Kurs administrowania systemem Linux 2024

Lista zadań na pracownię nr 10

Na zajęcia 6 i 15 maja 2024

Uwaga: w większości poniższych zadań eksperymentowanie z systemem w maszynie wirtualnej może być znacznie wygodniejsze niż na gołym metalu.

Zadanie 1 (3 pkt). Uwaga: w niektórych dystrybucjach konfigurowanie czcionki jest dostępne w standardowej konfiguracji initramfs-tools. To zadanie ma sens jedynie w dystrybucjach standardowo pozbawionych tej możliwości (np. w Debianie). Zapoznaj się z pakietami `kbd` (zobacz `kbd-project.org`) i `console-tools` (zobacz `lct.sourceforge.net`) wykorzystywanymi przez większość dystrybucji Linuksa do konfigurowania czcionek i układu klawiatury w konsolach tekstowych oraz pakietem `console-setup` specyficznym dla Debiana i przygotuj krótkie omówienie sposobu konfiguracji czcionek konsolowych. Zapoznaj się w szczególności z programami `setfont(8)` i `setupcon(8)`.

Zapoznaj się z formatami PSF, BDF oraz PCF i przygotuj ich omówienie. Znajdź (np. w Internecie) ładną czcionkę rastrową i zainstaluj ją w swoim komputerze (ja np. poza typowymi czcionkami VGA, Fixed i Terminus używam też ZAP LCF i Leggie). Jeśli czcionka jest dostępna w formacie BDF lub PCF, to przydadzą się programy `pcf2bdf(1)` i `bdf2psf(1)`, zapoznaj się więc także z nimi.

Zapoznaj się z urządzeniami `/dev/tty*`, `/dev/vcs*` i `/dev/fb*` i programami `fbgrab(1)`, `fbcat(1)`, `fbi(1)`, `fbset(8)`, `chvt(1)`, `openvt(1)` (`open(1)`). Dowiedz się, co to jest *Linux framebuffer*, *kernel mode setting* oraz *hardware text mode*. Przygotuj krótkie omówienie. Znajdź odpowiedź na pytanie: dlaczego w konsoli tekstowej nie można używać czcionek wektorowych i *antialiasingu*, tak jak w terminalu graficznym.

Dokończ przedstawioną na wykładzie konfigurację czcionki w *initramfs* — przeczytaj wcześniej dokumentację `initramfs-tools`, gdyż szkice przedstawione na wykładzie nie zawierają wielu ważnych szczegółów i mogą nie działać poprawnie! Wygeneruj obraz *initramfs*, który konfiguruje taką samą czcionkę, jaka jest używana w konsoli tekstowej po uruchomieniu systemu.

Zadanie 2 (3 pkt). Ponad 10 lat temu Joanna Rutkowska zaprezentowała atak na chroniony kryptograficznie sprzęt pozostawiony bez dozoru (zwany „Evil Maid Attack"). Pokaż, jak łatwo można taki atak przeprowadzić w przypadku standardowej konfiguracji Debiana. Scenariusz:

1. Właściciel pozostawia komputer bez dozoru. Na dysku są trzy partycje: EFI System Partition (ESP), niezaszyfrowany `ext4` z katalogiem `/boot` i zaszyfrowany kontener z *rootfs*.

2. Atakujący uruchamia system ratunkowy na tym komputerze, montuje partycję `/boot` i modyfikuje *initramfs* w ten sposób, że skrypt, który pyta o hasło do odszyfrowania dysku zapisuje je także w jakimś niewidocznym miejscu (np. w pliku `/boot/grub/locale/tu.mo`).

3. Właściciel powraca do urządzenia, uruchamia je i podaje hasło. Po zakończonej pracy znów pozostawia urządzenie bez dozoru.

4. Atakujący powraca, odczytuje hasło, usuwa dodany plik, przywraca oryginalny *initramfs*, odszyfrowuje i montuje dysk twardy (w trybie `ro`), wykonuje jego kopię, po czym wyłącza urządzenie i się oddala.

Jeśli masz standardową instalację, to możesz eksperymentować na systemie produkcyjnym — uszkodzisz co najwyżej *initramfs*, który da się łatwo odtworzyć.

Z powyższych powodów niektóre dystrybucje (np. Manjaro) przechowują katalog `/boot` wewnątrz zaszyfrowanego kontenera (w medycynie ta przypadłość nosi łacińską nazwę *cryptorchismus*), a Grub2

pyta o hasło i go odszyfrowuje. Zauważ, że takie rozwiązanie również jest podatne, tylko zamiast *patch*-ować *initramfs* należy dodać niewielką modyfikację Grub-a. Ten atak można jednak zablokować za pomocą *Secure Boot*.

Zadanie 3 (3 pkt). Rozważmy mały komputer, który ma tylko jeden dysk niewielkiego rozmiaru, np. 32 GB. Nie warto dzielić go na partycje. Chcemy zainstalować na nim system w taki sposób, żeby był mniej podatny na atak *Evil Maid*. W tym celu cały dysk wewnętrzny komputera przeznaczymy na jeden zaszyfrowany kontener bez nagłówka. Jeśli jest to dysk mechaniczny, to możemy go najpierw zamazać losowymi danymi. Wówczas jeśli będziemy na nim przechowywać dane w szyfrowaniu AES-XTS, to *nie da się* nawet stwierdzić, czy tam są jakiekolwiek dane. Jeśli mamy dysk SSD i nie chcemy go przeciążać, to włączymy trimowanie. Wówczas wcześniejsze zamazywanie dysku *nie ma sensu*. Teraz jednak na podstawie analizy używanych sektorów atakujący będzie mógł wywnioskować co nieco na temat danych zawartych na dysku (ilość tych danych, typ systemu plików itp.). Klucz szyfrujący AES chcemy przechowywać w nagłówku LUKS2 na osobnym pendrivie.

Poniżej zakładamy, że komputer uruchamia się w trybie UEFI z wyłączonym *Secure Boot*. Uruchom z pendive'a system ratunkowy. Podłącz do komputera drugi pendrive. Utwórz na nim system partycji GPT i dwie partycje: ESP z FAT32 i linuksową z ext4. Poleceniem `grub-install` zainstaluj Grub2 na tym pendrivie. Aplikacja EFI (`grubx64.efi` itp.) powinna znaleźć się na partycji ESP w katalogu `/EFI/boot/`. Rootfs Grub-a powinien znaleźć się na drugiej partycji w kalatogu `/grub/`. Przygotuj też na drugiej partycji katalog `/boot/` do którego skopiujesz nagłówek LUKS2, jądro i initramfs instalowanego systemu. Na pendrivie pozostanie jedynie skonfigurowanie pliku `/grub/grub.cfg` aby uruchamiał podane jądro z initramfs-em.

Na komputerze zainstalujemy teraz Debiana. W tym celu za pomocą polecenia `cryptsetup` utwórz na dysku wewnętrznym komputera zaszyfrowany kontener (*de facto* `cryptsetup` nie zapisze na ten dysk ani bajta, ale potrzebuje go jako parametr), którego klucz szyfrujący znajdzie się na pendrivie w nagłówku LUKS2 zapisanym do pliku `/boot/hdr.img`. Uruchom ten kontener poleceniem `cryptsetup open` nazywając go `root`. Załóż na nim system plików ext4. Zamontuj go w katalogu `/target` systemu ratunkowego. Za pomocą polecenia `debootstrap(8)` zainstaluj na nim Debiana Bookworm. Utwórz katalog `/pendrive` i skonfiguruj go w `fstab` jako punkt montażowy dla drugiej partycji pendrive'a. Plik `fstab` powinien wyglądać mniej więcej tak:

```
/dev/mapper/root / ext4 noatime,nodiratime,discard,errors=remount-ro 0 1
UUID=e4c9cd7d-c16d-86c8-df8f-b4a0662fc437 /pendrive ext4 noatime,nodiratime,noauto 0 0
```

zaś plik `crypttab` następująco:

```
root /dev/disk/by-id/mmc-RAB773-0xbcb3f278 none luks,discard,header=/pendrive/boot/hdr.img
```

Dysk wewnętrzny komputera nie posiada systemu partycji, więc możemy się do niego odwołać jedynie poprzez `id`.

Podczas startu systemu initramfs musi tymasowo zamontować partycję pendrive'a. Wówczas standardowa konfiguracja initramfs potrafi już znaleźć nagłówek i rozszyfrować dysk.[1] Przed zakończeniem pracy initramfs musi jeszcze odmontować pendrive. Napisz odpowiednie hooki i skrypty initramfs, które to zrobią. Dokończ instalację (w szczególności skopiuj zainstalowane jądro i initramfs z katalogu `/target/boot` na pendrive) i zademonstruj działający system. Obecnie wszystkie najbardziej wrażliwe elementy systemu (bootloader, jądro, initramfs, nagłówek LUKS2) przenieśliśmy na pendrive'a, który użytkownik powinien mieć zawsze przy sobie. Atakujący, który ma dostęp jedynie do komputera, nie zmodyfikuje ani initramfs, ani bootloadera, ani nie dokona ataku słownikowego na hasło zaszyfrowane w nagłówku LUKS2. Jeśli atakujący uzyska dostęp do pendrive'a i komputera, to zabezpieczenia zdegradują się do standardowego rozwiązania debianowego.

W zadaniach z kolejnych list będziemy dalej rozwijać naszą instalację. Za tydzień dodamy zabezpieczenia pendrive'a: jego drugą partycję zaszyfrujemy w trybie `plain`. Kluczem szyfrującym będą pierwsze 32 bajty wewnętrznego dysku komputera. Podczas formatowania systemu ext4 program `mke2fs` zapisuje pierwsze dwa sektory dysku zerami. „Na zewnątrz" są one więc idealnie losowymi danymi. Można

[1] Trwają prace nad zautomatyzowaniem tej czynności, ale rozwiązanie nie jest jeszcze dostępne w Debianie Stable. Poza tym nasze rozwiązanie jest bardziej przyszłościowe — będziemy potem zmierzać do zaszyfrowania tej partycji.

także je nadpisać dowolnymi losowymi danym (wewnątrz lub na zewnątrz kontenera) — system ext4 nigdy ich nie używa (oryginalnie były przeznaczone na *bootloader*). Jeśli teraz atakujący uzyska dostęp do pendrive'a (ale nie do komputera), to niewiele zobaczy (choć nadal będzie mógł zmodyfikować Grub-a).

Kolejnym etapem (za dwa tygodnie) będzie wygenerowanie własnych certyfikatów Secure Boot, usunięcie standardowych certyfikatów Microsoftu z komputera (formalnie nie da się ich usunąć — można jedynie skonfigurować system, by ich nie używał), wgranie własnych certyfikatów na płytę główną, włączenie Secure Boot, zahasłowanie BIOS-u (żeby postronna osoba nie mogła wyłączyć Secure Boot) i podpisanie własnym certyfikatem Grub-a na pendrivie. W ten sposób komputer będzie można uruchomić *tylko* za pomocą naszego pendrive'a. Poza tym atakujący, który uzyska dostęp do naszego pendrive'a nie będzie mógł zmodyfikować jedynego nie zaszyfrowanego pliku — Grub-a.

Zadanie 4 (1 pkt). Skonstruuj ręcznie prosty *initramfs*, który będzie działał tylko w pamięci (nigdy nie wykona `switch_root`) i uruchomi powłokę systemową (program *Suckless Init* może być pomocny). Dodaj kilka programów, które będziesz uruchamiać za pomocą tej powłoki (`busybox`?).

Zadanie 5 (1 pkt). Rozbuduj system z poprzedniego zadania o możliwość zamontowania jakiegoś systemu plików w podkatalogu. System dalej pracuje z RAM, ale ma dostęp do dodatkowego systemu plików.

Zadanie 6 (3 pkt). Przygotuj *initramfs* działający według następującego schematu: po uruchomieniu `/init` lokalizuje system plików na dysku i montuje go w trybie `ro`, a na nim tworzy *OverlayFS* (w skrócie: `mount` z opcją `overlay`) na *tmpfs*-ie działającym w trybie `rw`. Na koniec `switch_root`-uje się do niego. System powinien działać tak, jak większość zwykłych dystrybucji *LiveCD*.

Zadanie 7 (1 pkt). W poprzednim zadaniu zamiast *OverlayFS* można użyć *aufs*. Dowiedz się, co to jest. W Debianie musisz obecnie zainstalować pakiety `aufs-dkms` i `aufs-tools`. Zaprezentuj na przykładzie z poprzedniego zadania jego działanie.

Zadanie 8 (3 pkt). Zapoznaj się z programami `wakeonlan(1)` i `etherwake(1)`. Skonfiguruj BIOS i kartę sieciową swojego komputera tak, żeby można było go zdalnie uruchomić za pomocą protokołu *wake-on-lan*.

Jeśli system powinien wystartować z zaszyfrowanej partycji, to pojawia się problem — potrzebne jest wówczas hasło do odszyfrowania. Hasło może zostać przekazane za pomocą SSH: po otrzymaniu ramki WOL system uruchamia się z *initramfs*, który uruchamia prosty serwer SSH i oczekuje na połączenie w celu przesłania hasła. Po otrzymaniu hasła wyłącza serwer SSH, odszyfrowuje partycję z *rootfs* i dalej postępuje normalnie (wykonuje `switch_root(8)` i kontynuuje rozruch).

Zapoznaj się z serwerem Dropbear. Spośród licznych implementacji serwera SSH ten wydaje się najlepszy do osadzenia w *initramfs*.

Skonfiguruj initramfs tak, by po otrzymaniu ramki WOL uruchomił serwer Dropbear. Skonfiguruj zdalną maszynę tak, żeby po wysłaniu ramki WOL łączyła się z uruchamianą maszyną i przekazywała hasło do odszyfrowania dysku.

Zadanie 9 (2 pkt). Zainstaluj Debiana lub inną dystrybucję z `dracut`-em zamiast `initramfs-tools`. Poeksperymentuj z konfiguracją za pomocą `dracut`-a. Przygotuj krótkie omówienie.

Zadanie 10 (3 pkt). Skonfiguruj jądro do uruchamiania bez *initramfs*. Główny system plików powinien być łatwo dostępny (np. nie szyfruj go!). Możliwe, że będzie trzeba przekompilować jądro. Przygotuj krótką prezentację działania systemu. *Wskazówka:* Gentoo jest w tym zadaniu dobrym wyborem.

Zadanie 11 (1 pkt). Zainstaluj jedną z dystrybucji pozwalającą uruchamiać system tylko z RAM (Alpine Linux jest obecnie najmodniejszy). Przygotuj krótkie omówienie struktury jego *rootfs*-a.