APLIKASI PEMBATASAN PENGAKTIFAN EXECUTABLE DENGAN SYSTEM SERVICE DISPATCH TABLE (SSDT) HOOK

Hendra[1]
Teknik Informasi
STMIK IBBI
Jl. Sei Deli No. 18 Medan 20114
Telp.: +6261-4567111, e-mail: [1]hendra.soewarno@gmail.com

Abstract

Antivirus technology uses patterns approach and heuristic approach to detecting the presence and malware attacks. The pattern approach has one step late to the malware existence, this approach is available when the malware samples can be obtained for pattern extraction, whereas heuristic approach runs into problems of false alarms and tend to disturb the comfort of the user. Nowadays, malware authors tend to use attacks that exploit zero-day vulnerabilities that are not known by the user or software creator. Based on data from Symantec that the highest mechanism for distributing malware in 2010 is through the file exchange using a USB flash media, exploitation of the AutoRun facility and the MS08-067 vulnerability. In this paper the authors would like to offer an approach to the malware infections prevention by limiting launch of executable files from the folder %SystemRoot % and %ProgramFiles% using kernel level SSDT API hooking.

Keyword: *malware infection prevention.*

1. PENDAHULUAN

Berdasarkan laporan dari Symantec bahwa pada tahun 2010 terdapat tiga juta serangan malware, dan memberikan perhatian khusus kepada malware Stuxnet[1] yang menyebar dengan melakukan eksploitasi terhadap kerentanan MS08-067 yang merupakan suatu cacat pada Windows shortcut (.LNK), dan sebuah kerentanan zero-day pada layanan Print Spooler yang memungkinkan untuk kode berbahaya dikirim, dan dijalankan pada mesin remote tanpa intervensi pemakai. Serangan dengan memanfaatkan kerentanan zero-day yang merupakan kelemahan yang tidak diketahui pemakai maupun pembuat software menyebabkan malware dapat diinstalasi pada komputer tanpa sepengetahuan korban, disamping itu malware ini juga dilengkapi dengan rootkit untuk kemampuan stealth dan memata-matai sistim industri Siemens SIMATIC WinCC database applications [a].

Pendekatan software antivirus untuk mengenali malware dengan menggunakan pattern adalah kurang efektif, karena senantiasa satu langkah terlambat dari keberadaan malware, sedangkan pendekatan heuristik tidak populer dikalangan pemakai karena sering menimbulkan permasalahan *false alarm* dan kadang-kadang dirasakan menganggu[2]. Berdasarkan data dari Symantec bahwa mekanisme penyebaran malware tertinggi pada tahun 2010 adalah melalui mekanisme pertukaran file executable melalui media USB flashdisk dalam bentuk autorun, dan eksploitasi terhadap kerentanan MS08-067[1].

Pembatasan pengaktifan file *executable* dapat dilakukan dengan melakukan interception terhadap fungsi-fungsi API pada sistim operasi yang berkaitan dengan pembuatan process executable. Proses interception ini dapat dilakukan melalui hooking pada *level user* maupun pada *level kernel*. Proses hooking pada level kernel memiliki keunggulan yaitu hasil hooking akan mempengaruhi sistim secara keseluruhan, sedangkan pada *level user* hasil hooking hanya mempengaruhi suatu aplikasi tertentu[3].

Perumusan masalah pada penelitian ini adalah bagaimana pembatasan pengaktifan eksekusi file executable dengan dapat dilakukan melalui hooking pada level kernel, dan bagaimana diaplikasikan dalam bentuk solusi perangkat lunak pembatasan pengaktifan *executable* ?.

Secara umum penelitian ini adalah merancangan algoritma pembatasan pengaktifan dan mengembangkan suatu aplikasi yang dapat membatasi eksekusi file executable berdasarkan folder.

Penelitian ini dibatasi pada file executable yang memiliki format Portable Executable (PE), dan perangkat lunak pembatasan eksekusi ditujukan pada Sistim Operasi 32-bit Windows XP, karena sistim operasi Windows merupakan sistim operasi yang sering menjadi target dari

pembuat malware. Pembatasan executable dilakukan berdasarkan lokasi keberadaan executable file. Adapun asumsi yang digunakan pada penelitian ini adalah tidak adanya kepentingan pemakai untuk menjalankan executable selain dari folder %SYSTEMROOT% dan %ProgramFiles%.

2. METODE

Suatu file executable menyebabkan komputer untuk melakukan suatu tugas berdasarkan instruksi yang dikodekan, dan berbeda dengan suatu file data dimana harus dibentuk oleh suatu program menjadi berarti. Instruksi ini secara tradisional adalah instruksi-instruksi kode mesin untuk suatu CPU secara fisik. Walaupun anggapan secara umum, suatu file mengandung instruksi (seperti bytecode) untuk suatu software interpreter dapat juga dipertimbangkan sebagai executable; bahkan suatu file sumber bahasa scripting dapat dianggap sebagai executable.

Beberapa sistim operasi mengenali *file executable* berdasarkan nama ekstension (seperti .com, .exe, .dll, .cpl, .bin, .ocx, dan .sys) walaupun sebenarnya pada Windows, suatu executable dapat memiliki ekstension apa saja sehingga pengenalan terhadap executable pada Windows tidak dapat dilakukan hanya dengan pemeriksaan terhadap ekstension file, berbeda dengan sistim operasi unix maupun menyerupai unix menggunakan suatu catatan bersamaan dengan file didalam *metadata*-nya (seperti menandai suatu permisi execute).

Portable Executable Format *Portable Executable* (PE) adalah suatu format file untuk binari executable, kode object dan DLLs yang digunakan pada versi sistim operasi Windows 32-bit dan 64-bit. Format PE adalah suatu struktur data dimana membungkus informasi yang diperlukan oleh loader Windows OS untuk menangani kode executable yang terbungkus. Hal ini meliputi referensi dynamic library untuk linking, table API export dan import, data manajemen resource dan thread-local-storage (TLS). Pada sistim operasi Windows, format PE digunakan untuk EXE, DLL, SYS (device driver), dan tipe file eksekusi lainnya. Pengenalan terhadap executable dapat dilakukan dengan pemeriksaan keberadaan signature MZ dan PE pada PE header

Process Process adalah instant dari suatu program komputer yang sedang dieksekusi, walaupun program dan process nampaknya sama pada permukaan, mereka secara dasarnya adalah berbeda, suatu program adalah urutan statik dari suatu instruksi, sedangkan process adalah suatu kontainer dari suatu himpunan sumber daya yang digunakan oleh thread yang mana menjalankan instant dari program tersebut. Process pada windows terdiri dari:

1) Suatu *private virtual address space*, yang mana adalah suatu himpunan dari alamat memori virtual yang mana dapat digunakan oleh process.
2) Suatu program executable, yang mana mendefinisikan kode awal dan data yang dipetakan kedalam virtual address space dari process.
3) Suatu daftar dari handle yang terbuka untuk berbagai sumber daya sistim seperti semaphores, port komunikasi, dan file yang mana diakses oleh semua thread didalam process.
4) Suatu konteks security yang disebut sebagai suatu access token yang mana mengidentifikasikan user, security group dan privileges yang berasosiasi dengan process.
5) Suatu pengenal unik yang disebut sebagai process ID (secara internal disebut sebagai client ID)
6) Paling sedikit satu thread dari eksekusi. [b]

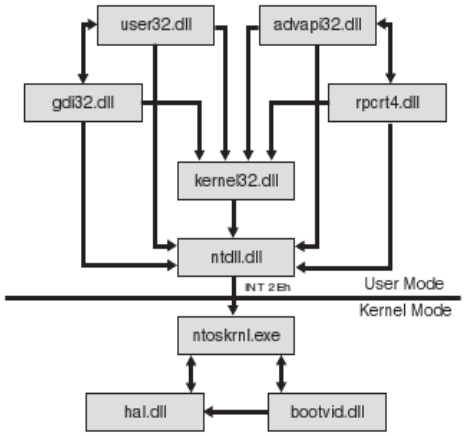
Pembuatan Process Suatu sub sistim process di Windows dibuat ketika suatu aplikasi memanggil satu dari fungsi pembuatan process pada user mode, seperti CreateProcess, CreateProcessAsUser,CreateProcessWithTokenW , atau CreateProcessWithLogonW yang merupakan fungsi pada Win32 library [c], Pembuatan proses pada Windows melalui beberapa tahapan dan berikut ini merupakan ringkasan dari tahapan tersebut:

1) Validasi parameter,
2) Membuka file image (.exe) yang akan dieksekusi didalam process.
3) Membuat object Windows executive process.
4) Membuat thread awal (stack, konteks, dan object Windows executive thread).
5) Membuat post-creation, inisialiasi proses Windows-subsystem-specific).
6) Memulai eksekusi pada awal thread-local
7) Didalam konteks dari proses baru dan thread, menyelesaikan inisialisasi dari address space (seperti memuat DLLs

yang diperlukan) dan memulai eksekusi dari program.

Karena sistim operasi Windows merupakan mikrokernel, maka pemakaian API CreateProses tersebut diatas pada user mode pada kenyataannya akan menggunakan berbagai native API yang disediakan pada kernel mode seperti: NTCreateFile(),NTCreateSection(),NTMapView OfSection(), NtCreateProcessEx, NTCreateThread.

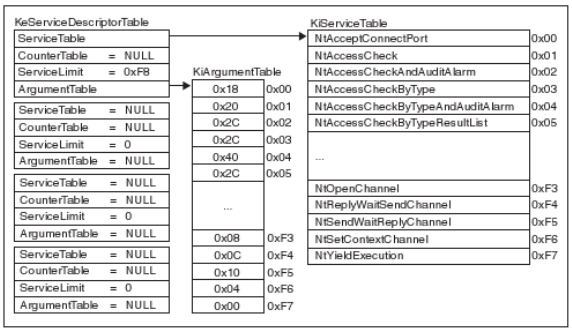
Kernel-Mode Untuk melindungi aplikasi user dari mengakses dan memodifikasi data sistim operasi yang kritis, Windows menggunakan dua modus akses processor, yaitu user mode dan kernel mode [4]. Kode aplikasi user berjalan pada user mode, sedangkan kode sistim operasi berjalan pada kernel mode. Kernel mode mengacu kepada suatu modus dari eksekusi didalam suatu processor dimana memberikan akses kepada semua memori sistim dan semua instruksi CPU. Dengan menyediakan kepada perangkat lunak sistim operasi suatu tingkat hak yang lebih tinggi daripada perangkat lunak aplikasi, processor menyediakan suatu dasar penting bagi perancang sistim operasi untuk memastikan bahwa suatu masalah pada aplikasi tidak menganggu stabilitas dari sistim secara keseluruhan, ntdll.dll merupakan sisi depan dari native api yang diimplementasikan pada ntoskrnl.exe, dan ntdll berfungsi agar sebagian dari native api tersedia bagi user mode melalui gerbang interrupt untuk beralih dari user mode ke kernel mode, dalam hal ini adalah INT 2EH sebagaimana ditunjukan pada Gambar 1.



Gambar 1. Ketergantungan modul dalam sistim

API Hooking Didalam pemrograman komputer, istilah hooking meliputi sejumlah teknik yang digunakan untuk mengubah atau menambah prilaku dari sistim operasi, aplikasi maupun komponen software lainnya dengan pencegatan terhadap pemanggilan fungsi atau message ataupun kejadiaan yang dilewatkan antar komponen software. Kode yang menangani pencegatan terhadap pemanggilan fungsi, kejadian atau message disebut sebagai sebuah "hook". Hooking digunakan untuk berbagai tujuan, meliputi debugging dan penambahan fungsi. Hooking dapat juga digunakan oleh kode jahat seperti rookit, suatu software yang membuat dirinya tidak kelihatan dengan memanipulasi keluaran dari pemanggilan API yang menunjukan keberadaannya.

Hooking dapat dilakukan secara fisik maupun runtime, secara fisik hooking dilakukan dengan modifikasi terhadap binari executable maupun library[5]. Perubahan secara runtime dilakukan terhadap process dari software, yang mana hal ini diperbolehkan di sistim operasi Windows maupun Linux. Hooking secara fungsi dilakukan dengan mengubah beberapa kode pada awal dari instruksi pada fungsi target untuk melompat kepada kode yang disuntikan, secara alternatif dapat dilakukan dengan konsep share library, tabel vektor interrupt, import descriptor table, dan SSDT.



Gambar 2. KiServiceTable direferensikan oleh KEServiceDescriptorTable

SSDT *System Service Dispatch Table* dikenal juga sebagai KiServiceTable merupakan suatu array yang menyimpan semua alamat *system service* yang menjadi mekanisme yang digunakan untuk mengarahkan suatu aliran eksekusi program ketika *system service* diminta. Suatu system service adalah secara fungsional disediakan oleh sistem operasi yang mana diimplementasikan didalam eksekutif seperti operasi file dan I/O lainnya, permintaan manajemen memori, dan konfigurasi manajemen operasi yang dikenal sebagai *native API*, secara singkat bahwa pada saat program user-mode perlu mengeksekusi suatu fungsi *native API*, mereka harus memiliki cara untuk transisi dari user-mode ke kernel mode. SSDT tersedia sebagai tabel pencarian bagi

sistim operasi dimana sebagai respon permintaan user mode ke system-service berdasarkan *dispatch ID*. Keseluruhan proses ini dikenal sebagai S*ystem Service Dispatching,* KiServiceTable direferensikan oleh KEServiceDescriptorTable sebagaimana ditunjukan pada Gambar 2.

```
typedef struct _SERVICE_DESCRIPTOR_TABLE {
SERVICE_DESCRIPTOR_ENTRY Descriptors[2];
} SERVICE_DESCRIPTOR_TABLE;

typedef struct _SERVICE_DESCRIPTOR_ENTRY {
PVOID KiServiceTable;
PULONG CounterBaseTable;
LONG ServiceLimit;
PUCHAR ArgumentTable;
} SERVICE_DESCRIPTOR_ENTRY;
```

Untuk melakukan interception terhadap suatu fungsi native API, dapat dilakukan dengan mengubah pointer fungsi pada array KiServiceTable[Dispatch] ID ke alamat fungsi tujuan, SSDT hook adalah mudah diinstalasi dan bersifat system-wide [d]. Dispatch ID untuk NtCreateProcessEx berdasarkan sistim operasi ditunjukkan pada Tabel 1.

Tabel 1. Dispatch ID untuk NtCreateProcess

Sistim Operisi	Nilai
WinNT 4	0x001f
Win2000	0x0029
WinXP	0x002f
Win2k3S	0x0031
WinVista	0x0046
Win7	0x009f

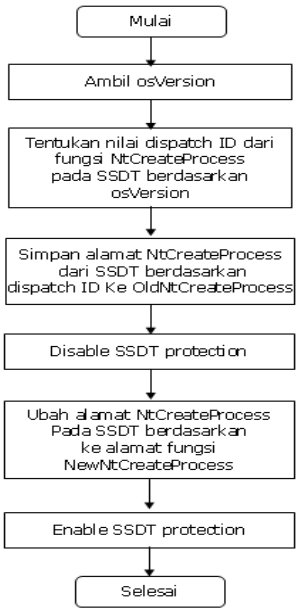
(sumber: MetaSploit)

3. DISKUSI

Adapun langkah-langkah pada penelitian ini adalah sebagai berikut:

1) Melakukan studi pustaka dengan menggambil referensi dari berbagai literatur, jurnal dan situs yang relevan yang membahas tentang executable, process dan pembuatan process pada sistim operasi Windows, fungsi-fungsi API yang berkaitan dengan pembuatan process, native api hooking, pembuatan device driver sistim dan instalasi device driver ke device-control-manager.
2) Merancang algoritma aplikasi pembatasan pengaktifan executable, struktur dari aplikasi dalam wujud aplikasi device driver sistim, dan pembuatan aplikasi instalasi sehingga menjadi solusi sesuai dengan tujuan penelitian.
3) Melakukan pengujian terhadap prototipe perangkat lunak atas berbagai teknik pengaktifan executable yang mungkin pada user mode, dan pengujian dengan berbagai malware yang melakukan eksploitasi fasilitas autorun maupun yang memanfaatkan kerentanan MS08-067 akan efektifitas hasil rancangan.

Untuk mendukung proses penelitian, penulis menggunakan Windows Driver Kit (WDK) untuk kompilasi komponen sistim driver, dan Dev-Cpp untuk kompilasi komponen installer.

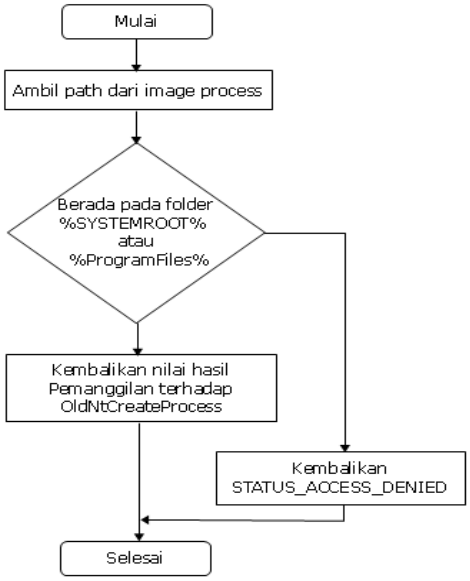


Gambar 4. Flowchart hooking terhadap native API NtCreateProcessEx

Algoritma Adapun proses pembatasan program executable dilakukan dengan melakukan hooking terhadap fungsi native API NtCreateProcessEx dengan melakukan modifikasi penunjuk fungsi pada SSDT, sehingga setiap pemanggilan terhadap fungsi tersebut akan dialihkan ke fungsi hook yang melakukan interception NewNtCreateProcessEx, algoritma proses hooking ditunjukan pada Gambar 4.

Pada fungsi NewNtCreateProcessEx akan melakukan pemeriksaan terhadap path image dari process executable yang akan dijalankan, jika image dari process executable berapa pada path folder %SYSTEMROOT% maupun %ProgramFiles%, maka fungsi akan mengembalikan hasil call terhadap OldNtCreateProcessEx, dan sebaliknya akan

mengembalikan status STATUS_ACCESS_DENIED, algoritma dari proses pembatasan executable ditunjukkan pada Gambar 5.



Gambar 5. Flowchart proses pembatasan executable

Rancangan Secara umum aplikasi pembatasan executable akan terdiri dari dua komponen perangkat lunak yaitu pertama adalah perangkat lunak device-driver (executeFilter.sys) yang berfungsi untuk melakukan hooking proses hooking dan proses pembatasan executable, dan yang kedua adalah perangkat lunak untuk menginstalasi device-driver ke Service Control Manager, sehingga device-driver akan dimuat dan dijalankan setiap kali startup sistim.

Komponen Device-Driver Sebagaimana program device-driver umumnya executeFilter.sys memiliki rutin sebagai berikut :

```
NTSTATUS OnStubDispatch(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp )
{
    Irp->IoStatus.Status = STATUS_SUCCESS;
    IoCompleteRequest(Irp,
    IO_NO_INCREMENT );
    return STATUS_SUCCESS;
}

VOID OnUnload( IN PDRIVER_OBJECT DriverObject )
{
//bagian ini otomatis dijalankan pada saat driver di unload
    Disable proteksi terhadap SSDT protection
    Ubah alamat NtCreateProcessEx pada SSDT ke OldNtCreateProcessEx
    Enable proteksi terhadap SSDT protection
}

NTSTATUS DriverEntry( IN PDRIVER_OBJECT theDriverObject,
IN PUNICODE_STRING theRegistryPath )
{
    //bagian ini otomatis dijalankan pada saat driver di load

    int i;
    theDriverObject->DriverUnload  = OnUnload;

    for(i=0;i< IRP_MJ_MAXIMUM_FUNCTION; i++ )
    {
    theDriverObject->MajorFunction[i] = OnStubDispatch;
    }

    Ambil osVersion
    Tentukan Dispatch ID berdasarkan osVersion
    Simpan alamat NtCreateProcessEx ke OldNtCreateProcessEx
    Disable proteksi terhadap SSDT protection
    Ubah alamat NtCreateProcessEx pada SSDT ke NewNtCreateProcessEx
    Enable proteksi terhadap SSDT protection

    return STATUS_SUCCESS;
}
```

Kemudian ditambahkan rutin NewNTCreateProcessEx yang berfungsi melakukan pemeriksaan terhadap executable.

```
NTSTATUS newNtCreateProcessEx(
  OUT PHANDLE ProcessHandle,
  IN ACCESS_MASK DesiredAccess,
    IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
  IN HANDLE ParentProcess,
  IN BOOLEAN InheritObjectTable,
  IN HANDLE SectionHandle OPTIONAL,
  IN HANDLE DebugPort OPTIONAL,
  IN HANDLE ExceptionPort OPTIONAL,
        IN DWORD saferPolicyMask
  )
{
NTSTATUS rc;

DbgPrint("NtCreateProcessEx\n");

Ambil Process Image Path berdasarkan SectionHandle
Periksa kesesuaian Path

Jika sesuai

rc = oldNtCreateProcessEx (
        ProcessHandle, DesiredAccess, ObjectAttributes,
        ParentProcess, InheritObjectTable,
        SectionHandle, DebugPort,
        ExceptionPort, saferPolicyMask);

Jika tidak
    rc = STATUS_ACCESS_DENIED;

        return rc;
}
```

Perangkat lunak device driver ini perlu dikompilasi dengan menggunakan n Windows Driver Kit (WDK) menjadi executeFilter.sys.

Komponen Installer Perangkat lunak installer berfungsi mendaftarkan executeFilter.sys ke Service Control Manager sebagai service auto start, berikut ini adalah rutin yang berfungsi melakukan proses pendaftaran.

```
int createService(char* serviceName, char* executablePath)
{
   printf("Creating service %s ", serviceName);

   SC_HANDLE sh = OpenSCManager(0, 0,
            SC_MANAGER_ALL_ACCESS);

   if(sh == INVALID_HANDLE_VALUE)
   {
      printf(" gagal buka handle.\n");
      return -1;
   }
   SC_HANDLE hService =
       CreateService(sh, serviceName, serviceName,
            SERVICE_ALL_ACCESS,
            SERVICE_KERNEL_DRIVER,
            SERVICE_AUTO_START,
            SERVICE_ERROR_NORMAL,
            executablePath,
            0, 0, 0, 0, 0);

   CloseServiceHandle(sh);

   if(hService == 0)
   {
      printf("Gagal buat service.\n");
      return -1;
   }

   printf("service berhasil didaftarkan);

   CloseServiceHandle(hService);

   return 1;
}
```

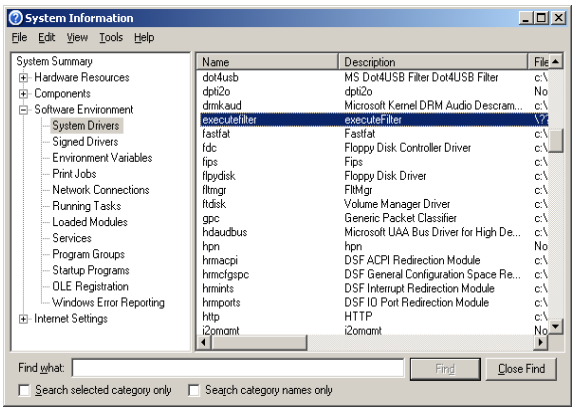
Perangkat lunak installer ini dapat dikompilasikan dengan Dev-cpp.

Pengujian

Pengujian dilakukan dengan menginstalasi komponen device-driver pada device-driver-manager dengan perintah :

installer load

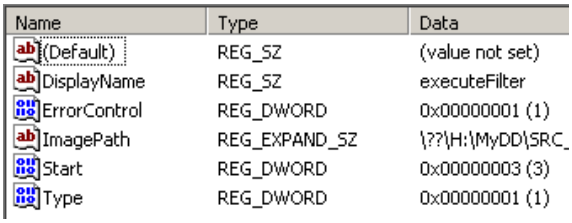
Sesaat setelah proses instalasi, maka keberhasilan proses instalasi dapat diperiksa pada fasilitas System Information yang tersedia pada Windows XP sebagaimana ditunjukkan pada Gambar 6.



Gambar 6. Hasil instalasi komponen executeFilter sebagai system-driver

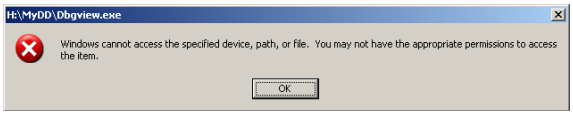
Atau dapat juga diverifikasi pada registry key sebagaimana yang ditunjukkan pada Gambar 7.

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\executeFilter



Gambar 7. Hasil instalasi pada registry key

Pengujian terhadap keberhasilan aplikasi pembatasan executable dilakukan dengan mencoba menjalankan program executable Dbgview.exe yang berada pada drive H:, dan tidak berhasil dijalankan dengan pesan sebagai berikut.



Gambar 8. Pesan penolakan akses executable file

Sedangkan pengujian dengan menjalankan Notepad.exe yang berada pada C:\Windows berhasil dijalankan.

4. HASIL

Pengujian dilakukan dengan berbagai pendekatan pengaktifan eksekusi file executable pada user mode terhadap file Dbgview.exe yang

ditempatkan di drive D, maupun dengan menggunakan USB flashdisk yang telah terinfeksi worms maupun virus yang melakukan eksploitasi terhadap autorun dan kerentanan MS08-067, dan hasil pengujian yang ditunjukan pada Tabel 2.

Tabel 2. Pengujian efektifitas aplikasi pembatasan executable

Metode pengaktifan executable	Berhasil
1. Double klik pada Dbgview.exe	Tidak berhasil dieksekusi
2. Double klik pada shortcut ke Dbgview.exe	Tidak berhasil dieksekusi
3. Mengetikan H:\DbgView.exe pada commmand prompt (cmd.exe)	Tidak berhasil dieksekusi
4. Pengaktifkan secara autorun.inf dengan flashdisk terinfeksi Win32.Brontok.Worms	Tidak berhasil dieksekusi
5. Pengaktifkan secara autorun.inf dengan flashdisk terinfeksi Win32.Sality.Virus	Tidak berhasil dieksekusi
6. Eksploitasi terhadap kerentanan MS08-067, melalui flashdisk terinfeksi Win32.Stuxnet.Worms	Tidak berhasil dieksekusi
7. Eksploitasi terhadap kerentanan MS08-067, melalui flashdisk terinfeksi Win32.Ratmit.Virus	Tidak berhasil dieksekusi
8. Dengan perintah Shell pada Visual Basic	Tidak berhasil dieksekusi
9. Dengan fungsi ShellExecute pada Shell32	Tidak berhasil dieksekusi
10. Dengan fungsi Winexec pada Kernel32	Tidak berhasil dieksekusi
11. Dengan fungsi CreateProcess pada Kernel32	Tidak berhasil dieksekusi

(Sumber: Pengujian oleh peneliti, 2011)

Berdasarkan hasil pengujian dan evaluasi yang dilakukan dapat disimpulkan bahwa aplikasi pembatasan executable dengan pendekatan hooking terhadap fungsi native API NtCreateProcessEx dapat secara efektif mencegah pengaktifkan executable dari path diluar dari yang ditentukan baik pengaktifan yang dilakukan oleh pemakai, maupun pengaktifan yang dilakukan dengan menggunakan fasilitas Autorun dan eksploitasi terhadap kerentanan MS08-067 yang dimanfaatkan oleh malware seperti W32.Brontok, W32.Sality, W32.Stuxnet dan W32.Ratmit.

5. DAFTAR PUSTAKA

[1] Symantec Corp., *Internet Security Threat Report*, Vol. 16., 2010

[2] Szor Peter, The Art of Cumputer Virus Research and Defense, Addison Wesley Professional, 2005

[3] G. Hoglund, J. Butler, *Rootkits: Subverting the Windows Kernel, Addison Wesley Professional*, ISBN: 0-321-29431-9, 2005.

[4] M. Russinovich, *Windows Internals 5th edition*., Microsoft Press, 2009.

[5] M. Jakobsson, Z. Ramzan, Crimeware: understanding new attacks and defenses, Addison-Wesley Professional, 2008.

Websites:

[a] F-Secure., Trojan-Dropper: W32/Stuxnet, http://www.f-secure.com/v-descs/trojan-dropper_w32_stuxnet.shtml, 2011.

[b] Wikipedia, Executable, http://en.wikipedia.org/wiki/Executable, 2011.

[c] A. Bassov, Hooking the native API and controlling process creation on a system-wide basis,http://www.codeproject.com/KB/system/soviet_protector.aspx, 2005.

[d] B. V. Doren (2006), Building and deploying a basic WDF Kernel Mode Driver, http://www.codeproject.com/KB/system/wdf_kmdf_basic.aspx, 2006.

Paper ID: 359

Title: APLIKASI PEMBATASAN PENGAKTIFAN EXECUTABLE DENGAN SYSTEM SERVICE DISPATCH TABLE (SSDT) HOOK
Student: F

Author 1 (CONTACT AUTHOR)
  Name: Hendra Soewarno
  Org: STMIK IBBI
  Country: Indonesia
  Email:hendra.soewarno@gmail.com
Author 2
  Name:
  Org:
  Country:
  Email:
Author 3
  Name:
  Org:
  Country:
  Email:
Author 4
  Name:
  Org:
  Country:
  Email:
Author 5
  Name:

Org:
Country:
Email:
Other Authors:
Contact Alt Email:
hendra.soewarno@gmail.com
Contact Phone: 081533113285
Keywords: malware infection prevention
Abstract: Antivirus technology uses patterns approach and heuristic approach to detecting the presence and malware attacks. The pattern approach has one step late to the malware existence, this approach is available when the malware samples can be obtained for pattern extraction, whereas heuristic approach runs into problems of false alarms and tend to disturb the comfort of the user. Nowadays, malware authors tend to use attacks that exploit zero-day vulnerabilities that are not known by the user or software creator. Based on data from Symantec that the highest mechanism for distributing malware in 2010 is through the file exchange using a USB flash media, exploitation of the AutoRun facility and the MS08-067 vulnerability. In this paper the authors would like to offer an approach to the malware infections prevention by limiting launch of executable files from the folder %SystemRoot% and %ProgramFiles% using kernel level SSDT API hooking.
Comments:
Paper: included