Hiding in open sight: Dynamic profiles for malware communication

Request For Comments

João Carlos de Novais Marques
joao.marques@os3.nl
University of Amsterdam
Faculty of Science (FNWI)
MSc System & Network Engineering

M.J.R.Cox
mick.cox@os3.nl
University of Amsterdam
Faculty of Science (FNWI)
MSc System & Network Engineering

Abstract—**Communication between Command & Control servers (C&C) and its slave systems occur in botnet environments and during advanced & targeted attacks (APTs). Due to the attackers intending not to be detected, communication methods used during such attacks commonly utilize advanced evasion techniques. Current network intrusion detection mechanisms are generally either based on signature based detection, or anomaly based methods. Where the former is accurate in true positive classification, it's also practically unable to detect any previously unknown intrusions; whilst the latter is able to detect from the standard deviation of normal behavior. Anomaly detection is also notoriously known for generating a high rate in false positives, making many of such techniques hard to implement. In this research the authors propose a highly modular and configurable software framework for simulating advanced and stealthy C&C to beacon communication. By focusing on three critical aspect: - local host and network reconnaissance; - message obfuscation, and the utilization of popular services and protocols as covert channels, communication between such a beacon and its C&C should be unrecognizable from an infected hosts' regular behavior.**

1. Introduction

With the ever increasing growth of the internet, malicious use of computer systems has been on the rise. Botnets and more recently, Advanced Persistent Threats (APT), play a big part on modern day cyber-crime and have a great impact on the security landscape to the current day. Both forms of malware have been known for their practice in infecting computers by exploiting vulnerabilities in its system, mostly exploiting such vulnerabilities in popular and widely used software applications. Once set up, these malwares establish a communication channel with a Command & Control server, providing a backdoor over which an attacker can take control over remote computers. Although both techniques share this common trait, it has to be said that where a botnet usually aims at infiltrating as many systems as possible, an APT is significantly different in that these usually form a targeted attack against a specific target.

In this paper, we are interested in C&C to bot communication channels; in currently used, and literature described, methods for detecting such channels; and last, in the possibilities of evading such detection methods, by means of dynamically changing the patterns used for communication, which we will refer to as communication profiles.

We initially set out to create such a network profile ourselves, phrasing the research question as follows:

> *"Is it possible to construct a dynamic network profile between a Command & Control server and the beacon, which is undetectable by state-of-the-art detection frameworks?"*

By starting with researching the inner workings of the popular network and mostly signature based IDS Snort 3, we set out to identify any weakness in its detection engine which we could then further evaluate in its effectiveness against anomaly based detection measures and its ability to act as a dynamic system/profile. These experiments would be have been conducted in an already fully operational testing environment. However, during the research we came to realize some of the limitations of this approach, being: developing a new network profile will by default evade any already existing rule or signature; a successfully implemented dynamic profile will by its nature change and therefore negate the effectiveness of any signature based NIDS; finding a vulnerability in the detection engine of Snort would be of arbitrary value considering the modular approach in which Snort ++ (3) is being build and its likelihood of such a shortcoming being patched; and lastly, the time constrictions of our research and the unlikelihood of us finding such a weakness. It is therefore that we decided to deviate from this original idea by focusing on a literature review in which we aim to point out the areas in which a dynamic profile can be made, by comparing ways current detection techniques function and countermeasures which can be taken to bypass these detection methods. We will then evaluate ways in which employ countermeasures in a dynamic fashion. Furthermore, we have set out to develop a system which will utilize these countermeasures in an intelligent and dynamic way. By developing such a system, we intend to enable researchers in simulating environments which can help in improving old and developing new detection systems.

Paper outline

In Section 2 we will give a review of the relevant literature on the topics of advanced malwares including both botnets and APT attacks, on detection techniques for these communication profiles and techniques these malwares have used and possibly utilize for hiding or evading detection. In Section 3 we will describe the method used set all of the environments and the development of the program. In Section 4 we will describe the results obtained by running the program in the testing environment. In Section 5 we will discuss our findings, our proposal and any possible countermeasures which we suspect are able to counter the developed program. In Section 6 we present possible future work and in Section 7 we will give our concluding remarks.

Definitions

Advanced Persistent Threats
: Advanced persistent threats (APT) are cyber attacks executed by sophisticated and well-resourced adversaries targeting specific information in high-profile companies and governments, usually in a long term campaign involving different steps, as defined by /citechen2014study. APTs usually involve Remote Access Trojans, and botnet like communication structures.

Beacon
: is the term used in this paper to denounce the subsystem of malicious software, which is active on a bot, and responsible for establishing a communication profile with the C&C.

Botnet
: a bot is computer system, usually workstation, infected by a remote access trojan,

Command & Control
: is a computer system with communication streams to its clients, most common infected clients which act as bot in a botnet.

Communication profile
: A communication profile are the combined properties of an established communication channel between C&C and beacon.

Overt channel
: An overt channel is an open and visible communication channel which is generally open for any bystander to see, usable as hidden data carrier.

Covert channel
: A covert channel is a communication channel that is not intended for information transfer at all [Lampson, 1973]. Networking covert channels are created using a network steganography technique.

Intrusion Detection Systems
: Systems for monitoring and evaluating data from a system with the goal of recognizing and possibly preventing intrusions; denoted by the abbreviation IDS, HIDS for Host based- and NIDS for Network based intrusion detection systems.

Overlay network
: is usually the term to denounce peer to peer networks in decentralized networks. In this paper we

Signature detection
: is one of the detection methods an IDS can employ in its search for intrusions. Signature based detection methods use signatures, also called rules. These rules are a fingerprint of previously detected intrusions and malware.

Anomaly detection
: is one of the detection methods an IDS can employ in its search for intrusions. Anomaly based detection methods use an algorithm which is first trained with a set of normal data, after which it will detect

2. Literature Review

In this section, we hope to facilitate the reader with an overview of relevant and recent research on the topics of information hiding, remote access malware, by which we mean botnets and APT attacks, the methods they utilize to remain undetected and some of the techniques one could use to detect them, which are often employed in intrusion detection systems.

We start with the act of information hiding. In the book [Katzenbeisser and Petitcolas, 2016], the authors gave an extended overview of the act of hiding information in communication networks. In this, he classifies three main types of information which can be subjected to such hiding mechanisms, which we will describe one by one:

- Identities of communication parties
- Communication process
- Communication content

2.1. Hiding the identities

Identities of parties involved in secret communication can be hidden by making the source and final address hard to retrieve. Known methods for to establish this is using tunneling techniques such as a VPN, proxies such as HTTP proxies, or anonymity networks such as TOR. In [Rodríguez-Gómez et al., 2013], the authors report botnets commonly using multi-hopping techniques, fast flux networks.

2.2. Hiding the processes

This subsection has been modeled after the book [Katzenbeisser and Petitcolas, 2016], which gives an extended scientif review of purely this topic.

When two parties decide to communicate in secret, they can do so by establishing a covert channel, which they indent to remain secret.

2.2.1. Network steganography.

Network steganography hides data inside an overt communication in a way that minimizes the impact on the over transmissions and thus it effectively conceals the existence of the covert transmission (CT). It aims at hiding secret data in the normal transmissions of users without significantly altering the carrier used. A carrier is defined as one or more overt traffic flows that pass between a covert sender and a covert receiver. A subcarrier is defined as a place or a timing of events in a carrier, where secret information can be hidden using a single steganographic technique. A subcarrier typically takes the form of a storage or timing covert channel.

A classification of network steganography is given, in which it defines two distinct groups: timing-, and storage based techniques. First, timing based techniques are further divided in protocol aware-, and protocol agnostic methods. A protocol aware method could for instance introduce artificial loss, retransmissions, manipulate the order of messages, frame collisions or changing the temperature, CPU clock and timestamps, in order to hide information; whereas a protocol agnostic method does not need to know the in depth characteristics of a protocol, utilizing for instance the rate/throughput, interpacket times and message timing.

Second, storage based methods are divided into methods which modify user data, and those methods which modify protocol specific fields: Protocol Data Units (PDU). The latter one can then be subdivided in methods which either preserve the structure of protocol fields and those who don't. Structure modifying methods can be found in size modulation, altering the sequence in position or in the number of elements, and by introducing redundancy. Structure preserving methods can be found in the usage of random numbers, value modulation such as modifying character case or modifying the least significant bit, or modifying reserved and/or unused fields.

Each of these methods can be further described with by three characteristics: (1) Steganography bandwidth, which describes how much secret information one is able to send per time unit; (2) Undetectibility, defines the inability of an adversary to detect a steganogram inside a carrier; and (3) Robustness, which describes how much alteration the steganogram can withstand whilst not destroying the secret data.

Furthermore, there is the concept of steganography cost. Steganography cost is defined as the degradation or distortion of the carrier, when steganographic methods are applied. One can imagine that this cost has an effect on the detectability. If the cost of applying steganography exceeds a certain treshold, the steganographic act becomes detectible, with an ever increasing likelihood if the cost grows.

Finally, the author introduces the idea of steganographic control protocols, which is a communication protocol that is embedded into a steganographic carrier and regulates the communication between distributed steganographic processes. Features include: sending acknowledgment flags, sequence numbers and measures for session management over covert data transfers. Control protocols can also enable the usage of deep hiding techniques (DHT), such as switching between carriers, subcarriers, and hiding methods. In short, control protocols increase reliability and when combined with DHT, they can increase covertness. However, it must be said that control protocols can potentially also increase the steganographic cost.

2.2.2. Traffic Type Obfuscation.

Traffic type obfuscation alters the patterns and contents of network traffic between two network entities so that a third entity can not identify the type of their communication, which is the network protocol. Traffic type obfuscation, can be categorized in two classes.

First, Traffic de-identification aims to make the network protocol hidden. Methods include: the encryption of packet headers; padding encrypted packets, using various padding strategies, such as already is implemented in GnuTLS; and HTTP obfuscation (HTTPOS) which implements features to obfuscate patterns in client side HTTP traffic.

Second, Traffic impersonation, which aims to alter traffic to hide the underlying protocol and also pretends to be using another protocol. Proposed methods include: SkypeMorph, which is a framework for disguising traffic from the TOR anonimity network as being Skype traffic; FreeWave, which similarly to SkypeMorph disguises any network traffic as Skype traffic, in order to resist any firewall blocks;

for instance when BitTorrent traffic tries to impersonate HTTP traffic. This can be done by imitating the other protocol's packet frequency and/or/size distributions.

TTO is commonly used for blocking resistance, such as bypassing a firewall; and for privacy protection.

2.3. Hiding the content

The most common method for hiding communication content is the usage of encryption. Encryption algorithms modify plain text into ciphertext using an encryption key. The cipher text cannot be interpreted and can only be reversed to its original state if one has the decryption key. There are many algorithms and methods to use encryption, which are not covered here. In symmetric encryption, the encryption and decryption key are the same. In asymmetric encryption, they are not.

2.4. Remote Access Malwares

In this section we will take a look at malwares which implement backdoors in compromised systems over which they establish a communication channel with a C&C, which are advanced persistent threats and botnets. We will take a look at their lifecycle, and how they operate.

2.4.1. Advanced Persistent Threats.

A definition of APT attacks is given by the authors of [Chen et al., 2013], namely: *Advanced persistent threats (APT) are cyber attacks executed by sophisticated and well-resourced adversaries targeting specific information in high-profile companies and governments, usually in a long term campaign involving different steps.* The authors of the same paper describe the life of an APT accordance with the following stages:

1) Reconnaissance and Weaponization
2) Delivery
3) Initial intrusion
4) Command & Control
5) Lateral movement
6) Data Exfiltration

During the *Reconnaissance and Weaponization* stage, in which APT actors gather information about a target mostly using OSINT (open source intelligence) and Social Engineering measures;

During the *Delivery* stage, in which attackers deliver malware exploits using directly targeted spear finishing techniques or watering hole attacks, in which the attacker lures the victim into using exploited but already trusted resources, such as we compromised website;

During the *Initial intrusion* stage, in which the attacker often exploits vulnerabilities in often wide used software such as Adobe Acrobat, Flash, or Microsoft Office, which can but not need be zero day vulnerabilities.

During the *Lateral Movement* stage, in which the APT actor performs internal reconnaissance and compromise additional systems to gain escalated privileges over commonly a longer period of time, in order to avoid detection.

During the *Command & Control* stage is the stage in which the APT actor, upon successfully establishing a backdoor, takes control over one or more compromised computers by communicating over various legitimate services and publicly available tools such as social networking sites in supplying control information, communicating via- and even hosting the C&C in anonymity networks such as TOR, or by using remote access tools (RAT) which are often used by system administrators but in such attacks misused to facilitate control over an infected host.

During the *Data Exfiltration* stage, in which the attacker is out to exfiltrate sensitive information, typically encrypted using SSL/TLS, funneled through an internal staging server and sent out using an anonymity network.

in which the authors propose data loss prevention systems (DLP) which act as a last line of defense, monitoring for sensitive data in different states of transit; and lastly, *Intelligence-driven Defense* which more than a defense mechanism is a strategy which would entail the creation of an intelligence feedback loop, enabling the defender to better recognize patterns of attacks.

Next to the stages, the authors also supplied countermeasures though explicitly mentioning no single solution can offer effective protection. Countermeasures include: *Security Awareness Training*, in which industry should be made aware to the difference between APT- and regular attacks; *Traditional Defense Mechanisms*, which can be utilized to block known attack vectors, such as patch management, anti-virus software, firewalls, HIDS, NIDS, IPS, SIEM and content filtering software; *Advanced Malware Detection*, in which by means of sandboxing execution malwares behavior can be analyzed, but also specify one should account for known sandbox execution evasion techniques; *Event Anomaly Detection*, which entails one utilize anomaly based detection, instead of looking for known bad (red: signature based detection), specifying this by proposing big data analytic and referring to previous succesfull studies which looked at HTTP requests and event processing based on MapReduce; *Data Loss Prevention*,

2.4.2. Botnets. Similarly to APT attacks, the lifecycle of botnets follow roughly the same stages, as defined by [Rodríguez-Gómez et al., 2013]:

1) Botnet conception, in which the botnet is designed and developed;
2) Botnet recruitment, in which the malware is spreaded
3) Botnet interaction, in which a bot communicates with its C&C
4) Botnet marketing, in which the botmaster goes to market with his botnet
5) Attack execution
6) Attack success

During the conception stage, the botnet is developed, for which three design types can be identified: centralized- , decentralized-, and a hybrid botnets; which represent the communication patterns between bot and C&C.

During the interaction stage, the authors identify internal and external interactions. First, internal interaction is further divided into a registration part and a C&C communications part. During the registration part, the bot becomes part of the botnet, which can happen either statically by hard coding addresses or algorithms which intend to find addresses; or dynamically, which can happen in a number of different ways, such as downloading files which include a list of peers. C&C communications happens after the registration part, represents the bulk of interactions, and can be grouped in accordance with three characteristics: the type of messages, direction of the information and the communication protocol. The *types of messages* can either be commands (orders), instructing a bot to do a certain action; or control messages intended for administrative purposes, such as updates or membership administration. The *direction of information* can either be pull or push. The *communication protocol* identifies the carrier of the messages, which is commonly recognized to be IRC, HTTP or P2P protocols. Second, *external interactions* entail the interactions between an infected bot and any noncompromised system, often to access publicly available services on the internet. Commonly, three services are identified: DNS, Services in P2P networks and Botnet monitoring services. DNS is usually used for C&C address resolvent. P2P network services are used as an intermediate layer to hide C&C connections, such as the case where only the name of a resource is given, which should be downloaded from a P2P network, which then contain instructions or otherwise new updates. Botnet monitoring services can be used by a botmaster to obtain useful information about the botnet itself, such as the availability of the domains used by the C&C in Whois or DNS systems. Within the marketing stage, the authors identify multiple scenarios in which a botmaster can monetize its botnet. In the attack execution, the authors identify multiple possible attack scenarios, such as distributed denial of service, spamming, phishing, data stealing and click fraud.

Furthermore, the authors identify some of the many hiding mechanisms used by botnets to frustrate the discovery of its components (bot, botmaster, C&C channels): Multi-hopping, Ciphering, Binary Obfuscation, Polymorphism, IP spoofing, Email spoofing, Fast-flux Networks. Multihopping is the usage of proxies or anonymity networks to conceal the final IP address. Ciphering is described as the usage of encryption in C&C communication channels, and is further reported to make the development of detection techniques increasingly different. Binary obfuscation techniques are used to complicate and conceal the bot source code, making reverse engineering and analysis on its behavior a difficult task. Polymorphism is utilized to bypass signature based virus scanners, as it entails the creation of different versions of the source code, which change, whilst the functionality remains the same; for example by using packers. IP Spoofing is the act of sending IP packets with a fake source address, often used to bypass IP filters during denial of service attacks. Similarly, E-mail spoofing is used to fake a source (from:) e-mail address or other e-mail header fields; often used in phishing and spamming attacks. Finally, Fast Flux Networks is the usage of rapidly changing DNS resource records in order to have multiple IP addresses answer for a certain fully qualified domain name; this method can be used to hide a final host in the network, such as a central C&C.

2.5. Detection methods

In this subsection, we will loosely describe methods of detecting C&C to beacon communications. As these concepts are often coined within the context of intrusion detection systems, they not need to be. As, for instance, such detection methods also apply for detecting network steganography and covert channels, we have decided to describe the methods individually from the systems in which they are often implemented.

[Limarunothai et al., 2015] identify four distinct botnet detection techniques, namely: (1) *Signature based detection*, (2) *Anomaly based detection*; (3) *DNS based detection*; (4) *Data-mining based detection*. Furthermore, they identify botnet challanges, which contribute frustrate the detection of such botnets.

2.5.1. Signature based. intrusion detection is based on pattern matching, such as regex, ngram or rule based matching. For the detection of intrusions, covert channels or malware, one need predefined knowledge of such cases, to translate them into a fingerprint, rule or otherwise called signature, in order to be able to validate those. Considering that signature based matching is deterministic in its approach, it is very accurate in classifying known intrusions (high true positive). However, this also makes this method practically unable to detect novel intrusions.

Signature based is very scalable and considerably more easy to implement than other methods. Snort 2 is a mostly on signature based detection system, which is often used in practice. Other systems commonly also support snort based signatures [Liao et al., 2013].

2.5.2. Anomaly based. intrusion detection is based on identifying anomalies based on a set of normal behavior, on which such systems are based. Anomaly based methods can use machine learning (ML) or statistical approaches; which both require the setting of a boundary or threshold. [Liao et al., 2013] [Karim et al., 2014]. Looking at machine learning, we can see that: behavior is measured in form of features; a decision threshold or boundary is required for classification; an algorithm decides between two classes (benign/malicious or true/false), even when the test data consists of more than two classes; in that case it will do this binary classification in a recursive 'one vs. the others' kind of fashion. Examples of such algorithms are decision trees, neural networks, support vector machines.

2.5.3. Traffic normalization. Traffic normalization is the act of removing semantic ambiguities, such as setting unset, reserved an padding bits to zero. Most traffic normalization targets the Network & Transport layer protocols (IP/TCP/UDP). Some have proposed to eliminating covert channels in HTTP by enforcing protocol compliant behavior. Furthermore, network-aware traffic normalizers (NAAW) are proposed to function as an intelligent and statefull system as an active countermeasure. Snort IDS uses traffic normalization extensively. These are called preprocessors and are responsible for either reassembling the original stream, which frustrate timing based steganography channels, and resetting PDU fields to normal values. Other traffic normalization systems are norm and Wendsel

2.5.4. Comparing botnet detection papers. In the extensive study [García et al., 2014], the authors conducted a survey in which they did an analysis on previous surveys, proposed a new classification and comparison of detection proposals and a new analysis of previously proposed detection methods.

Their analysis on previous botnet related surveys showed that these surveys all maintained a different categorization of detection methods, and emphasized different aspects of each paper. The authors identified different dimensions on which they evaluated previous surveys, and later also used to compare botnet detection papers. Based on these dimensions, the authors proposed a topology map and a set of desired properties of which they think botnet detection proposal frameworks should cover.

different network based botnet detection surveys, giving a so called topology map, in which the authors compared proposed detection methods on three different aspects: (1) *Detection Algorithms*, such as the use of supervised-, semi-supervised, unsupervised machine learning algorithms (MLA), signal processing which includes filtering, correlation and pattern matching; *Detection Techniques*, such as anomaly based techniques which includes analyzing the behavior of individual bots, behavior multiple bots in a cluster or network, temporal behavior which is defined as behavioral changes over time in which the authors include protocol usage such as peer to peer (P2P), HTTP, IRC, DNS, port scans, network metrics and SMTP, and lastly, fingerprinting techniques such as string or byte sequence pattern matching for detection; *Detection Sources* in which they compare the origin of packets captured during other surveys, which include normal packets from controlled virtual network or real networks, botnet packets captured from honeypots, darknets, controlled virtual networks, modified and recompiled known malware or packet logs.

Desired properties a novel botnet detection papers should include, target verification issues such as validating the used dataset, the diversity in training data, as well as the reproducibility; results issues such as a better description of performance and results, a result comparison; theoretical issues such as clearly defining any assumptions made, as well as describing how a hypothesis; and issues regarding the detection method itself, including a description about how the dataset was prepared, identifying the main detection method, differentiation from other attacks, which malicious actions it can detect and how botnets are automatically detected.

Further, they revisit previously proposed botnet detection papers and compared on their anomaly-based behavior detection techniques. The following papers are included:

BotMiner	[Gu et al., 2008a]
BotSniffer	[Gu et al., 2008b]
BotHunter	[Gu et al., 2007]
N-gram	[Lu et al., 2011]
Stability	[Li et al., 2010]
Models	[Wurzinger et al., 2009]
Unclean	[Collins et al., 2007]
FluXOR	[Passerini E, 2008]
Tight	[Strayer et al., 2006]
Tamed	[Yen and Reiter, 2008]
Incremental	[Yu et al., 2010]
Markov	[Kim et al., 2012]
Synchronize	[García et al., 2011]

Methods used by the proposed techniques, as described by earlier domains, include: (bot behavior) monitoring packets per flow, mean bytes per packet, connections to SMTP servers and DNS MX record queries, binary downloads, stable P2P connections in regards to average bytes per flow, generating the same traffic within different time windows, multiple bots synchronizing their traffic; (botnet behavior) same commands/traffic to every bot, similar traffic patterns over multiple hosts, IRC answers are synchronized, anomalous changes in domain features, communication with the same address / the same C&C; (temporal behavior) recognizing mean bytes per second and amount of flows per hour, the frequencies of communication and connection over HTTP, similarity of communications within time windows, recognizing if a C&C sends orders to multiple hosts at the same time; (protocol behavior) recognition of port scans, recognition of MX record lookup.

The authors also perform a comparison over the dataset sources; the protocols supported by the proposed techniques, mostly HTTP, IRC and P2P; the algorithms used within the proposed technique, including X-means hierarchical clustering, Autocorrelation, Nave Bayes classifiers and more; what the ratio over different dataset sources is during the training of different algorithms; how the papers described their results and the results themselves; the ability to recognize botnets not included in the training data; the dependency on protocol specific features, such as only looking for the IRC protocol on port 6667; and lastly describe the inner workings of each of the covered proposals individually.

The authors then conclude that even though great advances have been made, the state of the field has to solve multiple problems, as a significant amount of covered papers have issues such as the lack of dataset publication or usage of a public dataset, unverified captures, inaccurate reporting of performance metrics and recognize a common overfit of preprocessing methods. They conclude in saying that their survey aims at pointing to the most useful approaches in complex botnet traffic detection, which they identify as time based dynamic approaches (temporal); which focus on features describing general behavior in a hybrid system using meta-learner which can weigh and decide over the results from different behavior detection algorithms, warning the reader at last that such implementing multiple approaches in parallel would require big data solutions.

[Soniya and Wilscy, 2016] identify most detection proposed botnet detection methods rely on a regularity of C&C to bot communication traffic, whilst stating state of the art bots randomize traffic properties to evade regularity based detection techniques. The authors present a detection system which uses traffic analysis of an end-point host to identify bot to C&C communications even when the communication patterns are randomized to evade detection and also aims at early detection of bots. The proposed technique utilizes a Multi Layer Perceptron (MLP) Classifier, which is trained to differentiate between normal user-generated HTTP traffic and bot control traffic over a user session, and a Temporal Persistence (TP) Classifier that utilizes temporal persistence, a measurement of how repeatedly a destination is contacted over time, to identify bot control traffic for time periods larger than a user session. The proposed technique, called RCC Detector, has been trained on-, and tested against multiple public and privately captured data sets, on which it achieved a high accuracy (99.8% TP, 0.48% FP), given the dataset is representative of a real world scenario.

[Rossow and Dietrich, 2013] argue current payload signature intrusion detection systems fall short in their ability to detect modern state of the art C&C communication profiles, due to the fact these systems are Dependant on the presence of invariants in network communication. Considering C&C and beacons commonly utilize encryption with dynamic keys, random bytes to serve as initialization vector, and cipher-block chaining or cipher feedback mode to encrypt their messages, invariants will not often be found in its communication profile.

In a recent survey, [Acarali et al., 2016] set out to identify botnets which utilized encryption for hiding network content. In this research, the authors have shown it to be possible to train a framework on plain text known and widely used botnets, in such a manner that after encryption, it is able to, in a pseudeo brute force manner, be able to correlate entropy with trained botnets.

2.6. Intrusion detection systems

Intrusion Detection and Prevention Systems monitor data for malicious traces among traces of normal behavior in logs, metrics and communication at packet level. Where a detection system will remain to only reporting an intrusion, a prevention system will take immediate action in order to halt the possible intrusion /citemodi2013survey. In practice, these systems are mostly combined into an IDPS; though in literature they are mostly classified with the term IDS.

Multiple types of IDS exist, though the two prevalent ones are host based (HIST) and network based **(NIST)** Intrusion Detection Systems. [Goldring, 2003]. In this research, we will focus on NIDS.

2.6.1. Host based IDS. A host based intrusion detection system (HIDS) can then either profile a system on metrics derived from log file(s correlation), system calls, memory or CPU usage; or profile a user based on biometric information as keystroke frequency or psychometric data as Unix knowledge or usage, to detect intrusions. The host based IDS requires a software agent which is installed on the machine. Signature based HIDS are anti virus software packages for example.

2.6.2. Network based IDS. A network based intrusion detection system (NIDS) monitors network data. It can have multiple sensors (sniffers) in the network, on which it can perform traffic normalization, and then validate to an engine. NIDS come in both signature based as well as anomaly based methods. However, due to the high amount of false positives within anomaly based measures, signature based systems are still more prevalent.

2.6.3. Others. Other types of IDS have been proposed, such storage based systems or purely DNS based systems. If one of domains has been flagged as malicious, an intrusion prevention system could halt every DNS query in which this domain is present. Therefor, blocking IP resolution to the infected host [Karim et al., 2014].

2.7. Summary

We have described multiple methods of hiding information in communication networks and described methods of detecting attempts in hiding information. As we have shown, the field of information hiding, remote access malwares, detection methods and systems deploying such methods is a very broad subject and its taxonomy is complex, multidimensional. By no means is this literature review extensive. The research conducted is used in the next sections, in which we have attempted to develop a framework which can be intelligent, dynamic and able to evade intrusion detection.

3. Methodology

3.1. Hardware/Software

- Server - Dell PowerEdge R230
  - Intel(R) Xeon(R) CPU E3-1240L v5 @ 2.10GHz
  - 16Gb DDR4 memory (2 Modules of 8Gb each)
  - Broadcom Gigabit Ethernet BCM5720 (Network Interface Controller)
  - VMware Vsphere 6.5 (Hypervisor)
    - Virtual Router - Ubuntu Server 16.04.1
    - Virtual Switch (internal network)
    - Victim 1 - Ubuntu Workstation 16.04.1
    - Victim 2 - Windows 10 Student Edition x64
    - IDS - Ubuntu Workstation 16.04.1
    - C&C - Ubuntu Workstation 16.04.1
- Development laptop - Ubuntu Workstation 16.04.1
- Snort - 3 (Alpha)
- Overwatch
- python 3 - 3.5.2
- python 3 extra libraries:
  - appdirs - 1.4.0
  - dnslib - 0.9.7
  - netifaces - 0.10.5
  - packaging - 16.8
  - pyparsing - 2.1.10
  - six - 1.10.0
- pip3 - 9.0.1
  - virtualenv - 15.1.0
  - virtualenvwrapper - 4.7.2
- Github
- openVPN

3.2. Setting up the development Environment

The development environment was setup using python's package manager called pip (pip3 for python3). This tool facilitates the installation of python packages, such as community libraries[Wikipedia, 2017]. To setup the environment two packages where installed:

- virtualenv - enables the creation of virtual environments to work on separate projects
- virtualenvwrapper - wrapper for virtualenv that facilitates the creation and the management of virtual environments

The installation was achieved by issuing the following command:

```
sudo -H pip3 install virtualenv virtualenvwrapper
```

Once the packages finished installing the virtualenvwrapper had to be configured by exporting the directory where the project is to be saved and sourcing the configuration. This was done in the following way:

```
export WORKON_HOME=<virtual environment dir>
export PROJECT_HOME=<project dir>
echo ". /usr/local/bin/virtualenvwrapper.sh" >>
    $HOME/.bashrc
```

The project folder was then created by issuing the command:

```
mkproject dynamic_malware
```

Finally the last step was to install all of the required libraries to work on the program. This was done as the program was developed, but for concision the following command would install all of the required libraries:

```
pip3 install appdirs dnslib netifaces packaging
    pyparsing six
```

With the development environment properly set the development of the Intelligent malware could be started.

3.3. Setting up the testing Environment

To set up the test environment a Dell PowerEdge R230 server at university was used. The first step was installing an hypervisor to be able to create a full virtual environment. For this step the VMWare vSphere was chosen and installed. Then using the web management console the following virtual hardware was created and setup:

- Virtual Switch - Internal vSwitch
- Virtual Group - Internal VM Network
- Virtual Group - Internal Sniffer (Promiscuous mode)
- Virtual Machine - Router
- Virtual Machine - C&C
- Virtual Machine - IDS
- Virtual Machine - Victim 1
- Virtual Machine - Victim 2

The topology can be seen bellow on figure 1.

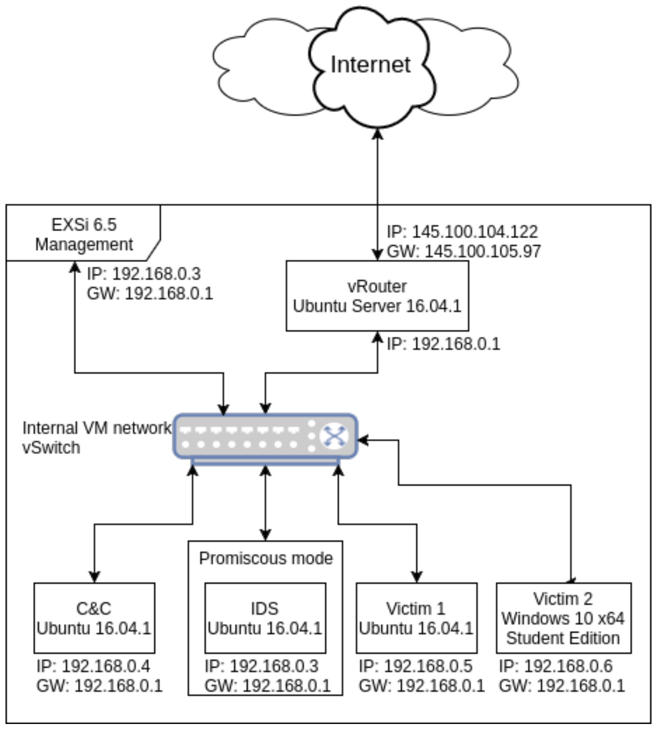

Figure 1. Testing environment network topology.

The Virtual group Internal sniffer was set in promiscuous mode so it can receive a copy of the switch's traffic and broadcast it to all VMs that are in the group. For our setup the only virtual machine in the internal sniffer group is the IDS. This prevents the other machines to get flooded with a copy of the traffic that is not supposed to be for them which would just result in a slower network without any advantages. As long as the IDS can receive a copy of the traffic, it can do its normal operations.

The Victim 1 and 2 were setup with Ubuntu and Windows respectively, updated and antivirus solution installed on the windows machine, for a non-technical user standard of security.

The Virtual router had to be set for access to the internal network virtual machines. This was accomplished by temporarily borrowing another server's public IP to access the router. Furthermore, a VPN solution (openVPN) has to be implemented in the router due to the server only having access to one public IP. With the router set with NAT forwarding and openVPN it was possible to contact each machine independently due to them being just one network hop away. This also enables the server management console to be contacted through an internal address which in turn enables for the setup to only need one public IP, releasing the borrowed one. With this the environment was set and ready to test the Intelligent malware.

The C&C was set up with an Ubuntu server and Overwatch as the C&C software that is going to act as the master to the malware infected devices. Overwatch was setup by cloning the git repository from Joey Dreijer and running the setup script it contains. The script downloads all necessary packages, sets up the database, creates a user and starts. It then can be accessed by browsing to its web service in a browser.

3.4. Intelligent Malware

The intelligent malware is simply a python script that gets executed after exploiting a vulnerability in the victim as seen in appendix A. The difference between the typical malware is that it does not activate its beaconing capabilities right away. It first scans the host for some extra information about the environment it is running. It then starts ”sniffing” for all incoming and outgoing traffic. This enables the scanner to look for specific methods of communication, such as:

- SSH (when C&C inside of internal network)
- Specific DNS queries/response
  - Signal
  - Whatsapp
  - Telegram
  - Facebook Messager
  - ...Any public service that does end to end encryption of data...

By looking specifically into DNS packets that have been captured, not only it can see that the host is interested in the service it requested the name resolution for, but also the name resolution itself. This enables, at a later stage, the beaconing operation to start without having to make a request by itself.

The implemented functionality is:

- Check for Root access
- Gather platform information
- Gather network interfaces information
- Sniffs on the network

To test the functionality the scanner just needs to be called using the python3 binary that points to the necessary libraries. To scanner can be called in the following way:

```
python3 scanner.py
```

For testing purposes where the script is called without a vulnerability being exploited the ”sudo” command needs to be used to grant root privileges. For testing it in the testing environment two virtual machines; victim1 (Ubuntu) had to call it with ”sudo” while for victim2 a command line just had to be opened with administrator rights.

4. Results

When running the experiment on victim 1, as the primary development platform, the scanner starts correctly gathering the correct information of the host, followed by the information of the network interface. Finally it enters sniffing mode and sniffs until interrupted. The output of the command can be found, partially, in appendix B.

The solution was then tested on Victim 2, as seen on appendix C, it starts the script correctly with the gathering of information about the host. It then gathers information about the network interfaces, but returns a registry key instead of the name of the network interface. It then creates successfully the network sockets (TCP and UPD) to start sniffing and crashes when it tries to read from them.

5. Discussion

For this project, a literature review was conducted. This led us to try to find weaknesses in the engine of snort 3, the most widely used IDS. This did not result in any findings, which led the research to change into the development of an intelligent solution. This idea has been well thought out, but development did not reach its final stages.

The research for weaknesses in the snort 3 engine did not reach its full potential due to the lack of time for such a task. We searched through all of the documentation of both snort 3 and its predecessor, snort 2, in hope to find a logical weakness in the rule making and/or checking operations that could be exploited. This proved to be a tedious and prone to error task that could take a very long time to accomplish. If the project would have more time to be researched this approach could have possibly wielded good results. These results would then have to be researched and analyzed properly to find a mean in which they could be used to evade IDS systems.

With the decision to change approach we started with the development of a one of a kind Intelligent malware that can respond to the differences in environment where it is executed. This was done by adding the functionality of checking for host information and by sniffing on the traffic of the host. With the possibility of knowing what traffic is supposed to leave the host, it is possible to prevent the use of any anomalous type of communication. Furthermore, the use of public service enables the malware to stay hidden since it does not touch the protocols of the services. The only thing the malware changes in the communication is the data that is sent. Due to end to end encryption of the services is is impossible to inspect the data rendering it impossible to detect differences between malicious and regular traffic.

Python was chosen for this task due to it providing many packages for cross platform use. Furthermore, it is easy to provide a portable version of it with the required libraries for the script to work out of the box. The script shows an unfinished program. Due to the lack of time it was not possible to implement all of the functionality. The script was tested on both Linux and Windows platform but due to the development of the script have been focused on the Linux platform it is not fully compatible with Windows. This can be seen in the Windows results in appendix C. Furthermore, in Linux despite the program running perfectly fine and sniffing the network it was detected too late that the captured packets are only of incoming traffic, unless it is a response to a query. This is not desired due to the need of knowing if the host is starting any outgoing ssh connections during the information gathering period. This means that a big part of the code needs to be changed to accommodate the change in the capture behavior.

Despite not having been tested for full proof of it working the base program as some countermeasures against it, such as the use of Host Intrusion Detection systems (HIDS). With the use of a HIDS it is possible to detect the running script before it event starts communicating. The HIDS if deployed in a centralized fashion could be used to isolate the infected host from the network. This would render the communication impossible. A possible solution against this countermeasure would be to only act when the CPU or Network would have an activity spike. Another solution would be having a countdown timer between captured traffic. This would act as a way to detect network isolation, and would activate the self deletion if it would reach 0 before the next packet would be received.

6. Future Work

The field is very broad and so much more work can be put into it. One such possibility is to take again the task of finding a weakness in the snort 3 engine which could render it unable to create a signature. Due to snort being open source, and With more time it could be beneficial the review of the source code.

The project was heavily based on signature detection, but it would be a great contribution if work would also be done on other types of detection such as anomaly based.

As explained in the discussion the script, that can be found in appendix A, is far from being finished. The following functionality still needs to be implemented:

- Outgoing traffic capture
- Capture timer
- Data analysis
- Decision making
- Download of appropriate communication module
- Windows compatibility
- No root access operations
- Self-deletion

The above functionality is the original meant to be included as the base. Further functionality could be added such as:

- Self-preservation (across reboots)
- Process masking (changing the name of the process so it is more difficult to be found)
- CPU and Network activity exploitation (Act on activity spikes of the CPU and Network)
- Increase the List of public services it can use
- Analise specific configurations such as SSH (to dynamically change what port it checks)
- execution option for a small degree of control

Lastly the scanner only does the scanning of the host but not the communication in itself. Each type of communication is meant to have its own script that is downloaded and executed by the scanner. For this reason work could also be done in making the cli of each service portable and usable by the script or making manually the implementation of the different protocols.

7. Conclusion

In Conclusion, a high amount of work and effort has been put into this project on dynamic profiles for malware communication. Different approaches were conducted, including finding weaknesses in IDS software that could be exploitable, with little to no results. Furthermore, a program that dynamically adapts to its environment was devised and partially implemented. The idea seems to be viable but needs further work. Due to time constraints, the program was not finished, it is not cross platform and requires some polishing on what has been implemented. Future work could be done into different methods of malware detection, improving the solution with extra functionality and finding weaknesses in IDS software.

Acknowledgments

The authors would like to thank Cedric van Bockhaven & Joey Dreijer for their, very appreciated, advice and contribution.

References

[Acarali et al., 2016] Acarali, D., Rajarajan, M., Komninos, N., and Herwono, I. (2016). Survey of approaches and features for the identification of http-based botnet traffic. *Journal of Network and Computer Applications*, 76:1–15.

[Chen et al., 2013] Chen, P., Desmet, L., and Huygens, C. (2013). A study on advanced persistent threats. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 248–254. IEEE.

[Collins et al., 2007] Collins, M. P., Shimeall, T. J., Faber, S., Janies, J., Weaver, R., De Shon, M., and Kadane, J. (2007). Using uncleanliness to predict future botnet addresses. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 93–104. ACM.

[García et al., 2011] García, S., Zunino, A., and Campo, M. (2011). Botnet behavior detection using network synchronism. *Privacy, Intrusion Detection and Response: Technologies for Protecting Networks*, pages 122–144.

[García et al., 2014] García, S., Zunino, A., and Campo, M. (2014). Survey on network-based botnet detection methods. *Security and Communication Networks*, 7(5):878–903.

[Goldring, 2003] Goldring, T. (2003). User profiling for intrusion detection in windows nt. *Computing Science and Statistics*, 35.

[Gu et al., 2008a] Gu, G., Perdisci, R., Zhang, J., Lee, W., et al. (2008a). Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX Security Symposium*, volume 5, pages 139–154.

[Gu et al., 2007] Gu, G., Porras, P. A., Yegneswaran, V., Fong, M. W., and Lee, W. (2007). Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Usenix Security*, volume 7, pages 1–16.

[Gu et al., 2008b] Gu, G., Zhang, J., and Lee, W. (2008b). Botsniffer: Detecting botnet command and control channels in network traffic.

[Karim et al., 2014] Karim, A., Salleh, R. B., Shiraz, M., Shah, S. A. A., Awan, I., and Anuar, N. B. (2014). Botnet detection techniques: review, future trends, and issues. *Journal of Zhejiang University SCIENCE C*, 15(11):943–983.

[Katzenbeisser and Petitcolas, 2016] Katzenbeisser, S. and Petitcolas, F. A. (2016). *Information hiding*. Springer.

[Kim et al., 2012] Kim, D. H., Lee, T., Kang, J., Jeong, H., and In, H. P. (2012). Adaptive pattern mining model for early detection of botnet-propagation scale. *Security and Communication Networks*, 5(8):917–927.

[Lampson, 1973] Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM*, 16(10):613–615.

[Li et al., 2010] Li, Z., Wang, B., Li, D., Chen, H., Liu, F., and Hu, Z. (2010). The aggregation and stability analysis of network traffic for structured-p2p-based botnet detection. *JNW*, 5(5):517–526.

[Liao et al., 2013] Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

[Limarunothai et al., 2015] Limarunothai, R., Munlin, M., et al. (2015). Trends and challenges of botnet architectures and detection techniques. *Journal of Information Science & Technology*, 5(1).

[Lu et al., 2011] Lu, W., Rammidi, G., and Ghorbani, A. A. (2011). Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, 34(3):502–514.

[Passerini E, 2008] Passerini E, Paleari R, M. L. B. D. (2008). Fluxor : detecting and monitoring fast-flux service networks. *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 186–206.

[Rodríguez-Gómez et al., 2013] Rodríguez-Gómez, R. A., Maciá-Fernández, G., and García-Teodoro, P. (2013). Survey and taxonomy of botnet research through life-cycle. *ACM Computing Surveys (CSUR)*, 45(4):45.

[Rossow and Dietrich, 2013] Rossow, C. and Dietrich, C. J. (2013). Provex: Detecting botnets with encrypted command and control channels. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 21–40. Springer.

[Soniya and Wilscy, 2016] Soniya, B. and Wilscy, M. (2016). Detection of randomized bot command and control traffic on an end-point host. *Alexandria Engineering Journal*, 55(3):2771–2781.

[Strayer et al., 2006] Strayer, W. T., Walsh, R., Livadas, C., and Lapsley, D. (2006). Detecting botnets with tight command and control. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 195–202. IEEE.

[Wikipedia, 2017] Wikipedia (2017). Pip (package manager) — wikipedia, the free encyclopedia. [Online; accessed 11-February-2017].

[Wurzinger et al., 2009] Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., and Kirda, E. (2009). Automatically generating models for botnet detection. In *European symposium on research in computer security*, pages 232–249. Springer.

[Yen and Reiter, 2008] Yen, T.-F. and Reiter, M. K. (2008). Traffic aggregation for malware detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 207–227. Springer.

[Yu et al., 2010] Yu, X., Dong, X., Yu, G., Qin, Y., Yue, D., and Zhao, Y. (2010). Online botnet detection based on incremental discrete fourier transform. *JNW*, 5(5):568–576.

Appendix A.

Github Repository: https://github.com/JCMarques15/Dynamic_Malware

Appendix B.

Linux results:

```
jmarques@desktop-12 ~/git/Dynamic_Malware   m a s t e r *    sudo python3 scanner.py
[INFO] The scanner has root access
[INFO] The host machine is Linux based, with the architecture ('64bit', 'ELF') and kernel 4.4.0-62-
    generic
[INFO] It is running Ubuntu version 16.04
[INFO] Ip address for eno1 is: 145.100.102.142
[DEBUG] [['eno1', '145.100.102.142']]
[INFO] TCP Socket created
[INFO] UDP Socket created
#####################################
TCP Packet Num: 1
Version: 4
IP Header Length: 5
TTL: 60
Protocol: 6
Source Address: 104.98.141.72
Destination Address: 145.100.102.142
Source Port: 443
Dest Port: 58700
Sequence Number: 1902228493
Acknowledgement: 2165171968
TCP header length: 10
Data:
<omitted>
#####################################
TCP Packet Num: 2
Version: 4
IP Header Length: 5
TTL: 60
Protocol: 6
Source Address: 104.98.141.72
Destination Address: 145.100.102.142
Source Port: 443
Dest Port: 58700
Sequence Number: 1902228494
Acknowledgement: 2165172485
TCP header length: 8
Data:
<omitted>
#####################################
TCP Packet Num: 3
Version: 4
IP Header Length: 5
TTL: 60
Protocol: 6
Source Address: 104.98.141.72
Destination Address: 145.100.102.142
Source Port: 443
Dest Port: 58700
Sequence Number: 1902228494
Acknowledgement: 2165172485
TCP header length: 8
Data:
<omitted>
#####################################
TCP Packet Num: 4
Version: 4
IP Header Length: 5
TTL: 60
Protocol: 6
Source Address: 104.98.141.72
Destination Address: 145.100.102.142
Source Port: 443
Dest Port: 58700
Sequence Number: 1902228640
Acknowledgement: 2165172589
TCP header length: 8
```

```
Data:
<omitted>
...
```

Appendix C.

Windows results:

```
C:\Users\jmarques\Downloads\Dynamic_Malware-master\Dynamic_Malware-master>python scanner.py
[INFO] The scanner has root access
[INFO] The host machine is Windows based, with the architecture ('32bit', 'WindowsPE')
[INFO] It is running Windows 10 version 10.0.14393
[INFO] Ip address for {2EF7B488-0AA1-4594-AFAD-C9C0BA083157} is: 192.168.0.6
[INFO] Ip address for {D410E957-E244-11E6-8000-806E6F6E6963} is: 127.0.0.1
[DEBUG] [['{2EF7B488-0AA1-4594-AFAD-C9C0BA083157}', '192.168.0.6'], ['{D410E957-E244-11E6-8000-806
    E6F6E6963}', '127.0.0.1']]
[INFO] TCP Socket created
[INFO] UDP Socket created
Exception in thread Thread-1:
Traceback (most recent call last):
  File "C:\Users\jmarques\AppData\Local\Programs\Python\Python36-32\lib\threading.py", line 916, in
      _bootstrap_inner
    self.run()
  File "C:\Users\jmarques\AppData\Local\Programs\Python\Python36-32\lib\threading.py", line 864, in run
    self._target(*self._args, **self._kwargs)
  File "scanner.py", line 128, in tcp_sniffer
    packet = tcp_soc.recvfrom(65565)
OSError: [WinError 10022] An invalid argument was supplied

Exception in thread Thread-2:
Traceback (most recent call last):
  File "C:\Users\jmarques\AppData\Local\Programs\Python\Python36-32\lib\threading.py", line 916, in
      _bootstrap_inner
    self.run()
  File "C:\Users\jmarques\AppData\Local\Programs\Python\Python36-32\lib\threading.py", line 864, in run
    self._target(*self._args, **self._kwargs)
  File "scanner.py", line 154, in udp_sniffer
    packet = udp_soc.recvfrom(65565)
OSError: [WinError 10022] An invalid argument was supplied
```