# Project report

The follow report details the action plan for the development of MBRProtect (see requirements). The first part documents work done in determining the difficulty of each requirement. The second part is an action with estimated timeframe.

The report contains a code base for a linux kernel module built for generic 4.10 kernel on an x86 machine. We built the code to determine the difficulty and feasibility of the requirements. Run dmesg to see log output for the module. In the actual module, the log commands can be replaced with actual code.

## Kernel module files

* ktest.h  
  Module header used by ktest.c  
  Modify this file to chance defined constants. Specifically:
  + SHELL location of the revshell
  + CLEANUP location of the kill\_shell.sh
  + MBR\_DEVICE reference the device containing MBR
  + MBR\_BACKUP location where to store the MBR backup
* ktest.c  
  Module code.
* shells/revshell.c  
  User space code for a ping listener.  
  Modify this file to chance define constants. Specifically:
  + KEY filter for ICMP messages
  + KTEST\_LOG location of log file
* *scripts/make\_device.sh*  
  Shell commands to mount a device.   
  note: The script immediately adds a device before compiling. If you **rmmod** the kernel module, the device will still be present. If you want to completely remove the device after testing, you should do that manually. You can remove the device with:  
  **rm –f /dev/.ktest**
* Makefile  
  Basic Makefile

## How to use the module.

1. Compile by running **make**.
2. Install the mod with **insmod**.
3. You can send commands to the module by echoing a command to the device.  
   The module accepts the following commands:
   * hide Hides the module
   * show Shows the module (code is buggy, don’t use)
   * shellUp Activates the ping listener (stops after receiving on valid ping)
   * backup Writes a copy of the MBR to a specified file (see ktest.h)
   * overwrite Assembly implementation to write to MBR (not implemented)

## How to test the module

1. Install mod with **insmod ktest.ko**
2. Run **dmesg** to see the kernel info logs.
3. Run **echo hide > /dev/.ktest** to hide the module
4. Check **lsmod**, **ps aux**, etc. to check if the module is visible
5. Run **echo shellUp > /dev/.ktest** to bring up the listener.
6. Send a ICMP packet with the KEY (see revshell.c)  
   n**ping --icmp -c 1 -dest-ip 127.0.0.1 --data-string 'KTEST 127.0.0.1 3030'**
7. There should be a log file at KTEST\_LOG (see revshell.c)
8. Run **echo backup > /dev/.ktest** to backup the MBR
9. Check the MBR\_BACKUP to see if we’ve made a backup of the MBR  
   You can use **sudo dd if=/dev/sda of=mbr\_backup bs=512 count=1** to get a copy of the MBR
10. Run **echo overwrite > /dev/.ktest** to overwrite the MBR.
11. Overwrite is not implemented. See code for possible assembly implementation.
12. Notes:  
    You can always run **dmesg** to see the kernel info logs.

If the module is hidden, you can’t use **rmmod** to remove it.

Ignore the error messages when echoing to /dev/.ktest

## Functional requirements

1. MBRProtector is a kernel module / device driver (see code). Developed on/for Ubuntu 16.04. Tested on centOS 7 (3.10 kernel), most functionality did not work. The code can be customized for different kernels/versions by adding conditional statements.
2. MBRProtector is a kernel module (see code) and has root access.
3. Hiding features are achieved by removing the module from the module list (see code). More creative methods should be explored. TODO: solve problem of tainting kernel by “module verification failed”
4. IRP is a windows feature. As far as we can tell, we should be able to detect any attempt to read/write to the MBR by using our open hook from the sys\_call.
5. We intend to monitor integrity by either adding a task to the cron table (probably a bad idead, will most likely be detected) or using one of the hooks (more probable).
6. MBRProtector can call other programs and expose methods. Injecting a library should be no problem. TODO: “figure out how to circumvent module verification (see 3)” to trick the antivirus program.
7. Hiding can be done by removing the symbolic link to the file or saving the binary to the *.ktest* device. We can then read the file via the device driver.
8. Interface to be determined in cooperation with customer
9. Needs more research. We have yet to find a way of running the kernel in SELinux
10. Needs more research. We have yet to find a good way of not being detected by AV.

## Conclusion

The project is feasible and should be doable in 3-4 months. This was a very fun project, despite us clearly not being experts in OS. We think if we’re to proceeded with the project, we suggest you take into consideration the hurdles that we may encounter in the future and possible timeline extension. A lot of what we’ve done is built on work by others and we may get stumped at the harder problems (specifically points 9 and 10). Nonetheless, we’ve present a plan of action with rough time estimates. The provided code and action plan should be beneficial to completion of the project.

## Action plan

Actions are in implementation order. The number in the first column refers to the requirement number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Objective | Task | Output | Duration | Resources |
| 2 | MBRProtect is a kernel module | Create a kernel module | code | done | - |
| 3 | MBRProct should be hidden from user and applications | Determine how visible the module is and try to hide it from everyone/everything | code | partially done  1 week(?) |  |
| 4 | MBRProtect should stop applications from modifying the MBR | Use hooks to intercept read commands.  Determine how to deal with MBR location | code | partially done  1 weeks(?) |  |
| 5 | Undo modification / verify MBR | Test different methods of changing MBR and how to counteract. Look into cron table | code | partially done  2 weeks(?) |  |
| 5 | MBRProtect should write customized content to the MBR | Test writing to MBR | code | partially done  < 1 week |  |
| 7 | Hide MgrCode | Test how to hide code (either in device or hidden address) | code | partially done  < 1 week |  |
| 6 | Inject MgrCode into MBRMgr | Move the ping listener to the module. Determine how to hide/protect running process. Determine how to install necessary libraries. | code  documentation | partially done  1 month(?) | MBRMgr app specifications |
| 8 | MBRMgr and MBRProtect should interact | Design interface with customer | code  documentation | partially done  1 week(?) | MBRMgr app specifications |
| 3 10 | MBRProtect should not be visible to antivirus. | Determine how to best to hide MBRProtect from antivirus. | code | partially done  1 month(?) | Antivirus programs |
| 3  5 | MBR protect should load on startup/boot. | Write assembly code that modifies the boot record to allow for mod install on startup/boot. | code | 2 weeks(?) |  |
| 9 | MBRProtect should work with SELinux | Determine how best to run MBRProtect with SELinux | code documentation | 2 weeks(?) |  |
| 1 | MBRProtect works on multiple OS | Testing on multiple OS and implementing conditionals | MBR code | partially done  2 weeks | - |