

Storm Worm Process Injection from the Windows Kernel

Danny Quist
dquist@offensivecomputing.net

Offensive Computing, LLC

March 9, 2008

Abstract

This paper will detail the analysis methods of W32/StormWorm.gen1 and show a process injection method it uses to run malicious code in user-space. This variant loads a driver into the kernel which then injects itself into the running services.exe process. The worm then connects to a P2P network sending spam, initiating DDoS from the infected computer. This technique does not use a packer in the traditional sense but a two-stage loader to inject itself into a running process from kernel space. I will show the decoding process and methods for extracting the true malicious code from the driver executable.

Introduction

The storm worm is one of the most prolific viruses ever written. It is a front-end for a variety of illicit activities including distributed denial of service, spam, and phishing. It has sustained a near constant presence on the Internet and currently accounts for approximately 2% of all spam.¹ The authors of the storm worm have used a variety of techniques to elude analysis. This article will cover one of the latest techniques that storm uses: Process injection from the Windows kernel. I will illustrate the analysis method used to harvest the injected process and show how to extract the working payload from the kernel code.

Analysis

Finding a copy of this worm was not difficult. I simply had to open up my spam folder inside of Gmail and search for the word “ecard” in the subject line. The resulting page contained an automatic download which saved the file on my hard disk and waited for me to click the link. I downloaded the file inside of my Windows XP Vmware image and started analysis. I’ve uploaded the file to Offensive Computing.²

The first method of analysis I used was iDefense’s system call monitoring tool called Sysanalyzer.³ This is a good tool to use for determining what a particular sample is doing. Unfortunately results from this tool were very limited. The only information provided was that a file was dropped called C:\Windows\System32\diperto4417-e33.sys⁴. Looking at the file inside of IDA Pro did not reveal any useful information either. Filemon showed that another file diperto.ini was written and contained. According to analysis from other researchers⁵ this contains a list of Overnet⁶ P2P nodes to connect to.

Analysis of diperto4417-e33.sys

The next technique was to load the device driver into IDA and see what could be learned. This is where things got more interesting. The driver was not packed at all and analysis was possible without too much trouble. The file was not encrypted or packed and there were only 14 function calls.

¹ <http://asert.arbornetworks.com/2008/02/mega-d-botnet-or-mega-confusion/>

² <http://www.offensivecomputing.net/?q=ocsearch&ocq=591258adc48b422c86730214aef81989>

³ <http://labs.iddefense.com/software/malcode.php>

⁴ <http://www.offensivecomputing.net/?q=ocsearch&ocq=f75ced55ddf2005bf949a35534057887>

⁵ F-Secure, http://www.f-secure.com/v-descs/email-worm_w32_zhelatin_tq.shtml

⁶ <http://en.wikipedia.org/wiki/Overnet>

Before the xor decoding, the data at dword_10A40 is obviously garbage:

```
.data:00010A40 dword_10A40 dd 0D242889Fh
.data:00010A40
.data:00010A44 db 0D1h ; -
.data:00010A45 db 0D2h ; -
.data:00010A46 db 0D2h ; -
.data:00010A47 db 0D2h ; -
.data:00010A48 db 0D6h ; +
.data:00010A49 db 0D2h ; -
.data:00010A4A db 0D2h ; -
.data:00010A4B db 0D2h ; -
.data:00010A4C db 2Dh ; -
.data:00010A4D db 2Dh ; -
```

Figure 3: dword_10A40 before decoding

The next step is to attach to the process space and allocate memory. KeAttachProcess and ZwOpenProcess are used to prepare access to the process's memory. The function sub_105DC handles the insertion of code into the running process. Decoding the payload is addressed later.

To execute code inside of the process, the undocumented asynchronous procedure call API is used. These calls are typically used to handle a completed I/O request from a device driver. Callback code can be registered to handle completed I/O events. In the case of the Storm worm, the decoded memory is created inside the user process space and then registered as a NormalRoutine inside of the APC data structure. This callback code is executed in the context of the userspace process instead of the kernel. There are many sources of documentation^{7 8 9} of this attack and it illustrates the future of these attacks.

Decoding the Payload using x86emu

There are many methods to decode the payload data. The first is to write a script inside of IDA using any number of methods. (IDC scripting, IDAPython, etc.) Second you can manually decode the data by hand manually. Given that this is a large 122k file, it is best to let a tool perform the decoding for you. Another option you can use is the excellent x86emu¹⁰ tool by Chris Eagle. X86emu partially emulates an Intel instruction set which allows you to run small portions of the code. This was the method I chose to decode the data.

The goal for decoding this portion of the payload is to extract the data that is being injected into the services.exe application. X86emu helps to ease this process. The first step is to install x86emu using the

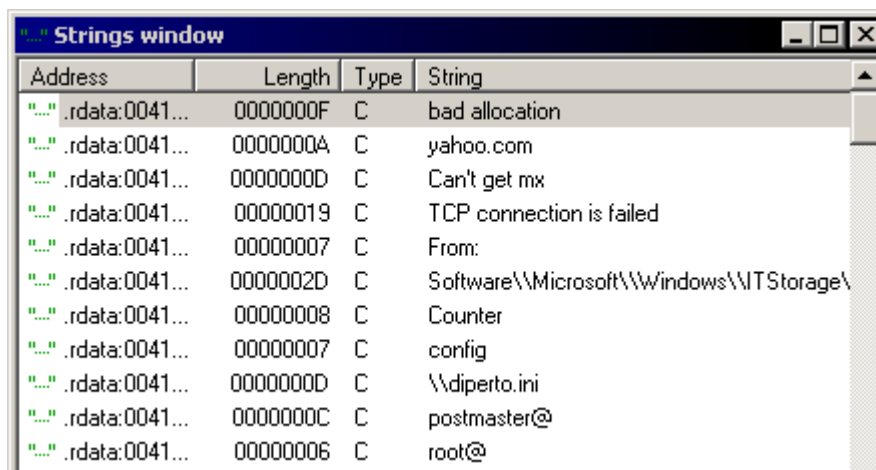
⁷ http://www.virusbtn.com/pdf/conference_slides/2007/BaumgartnerVB2007.pdf

⁸ <http://www.ddj.com/windows/184416590>

⁹ B. Jack, "Step Into Ring 0", Blackhat USA 2005

¹⁰ <http://ida-x86emu.sourceforge.net/>

starting at the address specified. To dump the contents of the memory simply take the starting address, 0x00010A40 and add the size 122,880 to it (0x1E000). This will yield the address of 0x0002EA40. Provide a filename and you can get a copy of the code that is injected into service.exe. This executable, when loaded into IDA, yields the actual program with good strings.



Address	Length	Type	String
"..." .rdata:0041...	0000000F	C	bad allocation
"..." .rdata:0041...	0000000A	C	yahoo.com
"..." .rdata:0041...	0000000D	C	Can't get mx
"..." .rdata:0041...	00000019	C	TCP connection is failed
"..." .rdata:0041...	00000007	C	From:
"..." .rdata:0041...	0000002D	C	Software\Microsoft\Windows\ITStorage\
"..." .rdata:0041...	00000008	C	Counter
"..." .rdata:0041...	00000007	C	config
"..." .rdata:0041...	0000000D	C	\diperto.ini
"..." .rdata:0041...	0000000C	C	postmaster@
"..." .rdata:0041...	00000006	C	root@

Figure 6: Strings from dumped payload

Conclusions

The methods used by storm worm represent the latest advances in malware. The kernel payload method is a useful mechanism to subvert analysis and make reverse engineering more difficult. The sophistication of evasion tactics is increasing and will require further innovation to be able to maintain automated analysis techniques. In many cases traditional packers are being replaced with simpler encoding techniques combined with more complicated subversion methods.