Capsicum: A Unix Process Isolation (Sandboxing) Mechanism

ShengYi Hong(aokblast@FreeBSD.org)
黃冠棠(i0905108390@gmail.com)
黃定凡(din2009siuc@gmail.com)

1. Introduction

a. 虛擬化

Sandbox 是一種虛擬化的技術，一般來說，虛擬化的技術可以分成三類，一是硬體虛擬化，必須有硬體指令集的的支援或者 syscall translation 。代表性實作有 bhyve, kvm, 跟 vmware，另一種是系統虛擬化，在同一個作業系統核心當基底的情況下對檔案系統跟其他資源 (e.g. CPU 核心，記憶體) 做切割分離，系統虛擬化的代表性實作有 jail, zone, 跟 lxc，第三種就是今天的主題，基於執行緒隔離的 Sandbox，他以應用程式的角度隔離資源，這種虛擬化的方式隔離資源最輕量，因為不需要虛擬出任何系統資源，完全以限制的方式產生一個分離的環境，這樣的方式符合了資訊安全的最小權限原則，代表性實作有 pledge 跟今天的主角 capsicum，這三種虛擬化的方式都同時做到了隔離這件事，但粒度跟實作的方式不同。

b. Seccomp(2)

Seccomp(2) 全名是 Secure Computing with filters 是早期的 Sandboxing 機制，他在 2005 年由 Linux 實作並提出，使用方式有兩種，第一種是 SECCOMP_SET_STRICT 模式 (以下簡稱嚴格模式)，在嚴格模式下，只有固定的 system call 可以被使用，分別是 read, write, exit, 跟 sigreturn 這幾個 system call，另一種則是 SECCOMP_SET_MODE_FILTER 模式 (以下簡稱 BPF 模式)，透過編寫 BPF 規則來限制系統的行為。在之後出了一個包裝產物 libseccomp，在 seccomp 之上包裝了一層，使 seccomp 比較容易使用。

由上述的說明可以得知，在嚴格模式下，作業系統核心很暴力的鎖死了所有系統呼叫，因此粒度只限於應用級別，而不能對個別的資源做限制，在 BPF 模式下，則因為 BPF 規則難以編寫而遭很多人詬病，且 BPF 在使用上就是給網路 socket 使用的，使用 socket 的皮套工具不僅不符合 Unix 哲學，且也有偷工減料之嫌疑，OpenBSD 的開發者甚至稱這種東西根本不是在保護作業系統安全，而是在保護維運工程師的飯碗安全，並且這整個框架在沒有 libseccomp 的幫助之下都難以使用。在這種環境之下大家開始在尋找其他更好的解決方案，也就有了後來的 Pledge 跟 Capsicum 的出現。

OpenBSD's unveil()

Posted Sep 29, 2018 5:33 UTC (Sat) by **flussence** (subscriber, #85566)
Parent article: OpenBSD's unveil()

The real value of this syscall is that it'll become a new sane default across the entire OS, as was the case with pledge().

AFAIK, Linux offers no comparable "seatbelts and airbags" API for application code to opt in to. Seccomp exists now, but it's far too hard for the average developer to use, and so they don't. And the existing security subsystems seem more focused on creating sysadmin job security than system security; it's a full-time effort to understand and configure some of them. This whole situation really ought to change.

圖(一) **OpenBSD** 開發者的評論，他認為 **Seccomp** 是在保護維運工程師的飯碗安全

c. Capsicum

Capsicum 是劍橋大學 Computer Laboratory 的一個計畫，是 Robert Watson 博士的博士論文，該實作的論文有上 USENIX Security Symposium 會議作為一個 talk，並且整個計畫由 Google 贊助。

在 Unix 哲學裡面，萬物皆是檔案，從一般的 File，Socket，Pipeline，跟 Prcoss (/proc, sysctl) 都是檔案，甚至 IPC 如 Uinx Domain Socket 也是檔案，這樣的哲學不止 Unix 有，甚至延伸到所有的 Unix，類 Unix 系統，例如 Linux, BSD, AIX。

在這樣的哲學之下，也就是說，我們只要在檔案的 handler 上做限制，也就能控制系統大部分的功能了，在 Unix 的所有派生系統核心中，檔案的 handler 就是 file descriptor，那 Capsicum 就是在這樣子的邏輯應運而生的一個 sandbox 框架，他在 file descriptor 上做限制，限制他能使用的 system call，把粒度從應用級別提升到檔案級別，或者說讓這個框架有了管理資源的能力，畢竟像網路之類的功能 (e.g. sendto, receive) 也被抽象成檔案了。

在這邊需要注意一點，由於 Capsicum 本質仍然是沙盒框架，所以他仍然限制了大部分的系統呼叫如 fork, exec 之類的不能使用。

那由於這個機制是限制在 file descriptor 上，我們要複製或者刪除一個 file descriptor 上的權限，其實只需要簡單的使用 dup 跟 close 就好，這也是開發使用上面更加的簡易。

在 Capsicum 裡面有一個封裝過的 Library 叫做 Casper，Casper 替換了 libc 裡面的部分 function，使這些 libc 的 function 需要經過 Casper Capability 的檢查，也就是說透過 Casper，這個 Capability 的就不限制在系統呼叫上，而是會擴大到一些 c 的 std function。

d. Capsicum vs Linux Capability

在原始的論文裡，Capsicum 被描述成 Unix 的 Capability，那就會讓大家聯想到 Linux 裡面好像也有一個 Capability，但 Linux 的 Capability 跟 Capsicum 其實是完全不同的概念。

Linux 的 Capability 旨在解決 root 權限過大的問題，因為 root 權限過大，而有些的系統功能又需要存取 root 權限，在以前的作法裡面，就必許使用 SUID，但是 SUID 又可能造成資安上的問題，當你的程式碼有 Bug 的時候，SUID 的權限很容易是駭客攻擊的一個跳板。那 Linux 的 Capability 把 root 權限拆分成多個不同的子權限，而當應用程式需要某

些權限的時候，他們只需要給予那個部分的 Capability，而不需要給應用程式 SUID。這樣大幅度減少了被攻擊的可能。
如前面所述，Capsicum 他其實是一個沙盒框架，是一個限制權限的過程，那 Linux 的 Capability 則是一個拆分 root 權限的工具，對 User 來說是一個提權的過程。

2. Kernel Source Code

a. 簡介

Capsicum 共有兩種模式，分別是 capability mode 和 capabilities，前者限制 process 存取 global namespace 的權限(e.g. fork, exec 之類的操作)，後者則是針對 file descriptor，限制特定 file descriptor 所能進行的操作。

b. Capability mode

cap_enter() 這個指令讓程式進到 capability mode，之後就不能存取 file system 或 IPC name spaces 等 global name spaces，只能透過針對 file descriptor 的 system call 來進行操作。

```
101  int
102  sys_cap_enter(struct thread *td, struct cap_enter_args *uap)
103  {
104          struct ucred *newcred, *oldcred;
105          struct proc *p;
106
107          if (IN_CAPABILITY_MODE(td))
108                  return (0);
109
110          newcred = crget();
111          p = td->td_proc;
112          PROC_LOCK(p);
113          oldcred = crcopysafe(p, newcred);
114          newcred->cr_flags |= CRED_FLAG_CAPMODE;
115          proc_set_cred(p, newcred);
116          PROC_UNLOCK(p);
117          crfree(oldcred);
118          return (0);
119  }
```

圖（二）freebsd-src/sys/kern/sys_capability.c 中 sys_cap_enter() 的實作

ucred 是 freebsd kernel 中用來儲存用戶權限（user credential）的 structure，cap_enter() 最重要的實作在第 114-115 行，把 ucred 中其中一個 flag（CAPMODE）on 起來，當然在 copy 和更動之前要先取得 PROC_LOCK 以避免 race condition。

c. Capabilities

capabilities 是用來限制 file descriptor 只能進行特定的操作，而這些操作的種類用 cap_rights 這個 structure 來表示。

```
45  #define CAP_RIGHTS_VERSION_00   0
46  /*
47  #define CAP_RIGHTS_VERSION_01   1
48  #define CAP_RIGHTS_VERSION_02   2
49  #define CAP_RIGHTS_VERSION_03   3
50  */
51  #define CAP_RIGHTS_VERSION      CAP_RIGHTS_VERSION_00
52
53  struct cap_rights {
54          uint64_t        cr_rights[CAP_RIGHTS_VERSION + 2];
55  };
```

圖(三)freebsd-src/sys/sys/caprights.h 中 struct cap_rights 的宣告

struct cap_rights(cap_rights_t) 其實就是有兩個元素的 uint64_t 陣列，其中的 64 + 64 個 bits(事實上有些 bits 有其他功能)用來表示各個權限。詳細定義如圖(四)。

```
76  /* Allows for openat(O_RDONLY), read(2), readv(2). */
77  #define CAP_READ                CAPRIGHT(0, 0x0000000000000001ULL)
78  /* Allows for openat(O_WRONLY | O_APPEND), write(2), writev(2). */
79  #define CAP_WRITE               CAPRIGHT(0, 0x0000000000000002ULL)
80  /* Allows for lseek(fd, 0, SEEK_CUR). */
81  #define CAP_SEEK_TELL           CAPRIGHT(0, 0x0000000000000004ULL)
82  /* Allows for lseek(2). */
83  #define CAP_SEEK                (CAP_SEEK_TELL | 0x0000000000000008ULL)
84  /* Allows for aio_read(2), pread(2), preadv(2). */
85  #define CAP_PREAD               (CAP_SEEK | CAP_READ)
```

圖(四)freebsd-src/sys/sys/capsicum.h 中各個 rights 的定義

capabilities 中有兩個重要的指令，分別是 cap_rights_init() 和 cap_rights_limit，前者用來將 cap_rights_t 設定為特定的權限，後者則是將設定好的 cap_rights_t 帶到 file descriptor上。

如同 cap_enter()，cap_rights_limit() 的實作也只是將設定好的 cap_rights_t 存到 file descriptor table 的 entry 上，等到實際使用 file decriptor 的時候才會去檢查該操作有沒有符合其權限。

以 fgets() 為例，fgets() 會呼叫到 fget_unlocked()，而這個 function 會先從 file descriptor table entry 中把 rights 取出來(圖五第 3264 行)，接著再檢查這個權限是否有包含所需的權限(圖五第 3272 行)。因為 cap_rights_t 是用 bit set 來存，因此在檢查權限是否包含的時候也只需用簡單的位元運算即可達成。

```
3240    int
3241    fget_unlocked(struct thread *td, int fd, cap_rights_t *needrightsp,
3242        struct file **fpp)
3243    {
```

```
3261    #ifdef CAPABILITIES
3262            seq = seqc_read_notmodify(fd_seqc(fdt, fd));
3263            fde = &fdt->fdt_ofiles[fd];
3264            haverights = cap_rights_fde_inline(fde);
3265            fp = fde->fde_file;
3266    #else
3267            fp = fdt->fdt_ofiles[fd].fde_file;
3268    #endif
3269            if (__predict_false(fp == NULL))
3270                    goto out_fallback;
3271    #ifdef CAPABILITIES
3272            if (__predict_false(cap_check_inline_transient(haverights, needrightsp)))
3273                    goto out_fallback;
3274    #endif
3275            if (__predict_false(!refcount_acquire_if_not_zero(&fp->f_count)))
3276                    goto out_fallback;
3277
```

圖（五）freebsd-src/sys/kern/kern_descrip.c 中 fget_unlocked() 的實作

3. 用戶的補丁
   a. **Traceroute** 介紹：
   Traceroute 是一種網路診斷工具，用於追蹤封包從源頭到目的地的路由過程。它可以幫助你了解數據封包在互聯網上傳輸時經過的所有網絡節點。
   當你使用 Traceroute 時，它會向目標設備（通常是一個網站或 IP 地址）發送一系列封包，每個封包具有不同的 TTL（Time to Live）值。TTL 值決定封包在網絡上的存活時間，每通過一個節點，TTL 值就會減少。當一個封包的 TTL 值為零時，路由器將丟棄該封包並返回一個 "Time Exceeded" 的錯誤消息。通過追蹤這些錯誤消息，Traceroute 可以建立一條從源頭到目的地的路由路徑。
   Traceroute 會在每個 TTL 值的封包上顯示經過的節點的 IP 地址和主機名稱（如果可用）。它還顯示每個節點的往返時間（Round-Trip Time，RTT），即從源頭發送封包到接收到該節點的回應所需的時間。通常，Traceroute 會發送多個封包到每個 TTL 值，並顯示平均 RTT 值。
   b. 補丁內容
      1. 用 casper，限制 dns 只能用 name2addr 跟 addr2name，前者用在本機，後者用在其他主機，因為 Traceroute 的輸出支援 ip 和 host names，所以他需要能拿到路徑上所有電腦的這兩項資訊。而在自己電腦上，他會透過 system call gethostname 得到本地的 host name，之後透過name2addr 得到本地的 ip，至於其他主機，因為你是用 ip 連過去的，所以你已經有 ip 這個資訊了，只要再透過 addr2name 就能得到他們的 host name。
      2. 只能訪問 ipv4，ipv6 需要用 traceroute6 這個指令。
      3. 如果需要進 sandbox，就會進 Capability mode，並限制 socket，值得注意的是他判斷進 Capability mode 會跟 n 這個參數有關，這個參數會決定要不要輸出 host name，因為本地的 host name 需要使用 systemcall，所以如果需要輸出 host name 程式就不會進入 capbility mode，否則無法使用這個 systemcall。
      4. sndsock，負責 send 的 socket，需要設定 TTL 的權限。
      5. s，負責接收訊息的 socket，程式使用 CAP_EVENT 裡面的 select 來設定 s。

```
487  #ifdef HAVE_LIBCASPER
488      const char *types[] = {"NAME", "ADDR"};
489      int families[1];
490      cap_channel_t *casper;
```

```
522  #ifdef HAVE_LIBCASPER
523      casper = cap_init();
524      if (casper == NULL)
525          errx(1, "unable to create casper process");
526      capdns = cap_service_open(casper, "system.dns");
527      if (capdns == NULL)
528          errx(1, "unable to open system.dns service");
529      if (cap_dns_type_limit(capdns, types, 2) < 0)
530          errx(1, "unable to limit access to system.dns service");
531      families[0] = AF_INET;
532      if (cap_dns_family_limit(capdns, families, 1) < 0)
533          errx(1, "unable to limit access to system.dns service");
534  #endif /* HAVE_LIBCASPER */
```

圖(六) 初始化 casper，並限制權限

在 488 行，開放 dns name2addr 和 daar2name 的權限，並於526~530行初始化。

在531~533行，設置 dns 只能使用IPv4。

這段程式碼只有在啟用 capser 限制權限時才會執行。

```
1739  char *
1740  inetname(struct in_addr in) {
1741      register char *cp;
1742      register struct hostent *hp;
1743      static int first = 1;
1744      static char domain[MAXHOSTNAMELEN + 1], line[MAXHOSTNAMELEN + 1];
1745
1746      if (first && !nflag) {
1747          first = 0;
1748          if (gethostname(domain, sizeof(domain) - 1) < 0)
1749              domain[0] = '\0';
1750          else {
1751              cp = strchr(domain, '.');
1752              if (cp == NULL) {
1753  #ifdef HAVE_LIBCASPER
```

圖(七) 取得本地主機名稱

如果在本地，且需要印出主機名稱，就呼叫 gethostname 這個 system call，因為呼叫此 system call 需要權限，所以這段沒有被放在 HAVE_LIBCASPER 裡面。

```
1753  #ifdef HAVE_LIBCASPER
1754              if (capdns != NULL)
1755                  hp = cap_gethostbyname(capdns, domain);
1756              else
1757  #endif
1758                  hp = gethostbyname(domain);
1759              if (hp != NULL)
1760                  cp = strchr(hp->h_name, '.');
1761          }
1762          if (cp == NULL)
1763              domain[0] = '\0';
1764          else {
1765              ++cp;
1766              (void)strncpy(domain, cp, sizeof(domain) - 1);
1767              domain[sizeof(domain) - 1] = '\0';
1768          }
1769      }
1770  }
```

```
1771      if (!nflag && in.s_addr != INADDR_ANY) {
1772  #ifdef HAVE_LIBCASPER
1773          if (capdns != NULL)
1774              hp = cap_gethostbyaddr(capdns, (char *)&in, sizeof(in),
1775                                     AF_INET);
1776          else
1777  #endif
1778              hp = gethostbyaddr((char *)&in, sizeof(in), AF_INET);
1779          if (hp != NULL) {
1780              if ((cp = strchr(hp->h_name, '.')) != NULL &&
1781                  strcmp(cp + 1, domain) == 0)
1782                  *cp = '\0';
1783              (void)strncpy(line, hp->h_name, sizeof(line) - 1);
1784              line[sizeof(line) - 1] = '\0';
1785              return (line);
1786          }
1787      }
1788      return (inet_ntoa(in));
1789  }
```

圖(八) 取得 ip名稱或是位址

如果有啟用 casper，則會改為呼叫 cap_開頭的函數，增加安全性。

```
973  #ifdef HAVE_LIBCASPER
974      cansandbox = true;
975  #else
976      if (nflag)
977          cansandbox = true;
978      else
979          cansandbox = false;
980  #endif
```

圖(九) 判斷程式是否進入沙盒

如果有啟用 casper 就進入，沒有的話則檢查是否需要印出 ip 名稱，不需要的話也進入沙盒模式，以回收使用者不需要的權限。

```
987      if (cansandbox && cap_enter() < 0) {
988          Fprintf(stderr, "%s: cap_enter: %s\n", prog, strerror(errno));
989          exit(1);
990      }
991
992      cap_rights_init(&rights, CAP_SEND, CAP_SETSOCKOPT);
993      if (cansandbox && cap_rights_limit(sndsock, &rights) < 0) {
994          Fprintf(stderr, "%s: cap_rights_limit sndsock: %s\n", prog,
995              strerror(errno));
996          exit(1);
997      }
998
999      cap_rights_init(&rights, CAP_RECV, CAP_EVENT);
1000     if (cansandbox && cap_rights_limit(s, &rights) < 0) {
1001         Fprintf(stderr, "%s: cap_rights_limit s: %s\n", prog,
1002             strerror(errno));
1003         exit(1);
1004     }
1005
```

圖(十) 限制 socket 的權限

sndsock 和 s 是兩個 socket。

第 992 ~997 行，給與 sndsock 傳送訊息與設置 socket 的權限。

第 999 ~1004 行，給與 s 傳送訊息與使用 CAP_EVENT 函式的權限。

```
1281  #if !defined(IP_HDRINCL) && defined(IP_TTL)
1282      if (setsockopt(sndsock, IPPROTO_IP, IP_TTL,
1283                     (char *)&ttl, sizeof(ttl)) < 0) {
1284          Fprintf(stderr, "%s: setsockopt ttl %d: %s\n",
1285              prog, ttl, strerror(errno));
1286          exit(1);
1287      }
1288  #endif
```

```
1290      cc = send(sndsock, (char *)outip, packlen, 0);
1291      if (cc < 0 || cc != packlen) {
1292          if (cc < 0)
1293              Fprintf(stderr, "%s: sendto: %s\n",
1294                      prog, strerror(errno));
1295          Printf("%s: wrote %s %d chars, ret=%d\n",
1296                 prog, hostname, packlen, cc);
1297          (void)fflush(stdout);
1298      }
```

圖(十一) 設置 TTL，並用 sndsock 傳送訊息

4. SELinux vs Capsicum

SElinux（Security-Enhanced Linux）是一種 Linux 安全模組，在 Linux 原有的檔案權限判斷後，再加上一層強制訪問控制（MAC）機制，以增強 Linux 系統的安全性。SElinux 最早由美國國家安全局（NSA）開發，並在 2000 年發布時開源。

SELinux 引入了強制訪問控制（MAC）概念，它會在 inode 中紀錄 Security Context，規定每個程式能讀寫的檔案，只有當程式要操作的檔案符合規定時才能操作，這樣的好處是它能避免權力被濫用，譬如當系統上運行著一個 Web Server，該應用程式使用 root 權限運行。沒有 SELinux 的情況下，如果攻擊者拿到 shell，他們將獲得 root 權限，這將對系統的安全性造成極大威脅。但若有了 SELinux，即使攻擊者能夠入侵系統，也會因為 SELinux 的機制而碰不到檔案被限制權限。

因為 Capsicum 的設計理念是開發一個框架，使應用程式提升安全性，而 SELinux 則是針對系統提出的安全模組，兩者的目標並不相同，於以下說明兩者的優缺點。

移植性：由於 Capsicum 不與系統綁定，相比於專為 Linux 開發的 SELinux，利用 Capsicum 的程式具有更高的可移植性，事實上，Capsicum 也可用於其他類 Unix 系統。

切換成本：Capsicum 較為輕量，若要切換模式，付出的成本較小，不像 SELinux，一旦 Disabled，就需要重新讀取所有檔案 inode 的預設值才能重新開啟。

使用難度：若使用者想更細緻的規範應用程式的權限，使用 Capsicum 較為簡單，因為 Capsicum 的應用方式就和其他 libiary 一樣，使用定義好的 API 來達到效果，但 SELinux 則需要編寫 inode 中的 Security Context，但如果以低度客製化的角度來說 (規則不多) 的情況下，SELinux 會更加的開箱即用。

實作面向: 由於 Capsicum 是綁定在 Process 上，所以被實作的地方是 file descriptor，那 SELinux 是系統級別的維護，所以他是綁定在 inode 上，雖然說都是在檔案上動手腳，不過是在不同的抽象層。

結論：若要限制一個應用程式的權限，以提升其安全性，使用 Capsicum 會更適合，畢竟它是為了應用程式而開發的框架，而 SELinux 則是站在系統的角度去提升安全性。

5. 為什麼 **Capsicum** 是開發者決定是否限制權限

對於一個應用程式來說，程式開發者一定比系統維運還要了解程式的邏輯，因此 Capsicum 的實作方式相對於由系統維運來管理還要更來的符合實際情況，再者 Capsicum 的邏輯可以提供更細緻的保護，畢竟在 system level 的角度 SELinux 來說，Capsicum 能夠提供檔案語意，因為在程式裡面我們能夠分辨 socket 跟普通的檔案，但對系統來說, procfs 跟 /etc/passwd 是等價的。