

# Lecture 21

## November 7, 2022

# Bots

- *bot*: malware that carries out some action in co-ordination with other bots
- *botmaster*: attacker controlling the bots on one or more systems
- *command and control (C&C) server, mothership*: system(s) the attacker uses to control the bots
- *C&C channels*: communication paths used to communicate with bots
  - Distinguishing characteristic of bot is the use of this channel
  - Can be triggered, updated over this
- *botnet*: a collection of bots

# Life Cycle of a Bot in a Botnet

1. Bot infects system
2. Bot checks for a network connection, looks for either C&C server or another bot it can communicate with
3. Bot gets commands sent by C&C server or other bot
  - These may include adding components to add to what the bot can do
4. Bot executes these commands
  - May send results to somewhere else

# Organization of a Botnet

- *Centralized*; each bot communicates directly with C&C server
  - Potential problem: C&C server can become a bottleneck
- *Hierarchical*: C&C server communicates with set of bots, which in turn act as C&C servers for other bots, in a hierarchy
  - Good for controlling large botnets such as Torpig (over 180,000 bots) and Mirai (estimated to have 493,000 bots)
- *Peer-to-peer*: no single C&C server; bots act as peers, and botnet is a peer-to-peer network
  - High latency; to join the botnet, a bot scans addresses until it finds another bot, then forwards message to it

# Example: GTBot (Centralized)

- One of the earliest bots; used IRC channel as C&C channel
- Get it onto a Windows system running mIRC client
  - Installed scripts to monitor the channel looking for specific keywords
  - Also program to hide the bot, and possibly other programs to propagate the bot to other systems and install servers on them
- One instance (Backdoor.IRC.Aladinz) installs and hides itself, then notifies attacker (via IRC channel) it is resident
  - This has lots of tools to launch attacks

# Example: Torpig (Hierarchical)

- Based on Mebroot rootkit
- Bot installed itself so it is executed at boot, *before* the operating system loaded
- Then contacted Mebroot C&C server to get modules to add to bot
  - Compromises sensitive data like passwords
  - Data sent to Torpig C&C server
- Server acknowledges upload
  - Can also send file with addresses of backup Torpig C&C servers, and how often the bot should contact the server

# Example: Trojan.Peacomm (Peer-to-Peer)

- Bot gets on Windows system
- Uses peer-to-peer protocol to connect to peers
  - Addresses of at least 100 peers stored in bot
- Looks for a value encoding a UTL pointing to another component
  - Downloads and executes this component
  - Many possible component, including one for spamming, one for reaping email addresses for spam, one to launch a distributed denial of service attack

# Content Delivery Networks and IP Flux

- Content delivery networks (Netflix, Amazon, etc.) have many servers
  - These are invisible to the client
  - When DNS gets target system, it returns IP address to client
- As these networks grew, needed way to prevent any single server from being overloaded
- Mechanism is to change IP address associated with a particular host name over a very short period of time (called *IP flux*)
  - So, while traffic appears to go to one particular system, it is actually sent to whichever server has the address currently



# Applying This to Botnets

- Goal: make it harder to locate bots
  - Especially C&C servers
- Approach: associate list of IP addresses with host name
- *IP flux botnet*: change binding between host name (in bot) and IP address, going down the list (*flux*)
  - *Single flux botnet*: list of IP addresses is large; host name is registered to one of these addresses, and after a short time deregistered and reregistered using a different IP address
  - *Double flux botnet*: do the same with IP addresses of the DNS servers used to look up the IP addresses
- Also called *fast flux botnets*

# Example: Flame (Fast Flux Botnet)

- Bot gathers information from infected system, network
  - This is then retrieved over C&C channel to Flame server
  - Server can also send new modules for bot
- When Flame installed:
  - Check for network by trying to connect to well-known servers; assume successful
  - Flame had 5 domains initially for the Flame server; Flame server could add more to this list
    - Communication over SSL; Flame's certificate was self-signed
  - Preliminary analysis: more than 50 host names and more than 15 IP addresses related to C&C messages; later raised number of C&C hosts to around 100

# Variation: Domain Flux

- Instead of holding host name fixed and changing the associated IP address, change host name and keep associated IP address fixed
  - Called *domain flux*
- Advantage: host name associated with C&C server for a short period of time
  - With IP flux, finding host name identifies C&C server
- Example: Torpig
  - Compute a host name that is fixed for current week number and year; append “.com”, “.net”, “.biz” suffixes and probe for each
  - If no luck, generate name based on current day, append suffixes, probe for each
  - If no luck, go to fixed list of host names

# Rabbit, Bacterium

- A program that absorbs all of some class of resources
- Example: for UNIX system, shell commands:

```
while true
do
    mkdir x
    chdir x
done
```

- Exhausts either disk space or file allocation table (inode) space

# Logic Bombs

- A program that performs an action that violates the site security policy when some external event occurs
- Example: program that deletes company's payroll records when one particular record is deleted
  - The “particular record” is usually that of the person writing the logic bomb
  - Idea is if (when) he or she is fired, and the payroll record deleted, the company loses *all* those records

# Adware

- Trojan horse that gathers information for marketing purposes and displays advertisements
  - Often selects ads to display based on gathered information
- Believed to have originated with a company announcing it would make its software available for free, because it would pop up window advertising company
  - Benign as user had to opt in
  - Spread through distribution of program only

# Types of Behavior

- *Low severity behavior*: just display ads, don't transmit information
- *Medium severity behavior*: transmits information deemed low risk, such as location information, and may display ads based on this
- *High severity behavior*: transmits personal information, and displays ads tailored to devices, people with those characteristic
  - Typically very aggressive (annoying)
  - Sometimes called *malware*

# Getting Adware On a System

- Put on a web site user visits
  - Put it in a banner enticing the user to click on it; this installs the adware
  - Page may require user to install software to view parts of web site; software contains adware
  - If page refreshes automatically, it may direct browser to run an executable
    - Usually browser notifies user via a dialog box that may require a click; on click, program runs and installs adware
  - Some browser plug-ins download, execute files automatically; there may be no indication of this
    - Called *drive-by downloading*



# Getting Adware on a System

- Put into software that user downloads
  - Very common with mobile apps
- Problem: app asks for permission to carry out its tasks
  - Some may be unnecessary; often hard for users to minimize permissions set
  - Thus app may have access to camera, microphone, and may be able to make calls without going through dialing interface — and user does not realize this
- Example: survey of 900 Android apps
  - 323 had unnecessary permissions

# Economic Motives

- Used to target ads that use is most likely to respond to
- Purveyors get money for every ad displayed or clicked on
  - Web site owners display ads on their sites
  - Developers put adware libraries in their apps
  - Others take apps, modify them to include adware, and put them on unauthorized app stores

# Spyware

- Trojan horse that records information about the use of a computer for a third party
  - Usually results in compromise of confidential information like keystrokes, passwords, credit card numbers, etc.
  - Information can be stored for retrieval or sent to third party
- Put on a system the way any other malware gets onto system

# Example: Pegasus

- Designed for Apple's iPhone, attacker sends URL to victim who clicks on it, triggering attack that tries to gain control of iPhone
- First sends HTML file exploiting vulnerability in WebKit
  - Basis for Safari and other browsers
- This downloads software to gain control of iPhone
  - Software enciphered with different keys for each download
  - Includes a loader for the next stage
- Loader downloads dynamic load libraries, daemons, other software and installs Pegasus
  - If iPhone has previously been jailbroken, removes all access to the iPhone provided by the earlier break

# Example: Response to Pegasus

- Apple developed patches for the vulnerabilities exploited
  - Deployed them in update to iPhone's operating system, iOS
- Discovered when human rights activist received text messages with a suspicious link
  - Sent messages to Citizens Lab
  - Citizens Lab recognized links were associated with a manufacturer of spyware for government surveillance
  - Lookout carried out technical analysis

# Ransomware

- Malware inhibiting use of computer, resources until a ransom is paid
  - Ransom is usually monetary, and must be paid through some anonymous mechanism (BitCoin is popular)
- PC CYBORG (1989) altered AUTOEXEC.BAT to count number of times system was booted; on 90<sup>th</sup>, names of all files on main drive (C:) enciphered and directories hidden
  - User told to send fee to post office box to recover the system
- CryptoLocker (2013) encrypted files and gave victim 100 hours to pay ransom; if not, encryption keys destroyed
  - Used evasive techniques to make tracking more difficult
  - Spread via email as attachments

# Example Protocol

Goal: Angie wants to extort money from Xavier

- Angie generates asymmetric key pair; embeds public key in malware
  - She retains private key
- Malware infects Xavier's system
  - Generates symmetric key and uses that to encipher target data
  - Enciphers symmetric key with public key, erases all instances of symmetric key
  - Xavier sees message saying he needs to do something for Angie (usually send money); he does so and includes the encrypted symmetric key
- Angie then deciphers encrypted symmetric key with her private key, returns it to Xavier

# Phishing

- Act of impersonating legitimate entity in order to obtain information such as passwords or credit card information without authorization
  - Usually a web site associated with a business
- Usual approach: craft a web site that looks like the legitimate one
  - Send out lots of email trying to persuade people to go to that web site
  - Copy their login, password, and other information for later use
- More vicious attack: fake web site passes data on to real web site, and sends replies back to victim
  - Man-in-the-middle attack



# Example

- Heidi banks at MegaBank, with URL of <https://www.megabank.com>
- She receives a letter saying she needs to check her account for possible fraudulent activity
- Email includes link
  - Link is visible as [www.megabank.com](http://www.megabank.com)
  - But link actually connects to <https://www.megabank.crookery.com>
- Attacker records name, password, then give error
  - If very clever, client is redirected to actual bank's home page

# Spearphishing

- Phishing attack tailored for particular user
- Used to attack specific (types of) users to obtain information
- Example: some employees of a major cybersecurity company received email called “2011 Recruitment Plan”
  - They opened an attacked spreadsheet
  - This exploited a vulnerability in a supporting program to install a backdoor so attackers could control system remotely
  - Attackers used this as a springboard to compromise other systems in the company’s network, and ultimately stole sensitive information
  - Embarrassment, financial costs of recovery large

# Defenses

- Scanning
- Distinguishing between data, instructions
- Containing
- Specifying behavior
- Limiting sharing
- Statistical analysis

# Scanning Defenses

- Malware alters memory contents or disk files
- Compute manipulation detection code (MDC) to generate signature block for data, and save it
- Later, recompute MDC and compare to stored MDC
  - If different, data has changed

# Example: *tripwire*

- File system scanner
- Initialization: it computes signature block for each file, saves it
  - Signature consists of file attributes, cryptographic checksums
  - System administrator selects what file attributes go into signature
- Checking file system: run *tripwire*
  - Regenerates file signatures
  - Compares them to stored file signatures and reports any differences

# Assumptions

- Files do not contain malicious logic when original signature block generated
- Pozzo & Grey: implement Biba's model on LOCUS to make assumption explicit
  - Credibility ratings assign trustworthiness numbers from 0 (untrusted) to  $n$  (signed, fully trusted)
  - Subjects have risk levels
    - Subjects can execute programs with credibility ratings  $\geq$  risk level
    - If credibility rating  $<$  risk level, must use special command to run program

# Antivirus Programs

- Look for specific “malware signatures”
  - If found, warn user and/or disinfect data
- At first, static sequences of bits, or patterns; now also includes patterns of behavior
- At first, derived manually; now usually done automatically
  - Manual derivation impractical due to number of malwares

# Example: Earlybird

- System for generating worm signatures based on worm increasing network traffic between hosts, and this traffic has many common substrings
- When a packet arrives, its contents hashed and destination port and protocol identifier appended; then check hash table (called *dispersion table*) to see if this content, port, and protocol have been seen
  - If yes, increment counters for source, destination addresses; if both exceed a threshold, content may be worm signature
  - If no, run through a multistage filter that applies 4 different hashes and checks for those hashes in different tables; count of entry with smallest count incremented; if all 4 counters exceed a second threshold, make entry in dispersion table
- Found several worms before antimalware vendors distributed signatures for them



# Example: Polygraph

- Assumes worm is polymorphic or metamorphic
- Generates classes of signatures, all based on substrings called *tokens*
  - *Conjunction signature*: collection of tokens, matched if all tokens appear regardless of order
  - *Token-subsequence signature*: like conjunction signature but tokens must appear in order
- Bayes signature associates a score with each token, and threshold with signature
  - If probability of the payload as computed from token scores exceeds a threshold, match occurs
- Experimentally, Bayes signatures work well when there is little non-malicious traffic, but if that's more than 80% of network traffic, no worms identified

# Behavioral Analysis

- Run suspected malware in a confined area, typically a sandbox, that simulates environment it will execute in
- Monitor it for some time period
- Look for anything considered “bad”; if it occurs, flag this as malware

# Example: Panorama

- Loads suspected malware into a Windows system, which is itself loaded into Panorama and run
  - Files belonging to suspect program are marked
- Test engine sends “sensitive” information to trusted application on Windows
- Taint engine monitors flow of information around system
  - So when suspect program and trusted application run, behavior of information can be recorded in taint graphs
- Malware detection engine analyzes taint graphs for suspicious behavior
- Experimentally, Panorama tested against 42 malware samples, 56 benign samples; no false negatives, 3 false positives

# Evasion

Malware can try to ensure malicious activity not triggered in analysis environment

- Wait for a (relatively) long time
- Wait for a particular input or external event
- Identify malware is running in constrained environment
  - Check various descriptor tables
  - Run sequence of instructions that generate an exception if not in a virtual machine (in 2010, estimates found 2.13% of malware samples did this)

# Data vs. Instructions

- Malicious logic is both
  - Virus: written to program (data); then executes (instructions)
- Approach: treat “data” and “instructions” as separate types, and require certifying authority to approve conversion
  - Key are assumption that certifying authority will *not* make mistakes and assumption that tools, supporting infrastructure used in certifying process are not corrupt

# Example: Duff and UNIX

- Observation: users with execute permission usually have read permission, too
  - So files with “execute” permission have type “executable”; those without it, type “data”
  - Executable files can be altered, but type immediately changed to “data”
    - Implemented by turning off execute permission
    - Certifier can change them back
      - So virus can spread only if run as certifier

# Containment

- Basis: a user (unknowingly) executes malicious logic, which then executes with all that user's privileges
  - Limiting accessibility of objects should limit spread of malicious logic and effects of its actions
- Approach draws on mechanisms for confinement

# Information Flow Metrics

- Idea: limit distance a virus can spread
- Flow distance metric  $fd(x)$ :
  - Initially, all information  $x$  has  $fd(x) = 0$
  - Whenever information  $y$  is shared,  $fd(y)$  increases by 1
  - Whenever  $y_1, \dots, y_n$  used as input to compute  $z$ ,  $fd(z) = \max(fd(y_1), \dots, fd(y_n))$
- Information  $x$  accessible if and only if for some parameter  $V$ ,  $fd(x) < V$



# Example

- Anne:  $V_A = 3$ ; Bill, Cathy:  $V_B = V_C = 2$
- Anne creates program P containing virus
- Bill executes P
  - P tries to write to Bill's program Q; works, as  $fd(P) = 0$ , so  $fd(Q) = 1 < V_B$
- Cathy executes Q
  - Q tries to write to Cathy's program R; fails, as  $fd(Q) = 1$ , so  $fd(R)$  would be 2
- Problem: if Cathy executes P, R can be infected
  - So, does not stop spread; slows it down greatly, though

# Implementation Issues

- Metric associated with *information*, not *objects*
  - You can tag files with metric, but how do you tag the information in them?
  - This inhibits sharing
- To stop spread, make  $V = 0$ 
  - Disallows sharing
  - Also defeats purpose of multi-user systems, and is crippling in scientific and developmental environments
    - Sharing is critical here

# Reducing Protection Domain

- Application of principle of least privilege
- Basic idea: remove rights from process so it can only perform its function
  - Warning: if that function requires it to write, it can write anything
  - But you can make sure it writes only to those objects you expect

# Example: ACLs and C-Lists

- $s_1$  owns file  $f_1$  and  $s_2$  owns program  $p_2$  and file  $f_3$ 
  - Suppose  $s_1$  can read, write  $f_1$ , execute  $p_2$ , write  $f_3$
  - Suppose  $s_2$  can read, write, execute  $p_2$  and read  $f_3$
- $s_1$  needs to run  $p_2$ 
  - $p_2$  contains Trojan horse
    - So  $s_1$  needs to ensure  $p_{12}$  (subject created when  $s_1$  runs  $p_2$ ) can't write to  $f_3$
  - Ideally,  $p_{12}$  has capability  $\{ (s_1, p_2, x) \}$  so no problem
    - In practice,  $p_{12}$  inherits  $s_1$ 's rights, so it can write to  $f_3$ —bad! Note  $s_1$  does not own  $f_3$ , so can't change its rights over  $f_3$
- Solution: restrict access by others

# Authorization Denial Subset

- Defined for each user  $s_i$
- Contains ACL entries that others cannot exercise over objects  $s_i$  owns
- In example:  $R(s_2) = \{ (s_1, f_3, w) \}$ 
  - So when  $p_{12}$  tries to write to  $f_3$ , as  $p_{12}$  owned by  $s_1$  and  $f_3$  owned by  $s_2$ , system denies access
- Problem: how do you decide what should be in your authorization denial subset?

# Karger's Scheme

- Base it on attribute of subject, object
- Interpose a knowledge-based subsystem to determine if requested file access reasonable
  - Sits between kernel and application
- Example: UNIX C compiler
  - Reads from files with names ending in “.c”, “.h”
  - Writes to files with names beginning with “/tmp/ctm” and assembly files with names ending in “.s”
- When subsystem invoked, if C compiler tries to write to “.c” file, request rejected

# Lai and Gray

- Implemented modified version of Karger's scheme on UNIX system
  - Allow programs to access (read or write) files named on command line
  - Prevent access to other files
- Two types of processes
  - Trusted: no access checks or restrictions
  - Untrusted: valid access list (VAL) controls access and is initialized to command line arguments plus any temporary files that the process creates

# File Access Requests

1. If file on VAL, use effective UID/GID of process to determine if access allowed
2. If access requested is read and file is world-readable, allow access
3. If process creating file, effective UID/GID controls allowing creation
  - Enter file into VAL as NNA (new non-argument); set permissions so no other process can read file
4. Ask user. If yes, effective UID/GID controls allowing access; if no, deny access



# Example

- Assembler invoked from compiler

```
as x.s /tmp/ctm2345
```

and creates temp file /tmp/as1111

- VAL is

```
x.s /tmp/ctm2345 /tmp/as1111
```

- Now Trojan horse tries to copy x.s to another file
  - On creation, file inaccessible to all except creating user so attacker cannot read it (rule 3)
  - If file created already and assembler tries to write to it, user is asked (rule 4), thereby revealing Trojan horse

# Trusted Programs

- No VALs applied here
  - UNIX command interpreters: *cs**h*, *sh*
  - Program that spawn them: *getty*, *login*
  - Programs that access file system recursively: *ar*, *chgrp*, *chown*, *diff*, *du*, *dump*, *find*, *ls*, *restore*, *tar*
  - Programs that often access files not in argument list: *binmail*, *cpp*, *dbx*, *mail*, *make*, *script*, *vi*
  - Various network daemons: *fingerd*, *ftpd*, *sendmail*, *talkd*, *telnetd*, *tftpd*