Decentralized and adaptive botnets

Grzegorz Kuliński *

Abstract

Modern botnets incorporates increasingly elaborate evasion techniques, as a defensive countermeasures are improved. In recent years raise of machine learning techniques seen adaptation in many fields, including malware and botnets. As a part of masters thesis preparation, I have conducted following research. In first section (1) I explore general landscape of botnets. Common classification of botnets is presented and some challenges of creating functional network of bots are discussed. Second part of the following report, starting with Section 2, describes proposed laboratory setup, that can be used to conduct practical tests of botnets.

1 Overview of botnets

1.1 Classification

Botnets can be divided into several categories based on key features and their main purpose.

Choosing communication method is one of the most important decision during development of a Botnet. Commonly used architecture solutions are P2P and centralized models.

P2P Nodes communicates directly with each other. Routing in such networks can be based on a few peer servers, or be fully decentralized. Some networks are based on completely new protocols (like *FritzFrog*), other uses well known protocols like *Gnutella*.

Popular way of creating a botnet in early 2000s was to pollute existing P2P networks — method called by P. Wang et al. [14] as "parasite P2P botnet". Bot-master had to pollute selected shared files, those could then easily spread to other nodes, attacking vulnerable hosts (e.g.. Gnutella network). Author highlighted two more categories: "leeching P2P botnets" and "Bot-only P2P botnets". First term relates to botnets that consists of compromised hosts that eventually joins some existing P2P network. Finally a Bot-only P2P botnet uses its own P2P network for communication (e.g.. *Stormnet* [4], *FritzFrog* [3]).

Popular examples of P2P based botnets are: *Stormnet*, *Trojan.Peacomm*, *Gnuman*, *VBS.Gnutella*, *SdDrop* and recent *FritzFrog*.

Centralized In contrast to P2P networks, in centralized structure botnets each node generally communicates only with it's *Command and control* server. Sometimes hierarchical model is used, where similarly to pyramid, nodes connects to its command nodes, which connects to higher level nodes — creating pyramid. Botnets such as *Mirai*, *Hajime*, *Srizbi*, *Kraken* are popular examples of centralized architecture botnets.

Mixed Some botnets uses mix of architectures, taking features from both models. Nodes can communicate with each other, while they keep connection to a *CNC* server.

Botnets often are sold in "Botnet as a service" model, where client rents part of a botnet resources, similarly to how modern cloud solutions like *AWS* or *Microsoft Azure* operates. Subscribed client can use rented botnet for different tasks. Based on this, networks can be classified by its purpose.

DDOS Main purpose of such botnets is performing DDoS attacks.

Spam Botnets can be used to send spam messages. Often these contains advertisements and presents malicious behaviour, or are part of an affiliate program.

Crypto-mining With raising popularity of crypto-currencies, botnets are used to generate profit by mining crypto-coins on infected machines. These botnets are often focused on infecting more modern machines, as performance is important factor in crypt-mining.

1.2 Main botnets threads

With a new malware, researchers often analyse its internal mechanism, fixing vulnerabilities, bugs and rendering viruses harmless to updated devices. Malicious actors, as a response, creates more sophisticated solutions. It creates similar to cat-and-mouse game, where often hackers are one step ahead.

Botnet executes a few key tasks, that each presents its own challenges: expanding by infecting new devices, keeping existing nodes in botnet, subscribing to commands from bot-master and executing received commands.

Expanding Metric determining success factor is number of compromised devices joined to a botnet. There is no definite solution to how to achieve that.

New vulnerable systems can be discovered by network scans performed by joined nodes. Such scans often can be easily intercepted by Intrusion prevention systems, leading to compromising the node. One can implement some evasion techniques, making detection more difficult.

Another problem is enhancing effectiveness of attacks. In case of IoT focused botnets, first access to systems is often achieved by brute-forcing ssh/telnet login data.

* gkulinski@student.agh.edu.pl

More advanced botnets can use dictionaries of commonly used passphrases. Researchers creates countermeasures, including "whitehat" attacking tools, which tests IoT devices for known vulnerabilities and patches them. Such proof-of-concept was presented by Kelly, Pitropakis, McKeown, *et al.* [6].

Propagating commands To harvesting power of created botnet CNC server must create easy way to communicate with bot-nodes. The same time, it is considered to be one of the weak points of botnets. Disabling communication channels to botmaster is one of the typical defence measures, as it can render botnet unusable.

Communication channels are designed in many different ways. In centralised architecture this can mean that nodes periodically checks some known CNC endpoints. Addresses often are hardcoded into malware. If ISP blocks internet access to associated CNC servers, bots no longer can be used by a botmaster. *DeepC2* proposed by Wang, Liu, Cui, *et al.* [16] introduces interesting approach. CNC nodes publishes commands as a tweets, but instead of enigmatic encoding, tweets consists of properly constructed English sentences. That way outsider cannot distinguish bots from real users. Bots can identify bot-tweet by passing their profile avatar to embedded neural networks.

Reverse engineering Inevitably, sooner or later, compromised host will be identified and deployed malware analysed. Such software can be analysed statically and dynamically. Number of measures was proposed to hinder that process [1] [8] [10]. Certain tools can obscure debugging process. If malware detects it runs within sandbox, it can disable its malicious behaviour.

Another technique is to create polymorphic malware. Such software cam change its form when replicated. As a result identifying and classifying detected malware is more challenging, as simple signature based solutions won't work.

2 Testing methodology

Major difficulty in testing botnets is to find fine balance between realism and scalability. There is number of papers, where authors use simulation methods to research botnets.. Vignau, Khoury, Hallé, *et al.* [13] focuses on interactions between botnets. Wang, Ma, Zhang, *et al.* [15] used existing *OPNET* network simulator to create game theory based model to analyse DDoS attacks. Similarly Wu, Shiva, Roy, *et al.* [17] based their game theory model on *NS-3* simulation tool. Intresting example of agent-based simulator was proposed by Kotenko, Konovalov, and Shorov [7] using *OMNeT++*.

Papers focusing on botnet defense often uses prepared datasets. Such sets usually consists of realistic network traffic flow and includes multiple types of typical for botnets network traffic, such as DDoS attacks or service scans. One of the examples is The Bot-IoT Dataset [9] published by UNSW Canberra. It consists of 69.3 GB of pcap files, capturing DDoS, DoS, OS and Service Scan, Keylogging and Data exfiltration attacks.

2.1 Laboratory setup

Virtualization was selected solution for creating botnets environment. It offers realistic nodes behaviour, while keeping low cost of operating.

Main limitation for a virtualization approach is fairly small number of machines that can be run concurrently on a host machine.

2.1.1 *Virtualization types*

VMs technologies can be classified by hypervisor type. Type 1 - *bare metal*, where hypervisor runs directly on host's hardware. Type 2 - *hosted* hypervisor operates within the host's operating system. Examples of type 1 are: *VMware*, *Virtual box*, *Hyper-V*. Typical Type 2 technologies are *KVM*, *QEMU*. There are other types of virtualization such as *Docker*

From the botnet simulation point of view, depending on needs, different virtualization methods can be used. Machines designated to be infected should be run using Type 1 supervisors. Such *VM*s must convey the most realistic environment. *VM*s used as a network nodes can be minimal Linux machines, without many common system features.

2.2 Setup & tools

I selected Hyper-V as a main virtualization method. In order to easily reproduce lab setup *Vagrant* [12] was used. It is an cross-platform tool, that simplifies VMs configuration and management. Vagrant supports every major virtualization technology, making it perfect solution for mixing vm types. Example laboratory setup consists of several *target* nodes and one *CNC* machine.

Target Machines used as a target are based on *BusyBox* system - Apline linux distribution. Alpine is minimal Linux distribution. With total image size below 100MB (some versions can go as low as 5MB), it is ideal Linux distro to be deployed in multiple nodes.

CNC Command and control machine runs on *Ubuntu* 18.04 Linux distro. As *CNC* node often carries more services, daemons, etc., Ubuntu offers better support.

For automation *vagrant* startup script is used. It handles creating and configuring *VM*s. Example configuration of *CNC* machine is included at Figure 1.

Vagrant setup script can include multiple virtualization technologies, making possible to create network of several *Hyper-V VM*s and *Docker* instances. Example diagram of such topology is presented at Figure 2. Network uses virtual switch (*vSwitch*), which is system managed network adapter. Each node connected to *vSwitch* can freely communicate with each other.

To simulate network/nodes outages, one can connect directly to desired nodes cluster, disabling their *NIC*. Such task can be easily done using *Ansible* [11]. It is automation software, suited to speedup management process. Keeping connection to nodes can be achieved by creating two *NIC*s, one used for a management, second as a data interface. Similar solution can be found in networking — often called control plane, data plane and management plane. Botnet can be limited to

```
# ============================
# =       Configure CNC      =
# ============================
config.vm.define "cnc" do |cnc|
  cnc.vm.box = "generic/ubuntu1804"
  cnc.vm.hostname = "cnc"
  cnc.vm.network "public_network", bridge: "Default Switch"

  cnc.trigger.after :'Vagrant::Action::Builtin::SetHostname', type: :hook do |hook|
      hook.info = 'Sync files'
      hook.run = {inline: "python setup_scripts/run_after.py cnc"}
  end

  cnc.vm.provision "shell",
    inline: "ifconfig eth1 10.10.10.1 netmask 255.255.255.0"

  cnc.vm.provision "shell", path: "setup_scripts/cnc_setup.sh"

  cnc.vm.provider :hyperv do |h, override|
      h.maxmemory = 2048
      h.memory = 2048
      h.vmname = "cnc"

      override.trigger.before :'VagrantPlugins::HyperV::Action::StartInstance', type: :action do |trigger|
        trigger.run = { inline: "./add_net_int.ps1 -VmName cnc -SwitchName \"'#{secSwitch}'\" " }
      end
  end

end
```

Figure 1: Example *CNC* configuration

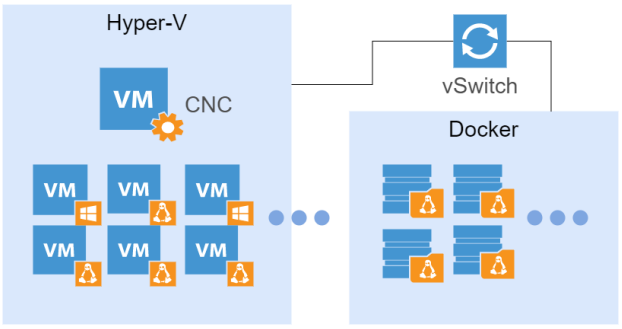

Figure 2: Topology of virtualization network on local machine

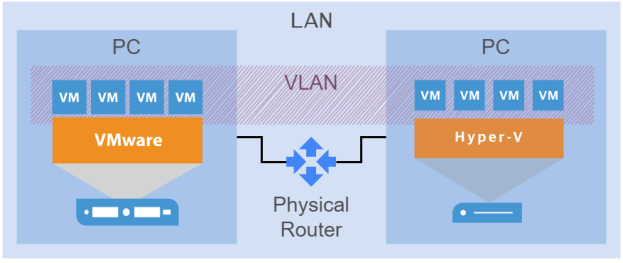

Figure 3: Topology of virtualization network based on local access physical network

use only data *NIC*, while management *NIC* can communicate without any disruptions.

2.3 Issues

Unfortunately *Vagrant* has several limitations, making workarounds necessary.

Network *Vagrant* is unable to setup network with *Hyper-V* (normally done with `config.vm.network <type>, <ip_addr>, <opts>`). As a result it is impossible to control IP address and interface setings in advanced, within *Vagrantfile* script. to solve this problem, powershell script, setting up extra *NIC* after VM creation must be run. Additionally some experimental features must be activated to allow triggering jobs between VM creation and start.

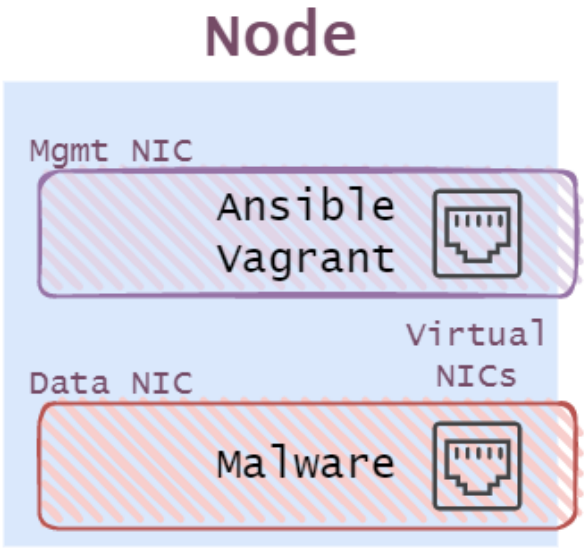

Figure 4: Diagram of node network configuration

Shared folders Windows host supports only *SMB* protocol. When experimental features are used, issues with *SMB* arises. I proposed a workaround by using powershell *scp* scripts, that sends necessary data to a *VM*.

Alpine releases Newer releases of Alpine Linux faces various problems during configuration using *Vagrant*. Setup script freezes on some steps, rendering *Vagrant* engine unresponsive. Older releases unfortunately often does not support latest versions of applications (such as *Node/Npm*, Cmake), making development much more difficult.

2.4 Limitations

Several limitations of virtualization approach can be pointed out.

Firstly, number of concurrently running VMs is limited. Botnets contains often thousands of infected machines. Virtualization handles only a dozens of VMs. Study of networks with fairly limited number of nodes can yield skewed results. Often large scale systems presents different behaviour, when compared to smaller, locally performed tests.

2.5 Alternative methods

Multiple solutions for testing botnets can be proposed. One can create simulator, which tries to imitate behaviour of such networks. This approach offers great scability, as when network is simulated, thousands of nodes can be run. Unfortunately, creating simulator often implies creating simplified model. Assumptions made during that process often limits accuracy of performed tests.

Another approach is to deploy physical machines that connects to a physical network hardware. Such solution is the most realistic one, but its scalability is main limiting factor.

3 Experiments

As a proof of concept, I have created two experimental configurations: one based on Miari botnet, second one using P2P network.

3.1 Mirai botnet

Starting with a botnet. *Mirai* [5] is centralized botnet, which was used in several big scale *DDoS* attacks. Main command and control server component is implemented in Golang, it handles *SQL* database, which contains credentials for Botnet users. *CNC* server includes also loader used for sending payload to vulnerable devices. Payload implemented in C, compiled for multiple architectures, was running inside memory. Mirai targeted mainly *IoT* devices, forcing connection using default login data. Simple IoT devices often used factory setting, exposing ssh/telnet ports for external connections. That allowed Mirai to gain impressive number of infected devices.

Currently source code of Mirai botnet is publicly available [5], after it was leaked by one of the (presumably) original creator.

3.2 P2P network

Second experiment focused on deploying simple P2P network. I have chosen a GUN [2] — An open source cybersecurity protocol for syncing decentralized graph data. In this case server was necessary to propagate the peers, from which nodes could sync data. Each node can run JavaScript CLI script, behaving as a nodes in P2P network.

3.3 Observability

Because nodes participating in Lab are fully functional operating systems, it is easy to observe desired metrics wihin those machines. One can record traffic-load in case of emulating DDoS attacks, or simply capture traffic on malware interface.

From controller point of view, network traffic can be recorded with external capturing tools like *Wireshark*. Hyper-V by default lets listen to sent/received traffic on created NIC.

4 Conclusions and future work

With a surge of advanced botnets it is important to explore possible challenges for a defenders to encounter. It is worth exploring possible implementation of swarm-based intelligence in botnets, or other machine learning techniques. As a result of this project, I have developed a limited survey on current botnets. Additionally I have identified common patterns found in botnets and major challenges that comes with designing one.

Another important part of the project was to propose laboratory setup. I have proposed virtualization method that uses Hyper-V and docker. I have incorporated automation tools to achieve reproducible environments in robust and fast fashion. To compensate limited scale of possible test botnets, I additionally proposed alternative topology that uses several physical devices connected to physical L2 switch.

Data Availability

Project is available as a git repo on github `https://github.com/grzes5003/botnet-sandbox`.

References

[1] B. Abrath, B. Coppens, I. Nevolin, and B. D. Sutter, "Resilient self-debugging software protection", in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&ampPW)*, IEEE, 2020. DOI: 10.1109/eurospw51379.2020.00088.

[2] amark, *Gun — an open source cybersecurity protocol for syncing decentralized graph data.* https://github.com/amark/gun, 2022.

[3] O. H. Ben Barnea Shiran Guez, *Fritzfrog: P2p botnet hops back on the scene*, https://www.akamai.com/blog/security/fritzfrog-p2p, [Accessed 01-Sep-2022], 2022.

[4] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm", in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, series LEET'08, San Francisco, California: USENIX Association, 2008.

[5] jgamblin, *Mirai source code*, https://github.com/jgamblin/Mirai-Source-Code, 2017.

[6] C. Kelly, N. Pitropakis, S. McKeown, and C. Lambrinoudakis, "Testing and hardening iot devices against the mirai botnet", in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pages 1–8. DOI: 10.1109/CyberSecurity49315.2020.9138887.

[7] I. Kotenko, A. Konovalov, and A. V. Shorov, "Agent-based modeling and simulation of botnets and botnet defense", 2010.

[8] J. Miljak, *An experimental study on which anti-reverse engineering technique are the most effective to protect your software from reversers*, 2017.

[9] N. Moustafa, *The bot-iot dataset*, 2019. DOI: 10.21227/r7v2-x988.

[10] K. B. R. Haritha Priya, *Anti –reverse engineering techniques employed by malware*, 2019.

[11] *Red Hat Ansible — Simple IT Automation — ansible.com*, https://www.ansible.com, [Accessed 01-Sep-2022].

[12] *Vagrant by HashiCorp — vagrantup.com*, https://www.vagrantup.com, [Accessed 01-Sep-2022].

[13] B. Vignau, R. Khoury, S. Hallé, and A. Hamou-Lladj, "The botnet simulator: A simulation tool for understanding the interaction between botnets", *Software Impacts*, volume 10, page 100 173, 2021. DOI: https://doi.org/10.1016/j.simpa.2021.100173.

[14] P. Wang, B. Aslam, and C. C. Zou, "Peer-to-peer botnets", in *Handbook of Information and Communication Security*, P. Stavroulakis and M. Stamp, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pages 335–350, ISBN: 978-3-642-04117-4. DOI: 10.1007/978-3-642-04117-4_18.

[15] Y. Wang, J. Ma, L. Zhang, W. Ji, D. Lu, and X. Hei, "Dynamic game model of botnet ddos attack and defense", *Security and Communication Networks*, volume 9, (16), pages 3127–3140, 2016. DOI: https://doi.org/10.1002/sec.1518.

[16] Z. Wang, C. Liu, X. Cui, J. Yin, J. Liu, D. Wu, and Q. Liu, "Deepc2: Ai-powered covert command and control on osns", 2020. DOI: 10.48550/ARXIV.2009.07707.

[17] C. Wu, S. Shiva, S. Roy, C. Ellis, and V. Datla, "On modeling and simulation of game theory-based defense mechanisms against dos and ddos attacks", 2010, page 159. DOI: 10.1145/1878537.1878703.