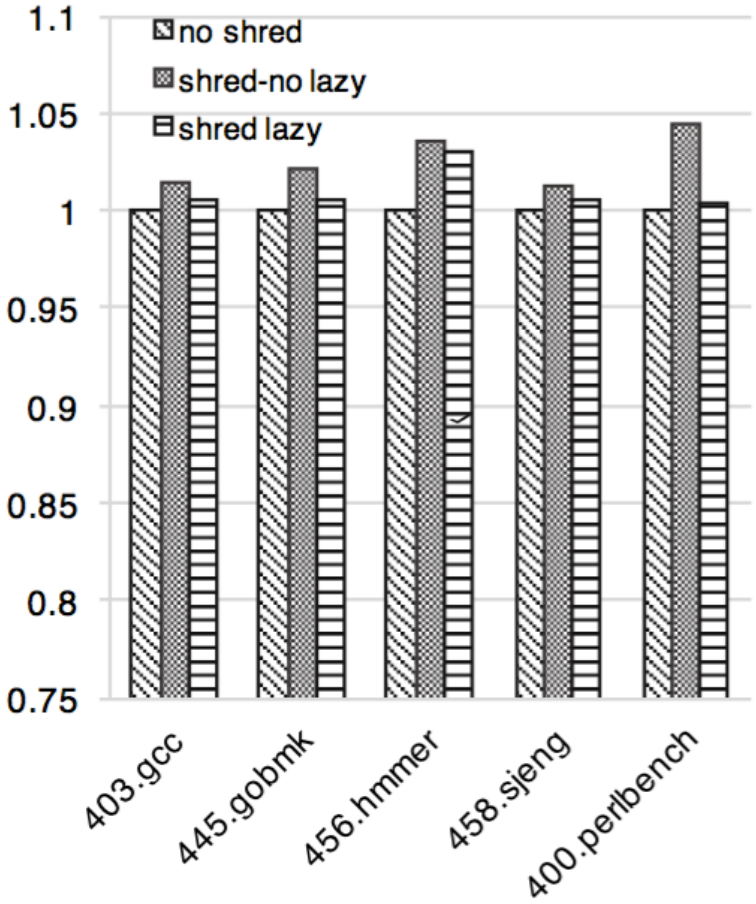


Fig. 8: Five SPEC2000 benchmark programs tested when: (1) no shred is used, (2) shreds are used but without the lazy domain adjustment turned on in S-driver, and (3) shreds are used with the lazy domain adjustment.