Extraction of Platform-unique Information as an Identifier

Kangwon Lee, Kyungroul Lee, Jaecheon Byun, Sunghoon Lee
Soonchunhyang University
Shinchang, Asan, Republic of Korea
{paul, carpedm, apple}@sch.ac.kr, joyce2@daum.net

Hyobeom Ahn
Kongju National University
Budae, Choenan, Republic of Korea
hbahn@kongju.ac.kr

Kangbin Yim*
Soonchunhyang University
Shinchang, Asan, Republic of Korea
yim@sch.ac.kr

Abstract

The Ethernet MAC address is known it is not changed and so highly considered as one of the platform-unique information. Because of the reason, the MAC address has been used as a platform identifier for several public services. This paper surveys, implements and analyzes the extraction methods for the MAC address in different levels on the PC platform. The methods considered include reading the registry database, calling the dedicated API functions, polling the I/O controller and communicating with external EEPROM. The result of the implementation will be helpful as a reference for developers who need to have a simple platform identifier.

Keywords: platform-unique information, platform identifier, designated platform, MAC address, hardware spoof.

1 Introduction

Most difficult thing in security area is to assure the entity's identity. In many cases, string based pass phrases are used for this purpose. However, this kind of pass phrases are easily sniffed by a sophisticatedly architected key logging malware. Because of the reason, the highly security-sensitive services such as online finance and government issues are trying to restrict their operational environment in different ways. As an emphatic example of such approaches, Korean government has introduced the designated platform policy for online banking services where users are requested to use several limited number of PCs for renewing their public certificates. The problem in this approach is how to prove that the PC currently in use is one of the designated set of PCs. For this policy to be successful, it is essential to achieve the uniqueness of a PC platform to register it as a designated one.

The uniqueness of a hardware platform can be achieved by deriving a platform-unique information from one or a combination of several hardware-dependent unique values. The Ethernet MAC address (MAC address hereafter) is considered the best one of such reasonable candidates as network IP address, serial numbers of hard disks, identifiers and mapping addresses of periphery devices and etc. because many people believe it is an unmodifiable and globally unique hardwired value. Although a MAC address needs to be unique only in a network segment, manufactures produce the Ethernet card with a pseudo globally-unique MAC address to eliminate the address conflicts when multiple cards are randomly deployed.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 3, number: 4, pp. 85-99

*Corresponding Author: LISA Laboratory, Soonchunhyang University, 9418 Engineering Building, 646 Eupnae, Shinchang, Asan, 336-745 Republic of Korea, Tel: +82-(0)41-530-1741, Web: `http://lisa.sch.ac.kr`

The MAC address also can be targeted by adversaries in the form of MAC spoofing attack, which is also known as ARP spoofing or ARP poisoning[1][2][3]. The ARP spoofing redirects network traffic and allows adversaries to sniff it as a result of the attack though many countermeasures against the ARP spoofing already effectively applied. Moreover, the problem caused by the ARP spoofing absolutely has no relation to the problem proving the platform-unique information. Therefore, we need to analyze and rationalize the designated platform policy and its implementations to count potential vulnerabilities we may face in the practices, especially related to the MAC address.

In case of the designated platform solution[4], it utilizes the MAC address as a factor for constructing the platform-unique information. Therefore, this solution is somewhat for a kind of multi-factor authentication because the platform-unique information is used as an additional factor in proving both of the user identifier and the platform identifier. This approach can improve the security level of the services such as the online games, file repositories, financial transactions, etc. by restricting the locations (authenticated platforms) of the authenticated users. When a specific service is used, it registers the platform identifier to the management server. When the user tries to use this service back, the trials issued only on the platforms of which identifiers have registered are allowed and other requests are denied.

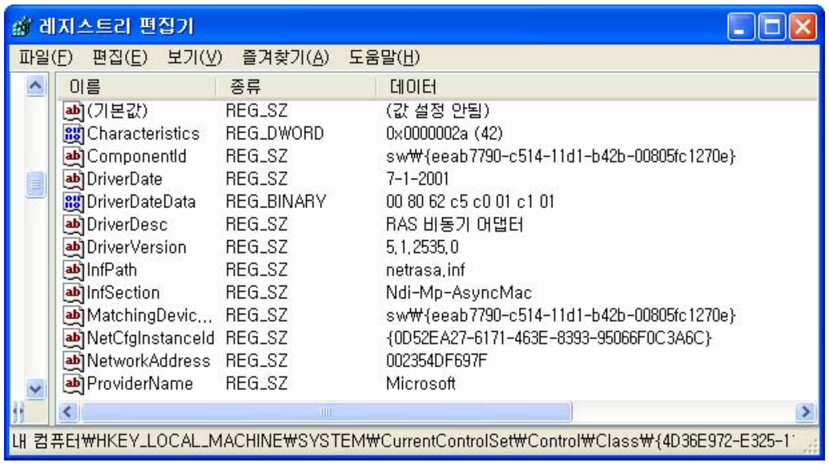
Practically, the MAC address is considered it should not be changed in an active service. Regarding the wireless gateway, it allows only the registered specific MAC addresses for network connections if configured especially in the case a mobile or vehicle machine makes an inquiry into a location in the dedicated network[5][6]. Finally, for the public certificate, a research on commercial designated platform services shows that they add to the authentication server a limited number of MAC addresses of the platforms for an authenticated user in a registration process[7]. However, the simple extraction of the MAC address is insufficient even though various services utilize the MAC addresses. Therefore, in this paper, we investigate possible extraction methods for the MAC address and analyze safety of these methods.

2 Extraction Methods for Platform-unique Information (MAC address)

Extraction of the MAC address can be done in several ways: reading the registry database, calling one of the dedicated APIs, polling registers in the I/O controller of the network interface card (NIC) and communicating with EEPROM in the NIC. This chapter shows how to implement them. The operational environment for each implementation includes Microsoft Windows XP service pack 3, Intel I/O controller hub 7 (ICH7) family and Realtek RTL8169/8110 family. Each code was programmed on Microsoft visual studio 2005 with Winddk 3790.1830.

2.1 Reading the Registry Database

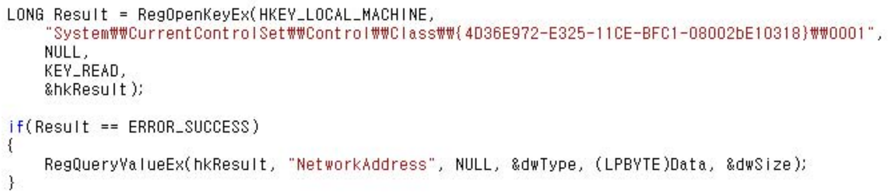
The entry `CurrentControlSet` in the system registry database includes the result of the last successful hardware enumeration. This means that the I/O information for every PnP compliant hardware resources resides as one of its key. The MAC address is also located at the `NetworkAddress` key in `My computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E972-E325-11CEBFC1-08002bE10318}\0001`. Figure 1 shows the registry state after the Ethernet card was installed and its MAC address was added to the registry key. Thus, the MAC address is able to be extracted by reading the above registry key. Opening the registry key returns a handle and reading the `NetworkAddress` field using the obtained handle returns the MAC address. Figure 2 shows a part of the implemented code and figure 3 shows the result[8][9].



이름	종류	데이터
(기본값)	REG_SZ	(값 설정 안됨)
Characteristics	REG_DWORD	0x0000002a (42)
ComponentId	REG_SZ	sw₩{eeab7790-c514-11d1-b42b-00805fc1270e}
DriverDate	REG_SZ	7-1-2001
DriverDateData	REG_BINARY	00 80 62 c5 c0 01 c1 01
DriverDesc	REG_SZ	RAS 비동기 어댑터
DriverVersion	REG_SZ	5.1.2535.0
InfPath	REG_SZ	netrasa.inf
InfSection	REG_SZ	Ndi-Mp-AsyncMac
MatchingDevic...	REG_SZ	sw₩{eeab7790-c514-11d1-b42b-00805fc1270e}
NetCfgInstanceId	REG_SZ	{0D52EA27-6171-463E-8393-95066F0C3A6C}
NetworkAddress	REG_SZ	002354DF697F
ProviderName	REG_SZ	Microsoft

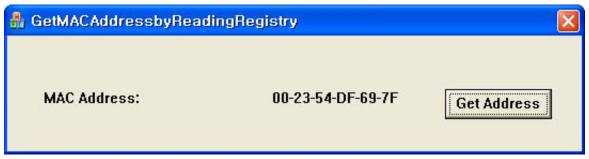
내 컴퓨터₩HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Control₩Class₩{4D36E972-E325-1

Figure 1: Registry state when MAC address is added



```
LONG Result = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
    "System₩₩CurrentControlSet₩₩Control₩₩Class₩₩{4D36E972-E325-11CE-BFC1-08002bE10318}₩₩0001",
    NULL,
    KEY_READ,
    &hkResult);

if(Result == ERROR_SUCCESS)
{
    RegQueryValueEx(hkResult, "NetworkAddress", NULL, &dwType, (LPBYTE)Data, &dwSize);
}
```

Figure 2: A part of implemented code by using the registry



MAC Address: 00-23-54-DF-69-7F

Figure 3: Implement result by using the registry

2.2 Calling the Dedicated APIs

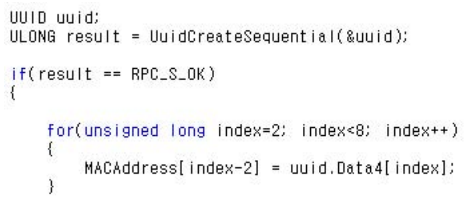
Windows provides several API functions that enable to achieve the MAC address: `UuidCreate`, `NetWkstaTransportEnum`, `GetAdaptersInfo`, `GetIfTable` and `Netbios`.

2.2.1 UuidCreateSequential()

UUID (Universally Unique IDentifier) supports the unique designation of an object such as interfaces, entry-point vectors, and client objects. The MAC address is stored in `data4` which is the 4th member of UUID structure. Two different versions `UUIDCreate()` and `UUIDCreateSequential()` provide the same function. Figure 4 is shown for a part of implemented code and 5 shows the result of the implementation [10][11][12].

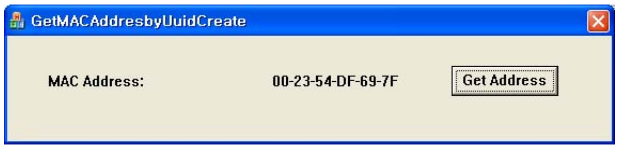
2.2.2 NetWkstaTransportEnum()

`NetWkstaTransportEnum()` provides information about the transport protocols. The MAC address is stored in one of the parameters named `bufptr`. after calling this function. Figure 6 shows a part of



```
UUID uuid;
ULONG result = UuidCreateSequential(&uuid);

if(result == RPC_S_OK)
{

    for(unsigned long index=2; index<8; index++)
    {
        MACAddress[index-2] = uuid.Data4[index];
    }
```

Figure 4: A part of implemented code using UUIDCreate

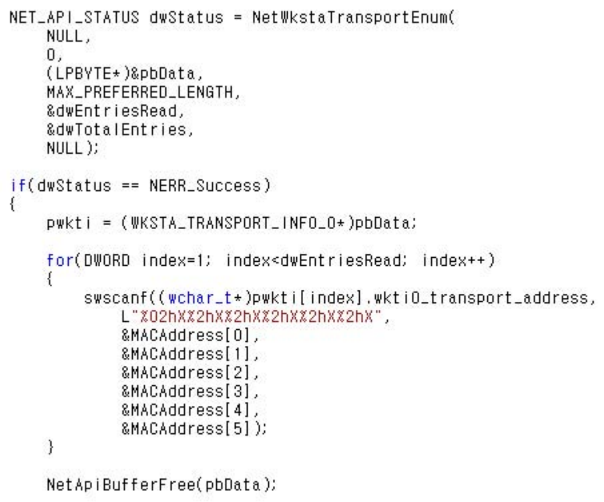


GetMACAddresbyUuidCreate

MAC Address: 00-23-54-DF-69-7F

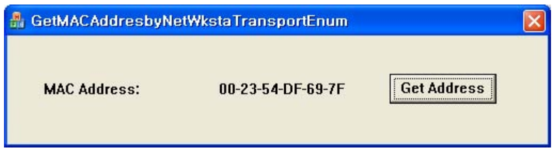
Figure 5: Result of the implementation using UUIDCreate

implemented code and figure 7 shows the result of the implementation[13].



```
NET_API_STATUS dwStatus = NetWkstaTransportEnum(
    NULL,
    0,
    (LPBYTE*)&pbData,
    MAX_PREFERRED_LENGTH,
    &dwEntriesRead,
    &dwTotalEntries,
    NULL);

if(dwStatus == NERR_Success)
{
    pwkti = (WKSTA_TRANSPORT_INFO_0*)pbData;

    for(DWORD index=1; index<dwEntriesRead; index++)
    {
        swscanf((wchar_t*)pwkti[index].wkti0_transport_address,
            L"%02hX%2hX%2hX%2hX%2hX%2hX",
            &MACAddress[0],
            &MACAddress[1],
            &MACAddress[2],
            &MACAddress[3],
            &MACAddress[4],
            &MACAddress[5]);
    }

    NetApiBufferFree(pbData);
```

Figure 6: A part of implemented codes using NetWkstaTransportEnum



GetMACAddresbyNetWkstaTransportEnum

MAC Address: 00-23-54-DF-69-7F

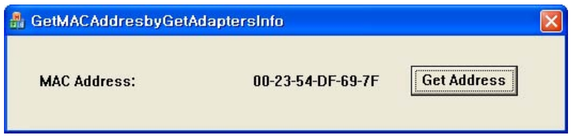
Figure 7: Result of the implementation using NetWkstaTransportEnum

2.2.3 GetAdaptersInfo()

`GetAdaptersInfo()` extracts the adapter information of a local computer. Address field in `pAdapterInfo` structure is one of the parameters and the MAC address is stored in this parameter after calling this function. Figure 8 shows a part of implemented code and figure 9 shows the implemented result[14][15][16].

```
IP_ADAPTER_INFO AdapterInfo;
DWORD dwBufLen = sizeof(AdapterInfo);
DWORD dwStatus = GetAdaptersInfo(&AdapterInfo, &dwBufLen);
```

Figure 8: A part of implemented code using GetAdaptersInfo



MAC Address: 00-23-54-DF-69-7F

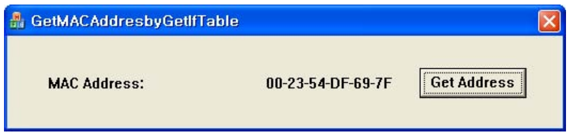
Figure 9: Result of the implementation using GetAdaptersInfo

2.2.4 GetIfTable()

`GetIfTable()` returns MIB-II interface table. The `bPhysAddr` field in the `pIfTable` structure is one of the parameters and this parameter includes the MAC address. Figure 10 shows a part of implemented code and figure 11 shows the result of the implementation[17][18][19].

```
DWORD sizeReq = 0;
PMIB_IFTABLE pMIB = NULL;

::GetIfTable(NULL, &sizeReq, FALSE) ;

pMIB = (PMIB_IFTABLE) new BYTE [sizeReq] ;
memset (pMIB, 0, sizeReq) ;

DWORD sizeToUse = sizeReq ;

DWORD dwStatus =  ::GetIfTable((PMIB_IFTABLE)pMIB,&sizeToUse, FALSE);
```

Figure 10: A part of implemented codes using GetIfTable function



MAC Address: 00-23-54-DF-69-7F

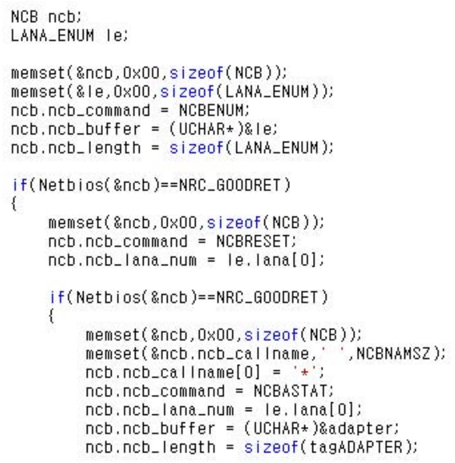
Figure 11: Result of the implementation using GetIfTable function

2.2.5 Netbios()

`Netbios()` interprets and executes the specified network control block(NCB). The `adapter_address` field in the `ncb_buffer` of the `pcnb` structure is one of the parameters and the MAC address is found in this parameter. Figure 12 shows a part of implemented code and figure 13 shows the result[20][21][22].

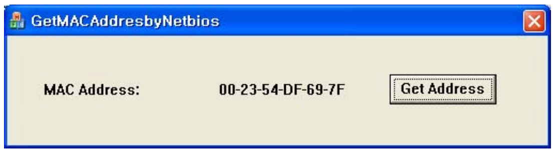
2.3 Polling the I/O registers

Extracting the MAC address through the registers of the I/O controller is dependent upon the specific hardware device. The input and output interfaces of a specific hardware device are allowed to be accessed through the PCI bus and thus `HalGetBusData` function can be used to extract the configuration information of a specific I/O device. NIC is assigned to bus 2, device 5, and function 0 on the test system and polling this address gives the NIC configuration information. Figure 14 shows the NIC information and Figure 15 shows the extracted NIC information[23][24].



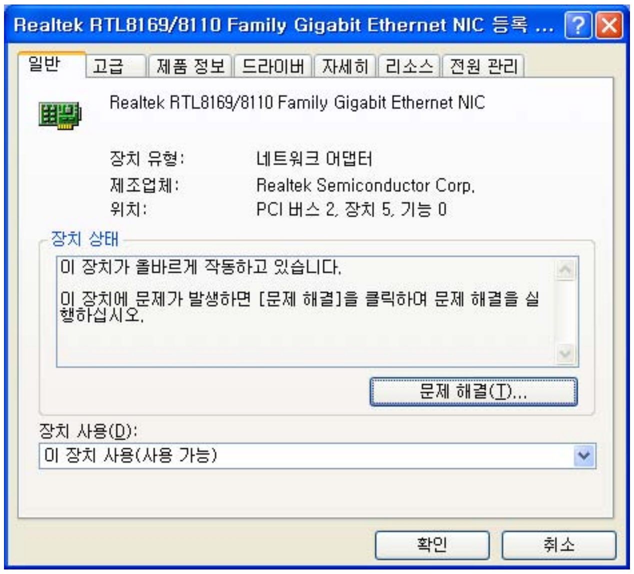
```
NCB ncb;
LANA_ENUM le;

memset(&ncb,0x00,sizeof(NCB));
memset(&le,0x00,sizeof(LANA_ENUM));
ncb.ncb_command = NCBENUM;
ncb.ncb_buffer = (UCHAR*)&le;
ncb.ncb_length = sizeof(LANA_ENUM);

if(Netbios(&ncb)==NRC_GOODRET)
{
    memset(&ncb,0x00,sizeof(NCB));
    ncb.ncb_command = NCBRESET;
    ncb.ncb_lana_num = le.lana[0];

    if(Netbios(&ncb)==NRC_GOODRET)
    {
        memset(&ncb,0x00,sizeof(NCB));
        memset(&ncb.ncb_callname,' ',NCBNAMSZ);
        ncb.ncb_callname[0] = '*';
        ncb.ncb_command = NCBASTAT;
        ncb.ncb_lana_num = le.lana[0];
        ncb.ncb_buffer = (UCHAR*)&adapter;
        ncb.ncb_length = sizeof(tagADAPTER);
```

Figure 12: A part of implemented code using Netbios



MAC Address: 00-23-54-DF-69-7F

Figure 13: Result of the implementation using Netbios



Realtek RTL8169/8110 Family Gigabit Ethernet NIC

장치 유형: 네트워크 어댑터
제조업체: Realtek Semiconductor Corp.
위치: PCI 버스 2, 장치 5, 기능 0

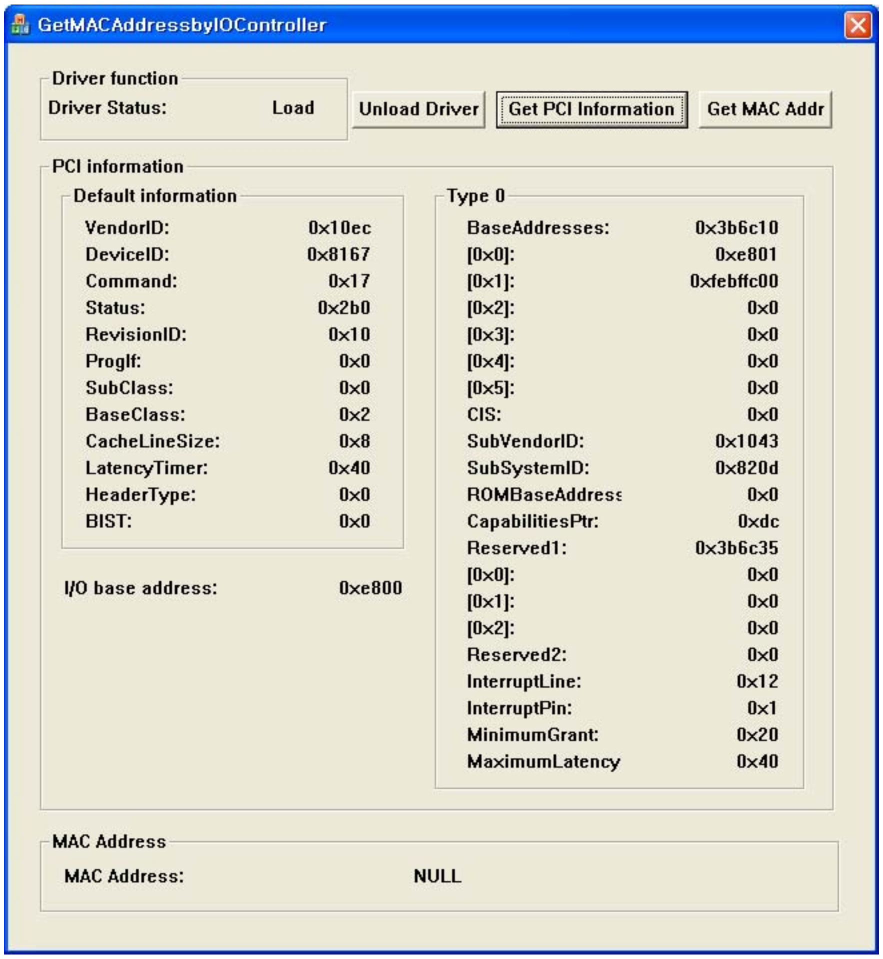
장치 상태

이 장치가 올바르게 작동하고 있습니다.

이 장치에 문제가 발생하면 [문제 해결]을 클릭하여 문제 해결을 실행하십시오.

장치 사용(D):
이 장치 사용(사용 가능)

Figure 14: NIC information



Driver function

Driver Status: Load

PCI information

Default information

VendorID:	0x10ec
DeviceID:	0x8167
Command:	0x17
Status:	0x2b0
RevisionID:	0x10
ProgIf:	0x0
SubClass:	0x0
BaseClass:	0x2
CacheLineSize:	0x8
LatencyTimer:	0x40
HeaderType:	0x0
BIST:	0x0

I/O base address: 0xe800

Type 0

BaseAddresses:	0x3b6c10
[0x0]:	0xe801
[0x1]:	0xfebffc00
[0x2]:	0x0
[0x3]:	0x0
[0x4]:	0x0
[0x5]:	0x0
CIS:	0x0
SubVendorID:	0x1043
SubSystemID:	0x820d
ROMBaseAddress	0x0
CapabilitiesPtr:	0xdc
Reserved1:	0x3b6c35
[0x0]:	0x0
[0x1]:	0x0
[0x2]:	0x0
Reserved2:	0x0
InterruptLine:	0x12
InterruptPin:	0x1
MinimumGrant:	0x20
MaximumLatency	0x40

MAC Address

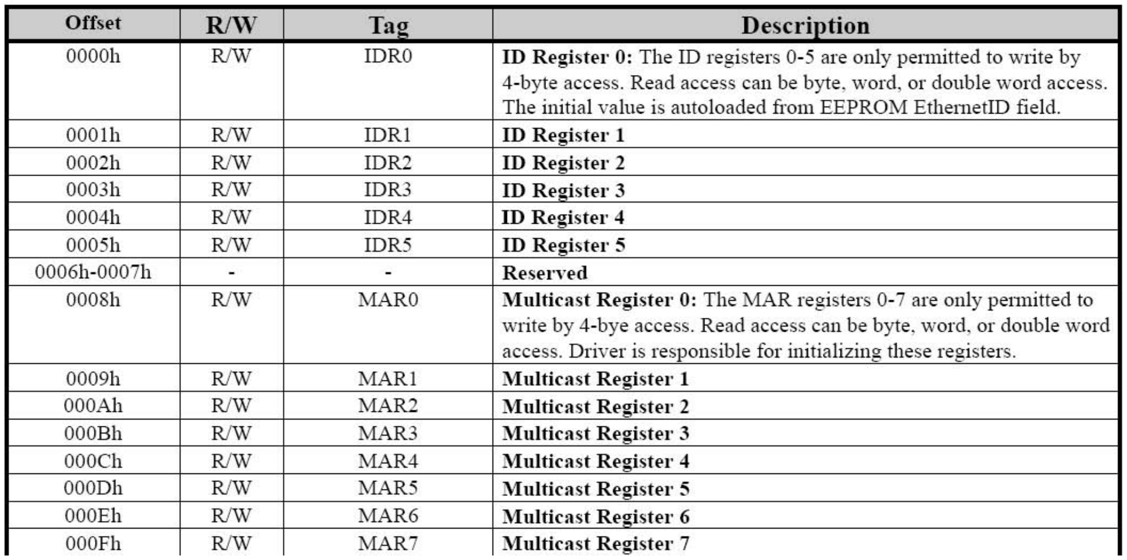
MAC Address: NULL

Figure 15: Extracted NIC information

As shown in figure 15, various PCI data such as vendorID, deviceID, and etc. can be extracted. To access the NIC register, we have to obtain the base of the I/O address or memory address which is mapped into the NIC registers. The BaseAddresses field is one of the extracted configuration information and extracted value is 0xE801, and 0xFEBFFC00. This information shows wether it is an I/O address or a memory address. If the least significant bit is 0, it means that the address is an I/O address and if not, this is a memory address. After extracting the address, you can access the NIC registers according to the NIC specification as shown in figure 16 in part. As shown in figure 16, a group of the six registers from IDR0 to IDR5 is called ID register and this stores the MAC address. Therefore, the MAC address is obtained by reading these six registers based on the extracted base address. Figure 17 shows a part of the implemented code and figure 18 shows the implemented result.

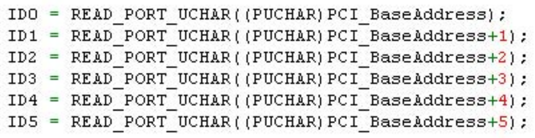
2.4 Communicating with EEPROM

Even though the MAC address is collected by accessing the NIC registers directly, this MAC address is just a copied content of the incorporated EEPROM in the NIC. These registers support accessing the MAC address easily by loading the original MAC address from the EEPROM to these registers. Hence, we can obtain the original MAC address by accessing the EEPROM contents directly. This is the lowest level of the extracting methods for the MAC address. The NIC utilizes a flash memory and an



Offset	R/W	Tag	Description
0000h	R/W	IDR0	**ID Register 0:** The ID registers 0-5 are only permitted to write by 4-byte access. Read access can be byte, word, or double word access. The initial value is autoloaded from EEPROM EthernetID field.
0001h	R/W	IDR1	**ID Register 1**
0002h	R/W	IDR2	**ID Register 2**
0003h	R/W	IDR3	**ID Register 3**
0004h	R/W	IDR4	**ID Register 4**
0005h	R/W	IDR5	**ID Register 5**
0006h-0007h	-	-	**Reserved**
0008h	R/W	MAR0	**Multicast Register 0:** The MAR registers 0-7 are only permitted to write by 4-bye access. Read access can be byte, word, or double word access. Driver is responsible for initializing these registers.
0009h	R/W	MAR1	**Multicast Register 1**
000Ah	R/W	MAR2	**Multicast Register 2**
000Bh	R/W	MAR3	**Multicast Register 3**
000Ch	R/W	MAR4	**Multicast Register 4**
000Dh	R/W	MAR5	**Multicast Register 5**
000Eh	R/W	MAR6	**Multicast Register 6**
000Fh	R/W	MAR7	**Multicast Register 7**

Figure 16: A part of the NIC specification[25]



```
ID0 = READ_PORT_UCHAR((PUCHAR)PCI_BaseAddress);
ID1 = READ_PORT_UCHAR((PUCHAR)PCI_BaseAddress+1);
ID2 = READ_PORT_UCHAR((PUCHAR)PCI_BaseAddress+2);
ID3 = READ_PORT_UCHAR((PUCHAR)PCI_BaseAddress+3);
ID4 = READ_PORT_UCHAR((PUCHAR)PCI_BaseAddress+4);
ID5 = READ_PORT_UCHAR((PUCHAR)PCI_BaseAddress+5);
```

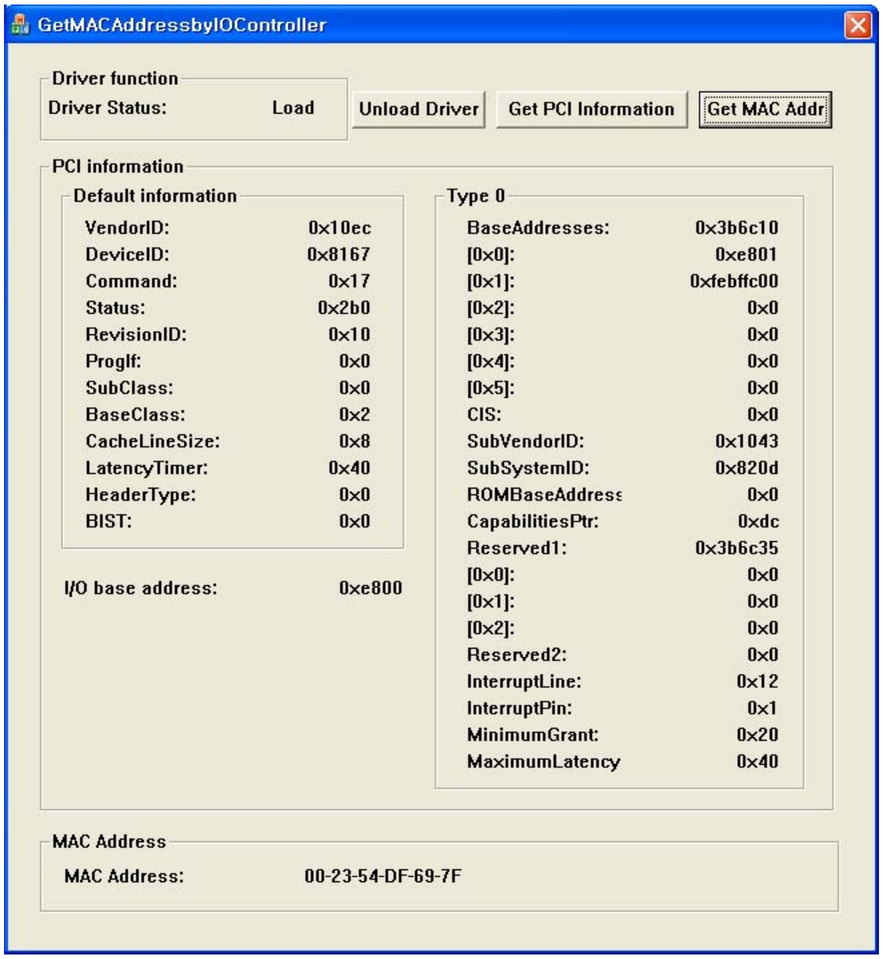
Figure 17: A part of implemented codes polling I/O registers

EEPROM for storing its firmware and the configuration information. In case of Realtek NIC in this test environment, 93C46 or 93C56 serial EEPROM is externally attached to the NIC and the NIC provides registers to directly access this serial EEPROM. Figure 19 shows the address of the command register for the EEPROM and and figure 20 shows the format of the command register.

As shown in these figures, six bits in the command register are meaningful. Bits 7:6 of the register indicate the operation mode and each of bits 3:0 respectively indicates EECS, EESK, EEDI, and EEDO meaning chip select, clock, data in and data out, which are directly connected to the dedicated pins to control the EEPROM. Realtek NIC supports four different operational modes such as normal mode, auto-load mode, programming mode, and configuration register write enable mode. The programming mode setting enables to read or write the EEPROM by controlling the EECS, EESK, EEDI, and EEDO. The actual access to a specific address of the 93C46 EEPROM is performed by sequentially controlling these pins and transmitting the address bit by bit according to the timing diagram as shown in figure 21. The MAC address can be obtained from the EEPROM by accessing the offset address 0eh-13h, the Ethernet ID field as shown in figure 22. Figure 23 and figure 24 show a part of implemented code and the result of the implementation, respectively.

3 Considerations on Safety of the Extraction Methods

Although the MAC address can be collected through the various extraction methods explained above, safety of the obtained information should be considered. Because each of the methods has its own different access level, they have a different safety level and it may affect the security level of the service that the information supports. The detailed levels of the methods are shown in figure 25.


GetMACAddressbyIOController

Driver function
Driver Status: Load

PCI information

Default information

VendorID:	0x10ec
DeviceID:	0x8167
Command:	0x17
Status:	0x2b0
RevisionID:	0x10
ProgIf:	0x0
SubClass:	0x0
BaseClass:	0x2
CacheLineSize:	0x8
LatencyTimer:	0x40
HeaderType:	0x0
BIST:	0x0

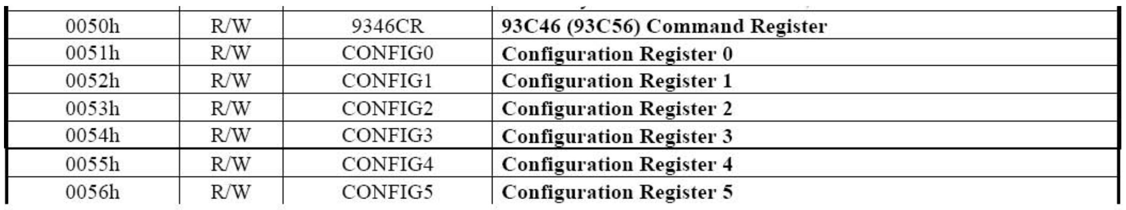
I/O base address: 0xe800

Type 0

BaseAddresses:	0x3b6c10
[0x0]:	0xe801
[0x1]:	0xfebffc00
[0x2]:	0x0
[0x3]:	0x0
[0x4]:	0x0
[0x5]:	0x0
CIS:	0x0
SubVendorID:	0x1043
SubSystemID:	0x820d
ROMBaseAddress	0x0
CapabilitiesPtr:	0xdc
Reserved1:	0x3b6c35
[0x0]:	0x0
[0x1]:	0x0
[0x2]:	0x0
Reserved2:	0x0
InterruptLine:	0x12
InterruptPin:	0x1
MinimumGrant:	0x20
MaximumLatency	0x40

MAC Address
MAC Address: 00-23-54-DF-69-7F

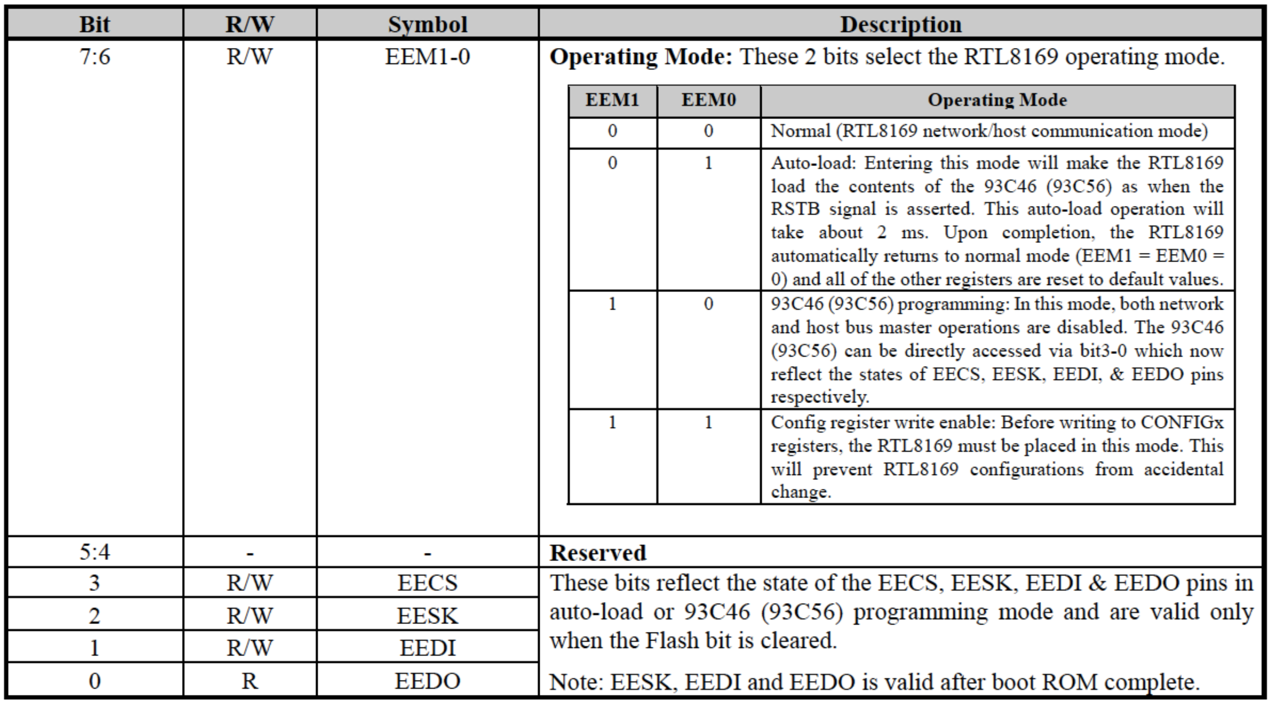
Figure 18: Result of the implementation polling I/O registers



0050h	R/W	9346CR	**93C46 (93C56) Command Register**
0051h	R/W	CONFIG0	**Configuration Register 0**
0052h	R/W	CONFIG1	**Configuration Register 1**
0053h	R/W	CONFIG2	**Configuration Register 2**
0054h	R/W	CONFIG3	**Configuration Register 3**
0055h	R/W	CONFIG4	**Configuration Register 4**
0056h	R/W	CONFIG5	**Configuration Register 5**

Figure 19: Base address of 93C46 EEPROM[25]

The method reading the registry database does not assure safety because the registry value is easily compromised. Similarly, the method calling the API functions also does not assure safety because these functions execute in the O/S layer and they are also fragile. It means that these methods can be hooked by adversaries. In case of polling I/O controller and communicating with the EEPROM, an attacker and a defender will compete for the information because they try to access I/O register or memory address at the same time. In case of the Intel processor, it can prepare an exception handler for debugging to be executed when a process accesses a specific memory address thus the attacker can utilize this facility to change the MAC address. Nevertheless, the defender can detect whether debug trap handler is set or not. On the other hand, the attack can also enable and disable the debug trap handler. This situation



Bit	R/W	Symbol	Description
7:6	R/W	EEM1-0	**Operating Mode:** These 2 bits select the RTL8169 operating mode.
5:4	-	-	**Reserved**
3	R/W	EECS	These bits reflect the state of the EECS, EESK, EEDI & EEDO pins in auto-load or 93C46 (93C56) programming mode and are valid only when the Flash bit is cleared. Note: EESK, EEDI and EEDO is valid after boot ROM complete.
2	R/W	EESK	
1	R/W	EEDI	
0	R	EEDO	

EEM1	EEM0	Operating Mode
0	0	Normal (RTL8169 network/host communication mode)
0	1	Auto-load: Entering this mode will make the RTL8169 load the contents of the 93C46 (93C56) as when the RSTB signal is asserted. This auto-load operation will take about 2 ms. Upon completion, the RTL8169 automatically returns to normal mode (EEM1 = EEM0 = 0) and all of the other registers are reset to default values.
1	0	93C46 (93C56) programming: In this mode, both network and host bus master operations are disabled. The 93C46 (93C56) can be directly accessed via bit3-0 which now reflect the states of EECS, EESK, EEDI, & EEDO pins respectively.
1	1	Config register write enable: Before writing to CONFIGx registers, the RTL8169 must be placed in this mode. This will prevent RTL8169 configurations from accidental change.

Figure 20: Command format for 93C46 EEPROM[25]

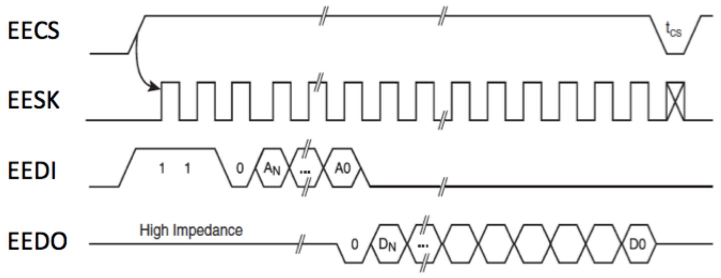
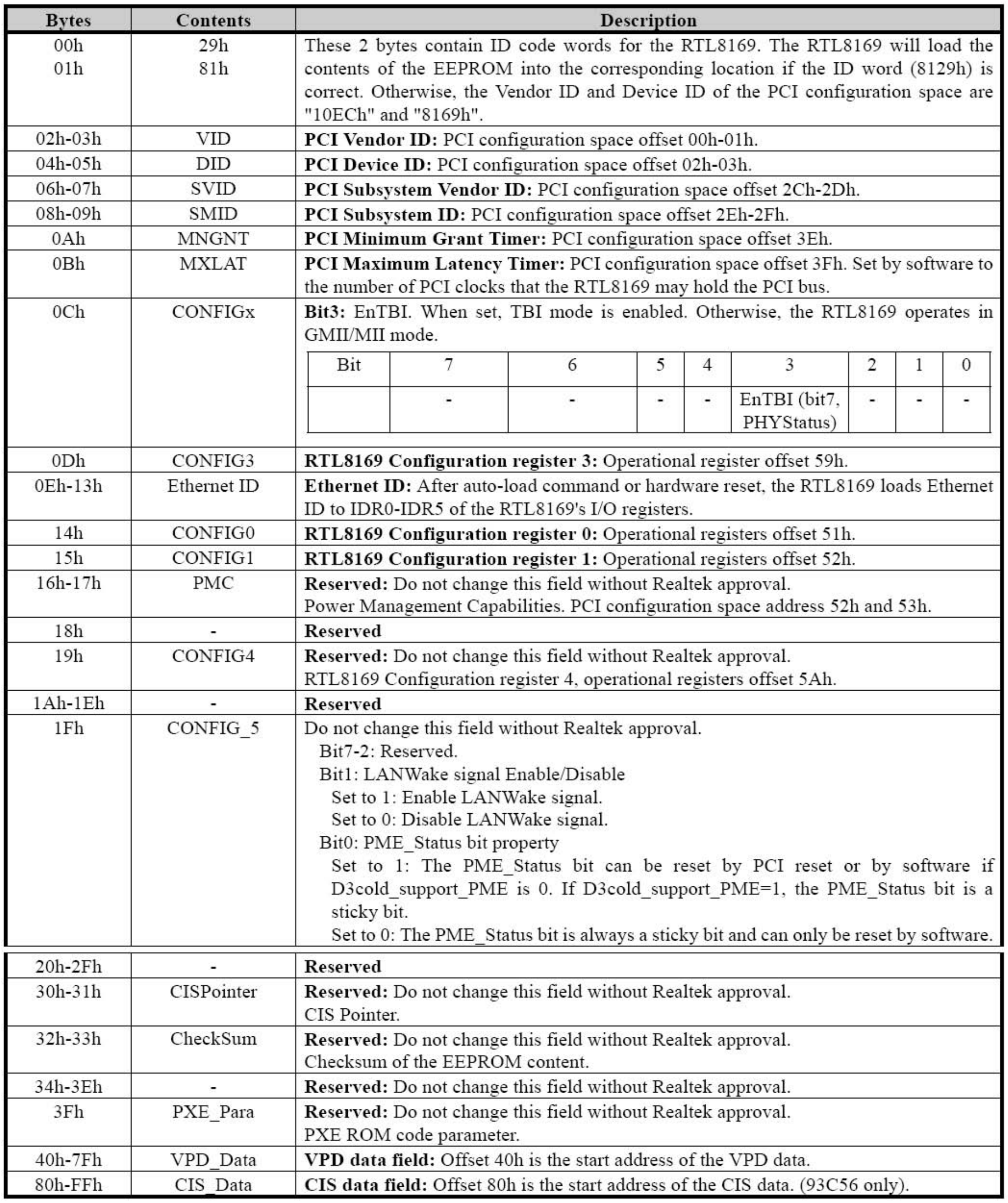


Figure 21: EEPROM interface timing[25]

makes a kind of race condition. However, if the original MAC address in the NIC registers is directly accessed then it is always fresh, which is the same as the one red when the system boots. Because of the reason, the method directly communicating with the EEPROM is comparatively most safe. However, we consider that this procedure may not exclude modifiability perfectly.

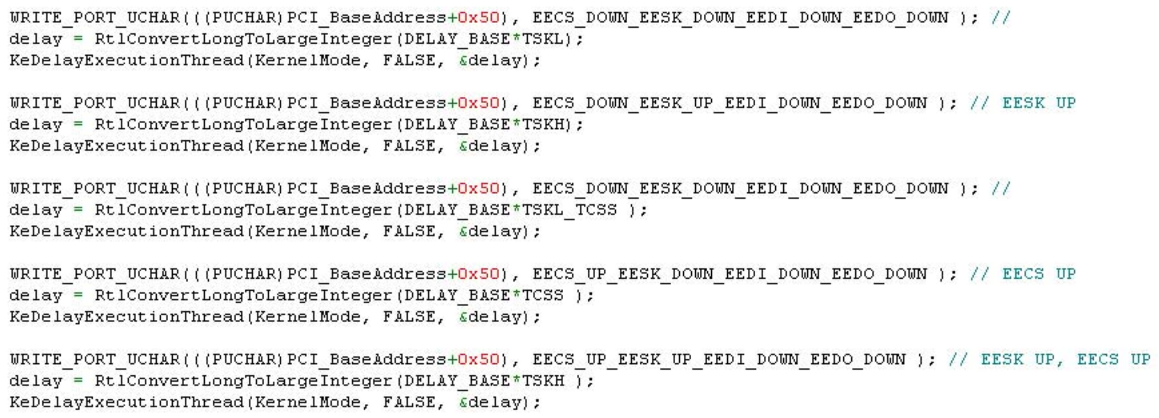
4 Conclusion and Future Work

In this paper, we survey various extraction methods for the MAC address that is being utilized in many applications as a platform-unique information. To show the details of the methods we implemented a set of sample software based on the surveyed extraction methods. Based on the result of the implementation, we conclude that the safety level of the method directly communicating with the EEPROM is qualified comparatively the best unless the content of the external EEPROM does not change. Additional research should be required to approve feasibility of the method for practical security applications. Although only the MAC address was analyzed in this paper due to its popularity, other hardware dependent values



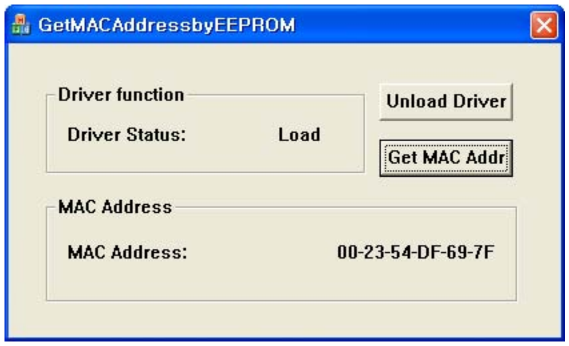
Bytes	Contents	Description
00h 01h	29h 81h	These 2 bytes contain ID code words for the RTL8169. The RTL8169 will load the contents of the EEPROM into the corresponding location if the ID word (8129h) is correct. Otherwise, the Vendor ID and Device ID of the PCI configuration space are "10ECh" and "8169h".
02h-03h	VID	**PCI Vendor ID:** PCI configuration space offset 00h-01h.
04h-05h	DID	**PCI Device ID:** PCI configuration space offset 02h-03h.
06h-07h	SVID	**PCI Subsystem Vendor ID:** PCI configuration space offset 2Ch-2Dh.
08h-09h	SMID	**PCI Subsystem ID:** PCI configuration space offset 2Eh-2Fh.
0Ah	MNGNT	**PCI Minimum Grant Timer:** PCI configuration space offset 3Eh.
0Bh	MXLAT	**PCI Maximum Latency Timer:** PCI configuration space offset 3Fh. Set by software to the number of PCI clocks that the RTL8169 may hold the PCI bus.
0Ch	CONFIGx	**Bit3:** EnTBI. When set, TBI mode is enabled. Otherwise, the RTL8169 operates in GMII/MII mode. Bit: 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 - | - | - | - | EnTBI (bit7, PHYStatus) | - | - | -
0Dh	CONFIG3	**RTL8169 Configuration register 3:** Operational register offset 59h.
0Eh-13h	Ethernet ID	**Ethernet ID:** After auto-load command or hardware reset, the RTL8169 loads Ethernet ID to IDR0-IDR5 of the RTL8169's I/O registers.
14h	CONFIG0	**RTL8169 Configuration register 0:** Operational registers offset 51h.
15h	CONFIG1	**RTL8169 Configuration register 1:** Operational registers offset 52h.
16h-17h	PMC	**Reserved:** Do not change this field without Realtek approval. Power Management Capabilities. PCI configuration space address 52h and 53h.
18h	-	**Reserved**
19h	CONFIG4	**Reserved:** Do not change this field without Realtek approval. RTL8169 Configuration register 4, operational registers offset 5Ah.
1Ah-1Eh	-	**Reserved**
1Fh	CONFIG_5	Do not change this field without Realtek approval. Bit7-2: Reserved. Bit1: LANWake signal Enable/Disable Set to 1: Enable LANWake signal. Set to 0: Disable LANWake signal. Bit0: PME_Status bit property Set to 1: The PME_Status bit can be reset by PCI reset or by software if D3cold_support_PME is 0. If D3cold_support_PME=1, the PME_Status bit is a sticky bit. Set to 0: The PME_Status bit is always a sticky bit and can only be reset by software.
20h-2Fh	-	**Reserved**
30h-31h	CISPointer	**Reserved:** Do not change this field without Realtek approval. CIS Pointer.
32h-33h	CheckSum	**Reserved:** Do not change this field without Realtek approval. Checksum of the EEPROM content.
34h-3Eh	-	**Reserved:** Do not change this field without Realtek approval.
3Fh	PXE_Para	**Reserved:** Do not change this field without Realtek approval. PXE ROM code parameter.
40h-7Fh	VPD_Data	**VPD data field:** Offset 40h is the start address of the VPD data.
80h-FFh	CIS_Data	**CIS data field:** Offset 80h is the start address of the CIS data. (93C56 only).

Figure 22: EEPROM contents[25]



```
WRITE_PORT_UCHAR(((PUCHAR)PCI_BaseAddress+0x50), EECS_DOWN_EESK_DOWN_EEDI_DOWN_EEDO_DOWN ); //
delay = RtlConvertLongToLargeInteger(DELAY_BASE*TSKL);
KeDelayExecutionThread(KernelMode, FALSE, &delay);

WRITE_PORT_UCHAR(((PUCHAR)PCI_BaseAddress+0x50), EECS_DOWN_EESK_UP_EEDI_DOWN_EEDO_DOWN ); // EESK UP
delay = RtlConvertLongToLargeInteger(DELAY_BASE*TSKH);
KeDelayExecutionThread(KernelMode, FALSE, &delay);

WRITE_PORT_UCHAR(((PUCHAR)PCI_BaseAddress+0x50), EECS_DOWN_EESK_DOWN_EEDI_DOWN_EEDO_DOWN ); //
delay = RtlConvertLongToLargeInteger(DELAY_BASE*TSKL_TCSS );
KeDelayExecutionThread(KernelMode, FALSE, &delay);

WRITE_PORT_UCHAR(((PUCHAR)PCI_BaseAddress+0x50), EECS_UP_EESK_DOWN_EEDI_DOWN_EEDO_DOWN ); // EECS UP
delay = RtlConvertLongToLargeInteger(DELAY_BASE*TCSS );
KeDelayExecutionThread(KernelMode, FALSE, &delay);

WRITE_PORT_UCHAR(((PUCHAR)PCI_BaseAddress+0x50), EECS_UP_EESK_UP_EEDI_DOWN_EEDO_DOWN ); // EESK UP, EECS UP
delay = RtlConvertLongToLargeInteger(DELAY_BASE*TSKH );
KeDelayExecutionThread(KernelMode, FALSE, &delay);
```

Figure 23: A part of implemented code communicating with the EEPROM



GetMACAddressbyEEPROM

Driver function

Driver Status: Load

Unload Driver

Get MAC Addr

MAC Address

MAC Address: 00-23-54-DF-69-7F

Figure 24: Result of the implement result communicating with the EEPROM

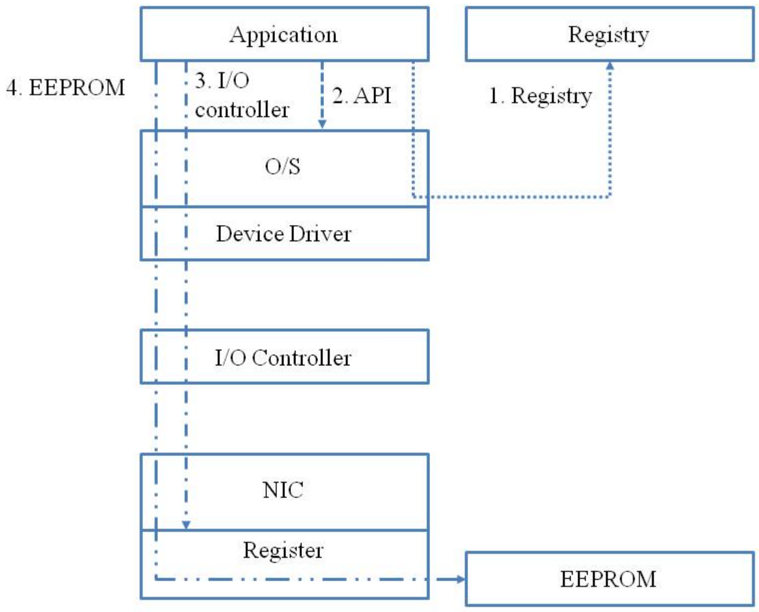
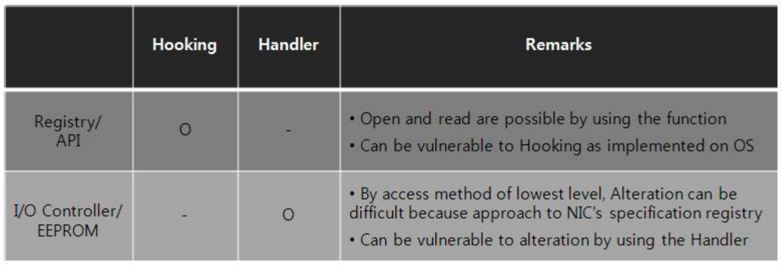


Figure 25: The extraction level of hardware unique information(MAC address)



	Hooking	Handler	Remarks
Registry/ API	O	-	• Open and read are possible by using the function • Can be vulnerable to Hooking as implemented on OS
I/O Controller/ EEPROM	-	O	• By access method of lowest level, Alteration can be difficult because approach to NIC's specification registry • Can be vulnerable to alteration by using the Handler

Figure 26: The chart of hardware unique information(MAC address)

including hard disk serial number, I/O map status, BIOS checksum and etc. are required to be analyzed as a future work.

References

[1] S. Jung, J. Kim, and S. Kim, “A Study on MAC Address Spoofing Attack Detection Structure in Wireless Sensor Network Environment,” *Journal of Advanced Communication and Networking Communications in Computer and Information Science*, vol. 199, pp. 31–35, June 2011.

[2] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell, “Detecting 802.11 MAC Layer Spoofing Using Received Signal Strength,” in *Proc. of the 27th IEEE Conference on Computer Communications (INFOCOM'08), Phoenix Arizona, USA*. IEEE, April 2008, pp. 1768–1776.

[3] W. Xing, Y. Zhao, and T. Li, “Reserch on the defense against ARP spoofing Attacks based on Winpcap,” in *Proc. of the 2nd International Workshop on Education Technology and Computer Science (ETCS'10), Wuhan, China*, vol. 1. IEEE, March 2010, pp. 762–765.

[4] K. Lee and K. Yim, “A Guideline for the Fixed PC Solution,” in *Proc. of the 2012 International Conference on Smart Convergence Technologies and Applications (SCTA '12), Gwangju, Korea*, August 2012, pp. 74–76.

[5] K. R. P. Association, “WPAN Alliance,” September 2010.

[6] M. Lei, Z. Qi, X. Hong, and S. V. Vrbsky, “Protecting Location Privacy with Dynamic Mac Address Exchanging in Wireless Networks,” in *Proc. of the 2007 Intelligence and Security Informatics (ISI'07), New Brunswick, New Jersey, USA*. IEEE, May 2007.

[7] S. Hong, “Secure MAC address-based Authentication on X.509 v3 Certificate in Group Communication,” *Journal of Korea Society for Internet Information*, vol. 9, no. 4, pp. 69–77, August 2008.

[8] Microsoft, “RegOpenKeyEx function,” http://msdn.microsoft.com/en-us/library/windows/desktop/ms724897(v=vs.85).aspx, 2012.

[9] ——, “RegQueryValueEx function,” http://msdn.microsoft.com/en-us/library/windows/desktop/ms724911(v=vs.85).aspx, 2012.

[10] ——, “UuidCreate structure,” http://msdn.microsoft.com/en-us/library/aa379358(v=vs.85).aspx, 2012.

[11] ——, “UuidCreate function,” http://msdn.microsoft.com/en-us/library/windows/desktop/aa379205(v=vs.85).aspx, 2012.

[12] ——, “UuidCreatesequential function,” http://msdn.microsoft.com/en-us/library/windows/desktop/aa379322(v=vs.85).aspx, September 2012.

[13] ——, “NetWkstaTransportEnum function,” http://msdn.microsoft.com/en-us/library/windows/desktop/aa370668(v=vs.85).aspx, 2012.

[14] ——, “GetAdaptersInfo function,” http://msdn.microsoft.com/en-us/library/windows/desktop/aa365917(v=vs.85).aspx, 2012.

[15] ——, “GetAdaptersAddresses function,” http://msdn.microsoft.com/en-us/library/windows/desktop/aa365915(v=vs.85).aspx, 2012.

[16] ——, "IP ADAPTER INFO structure," http://msdn.microsoft.com/en-us/library/windows/desktop/aa366062(v=vs.85).aspx, 2012.

[17] ——, "GetIfTable function," http://msdn.microsoft.com/en-us/library/windows/desktop/aa365943(v=vs.85).aspx, 2012.

[18] ——, "MIB IFTABLE structure," http://msdn.microsoft.com/en-us/library/windows/desktop/aa366842(v=vs.85).aspx, 2012.

[19] ——, "MIB IFROW structure," http://msdn.microsoft.com/en-us/library/windows/desktop/aa366836(v=vs.85).aspx, 2012.

[20] ——, "Netbios function," http://msdn.microsoft.com/en-us/library/bb870903(v=VS.85).aspx, 2012.

[21] ——, "NCB structure," http://msdn.microsoft.com/en-us/library/bb870902(v=vs.85).aspx, 2012.

[22] ——, "ADAPTER STATUS structure," http://msdn.microsoft.com/en-us/library/bb870890(v=VS.85).aspx, 2012.

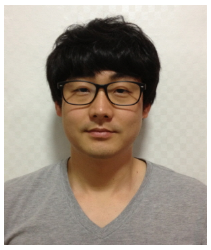
[23] ——, "HalGetBusData function," http://msdn.microsoft.com/en-us/library/windows/hardware/ff546599(v=vs.85).aspx, 2012.

[24] Intel, "Intel(R) I/O Controller Hub 7(ICH7) Family Datasheet," http://www.intel.com/content/www/us/en/io/i-o-controller-hub-7-datasheet.html, 2007.

[25] Realtek, "REALTEK GIGABIT ETHERNET MEDIA ACCESS CONTROLLER WITH POWER MANAGEMENT RTL8169," http://www.iitg.ernet.in/asahu/cs421/RealTek.pdf, 2002.

Kangwon Lee is currently working towards his B.S. in Information Security Engineering at Soonchunhyang University, Republic of Korea. Also, he is a member of the Lab. of Information Systems Security Assurance (LISA) led by Prof. Kangbin Yim. His research interests include vulnerability analysis, obfuscation, kernel mode root kit and insider threats.

Kyungroul Lee received his B.S and M.S. degrees from Soonchunhyang University, Republic of Korea in 2008 and 2010 respectively. As a member of the Lab. of Information Systems Security Assurance (LISA), he is currently working towards his Ph.D. degree in the same university. His research interests include vulnerability analysis, kernel mode root kit, obfuscation, platform security, access control and insider threats.



Jaecheon Byun received his B.S. from Soonchunhyang University, Republic of Korea in 2011 and he is currently working towards Master's degree. Also, he is a member of the Lab. of Information Systems Security Assurance (LISA) led by Prof. Kangbin Yim. His research interests include root kit, system security, access control and reversing.

Sunghoon Lee was a researcher at Samsung Electronics Co., Ltd. from 1989 to 1994, the CEO at ELTO Co., Ltd. from 1994 to 2006, and the Suwon Vocational College lecturer from 2007 to 2009. Since 2010, he has worked at EDA Korea co., Ltd as its CEO. In addition, he is now a member of the Lab. of Information Systems Security Assurance (LISA) led by Prof. Kangbin Yim. He has published several books on PCB design and H/W development. His research interests include secure system architecture and secure SoC design.

HyoBeom Ahn received the B.S. in Computer Science and M.S., and Ph.D. in Computer Science and Statistics from Dankook University, Korea in 1992, 1994 and 2002 respectively. Since then, he has been with the Dept. of Information and Telecommunication, Kongju National University, Republic of Korea. His main research interests include Computer Networks, Network Security and SmartGrid Security.

Kangbin Yim received his B.S., M.S., and Ph.D. from Ajou University, Suwon, Korea in 1992, 1994 and 2001, respectively. He is currently an associate professor in the Department of Information Security Engineering, Soonchunhyang University. He has served as an executive board member of Korea Institute of Information Security and Cryptology, Korean Society for Internet Information and The Institute of Electronics Engineers of Korea. He also has served as a committee chair of the international conferences and workshops and the guest editor of the journals such as JIT, MIS and JoWUA. His research interests include vulnerability assessment, code obfuscation, malware analysis, leakage prevention, secure platform architecture and mobile security. Related to these topics, he has worked on more than forty research projects and published more than ninety research papers.