Botnet Analysis - I

Amit Malik

www.SecurityXploded.com

Disclaimer

The Content, Demonstration, Source Code and Programs presented here is "AS IS" without any warranty or conditions of any kind. Also the views/ideas/knowledge expressed here are solely of the trainer's only and nothing to do with the company or the organization in which the trainer is currently working.

However in no circumstances neither the Trainer nor SecurityXploded is responsible for any damage or loss caused due to use or misuse of the information presented here.

Acknowledgement

- Special thanks to **Null** community for their extended support and co-operation.
- Special thanks to **ThoughtWorks** for the beautiful venue.
- Thanks to all the trainers who have devoted their precious time and countless hours to make it happen.

Advanced Malware Analysis Training

This presentation is part of our **Advanced Malware Analysis** Training program. Currently it is delivered only during our local meets for FREE of cost.

For complete details of this course, visit our Security Training page.

Who am I?

Amit Malik

- Member, SecurityXploded
- Security Researcher, McAfee Labs
- Reversing, Malware Analysis, Exploit Analysis/Development etc.
- E-mail: m.amit30@gmail.com

Content

- Bots and Botnets
- Important point #1 (Registers etc.)
- Important point #2 (Procedure calls etc.)
- Important point #3 (code injection etc.)
- Case Study: Waledac Botnet
  - Waledac C&C communication
  - Rapid Reversing Techniques
  - Waledac Analysis (C&C)
  - Summary

Bots and Botnets

- What is a Bot and Botnet?
- Commercialized and Infrastructure based malware class.
- C&C – Command and Control
- Centralized C&C communication
- Distributed C&C communication (e.g: P2P)
- Some Popular Bots
  - Kehlios, Carberp, Zeus, Nittol, Alureon, Maazben, Waledac etc.

Important point #1

- Pay very close attention to the following registers in the code.
  - Fs:[0]
  - Fs:[18]
  - Fs:[30]
- When function returns check EAX register
- Check ECX when you are in loop
- ESI and EDI locations during string copy ops.

Fs:[0] (SEH)

- Fs:[0] holds SEH (Structured Exception Handler) address
- Some malicious programs abuses SEH in order to fool novice reverse engineer.
  - Example code

    Mov eax, some address

    Push eax

    Push dword ptr fs:[0]

    Mov fs:[0], esp ; EAX addr now becomes SEH.

    Some invalid instruction that generate an exception and transfer the execution to the address (which was in EAX).

Fs:[18] and Fs:[30]

- Fs:[18] holds TEB (Thread Environment Block) Address
  - From TEB structure we can access stack base, stack end and PEB address.
  - Use **dt nt!_TEB –r1** command in windbg to see the TEB structure.
- Fs:[30] holds PEB (Process Environment Block) Address
  - From PEB we can access some key information like process heaps, loaded modules, some debugger detection flags.
  - Use **dt nt!_PEB –r2** command in windbg to see the PEB structure

Fs:[18] and fs:[30] example codes

- Loaded modules enumeration (Extremely important) – IAT rebuilding, get function addresses etc.

```
push dword ptr fs:[30h]  ; PEB
pop eax
mov eax,[eax+0ch]  ; LDR
mov ecx,[eax+0ch]  ; InLoadOrderModuleList
mov edx,[ecx]
push edx
mov eax,[ecx+30h]
```

- Debugger detection

```
MOV EAX,FS:[18]                  TEB
MOV EAX,[EAX+30]                 PEB
MOVZX EAX,BYTE PTR [EAX+2]       BeingDebugged Flag
RET
```

Important point #2

- Asynchronous procedure calls (APC)
  - IRQL (Interrupt Request Level) 1
  - Used by malicious softwares to fool beginner reverse engineers
  - WriteFileEx, ReadFileEx, SetWaitableTimer,
  - SetWaitableTimerEx, QueueUserAPC API can be used to execute code via APC
  - http://msdn.microsoft.com/en-us/library/windows/desktop/ms681951(v=vs.85).aspx

E.g. ReadFileEx

- Reads data from the specified file or input/output (I/O) device. It reports its completion status **asynchronously**, **calling the specified completion routine** when reading is completed or cancelled and the calling thread is in an **alertable wait state**.
- BOOL WINAPI ReadFileEx(_In_ HANDLE hFile, _Out_opt_ LPVOID lpBuffer, _In_ DWORD nNumberOfBytesToRead, _Inout_ LPOVERLAPPED lpOverlapped, **_In_opt_ LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine**);

Role of SleepEx and Similar APIs

- Suspends the current thread until the specified condition is met. Execution resumes when one of the following occurs:
  - An I/O completion callback function is called.
  - An asynchronous procedure call (APC) is queued to the thread.
  - The time-out interval elapses.
  - SleepEx, SignalObjectAndWait,MsgWaitForMultipleObjectsEx, WaitForMultipleObjectsEx, or WaitForSingleObjectEx
- DWORD WINAPI **SleepEx**(_In_ DWORD dwMilliseconds, **_In_ BOOL bAlertable**);
- If the parameter is TRUE and the thread that called this function is the same thread that called the extended I/O function (**ReadFileEx** or **WriteFileEx**), the function returns when either the time-out period has elapsed or when an I/O completion callback function occurs. If an I/O completion callback occurs, the I/O completion function is called. If an APC is queued to the thread (**QueueUserAPC**), the function returns when either the timer-out period has elapsed or when the APC function is called.

Important point #3

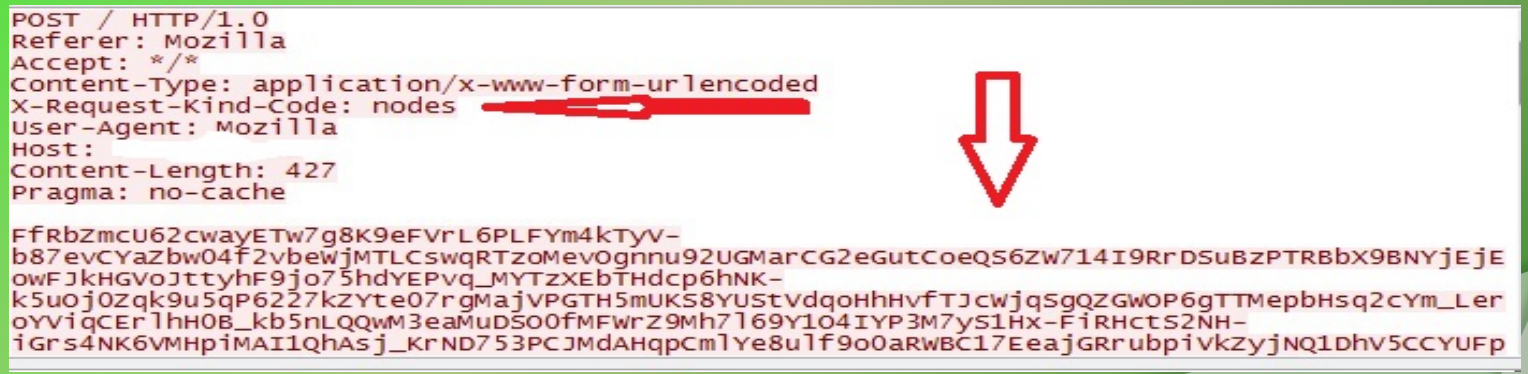
- Code Injection – Very well explained in our couple of articles, check article section at securityxploded.
- Malwares often inject code to some legitimate processes to bypass host based security solutions or to minimize the footprints
- Injected code can be direct executable code or can be a DLL(Dynamic Link Library).
- OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, NtCreateThreadEx, ZwCreateThread, APC APIs
- Services – CreateService, StartService.
- New Process – CreateProcessA/W, ShellExecute/Ex, WinExec etc.
- **Process Hollowing and variants, In-Memory execution, Packers etc.

Case Study: Waledac botnet

- Popular e-mail spam botnet
- Taken down by microsoft in march 2010
- Around 90,000 computers were infected
- Capable to send 1.5 billion spam messages a day. (1% of the total global spam volume)
- http://en.wikipedia.org/wiki/Waledac_botnet
- In this presentation we will focus only on C&C communication

Waledac C&C Communication

- Bzip, AES, base64 encryption/encoding algorithms used in the network communication
- Custom header field in HTTP protocol
- P2P style communication over HTTP protocol



Waledac C&C Communication

- Ok, We have enough information to start our analysis
- Now our task is to
  - Indentify which functions are responsible for encryption
  - Identify which function is responsible for sending the request
  - Identify what sent into the request (Clear Text)
- **Challenge**
  - Lots of code (around 950 KB, ~14 MB IDB binary)
  - More than 4,000 functions
  - Completely unaware about the malware behaviour (code based)

Rapid Reversing Techniques

- Tracing - most common approach
  - Instruction based tracing
    - Pros
      - Complete flow including branching /decisions made by program during execution
      - Powerful, given time and space provides solid results
    - Cons
      - Very slow (each instruction logging)
      - Huge output, difficult to analyze
      - For huge code base nearly impossible
  - Functions call based tracing
    - Pros
      - Functions call flow during execution
      - Provides fair details about the binary
    - Cons
      - Not very helpful in malware analysis (packers, run rime executable code generation etc.)

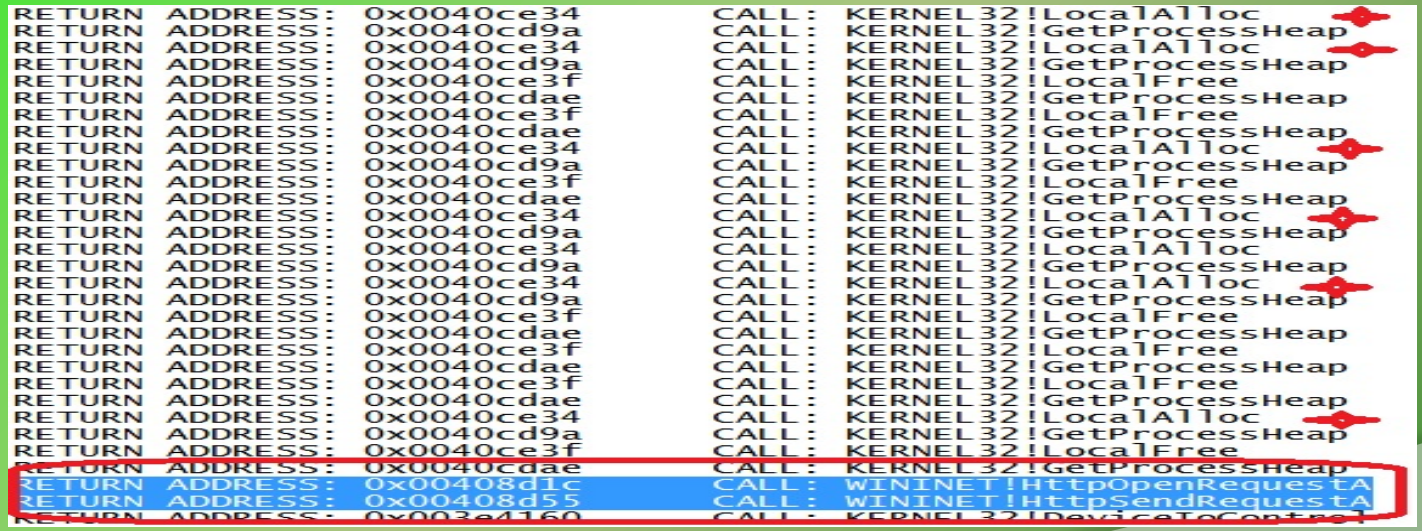
Rapid Reversing Techniques

- API call based tracing
  - Pros
    - Can log all API calls even from newly allocated regions i.e. run rime unpacked code or run rime generated code
    - Provides good details about the malware and system interaction
    - Excellent for malware analysis, more advanced and sophisticated techniques can be used like API call clustering, API call based execution path diffing, Automated unpacking etc.
  - Cons
    - Efficiency and flexibility is based on the actual tracing software otherwise it can be very painful like instruction based tracing (hook hopping, API calls in big loops etc.)

- These techniques are often used with conditions after a specific point/address

Waledac Analysis (C&C)

- Our encryption routines should be near this code segment. We will look into function call flow of the malware upto this code so that we can get better understanding of the nearby code.

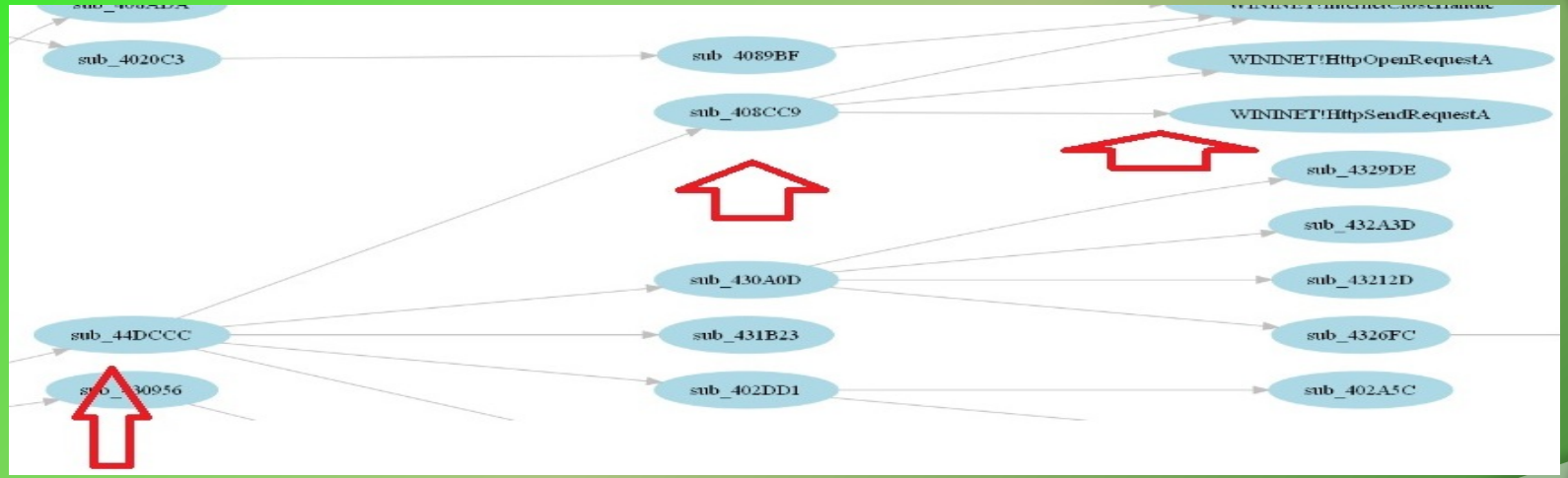


Continue...

- In snapshot we see following things
  - Calls to LocalAlloc API (Another point confirming encryption routines near the code)
  - Call to HttpSendRequestA API (send the encrypted payload to C&C server)
- Now identify the functions that are calling the above APIs, How ?
- Static cross references doesn't provide much details so we will use tracing.
- ** I am not going to share my tracing tools so please don't make request for them ☺.*

Continue...

- Below snapshot is from single shot function call (and partial API) tracing of the malware.



- We can mark sub_44DCCC as start point, because sub_408CC9 is directly calling our APIs.

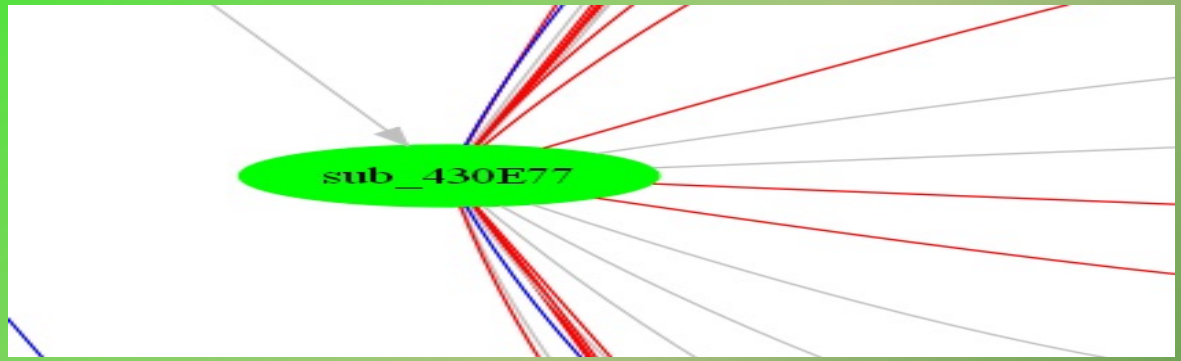
Tracing Function SUB_44DCCC

- Full function and API trace of the function sub_44DCCC.

- Inside sub_44DCCC we see an another call (zoom-next slide) to a function which seems to hold 60% of code for sub_44DCCC and also calls LocalAlloc API, so that is our function.

Important Function

- Put breakpoint on this function, run, put breakpoint on LocalAlloc, use single stepping (F8) and wait for LocalAlloc call.



- Monitor LocalAlloc returned pointer, and keep single stepping on. Eventually you will see the answer of all the questions.
- So pretty simple huh?? Try it yourself. ☺

C&C Payload (Clear Text)

- Waledac uses xml style in its payload as we can see in the below snapshot.

```
<lm><localtime>1357545840</localtime><nodes><node ip="83.254.252.65" port="80"
time="1357545822">ae1a45118c45c751531ea25fcd7638709f16213d69</node><node ip="81.172.5.183" port="80"
time="1357545815">a60aed60e315cc48220f9c61a7714f0370406d4c46</node><node ip="98.14.4.41" port="80"
time="1357545807">2e15c83c76192b5c4050162c426ae10c0b572e46d2</node><node ip="93.156.40.233" port="80"
time="1357545806">356d1f27524591514058236215 4a19237040751eb8</node><node ip="66.171.19.226" port="80"
time="1357545768">be413a5f5c2fe041874b2c4de6056d16aa19ab02ef</node></nodes></lm>
```

- However to know the meaning and purpose of this payload we need more analysis and threat intelligence, but most of the time it will suffice. ☺

Demonstration

https://vimeo.com/57755964

Summary

- Complete analysis and threat report can take weeks to months, depending on the threat.
- Tracing methods are extremely helpful in large code base malwares. Actually tracing is much more powerful, think about taint analysis.
- Reverse Engineering Automation is a fascinating and useful study area.
- VERA (ether dependent), Zynamics BinNavi (not very useful for malware analysis) etc. can be used to accelerate reverse engineering but they are not very flexible. (inclusion, exclusion, trace from one node to another node etc.)
- I know it's a lots of load on your brain but try to understand the malware yourself.
- Malware MD5 : 0A0E0AD7175B17A5D6AFFC73279BDA8B

Single Shot Function Call Trace

- In our analysis we discussed only the marked area in this snapshot.

- This trace still not covering entire malware calls (because c&c is down)

Reference

Complete Reference Guide for Advanced Malware Analysis Training

[Include links for all the Demos & Tools]

Thank You !

www.SecurityXploded.com