Analyzing Malware

Vlad Grigorescu

April 8, 2010

Office of Privacy and Information Assurance
University of Illinois at Urbana-Champaign

Contents

Overview

Project Outline

The goals of the project were to:

- Capture a large set of malware that was active in the wild
- Analyze one or two of the more interesting samples
- Document the control mechanism
- Look for vulnerabilities that would allow further research into the botnet

Results

Project Results

Two samples analyzed: an SDBot variant, and a Zeus trojan. The following documentation was created for the SDBot variant:

- A full list of the bot's commands
- How the owner is authenticated
- How the bot is controlled
- How the bot spreads
- How the bot can be detected/removed

Utility

Usefulness of Results

The collected data is useful for several reasons:

- Improved detection
- Better defending
- Learning about a virus' lifecycle
- Detecting botnets and controllers
- Subverting control mechanisms

Capture Options

Possible methods

- Honeypot
- Physical/virtual machine
- User machines
- Malware collections

Capture Options

Possible methods

- Honeypot
- Physical/virtual machine
- User machines
- Malware collections (`http://offensivecomputing.net`)

Nepenthes

Features

- Low interaction honeypot
- Handlers for 30+ common exploits
- Won't execute the malware - only download it
- Easy to write custom handlers
- Automatic submission to Norman Sandbox for analysis
- Low system resource use; no separate machine needed.

Nepenthes

Drawbacks

- Covers small surface area
- Captured binaries are usually well-known
- Initial infection stage only

Nepenthes

Setup

- No firewall
- Server in a datacenter
- Two IPs
- IPv6
- Fast connection

Nepenthes

Results

- Ran for 3 weeks
- 165,810 exploit attempts
- 273 unique samples
- 597 unique IPs

First Steps

Running the Bot

- In a safe environment, run the bot
- "MITM" attack
- Set up firewall rules
- Sniff its traffic
- Create a client and server emulator

Reverse Engineering

Unpacking the Bot

- Most bots are "packed" to avoid detection
- Anti-debugging features
- First step: unpack and disable anti-debugging protection
- Memory dump "cleaned" binary

Reverse Engineering

Knowing Assembly

Very little assembly knowledge is needed

- Calls to DLLs
- Strings
- Graphical views
- You can always just run it

Reverse Engineering

Extracting the Commands

- Command names were hashed
- Horrible hashing algorithm
- Brute-force attack
- Most names recovered this way

Reverse Engineering

Interesting Commands

- botfind/botkill
- encrypt/decrypt
- flashfxp/iexplore
- interesting
- ipv6toipv4
- packetsniff
- wormride

Zeus

Hooking

- Hides itself
- A/V and firewall protection
- Internet Explorer/Firefox hooks
- Usually in combination with MebRoot/Torpig