



دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

گزارش پروژه ۸
درس سیستم‌های عامل

آزمون حافظه توسط برنامه EFI و بوت امن

نگارش

محمد داودآبادی

علیرضا صمیمی

پارسا علیزاده

امیرمحمد شاهرضایی

استاد راهنما

دکتر اسدی و دکتر جلیلی

بهمن ۱۴۰۳

چکیده

در این پروژه بنا داریم تا در ابتدا با محیط برنامه نویسی EFI آشنا شویم به همین سبب کد Hello World با استفاده از EFI می‌نویسیم و گزارشی از آن ارائه می‌دهیم. در ادامه کمی در رابطه با آزمون حافظه، پیاده‌سازی آن و الگوریتم‌های آن تحقیق می‌کنیم. در آخر مقداری با Boot امن آشنا خواهیم شد.

فهرست مطالب

۱	نوشتن برنامه World Hello با EFI	۱
۱	۱-۱ آماده کردن محیط برنامه نویسی EFI	۱
۱	۲-۱ تحلیل کد	۱
۱	۳-۱ اجرای کد	۱
۲	آزمون حافظه	۲
۳	۱-۲ آشنایی با آزمون حافظه	۳
۳	۲-۲ آزمون‌ها	۳
۳	۱-۲-۲ Walking Ones	۳
۳	۲-۲-۲ Identity	۳
۴	۳-۲-۲ Rowhammer	۴
۴	۴-۲-۲ DMA	۴
۴	۳-۲ اجرای آزمون حافظه	۴
۵	نکات پیاده‌سازی	۳
۵	۱-۳ یکپارچه‌سازی با Shell	۵
۵	۲-۳ پیاده‌سازی چندهسته‌ای آزمون Identity	۵
۵	۳-۳ پیاده‌سازی GUI	۵

۷	۴ بوت امن
۷	۴-۱ شیه سازی
۸	۴-۲ آماده سازی برنامه
۸	۴-۳ آماده سازی بوت
۸	۴-۴ بوت با اثر انگشت برنامه
۹	۴-۵ بوت با امضای برنامه
۹	۴-۶ منابع

فهرست تصاویر

۲	۱-۱ اجرای برنامه‌ی hello.efi
۴	۱-۲ اجرای متوالی همه‌ی آزمون‌ها
۶	۱-۳ صدا زدن برنامه و اجرای تست Identity
۸	۱-۴ اضافه کردن sbat
۹	۲-۴ کپی کردن shim به ESP
۱۰	۳-۴ اضافه کردن boot option
۱۰	۴-۴ صفحه Mok Management
۱۱	۵-۴ وارد کردن امضای برنامه
۱۱	۶-۴ منوی انتخاب برنامه‌ها
۱۲	۷-۴ ساخت کلید
۱۲	۸-۴ امضای برنامه EFI

فصل ۱

نوشتن برنامه World Hello با EFI

در این قسمت برای شروع کار یک برنامه‌ی World Hello می‌نویسیم.

۱-۱ آماده کردن محیط برنامه نویسی EFI

برای این قسمت ما از edk2 استفاده کرده‌ایم که می‌توانید آن را در این [لینک](#) مشاهده کنید. این ابزار یک سری کتابخانه برای نوشتن و ساختن برنامه‌ها و درایورهای EFI است.

۲-۱ تحلیل کد

کد این قسمت در HelloWorld.c نوشته شده است. تابع UefiMain تابعی است که در شروع برنامه اجرا می‌شود. در خط اول توسط تابع Print رشته مورد نظر را چاپ می‌کنیم و دقت می‌کنیم که در UEFI باید از رشته‌های عریض (کاراکترهای ۲ بایتی) استفاده شود.

۳-۱ اجرای کد

برای اجرای کد با استفاده از QEMU و OVMF یک محیط EFI بالا می‌آوریم تا برنامه را در آن اجرا کنیم. این کار با دادن OVMF به عنوان آپشن bios به QEMU به سادگی قابل انجام است. این اجرا در ۱-۱ قابل مشاهده است.

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s):HD0a1::BLK1:
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/HD(1,MBR,0xBE1AFDFA,0x3F,0xFBFC1)
BLK0: Alias(s):
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK2: Alias(s):
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
Shell> fs:
'fs:' is not a valid mapping.
Shell> fs0:
FS0:\> hello.efi
Hello, world!
FS0:\> _
```

شکل ۱-۱: اجرای برنامه‌ی hello.efi

فصل ۲

آزمون حافظه

۱-۲ آشنایی با آزمون حافظه

آزمون حافظه به فرآیند تست کردن تایید کارکرد، درستی و کارایی حافظه سیستم گفته می‌شود. در این جا ما در فایل اصلی پروژه یعنی MemTest.c ۴ نوع تست آماده کرده‌ایم که هر کدام برای شرایطی خاص مناسب هستند.

۲-۲ آزمون‌ها

۱-۲-۲ Walking Ones

در این آزمون ما با یک الگو که شامل دقیقاً یک بیت ۱ است شروع می‌کنیم و هر بار مقدار آن را یک شیفت دوری می‌دهیم. سپس در نهایت همه مقادیر را چک می‌کنیم. این آزمون کمک می‌کند مشکلات data bus پیدا شود. دلیل ۱ بودن دقیقاً یک بیت این است که تفاوت یک بیت با دو بیت کناری شانس خطا را افزایش می‌دهد. این تست در تابع WalkingOnesTest پیاده‌سازی شده است.

۲-۲-۲ Identity

در این آزمون ما در هر خانه از حافظه آدرس خودش را می‌نویسیم. سپس در نهایت همه مقادیر را چک می‌کنیم. این آزمون بسیار ساده و کلی است و. در تابع IdentityTest پیاده‌سازی شده است.


```

UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s) :HD0a1:;BLK1:
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/HD(1,MBR,0xBE1AFDFA,0x3F,0xFBFC1)
BLK0: Alias(s) :
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK2: Alias(s) :
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell> memtest
Total number of pages 512109
WalkingOnesTest passed
IdentityTest passed
RowhammerTest passed
DMATest passed
Shell> _

```

شکل ۲-۱: اجرای متوالی همه‌ی آزمون‌ها

۳-۲-۲ Rowhammer

در این آزمون ما حساس بودن حافظه به آسیب پذیری Rowhammer را می‌سنجیم. این آسیب پذیری سعی می‌کند با دسترسی با فرکانس بالا به چند سطر خاص در حافظه باعث شود مقادیر سطری دیگر عوض شوند. در این جا ما با استناد به این پیاده‌سازی از Rowhammer دو طرفه، تابع RowHammerTest را پیاده‌سازی کرده‌ایم.

۴-۲-۲ DMA

در این آزمون ما درستی کارکرد زیرسامانه DMA برای انتقال داده بین دیسک و حافظه بدون دخالت پردازنده را می‌سنجیم. برای این کار در ابتدا یک فایل روی دیسک مورد نظر ساخته و با داده‌های خاص پر می‌شود. سپس این فایل در خانه‌هایی از حافظه خوانده شده و چک می‌شود که با چیزی که نوشته شده یکی باشد. این تست در تابع DMA پیاده‌سازی شده است.

۳-۲ اجرای آزمون حافظه

در ۲-۱ یک اجرا از تست‌های نوشته شده را نشان می‌دهیم.

فصل ۳

نکات پیاده‌سازی

۱-۳ یکپارچه‌سازی با Shell

در این قسمت ما با استفاده از کتابخانه ShellCEntryLib تابع ورودی کد را مانند یک تابع ورودی عادی در C کردیم. سپس با استفاده از دستور alias در شل آن را مانند دستورات عادی کردیم. این کار دائمی است و بین دو بوت متوالی از بین نمی‌رود. همچنین با صدا زدن کد مانند memtest testname می‌توان فقط یک تست خاص را اجرا کرد که در ۱-۳ قابل مشاهده است.

۲-۳ پیاده‌سازی چند هسته‌ای آزمون Identity

برای این کار ما از پروتکل MpService در UEFI استفاده کردیم. یک تابع کارگر ساختیم و سپس با استفاده از StartupAllAPs آن را روی هسته‌های مختلف فراخوانی کردیم.

۳-۳ پیاده‌سازی GUI

ما تلاش کردیم تا یک محیط گرافیکی ساده برای این برنامه ارائه دهیم اما به مشکل خوردیم. اصلی‌ترین مشکلی که داشتیم این بود که در حین آزمون نباید حافظه جدیدی اختصاص داده می‌شود، وگرنه ممکن بود آزمون ما با یک سامانه دیگر تداخل بخورد و آزمون و/یا روند اجرای یک تابع بیرونی به غلط انجام شوند. کتابخانه‌هایی که اجزای گرافیکی را پیاده‌سازی کرده بودند ممکن بود با اجرا شدن از سیستم

```

UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s):HD0a1::BLK1:
PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/HD(1,MBR,0xBE1AFDFA,0x3F,0xFBFC1)
BLK0: Alias(s):
PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK2: Alias(s):
PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
Shell> memtest identity
Total number of pages 512109
IdentityTest passed
Shell> _

```

شکل ۳-۱: صدا زدن برنامه و اجرای تست Identity

حافظه بگیرند و این باعث می‌شد که ما نتوانیم از آن‌ها وسط آزمون استفاده کنیم. به عبارتی دیگر برای پیاده‌سازی یک رابط گرافیکی باید کتابخانه‌های مربوطه را در سطح پایین تغییر می‌دادیم که از حافظه‌های از قبل اختصاص داده شده استفاده کنند که به علت کمبود زمان موفق به انجام این مورد نشدیم.

فصل ۴

بوت امن

بوت امن (Secure Boot) یکی از ویژگی‌های امنیتی سیستم‌های UEFI است که هدف آن جلوگیری از اجرای کدهای غیرمجاز در فرآیند بوت سیستم می‌باشد. در این فرآیند، تمام نرم‌افزارهای بوت باید دارای امضای دیجیتال معتبر باشند. ابزار shim یک واسطه است که به کاربران امکان می‌دهد گواهی‌های سفارشی یا کلیدهای اضافی را برای بوت امن اضافه کنند، به خصوص در سیستم‌های لینوکسی که ممکن است امضاهای استاندارد میکروسافت را نداشته باشند. این روش باعث حفظ امنیت در عین انعطاف‌پذیری می‌شود.

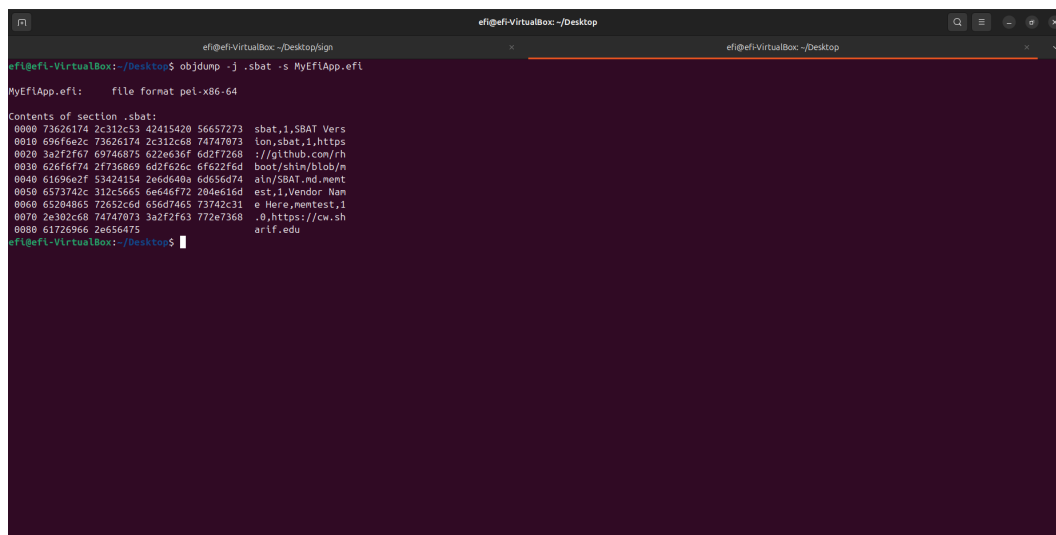
از آنجا که خروجی EFI ما امضای رسمی میکروسافت را ندارد، به shim برای اجرای آن در محیط بوت امن نیاز داریم.

۴-۱ شبیه‌سازی

برای شبیه‌سازی بوت امن از Virtualbox استفاده می‌کنیم. در این محیط می‌توانیم EFI و بوت امن را مشابه ماشین واقعی تست کنیم. یک سیستم عامل مجازی اوبونتو نصب می‌کنیم تا عملیات‌های مرتبط با راه‌اندازی برنامه را در آن انجام دهیم.

۲-۴ آماده‌سازی برنامه

باید به برنامه section به نام sbat اضافه کنیم. مهم است مقدار آن را shim بشناسد و به همین خاطر از بخش مشابهی در grub استفاده می‌کنیم.



```
efi@efi-VirtualBox: ~/Desktop
efi@efi-VirtualBox:~/Desktop$ objdump -j .sbat -s MyEfiApp.efi
MyEfiApp.efi:      file format pei-x86-64

Contents of section .sbat:
0000 73626174 2c312e53 42415420 56657273  sbat,1.SBAT Vers
0010 696f6e2c 73626174 2c312e68 74747073  lon,sbat,1.https
0020 3a2f2f67 69746875 622e636f 6d2f7268  ://github.com/rh
0030 626f6f74 2f736869 6d2f626c 6f622f6d  boot/shim/blob/n
0040 61696e2f 53424154 2e6d648a 6d656d74  aln/SBAT.md.ment
0050 6573742c 312c5665 6e646f72 204e616d  est,1.Vendor Man
0060 65204865 72652c6d 656d7465 73742c31  e Here,mentest,1
0070 2e302c68 74747073 3a2f2f63 772e7368  .0,https://cw.sh
0080 61726966 2e656475  artf.edu
```

شکل ۴-۱: اضافه کردن sbat

۳-۴ آماده‌سازی بوت

برنامه shim به طور پیشفرض در اوبونتو نصب می‌شود. لازم است فایل‌های shim و برنامه EFI را به پارتیشن ESP منتقل می‌کنیم. اسم برنامه را به grubx64.efi تغییر می‌دهیم. به صورت پیشفرض shim برنامه‌ای با این اسم را لود می‌کند.

۴-۴ بوت با اثرانگشت برنامه

یک راه برای بوت امن این است که اجازه دهیم shim بوت شود و سپس اثرانگشت برنامه را به MokList اضافه کنیم. بعد از انجام این کار shim برنامه را اجرا می‌کند و خطای 0x1a Security Violation چاپ نمی‌شود. مراحل اضافه کردن اثرانگشت hash در ادامه نشان داده شده است.

```

efi@efi-VirtualBox: /boot/efi/EFI/test
efi@efi-VirtualBox: ~/Desktop/hign
efi@efi-VirtualBox: /boot/efi/EFI/test

0 upgraded, 2 newly installed, 0 to remove and 163 not upgraded.
Need to get 160 kB of archives.
After this operation, 623 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ir.archive.ubuntu.com/ubuntu noble/main amd64 liburing2 amd64 2.5-1build1 [21.1 kB]
Get:2 http://ir.archive.ubuntu.com/ubuntu noble/universe amd64 plocate amd64 1.1.19-2ubuntu2 [139 kB]
Fetched 160 kB in 1s (230 kB/s)
Selecting previously unselected package liburing2:amd64.
(Reading database ... 152705 files and directories currently installed.)
Preparing to unpack .../liburing2_2.5-1build1_amd64.deb ...
Unpacking liburing2:amd64 (2.5-1build1) ...
Selecting previously unselected package plocate.
Preparing to unpack .../plocate_1.1.19-2ubuntu2_amd64.deb ...
Unpacking plocate (1.1.19-2ubuntu2) ...
Setting up liburing2:amd64 (2.5-1build1) ...
Setting up plocate (1.1.19-2ubuntu2) ...
update-alternatives: using /usr/bin/plocate to provide /usr/bin/locate (locate) in auto mode
info: Selecting GID from range 100 to 999 ...
info: Adding group 'plocate' (GID 124) ...
Initializing plocate database; this may take some time... done
Created symlink /etc/systemd/system/timers.target.wants/plocate-updatedb.timer → /usr/lib/systemd/system/plocate-updatedb.timer.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu6.3) ...
efi@efi-VirtualBox: /boot/efi/EFI/test$ cp /usr/lib/shim/
BOOTX64.CSV ls-not-revoked mok/ shlnx64.efl.shlnx64.efl.dualsigned shlnx64.efl.signed.latest
fbx64.efl mx64.efl shlnx64.efl.shlnx64.efl.signed shlnx64.efl.signed.previous
efi@efi-VirtualBox: /boot/efi/EFI/test$ cp /usr/lib/shim/(shlnx64.efl,mx64.efl) .
cp: cannot create regular file './shlnx64.efl': Permission denied
cp: cannot create regular file './mx64.efl': Permission denied
efi@efi-VirtualBox: /boot/efi/EFI/test$ sudo cp /usr/lib/shim/(shlnx64.efl,mx64.efl) .
efi@efi-VirtualBox: /boot/efi/EFI/test$ ll
total 1600
drwxr-xr-x 2 root root 4096 Jan 29 14:11 ./
drwxr-xr-x 5 root root 4096 Jan 27 09:10 ../
-rwxr-xr-x 1 root root 13696 Jan 29 14:09 grubx64.efi*
-rwxr-xr-x 1 root root 856288 Jan 29 14:11 mx64.efi*
-rwxr-xr-x 1 root root 857942 Jan 29 14:11 shlnx64.efi*
efi@efi-VirtualBox: /boot/efi/EFI/test$

```

شکل ۴-۲: کپی کردن shim به ESP

۴-۵ بوت با امضای برنامه

راه دیگر برای بوت امن این است که یک کلید شخصی بسازیم و برنامه EFI را با آن امضا کنیم. بعد از امضا کردن و اجرای دوباره shim، می‌توانیم فایل DER امضا را به MokList اضافه کنیم. مزیت این روش این است که اگر برنامه بعداً تغییر کند فقط لازم است یک بار دیگر آن را امضا کنیم و دیگر طی کردن مراحل enroll key لازم نیست.

۴-۶ منابع

https://wiki.archlinux.org/title/Unified_Extensible_Firmware_Interface/Secure_Boot

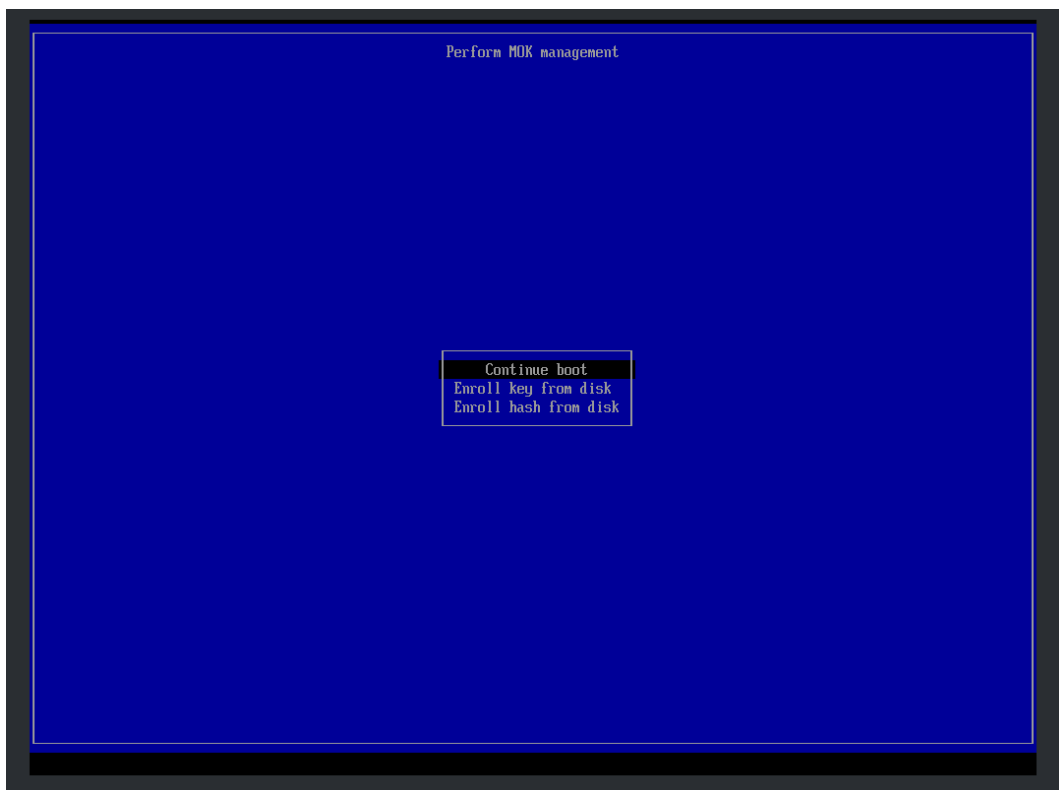
https://www.funtoo.org/UEFI_Secure_Boot_and_SHIM

<https://github.com/rhboot/shim/blob/main/SBAT.md>

<https://github.com/rhboot/shim/issues/376>

```
efi@efi-VirtualBox:~$ efibootmgr --unicode --disk /dev/sdX --part Y --create --label "Shin" --loader ^C
efi@efi-VirtualBox:~$ efibootmgr --unicode --disk /dev/sda --part 1 --create --label "Mentest" --loader /EFI/test/shinx64.efi
Could not prepare Boot variable: Permission denied
efi@efi-VirtualBox:~$ sudo efibootmgr --unicode --disk /dev/sda --part 1 --create --label "Mentest" --loader /EFI/test/shinx64.efi
[sudo] password for efi:
BootCurrent: 0003
Timeout: 0 seconds
BootOrder: 0007,0003,0002,0001,0000,0004,0005
Boot0000* UEFI FvVol(7c8b9d9-8ab-4f34-aaea-3ee4f6516a1)/FvFile(462caa21-7614-4503-836e-8ab5f4662331)
Boot0001* UEFI VBOX CD-ROM VBOX2-01700376 PciRoot(0x0)/Pci(0x1,0x1)/Ata(1,0,0)
Boot0002* UEFI VBOX HARDISK VBOX2-01700376 PciRoot(0x0)/Pci(0xd,0x0)/Sata(0,65535,0)
Boot0003* Ubuntu HD(1,GPT,9146273e-8178-44d7-a556-4fc4a8be95c7,0x800,0x219800)/File(\EFI\ubuntu\shinx64.efi)
Boot0004* UEFI PXEvd (MAC:08002736d7ca) PciRoot(0x0)/Pci(0x3,0x0)/MAC(08002736d7ca,1)/IPv4(0.0.0.0,0.0.0.0)
Boot0005* UEFI PXEv6 (MAC:08002736d7ca) PciRoot(0x0)/Pci(0x3,0x0)/MAC(08002736d7ca,1)/IPv6([::]:[::]:[::]:[::]:[::]:[::]:[::]:[::])
Boot0006* MyEfiApp.efi PciRoot(0x0)/Pci(0xd,0x0)/Sata(0,65535,0)/HD(1,GPT,9146273e-8178-44d7-a556-4fc4a8be95c7,0x800,0x219800)/File(\EFI\test\MyEfiApp.efi)
Boot0007* Mentest HD(1,GPT,9146273e-8178-44d7-a556-4fc4a8be95c7,0x800,0x219800)/File(\EFI\test\shinx64.efi)
efi@efi-VirtualBox:~$
```

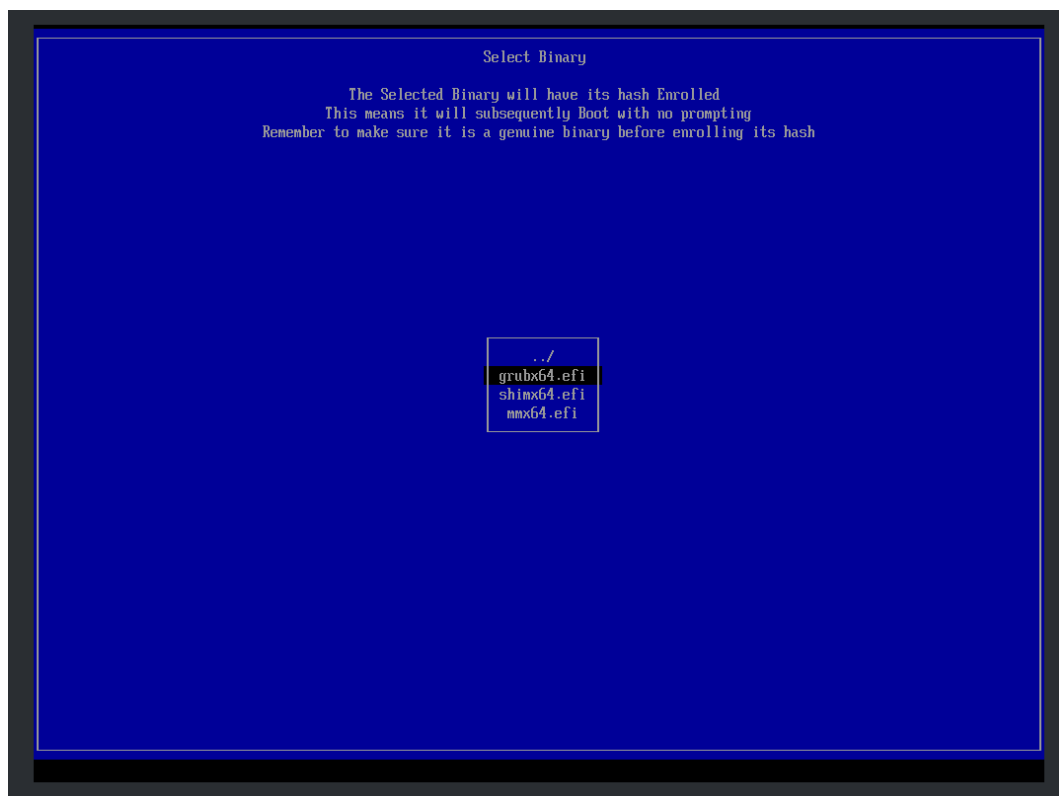
شکل ۴-۳: اضافه کردن boot option



شکل ۴-۴: صفحه Mok Management



شکل ۴-۵: وارد کردن امضای برنامه



شکل ۴-۶: منوی انتخاب برنامه‌ها

