# File system driver filtering against metamorphic viral coding

Ruo Ando, Hideaki Miura*,Yoshiyasu Takefuji
Graduate School of Media and Governance,Keio University,
5322 Endo Fujisawa, Kanagawa, 252 Japan
*SciencePark Corporation
1-1538-11 Iriya Zama, Kanagawa, Japan 228-0024
{ruo,takefuji}@sfc.keio.ac.jp http://www.neuro.sfc.keio.ac.jp
*miura@sciencepark.co.jp http://www.sciencepark.co.jp

*Abstract:* - Recent cyber attacks and viruses become more sophisticated. Metamorphic virus such as Win32 simile have a greate impact on anti-virus software developers, which evades signature mathing by inserting redundant fake assembler operation codes. In this paper we propose a file system device driver enhanced protection technique to prevent the metamorphic viral coding attack. File system driver filtering can detect the morphed viral code scattered over infected program in run-time environment without emulation. This technique enables us to apply systematic detection for the large number of malformed code by detecting a single event on device driver layer. In experiment, we focused on the metamorphic coding of buffer overflow exploit and it was showed that proposal system is effective in preventing the viral coding attack regardless of its metamorphic transformation. Our system is applicable only by replacing device driver enhanced of the inspecting buffer overlfow. Without modifying operating system, service software, application software, the trap in the kernel level is able to detect the buffer overflows in real-time and to provide process-control after obtaining the detailed data about malicious processes.

*Key-Words:* - Metamorphic computer virus, Device driver based protection, Buffer overflow, File system driver, Complexity of metamorphism, Filter driver, Real time nullification

## 1 Introduction

The number of security incidents is still constantly increasing, which imposes a great burden on both the server administrators and client users. Despite the short history, computer virus is becoming one of the most important issue. Although it has been about one decade since computer viruses became expected occurrence, Viruses, worms and Trojan damages personals, companies government. Code Red, Nimda and MsBlaster recently are a valid example showing we suffer the great damege if we keep using the computer unpatced. We have come to accept the updating a software regularly and uniformly. Besides, the cost of maintaining and updating vulnerable software is increasing gradually but certainly.

### 1.1 Metamorphism : viral code hiding

Recent cyber attacks and viruses become more sophisticated, while many users begin to equip anti virus scanners that is mainly relies on signature matching. Recently virus is improved rapidly so as to evade signature matching. Symantec Corporation published the paper [2] introducing metamorphic viruses against the impact of W32 simile computer virus, loading complex viral code hiding techniques. Viral code hiding techniques which avoid the string matching are not new phenomenon. This kind of technology is first appeared in early 1990's, called polymorphic computer virus. Polymorphic viral coding applies encryption for its body to nullify the virus scanning. Still now polymorphic virus is challenging to detect completely and effectively. However, there exists detection method and scanners that survives and improved from DOS 16bit days for Polymorphic coding attack [3].

Metamorphic virus can change whole its body with any encryption. Instead of the crypt engine, metamorphic viral code are generated by inserting redundant assembler, replacing register and changing magic word so as not to disturb the same action of pre-morphed virus in targeting operating system. As a result, after metamorphic coding, the virus can change its body while keeping the same function. As we discuss in section 3.1, once some virus become metamorphic, little piece of viral code are scattered over the whole infected program body. Consequently

virus scanner cannot detect it with the sequence matching.

Although metamorphic virus is not appeared newly like polymorphic viruses, with the rapid improvement and complication in 32 bit processors and operating system, metamorphism is applied studiously by virus writers. While there are actually the scanners that survived DOS polymorphic days, no one can find the effective way of detecting viral metamorphism. Besides, it is hard to automate the heuristic process of detecting metamorphic compared with another viral code hiding techniques.

### 1.2 Buffer overflow as higher action

There are many kinds of computer viruses. Among those, from the sections below, we focus the systematic approach to detect the malmormed viruses loading buffer overflow exploits. The thrust of this paper is to validate that there is the possibility of catching "higher actions" that are performed by computer viruses regardless of its metamorphic form. The proposal system introduces a layer built by device driver programming where we can cancel the redundant assembler instruction and translate some operation code into more abstract action. Adversely, the proposal techniques detect one higher action from many kinds of derivation of metamorphism, regardless of its forms on register transfer, that is, assembler level. For example, metamorphism is based on the point that multiple implementation of one action (such as call kernel32.dll) could be possible by using several combination of operation code.

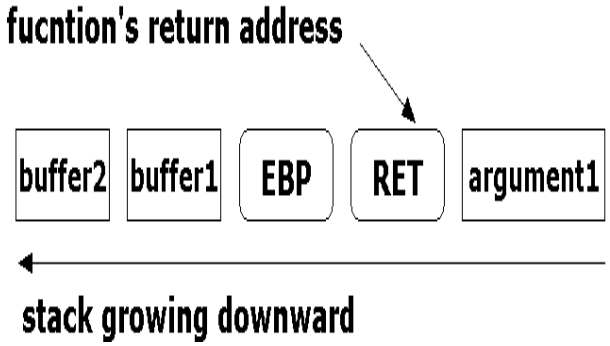


**Fig. 1 Allocation in stack**

In this paper we pick up buffer overflow as one example of higher actions. Buffer overflow is one of the most commonly exploited in software bugs, which is occurred when the memory reserved for data is not assigned enough size for processing inputs[1]. Figure1 illustrates the sample of allocating buffers and pointers when the local function is called. In most implementation in C, there could be the case that the longer input is allocated in the memory that it can handle by array length previously fixed. In the implementation that includes such bugs, a malicious user can rewrite the return address, which is indicated on ESP, by submitting of an extra-long input to the program. If ESP pointer is overwritten, it come to be possible to execute arbitrary malicious actions after operation is return from local function.

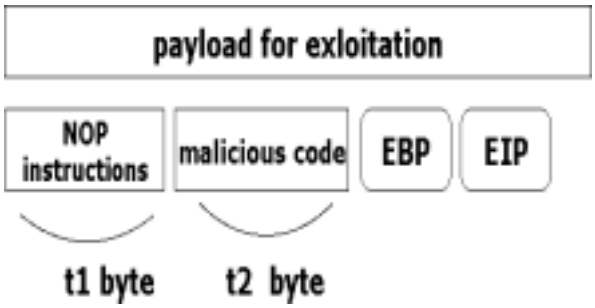


**Fig. 2. BOF exploits with nop instruction**

Figure 2 shows the illustration of buffer overflow using NOP instruction. In many cases of malicious code execution, it is almost impossible to estimate offset for the target program, so the virus writer inserts NOP instruction ahead of shellcode to improve the probabilities of exploitations. This could be:
*P(exploit)=1-(P(t2)-P(t1)) P(t2)>P(t1).*

Where P(exploit) is the probability of succeeding malicious code execution. P(t1) changes according to the size of buffer assigned in the target program. However, by using our system, we can catch the moment of returning from function. Consequently, proposal system enables us to prevent malicious code execution regardless of its probabilities of BOF exploitation.

Unfortunately or not, almost all the modern operating systems are constructed with the demand of high-level languages, which consists of the procedure and function. Consequently, stack and heap is irreplaceable implementation for this kind of system to control the flow of the call and return instructions. Nowadays, more than 70% of security incidents have been occurred by buffer overflow [10], which is still undiscovered despite the efforts of inspecting and verifying programs both manually and automatically. It is expected that the improvement of secure programming[11] is more required for the preventative against security incident. We will

discuss some countermeasures for buffer overflow in section 2.3.

# 2 Related work

In this section we discuss about some technologies detecting computer viruses. Detection method could be categorized into two types: 1)signature matchin*g* dealing with knows misuse viral coding and 2)heuristic scan applying discover method to find unregistered computer virus. Heuristic scan is almost as same meaning as anomaly detection adopting data-mining techniques for secure network [12][13]. Another type is adaptive protection, which is implemented compiler to inspect the integrity of call pointers when program is executed.

## 2.1 Signature matching

Signature matching is searching the particular byte sequences in files according to each format, .COM file, .EXE file, .out file, scripts and macros. If predefined sequences are found, some action such as alarm, nullifying is triggered. In the sense that signature matching hardly generate false positive alarm, this technique is still core mode of anti virus detection. Signature matching, while made more flexible by pre-qualifying files and type of infections, and using wild cards, still requires exact matches between infection and signature. Also [6] shows the effectiveness in anomaly detection of process behavior by tracing system call sequence

Although the Anti virus software have relies much on signature-based techniques including regular expression and wild card, it could be pointed out that this method is sometimes CPU intensive, and costs a lot in frequently managing signature. And to ensure signature definitions, these should be updated from server of each vendor regularly and uniformly. Consequently time lag of updating could be the cost and cause to be exploited when it is not updated.

## 2.2 Heuristic scan

Recently anti virus software began to equip heuristic scan. Heuristic scanning is the operation to complement signature matching in finding potentially malicious code (or actual viral code ) that have not been released and corresponded by anti virus software vendor. Instead of looking for specific strings, heuristic scanning deal with higher information such as assembler operation code or commands in order to find uncategorized viruses or possibility of malicious code.

The word heuristic (hyu-RIS-tik), which is originated from Greek word heuriskein, means the way to determine something in a methodic or experimental way. A skilled programmer can notice the sign of malicious operation from normal one when he inspects the program carefully by some debugging tools. Heuristic scan is applied so that the experience or knowledge of debug expert in to a anti virus software. These scanning techniques are now available in many popular anti virus software although there still many way proposed to evade heuristic scanners. When it is executed, heuristic scanner searches hundreds of operation code, instructions and behaviors that viral code may include and calculate possibility according to the threshold the user has set up. Nowadays, it is summarized by AV vendors that about 70-80% of unknown virus can be detected in heuristic scan.

## 2.3 Stack protection technologies

Adaptive protection mainly represented by compiler solution is loaded in major operating systems [8]. In this section we pick up the stack inspection and protection technologies. Static analysis of C language program is proposed in [5]. However, the longer the code to inspect becomes, it is more difficult to find a malformation point of return address or data area even for the skilled developer. StackGuard [4] is designed to check overwritten. This is a mechanism that can be embedded in standard GNU C compiler gcc, inserting value called “canary” just next to the return address. The key point behind it is that buffer overflow attacks is overwriting any local variables, old base pointer and finally the return address by overfilling the buffer intentionally. Also libsafe is the library version of stack protection techniques. Point guard is the same kind of techniques of Stack Guard, but it requires particular programmer intervention. In the recent version of OpenBSD, these stack protection technologies is available from the version 3.3. However, it could be pointed that t heir disadvantage lies in that kernels and software components must be rebuilt. Disadvantage of the compiler-based stack protection technologies are as follows:

[1] When the compiled program is loaded to the memory, its process is heavier to run than the previous process, because RA (return address) integrity check is executed whenever the function is called.

[2] There are still techniques proposed to overwrite return address without changing canary word.

[3] Every software and kernel modules must be recompiled whenever the bug or vulnerabilities are found in previous version of the source code.

Particularly concerning [3], we should consider these three constraints to maintain secure computer operation environment.

[1]It is impossible to find all vulnerabilities in the existing operating systems and application software programs.
[2]It is impossible to patch the vulnerable systems immediately.
[3]It is impossible to rebuild the risk-free system and replace the existing system.

The proposed scheme does not need the software-rebuilding. We only need to replace the driver module of the proposal version in existing operating system. As we discussed later, the concept of proposal method is operating system independent.

# 3 Metamorphic viral coding

As we discussed in section 1.2, there are two methods to evade the string template matching: polymorphic and metamorphic coding. Although the polymorphic viruses are hard to prevent still now, there exists a countermeasures that have been improve since DOS 16 bit era. Polymorphic virus must have a executable code section that operating system can recognize. Then, once decrypted or decrypting engine is discovered, this could be manageable by signature scanner and eradiated. On the other hand, through the last decade when the architecture of 32 bit processors or operating system becomes more sophisticated and complicated, metamorphism is studiously applied by virus writers. As matters stand, there is no decisive technique for detecting metamorphic viral coding. Anti virus software companies says that less than 70-80 % of viral metamorphism could be detected. In this section, we discuss four types of metamorphic viral coding, which are the same in mutating operating code and magic word form the same higher action.

## 3.1 Register replacement

As some simple techniques of metamorphic coding, we can exploit the exchangeability of some registers in IA 32 architecture.

```
POP EDX
MOV EDI, 0008H
MOV ESI,EBP
MOV EAX 000DH
ADD EDX, 005FH
MOV EDX,[EDX]
MOV [ESI+EAX*0000CCC9,EBX]

POP EAX
MOV EDX,0008H
MOV EDX,EBP
MOV EDI,000DH
ADD EAX,005FH
MOV ESI,[EAX]
MOV [EDX+EDI*0000CCC9],ESI
```

**List1. Register replacement**

List 1 shows the metamorphic coding of generating two different forms by replacing register. In this case, edx is replaced by eax, ebx by edi, edi by ebx, and esi by ebx. As a result, when this kind of code is translated in machine language, machine code is changed after compilation.

## 3.2 Magic number permutation

Some metamorphic virus mutates a new form by changing magic word. List2 shows the substitution of magic word into ESI is permutated. The line 1 is malformed by using register EDI and EDX. And in line 2, substitution of 110000FFH is translated through EDX and EBX.

```
MOV DWORD PTR [ESI] ,11000000H
MOV DWORD PTR [ESI+0004],110000FFH

MOV EDI,11000000H
MOV [ESI],EDI
POP EDI
PUSH EDX
MOV DH,40
MOV EDX,110000FFH
PUSH EBX
MOV EDX,EBX
MOV [ESI+0004],EDX
```

**List2. Register permutation**

Compared with the list2, which could be detected by crafted string matching such as half-byte wild cards, the next case in list 3 go further to change magic value 11000FFH.

```
MOV EDX,11000000H
MOV [ESI],EBX
```

*POP EDX*
*PUSH ECX*
*MOV ECX,11000000H*
*ADD ECX,000000FFH*
*MOV [ESI+0004],ECX*
**List3. Magic number permutation**

List3 shows dividing the magic word 110000FF into 11000000 and 000000FF. Consequently, wild card based string matching become disable to find the magic number.

### 3.3 Reordering instructions

Compared with polymorphic viruses which decrypt themselves to a constant virus body in memory, this type of metamorphic does not come to be constant because jump instruction is inserted at random.

*INSTRUCTION_A*
*INSTRUCTION_B*
*INSTRUCTION_C:*

*LABEL_2:*
*INSTRUCTION B*
*JMP*
*FAKE INSTRUCTIONS*
*START:*
*LABEL_3:*
*INSTRUCTION_C:*
*LABEL_1:*
*JMP*
*FAKE INSTRUCTIONS*
**List4. Reordering instructions**

List4 shows the obfuscation of entry point to avoid the searching of the beginning of the executable code section. As a result, signature is scattered in amongst the original code. Furthermore, in this technique virus can inserts fake instruction between core instruction and jump code. In extremely case of this kind of method, Win95 Zperm is the first virus to generate millions of iterations to surpress the anti viral emulation speed.

These four types of metamorphism are the same in the sense that there could be translated as the certain abstraction from both core and fake instructions regardless of its various malforming forms. In other words, whatever the code is permutated, the function that each morphed code has to achieve is the same, consequently a kind of higher action can be logged as event in device driver. Concretely, with the example in section 1.2, overflow is finally occurred despite its malformation of assembler code. Adversely, as long as we can only investigate on the assembler instruction level, we cannot go out of heuristic or data mining frameworks. From the next sections, we propose an insertion of new layer, called device driver based protection layer, to obtain the highly abstracted action of metamorphic code.

### 3.4 Complexity of metamorphism

In this section we discuss about the complexity to detect the morphed viral code. Through last decade, operating system and CPU have become more sophisticated and complicated accompanying with implementation of many function, consequently we do not use all instructions and operations at the same time for one purpose. Adversely, many combinations of routines come to be possible to achieve the same utilization. Metamorphic viruses are exploiting this point of modern computer system. Morphed virus writer implements n functions in order to generate n! variations. For example, if Win32 metamorphic virus such as W32/ghost has 16 routines, the combination could be:

*Combination = 16! = 20922789888000*

Besides, the computer viruses choose one sequence of instructions almost unpredictably, among possible combinations using random number based on some value such as TLB (thread information block). Also another register transition that is usually unpredictable could be the seed of random number.

*Selection = random (seed) Seed : FS:Och, EIP, etc*

Figure2 shows the illustration of list inserting fake instructions for redundant state in order to evade the signature matching.

*MOV DWORD_3,6E72654BH*
*MOV ESI,[EDX](*)*
*MOV DWORD_4, 32336C65H*
*MOV EDI,[EBX-04X](*)*
*MOV DWORD_5,0H*
*NOP(*)*
*CALL DS:[GETMODULEHANDLEA]*
**FAKE INSTRUCTION*
**List5. Inserting fake operations**

List3 shows the metamorphic code of calling API. As we discussed in section 1.2, the higher action of this code is “locate the kernel32 dll in the memory”. In Intel 32 bit architecture, there are eight generic registers available for programmers, all of which are not used in single operation. Particularly, ECX, EDX,

ESI and EDI are often applied for auxiliary use. It follows that at factorial of 4 combinations is possible without accounting order of fake instructions.

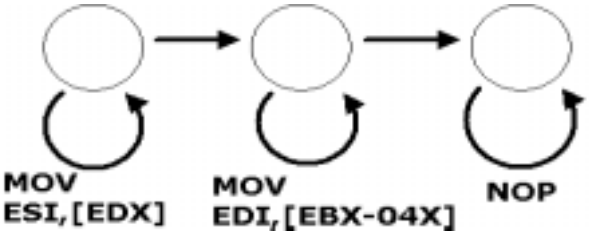


**Fig. 3 Redundant loop of list 5**

Concerning the magic number, every magic number in windows operating system can be decomposed arbitrarily into 32 bit memory address number. However, the hexadecimal numbers from 0xBFFFFFFF to 0x00000000 is preferable because the address from 0xFFFFFFFF from 0xC0000000 is number in kernel mode that is inclined to be hooked in heuristic scanning. Thus, metamorphic viral coder can generate the vast number of derivations using the large memory space and abundant availability of operations in IA and Win 32 elaborate architecture.

# 4 Proposal system

## 4.1 System architecture

Figure3 shows what we can focus on to detect malicious action in four layers: C language program, kernel mode in operating system, events in device driver, and assembly code register transfer level.

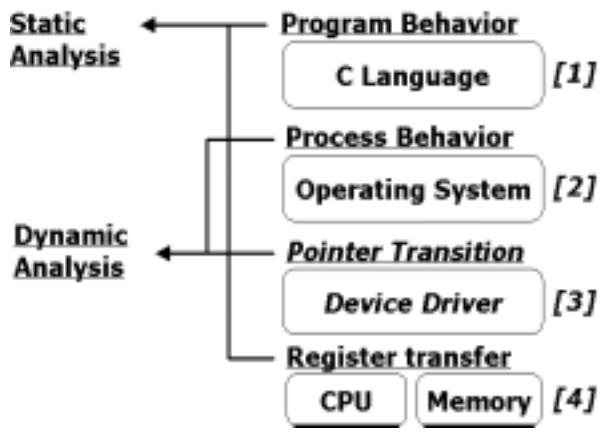


**Fig. 4 Detecting style and object in four layers**

Many traditional detection methods are applied on high-level language layer, kernel mode in operating system or low-level register transfer layer. It is an important point that each previous detection on layer [1][2][4], it is impossible to nullify the buffer overflow exploits because of the lack of adequate abstract data and events necessary to obtain for detecting buffer overflow in real-time. In other words, we have to get proper level of abstractness to get event information of malicious action. For previous detection techniques to detect morphed viral code, any detection in layer [1][2][4] is too abstract or concrete. To solve this problem, we built and equip the device driver based protection layer in the kernel mode where the existing device drivers can mutually communicate each other. As a result, we can protect the memory from the malicious operation directly. Besides, the new attributes can be added to memory management functions through this proposal layer.

## 4.2 File system driver filtering

Figure3 shows the simplified illustration of proposal system in windows 2000/NT Operating system structure. When certain file in storage device loaded in main memory, virtual memory manager and file system driver works mutually for the process management. In addition, it is important feature of Windows 2000 operation system that file system filter driver is placed on file system driver to observe the IRP (I/O request packet) information and stop the instruction flow if there are some errors.

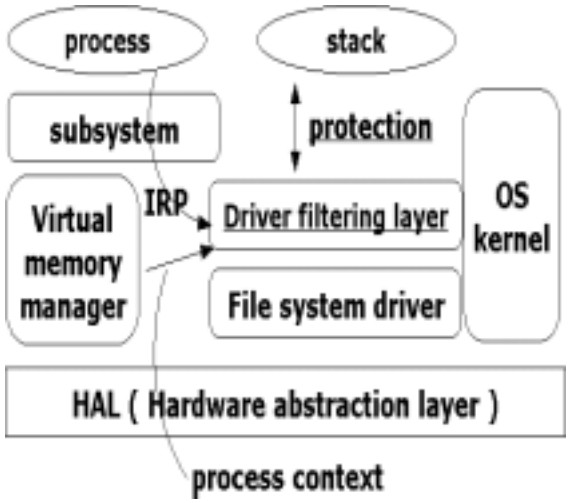


**Fig. 5. Proposal system**

The proposal technique applies the file system filter driver to inspect integrity of transition of instruction pointer before and after the function call. In windows operating system, virtual memory manager and file system driver coordinates to each other. Consequently we can obtain the detailed

process context by improving file system driver and filter driver with the help of virtual memory manager.

### 4.3 Real-time detection and prevention

In this paper we implemented the driver-based stack protection system detecting buffer overrun. As we discussed in section 1.2, buffer overflow is occurred when we overwrite the buffer with the longer strings even to reach the EBP ( base pointer ) and EIP (return address). In proposal system we can hook the function call, and loading memory of execution file into memory of focused application by improving file system driver and filter driver on it.

[1]When loading the execution program to main memory, the proposal protection layer traps for the call pointers in order to set the read-only memory attribute to the instruction pointer of return address on runtime.
[2]Malicious access to the read-only memory (instruction pointer of return address) is detected by system error where invalid memory access notification can be hooked by the proposal protection layer. Consequently, the malicious execution code is nullified.

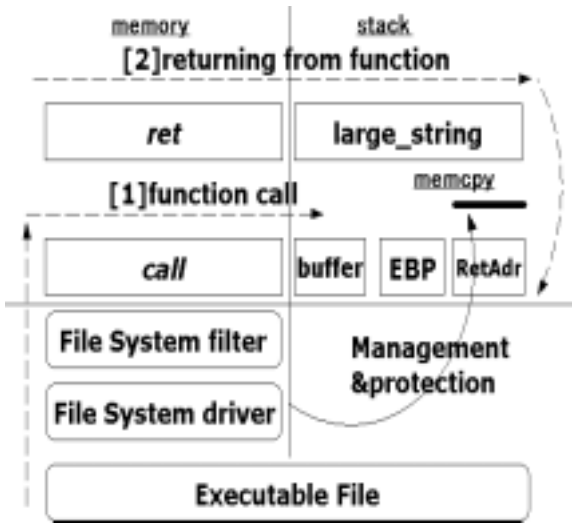


**Fig. 6. Real-time nullification of BOF**

Figure5 illustrates the real-time protection of return address in proposal method. In this system, any exploitation is aborted by malicious memory access with this function. Whether operating system has unknown security holes or not, malicious execution code on read-only memory access is nullified and aborted.

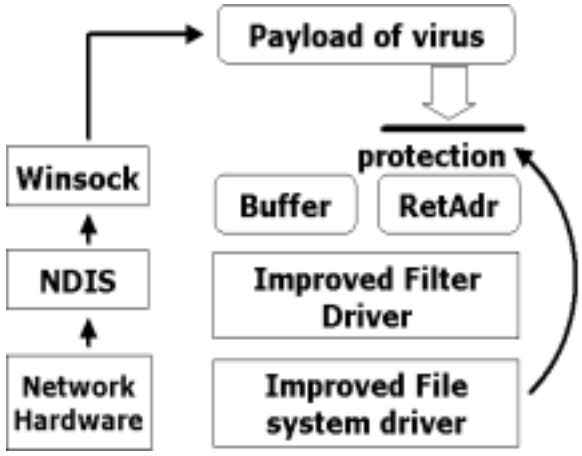


**Fig. 7. On-line virus protection**

Figure6 shows the protection system against network virus infection. There are many cases that the vulnerabilities of the software permitting inputs from both client and server without buffer bound checking. In our system, it is possible to prevent malicious code execution such as MSBlast, CodeRed in permitting illegally long inputs by file system based process handler.

## 4 Experimental results

### 4.1 Experimental Environment

In experiments, the proposal system has been implemented as service on Windows 2000 operating system. Our system consist of system file(SYS), registry editing script(REG), and executable(EXE). Executable file running as service stored in *C:\%systemroot%\system32*.
The device driver protection file is placed in driver directory, with the configuration of parameter in *C:\%systemroot%\HKEY_LOCAL_MACHINE \SYSTEMCurrentControlSet\Services\ \Antistackoverflow*.
With this prototype version of driver based protection system, we can observe a single process by specifying case-sensitive name for anti stack buffer overflow services.

### 4.2 Penetration program

Our attacker program is written in VC++ and MASM (Microsoft Macro Assembler) to occur buffer overrun. We adapt the simple form of buffer overflow. Program is verified in three types of metamorphic discussed above, based on the same forms as follows.
*void bof_proc()*
*retadr= arbitrary address;*

```
char large_string[];
memset();
large_string[]=arbitrary code;
large string[] = retadr;
memcpy(large_string,buffer,n);

main()
  __asm( label1:)
bof_proc();
  __asm(jmp label1)
```

**List6. Program for experiment**

After the buffer overrun, the instructional pointer is moved just after return address overwritten. As is list5, in order to execute particular application of Windows, we jump to the address of Winexec(). For obtaining reproducibility and many examples of situations on Windows operating system, we set up loop routine of buffer overflow using inline assembler in VC++.

### 5.3 Obtained results and analysis

In experiment we should consider that there could be many metamorphic coding way even in the simple overrun-based illustrated in section 1.2. In theory, if the virus has ten subroutines, the number of variations of metamorphic coding could be 10!. Although some signature for detecting viruses is expected to be the machine code of launching shell code[1], this time we focused the craft metamorphic coding for our own attacker program itself. The patterns we tested in the validation of the proposal driver-based protection system is as follows:

*Case1: Register and instruction replacement*

In the nature of the protection system, the stack segment is protected. We apply metamorphism for bof_proc() illustrated in section 3.1.

*Case2: Magic number permutation*

Many exploits code attempts to launch some application or API. In particular, the function Winexec() is often called after buffer overflow. We permute the magic number of fixed address of Winexec() in windows 2000 operating system without no service pack.

*Case3: Reordering instructions*

We scattered the jump instruction and insert the fake instructions just after it in order to malform the signature from which the morphed one is derived.

In these three cases, our system is effective to detect misuse and invalid the malicious process despite the forms of metamorphic coding. No matter how morphed the assembler code written in register transfer level is, the transition of pointers such as EBP, EIP, and ESP is eventually combining as then same value in driver-level observation. In other words, by inserting file system driver-based protection layer, we can observe the integrity of register transition directly, which is not the case in applying heuristic scanning or data mining techniques.

## 5 Conclusion

In today's IA32 and Win32 based computer system, we currently should consider following three constraints in maintaining secure environment.

[1]It is impossible to find all vulnerabilities in the existing operating systems and application software programs.
[2]It is impossible to patch the vulnerable systems immediately.
[3]It is impossible to rebuild the risk-free system and replace the existing system.

In this paper, the file system driver based protection was introduced in an attempt to detect and prevent metamorphic computer viruses without rebuilding vulnerable application and kernel source codes. The conventional anti-virus software, firewall, and IDS are all based on stored signatures. Consequently these schemes have the limitation against the new derivation using metamorphic coding discussed in section 3. The examples in this paper represent a possibility of detecting higher action against metamorphic viral coding regardless of its form of morphed viruses. The advantages in applying the proposal driver enhanced protection are as follows:

[1]Compared with the signature matching and heuristic scanning: previously, to detect the polymorphic and metamorphic viruses, heuristic scan is applied with the average probability rate of detection about 70%, which is claimed by anti virus software vendors. By using proposal method we can observe the integrity of the pointer transition and stack operation in run-time host environment. Consequently, this technique enables us to apply systematic detection for the large number of

malformed code by just a single detection of event on device driver layer.
[2]Compared with compiler based stack protection technologies: The proposed protection method does not need the software-rebuilding, while the compiler-based protection schemes need the software-rebuilding. The concept of proposal method is operating system independent.

For further work, the thrust of our next version of protection system is the process-control where malicious execution codes are nullified by the new attribute of read-only memory in the instruction pointer through file system driver layer.

**Acknowledgements**
We are indebted to K.Shoji, T.Kawade and T.Nozaki by courtesy of SciencePark Corporation[9]. The idea of the proposal system in this paper grew out of an ongoing collaboration with their team.
This effort is supported by AFOR Scientific Research with Grant Number AOARD 03-4049.

*References:*
[1] Aleph One,Smasing The Stack For Fun And Profit. Phrack, 7(49), November 1996.
[2] Szor, Peter and Ferrie, Peter. "Hunting for Metamorphic." Virus Bulletin Conference, September 2001.
[3] Stephen Pearce, “Viral Polymorphism”, paper submitted for GSEC version 1.4b,2003.
[4] C.Cowan, C.Pu, D.Maier, J.Walpole, P.Bakke, S.Beattie, A.Grier, P.Wagle, Q.Zhang, and H.Hinton. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In Proc. 7th USENIX Security Conference, pages 63--78, San Antonio, Texas, jan 1998
[5] David Larochelle and David Evans,Statically Detecting Likely Buffer Overflow Vulnerabilities,2001 USENIX Security Symposium, Washington, D.C., August 13-17, 2001.
[6] Kosoresow, Andrew P. and Steven A. Hofmeyr, "Intrusion Detection Via System Call Traces", IEEE Software, Sept.–Oct. 1997, pp 35-40.
[7]Diomidis Spinellis. Reliable identification of bounded-length viruses is NP-complete. IEEE Transactions on Information Theory, 49(1):280-284, January 2003.
[8]Openwall linux project.
http://www.openwall.com/linux
[9] 4th Eye: Robust security system against malicious employees.
http://www.sciencepark.co.jp/4thEye/Eng/
[10]NIST vulnerabilities database engine.
http://icat.nist.gov/
[11]Khaled.E.A.Negm,"Secure Mobile Code Computing in Distributed Environment", WSEAS TRANSACTIONS ON COMUTERS,2003,pp506-513
[12] Dimitris A. Karras, Vasilis Zorkadis, "Neural Network Techniques for Improved Intrusion Detection in Communication Systems" WSEAS CSCC,2001,pp318-323
[13]Ruo Ando, Yoshiyasu Takefuji, "Two-stage quantitative network incident detection for the adaptive coordination with SMTP proxy", Computer Network Security, Lecture note in computer science Springer 2003,pp424-428