OnionBots: Subverting Privacy Infrastructure for Cyber Attacks

Amirali Sanatinia, Guevara Noubir
College of Computer and Information Science
Northeastern University, Boston, USA
{amirali,noubir}@ccs.neu.edu

***Abstract*—Over the last decade botnets survived by adopting a sequence of increasingly sophisticated strategies to evade detection and take overs, and to monetize their infrastructure. At the same time, the success of privacy infrastructures such as Tor opened the door to illegal activities, including botnets, ransomware, and a marketplace for drugs and contraband. We contend that the next waves of botnets will extensively attempt to subvert privacy infrastructure and cryptographic mechanisms. In this work we propose to preemptively investigate the design and mitigation of such botnets. We first, introduce OnionBots, what we believe will be the next generation of resilient, stealthy botnets. OnionBots use privacy infrastructures for cyber attacks by completely decoupling their operation from the infected host IP address and by carrying traffic that does not leak information about its source, destination, and nature. Such bots live symbiotically within the privacy infrastructures to evade detection, measurement, scale estimation, observation, and in general all IP-based current mitigation techniques. Furthermore, we show that with an adequate self-healing network maintenance scheme, that is simple to implement, OnionBots can achieve a low diameter and a low degree and be robust to partitioning under node deletions. We develop a mitigation technique, called SOAP, that neutralizes the nodes of the basic OnionBots. In light of the potential of such botnets, we believe that the research community should proactively develop detection and mitigation methods to thwart OnionBots, potentially making adjustments to privacy infrastructure.**

I. Introduction

Over the last decade botnets rose to be a serious security threat. They are routinely used for denial of service attacks, spam, click frauds, and other malicious activities [1]. Both the research and industry communities invested a significant effort analysing, developing countermeasures, and products to effectively detect, cripple, and neutralize botnets. While some countermeasures operate on user computers, most are deployed at the ISP and enterprise levels. Many botnets were successfully neutralized by shutting down or hijacking their Command and Control (C&C) servers, communications channels (e.g., IRC), reverse engineering the algorithm used for the domain name generation (DGA) and preemptively blocking the access to these domains [2]. Such mitigation techniques exploit the fact that most current botnets rely on primitive communication architectures and C&C mechanisms. This forced botnet developers to continuously adapt raising the level of sophistication of their design from the early static and centralized IRC or fixed servers' IP addresses to more sophisticated fast-fluxing [3] and even straightforward use of Tor hidden services [4], [5].

In this paper, we are interested in investigating the next level of this arm-race. We contend that the next wave of botnets' sophistication will rely on subverting privacy infrastructure and a non-trivial use of cryptographic mechanisms. The Tor project was very successful in building an infrastructure that protects users identity over the Internet and allowing one to host Internet servers without revealing her or his location using the Tor hidden services feature. Evidence of our predictions can be found in the malicious use of hidden services for hosting the infamous silk road [6], instances of the Zeus [4] botnet, and the hosting of the CryptoLocker ransomware C&C server [7]. Interestingly, CryptoLocker combines Tor with the use of another privacy "infrastructure", bitcoin the crypto currency, for the ransom payment. The combination of Tor and bitcoin make it possible today to blackmail Internet users, anonymously be paid, and get away with it.

The current use of Tor and crypto-mechanisms in botnets is still in its infancy stages. Only hosting the C&C server as a hidden service still allows the detection, identification, and crippling of the communication between the bots. Recent research demonstrated that it is possible to successfully deny access to a single or few `.onion` server [8]. To assess the threat of crypto-based botnets, we advocate a preemptive analysis, understanding of their potential and limitations, and the development of mitigation techniques.

In this paper, we present the design of a first generation of non-trivial OnionBots. In this Basic OnionBot, communication is exclusively carried out through hidden services. No bot (not even the C&C) knows the IP address of any of the other bots. At any instant, a given bot is only aware of the *temporary* `.onion` address of a very small (constant) number of bots. Bots relay packets but cannot distinguish the traffic based on their source, destination, or nature. At the same time, the bot master is able to access and control any bot, anytime, without revealing his identity. We show that this design is resilient to current mitigations and analysis techniques from botnet mapping, hijacking, to even assessing the size of the botnet. We also show that the proposed Neighbors-of-Neighbor graph maintenance algorithm achieves a low diameter, degree, and high resiliency and repair in the event of a take-down (e.g., Tor DoSing or node capture/cleanup) of a significant fraction of the botnet nodes. Since our goal is to preemptively prevent the emergence of OnionBots, we also propose a novel mitigation technique against the Basic OnionBots. This technique exploits the same stealthy features of the OnionBot (namely peers not knowing each other's identities) to neutralize the bots. The technique called SOAP, gradually surrounds the bots by clones (or sybils) until the whole botnet is fully contained. Our goal

is to draw the attention of the community to the potential of OnionBots and develop preemptive measures to contain them and ideally prevent their occurence.

Our contributions are summarized as follows:

- A novel reference design for a OnionBotnet whose command, communication, and management are fully anonymized within the Tor privacy infrastructure.
- A communication topology with repair mechanisms that minimizes the nodes' degree, graph diameter, and maximizes resiliency.
- A performance evaluation and discussion of resiliency to various takedown attacks such as simultaneous denial of service attacks against multiple `.onion` botnet nodes.
- A Sybil mitigation technique, SOAP, that neutralizes each bot by surrounding it by clones.

We first survey the current state of botnet design and mitigation techniques in Section II, followed by a review of the key features of the Tor privacy infrastructure in Section III. In Section IV, we present our proposed reference design for an OnionBotnet. We evaluate the resiliency and performance of the OnionBotnet, using several metrics in Section V. We finally investigate potential mechanisms to prevent the rise of such botnets in Section VI.

II. Current Botnets & Mitigations

We first review the evolution of botnets and why we believe the next generation of botnets would subvert privacy infrastructures to evade detection and mitigation. Currently, bots are monitored and controlled by a botmaster, who issues commands. The transmission of theses commands, which are known as C&C messages, can be centralized, peer-to-peer or hybrid [9]. In the centralized architecture the bots contact the C&C servers to receive instructions from the botmaster. In this construction the message propagation speed and convergence is faster, compared to the other architectures. It is easy to implement, maintain and monitor. However, it is limited by a single point of failure. Such botnets can be disrupted by taking down or blocking access to the C&C server. Many centralized botnets use IRC or HTTP as their communication channel. GT-Bots, Agobot/Phatbot [10], and clickbot.a [11] are examples of such botnets. A significant amount of research focused on detecting and blocking them [12], [13], [14], [15], [16], [17]. To evade detection and mitigation, attackers developed more sophisticated techniques to dynamically change the C&C servers, such as: Domain Generation Algorithm (DGA) and fast-fluxing (single flux, double flux).

Single-fluxing is a special case of fast-flux method. It maps multiple (hundreds or even thousands) IP addresses to a domain name. These IP addresses are registered and de-registered at rapid speed, therefore the name fast-flux. These IPs are mapped to particular domain names (e.g., DNS A records) with very short TTL values in a round robin fashion [3]. Double-fluxing is an evolution of single-flux technique, it fluxes both IP addresses of the associated fully qualified domain names (FQDN) and the IP addresses of the responsible DNS servers (NS records). These DNS servers are then used to translate the FQDNs to their corresponding IP addresses. This technique provides an additional level of protection and redundancy [3]. Domain Generation Algorithms (DGA), are the algorithms used to generate a list of domains for botnets to contact their C&C. The large number of possible domain names makes it difficult for law enforcements to shut them down. Torpig [18] and Conficker [19] are famous examples of such botnets.

A significant amount of research focuses on the detection of malicious activities from the network perspective, since the traffic is not anonymized. For example [20], [21], [22], [23], [24], [25] inspect the DNS traffic and use machine learning clustering and classification algorithms. BotFinder [26] uses the high-level properties of the bot's network traffic and employs machine learning to identify the key features of C&C communications. DISCLOSURE [27] uses features from NetFlow data (e.g., flow sizes, client access patterns, and temporal behavior) to distinguish C&C channels. Other work [28], [29] focus on endpoints' static metadata properties and the order of the high-level system events for threat classification.

The next step in the arms race between attackers and defenders was moving from a centralized scheme to a peer-to-peer C&C. Storm [30], Nugache [31], Walowdac [32] and Gameover Zeus [33] are examples of such botnets. Some of these botnets use an already existing peer-to-peer protocol, while others use customized protocols. For example, earlier versions of Storm used Overnet, and the new versions use a customized version of Overnet, called Stormnet [30]. Meanwhile other botnets such as Walowdac and Gameover Zeus organize their communication channels in different layers.

Previous work studied specific mitigations against peer-to-peer botnets. For example, BotGrep [34] uses the unique communication patterns in a botnet to localize its members by employing structured graph analysis. Zhang et al. [35] propose a technique to detect botnet P2P communication by fingerprinting the malicious and benign traffic. Yen and Reiter [36] use three features (peer churn, traffic volume and differences between human-driven and bot-driven behavior) in network flow to detect malicious activity. Coskun et al. [37] propose a method to detect the local members of an unstructured botnet by using the mutual contacts. As we can see, some of these techniques rely on observing the unencrypted traffic, therefore by using a privacy infrastructure such as Tor they can be evaded.

Very recently the use of Tor received more attention from malware and botnet authors. For example, the new 64-bit Zeus employs Tor anonymity network in its botnet infrastructure [4]. It creates a Tor hidden service on the infected host and the C&C can reach these infected hosts using their unique `.onion` address through Tor. Another example is ChewBacca [5], which uses Tor, and logs the keystrokes of the infected host and reports them back to the botmaster. The C&C is an HTTP server that is hosted as a hidden service. Although using Tor and hidden services makes the detection and mitigation more difficult, these bots are still using the basic client-server model. This leaves them open to single point of failure.

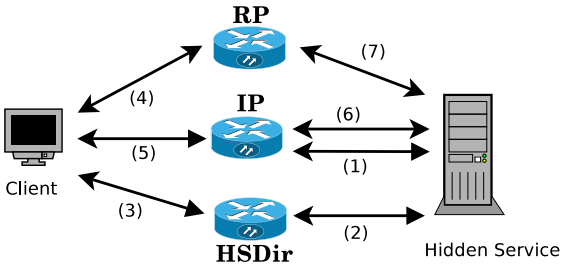


Fig. 1: Tor hidden service structure

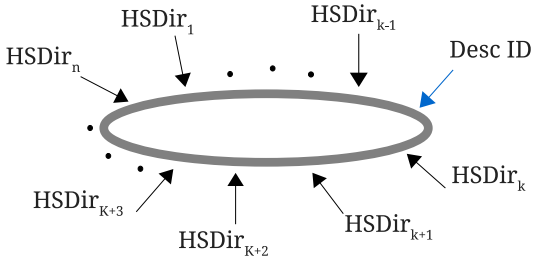


Fig. 2: Tor hidden service Directories (HSDir) Fingerprint

III. Privacy Infrastructure: Tor

We envision OnionBots to rely on Tor for their operation. To better understand their potential and limitations, we briefly review the structure of Tor and hidden services. Tor [38] is the most widely used distributed low-latency anonymity-network. It helps users to resist censorship, and protects their personal privacy. Furthermore, it allows users to hide their activities and location from government agencies and corporations. Clients establish anonymous communication by relaying their traffic through other Tor relays, called Onion Routers (OR). A client builds a circuit with the relays by negotiating symmetric keys with them. After building the circuit, the client sends the data in fixed sized `cells` and encrypts them in multiple layers, using the previously negotiated keys. Besides providing anonymous communication for clients, current implementation of Tor also offers anonymity for servers through hidden services.

The Tor hidden service architecture is composed of the following components:

- *Server*, that runs a service (e.g., a web server).
- *Client*, that wishes to access the server.
- *Introduction Points (IP)*, a set of Tor relays, chosen by the hidden service, that forward the initial messages between the server and the client's Rendezvous Point.
- *Rendezvous Point (RP)*, a Tor relay randomly chosen by the client that forwards the data between the client and the hidden service.
- *Hidden Service Directories (HSDir)*, a set of Tor relays chosen by the server to store its descriptors.

In order to make a service available via Tor, Bob (the service provider) generates an RSA key pair. The first 10 bytes of the SHA-1 digest of the generated RSA public key becomes the `Identifier` of the hidden service. The `.onion` hostname, is the base-32 encoding representation of the public key. As Figure 1 illustrates the following steps take place, in order to connect to a hidden service. Bob's Onion Proxy (OP) chooses a set of Tor relays to serve as his Introduction Points and establishes a circuit with them (step 1). After making the circuits, he computes two different service descriptors that determine which Tor relays should be chosen as its HSDirs (step 2). HSDirs are responsible for storing the hidden service descriptors, which change periodically every 24 hours and are chosen from the Tor relays that have the HSDir flag. This flag is obtained by relays that have been active for at least 25 hours. Later, in section VI we discuss the critical role that HSDirs can paly in mitigating OnionBots.

```
descriptor-id = H(Identifier || secret-id-part)
secret-id-part = H(time-period || descriptor-cookie
|| replica)
time-period = (current-time +
permanent-id-byte * 86400 / 256) / 86400
```

`H` denotes the SHA-1 hash digest. `Identifier` is the 80 bit fingerprint (truncated SHA-1 digest of the public key) of the hidden service. `Descriptor-cookie` is an optional 128 bit field. It can be used to provide authorization at the Tor network level, and it prevents unauthorized clients from accessing the hidden service. `Time-period` is used to periodically change the responsible HSDirs, and making the system more resilient. The `permanent-id-byte` prevents the descriptors from changing all at the same time. `Replica` takes values of 0 or 1. It is used to compute two different sets of descriptor IDs for a hidden service. The current implementation of Tor distributes each set of descriptor IDs in 3 different HSDirs. Therefore, for each hidden service there are a total of 6 responsible HSDirs. The list of Tor relays, which is called the `consensus` document, is published and updated every hour by the Tor authorities. If we consider the circle of the fingerprint of Tor relays as depicted in Figure 2, then if the descriptor ID of the hidden service falls between the fingerprint of HSDir_{k-1} and HSDir_k, it will be stored on HSDir_k, HSDir_{k+1} and HSDir_{k+2}.

When a client (Alice) wants to contact a hidden service, she first needs the hostname of the service. Then from the `.onion` hostname, she computes the `Descriptor ID` of the hidden service and the list of its responsible HSDirs (step 3). To communicate with the hidden service, Alice first randomly chooses a Tor relay as her Rendezvous Point and makes a circuit to it (step 4). She then builds a new circuit to one of Bob's Introduction Points, and sends it a message. This message includes the Rendezvous Point's info and the hash of the public key of the hidden service (step 5). If the public key (sent by Alice) is recognized by the Introduction Point, it will forward the information to the hidden service's OP (step 6). When Bob's OP receives the message it extracts the Rendezvous Point information and establishes a circuit with it

(step 7).

This approach protects client's (Alice) IP address from Bob, and hides hidden service's (Bob) IP address from Alice, thus providing mutual anonymity for both the client and the server.

IV. OnionBot: a Cryptographic P2P Botnet

In this section, we look at the details of the proposed OnionBot, a novel non IP based construction that is immune to the current mitigation techniques. We explore different bootstrapping approaches and a distributed, self-healing, low-degree, low-diameter overlay peer-to-peer network formation.

A. Overview

OnionBot retains the life cycle of a typical peer-to-peer bot [39]. However, each stage has unique characteristics that make OnionBots different from current peer-to-peer botnets. As a result, existing solutions are not applicable to them. For example, in the infection stage, each bot creates a `.onion` address and generates a key to indistinguishably encrypt the messages. In the rally stage, the bots dynamically peer with other bots that are the foundation of a self-healing network. Furthermore, while at the waiting stage, bots periodically change their address to avoid detection and mitigation. These new `.onion` addresses are generated from the key that is shared with the botmaster. This allows the botmaster to access and control any bot through the shared key, anytime, without revealing his identity.

Infection: is the first step in recruiting new bots. It can happen through traditional attack vectors such as phishing, spam, remote exploits, drive-by-download or zero-day vulnerabilities [40]. A great body of literature have focused on different spread mechanisms [41], [42], [43]. In this work we focus on the remaining stages of a bot's life cycle.

Rally: in order to join a botnet, the newly infected bots need to find the already existing members of the network. In peer-to-peer network this process is called bootstrapping. For clarity reasons we use the same terminology in describing OnionBots. Based on the requirements of the network, the complexity and flexibility of bootstrapping techniques varies significantly. OnionBots necessitate a distributed mechanism to maintain a low-degree, low-diameter network. Such requirements, demands a bootstrapping mechanism that is able to evolve with the network. In section IV-B we discuss different techniques and their ramifications in more detail.

Waiting: in this state, a bot is waiting for commands from the botmaster. Generally the command transmissions can be pull-based (bots make periodic queries to the C&C) or push-based (botmaster sends the commands to the bots), and there are trade-offs in each mechanism. For example, in the pull-based approach, if bots aggressively make queries for the C&C messages, it allows faster propagation of commands. However, it results in easier detection of C&C and the bots. In the push-based approach, it is important to be able to reach to each bot, within a reasonable number of steps. Furthermore, to prevent leakage of information about the botnet operation and topology, it should not be feasible for an adversary to distinguish the source, destination and the nature of the messages. Meanwhile, satisfying such requirements is not trivial in self-healing networks. Later in section IV-D we discuss how in OnionBots, the botmaster is able to access and control any bot.

Execution: at this stage the bots execute the commands given by the botmaster (e.g., bitcoin mining, sending spam [44] or DDoS attack [45], [46]), after authenticating them. Recently, botmasters started offering botnet-as-a-service [47] as it was previously predicted by researchers in 2008 [48]. Considering that the OnionBots make use of cryptographic block beyond the basic, trivial encryption/decryption of payloads, it allows them to offer the botnet for rent. In section IV-E, we explain how this can be done, in a distributed way, and without further intervention of the botmaster.

In the next sections we will focus on describing the key mechanisms of OnionBots.

B. Bootstrap

As mentioned previously the bootstrapping is an essential part of network formation in peer-to-peer networks. Additionally, in OnionBots, it provides the foundation for the self-healing graph construction. In the following, we study different approaches and their trade-offs. We discuss how these concepts should be adapted to the context of a privacy infrastructure such as Tor. Note that, the address of a peer list in our protocol refers to the `.onion` address of the peers, unless stated otherwise.

- *Hardcoded peer list*: in this setting each bot has a list of other bots to contact at the beginning. Since the infections can be carried out by the bots, the new peer lists can be updated. Each peer upon infecting another host sends a subset of its peer list. Each node in the original peer list will be included in the subset with probability p. In the conventional botnets this scheme is vulnerable to detection and blacklisting. However, in OnionBots, the `.onion` address is decoupled form IP address, and changes periodically as it is described in section IV-D. Therefore, the current mitigations are not applicable.
- *Hotlists (webcache)*: this is conceptually similar to the hardcoded peer list. However each bot has a list of peers to query for the other peers. In this setting, the adversary (defenders) will only have access to a subset of servers, since each bot only has a subset of the addresses, and these subsets can be updated upon infection or later in the waiting stage.
- *Random probing*: in this scheme a bot randomly queries the list of all possible addresses, until it finds a bot listening on that address. Although it can be used in IPv4 and IPv6 [49] networks, it is not feasible in the context of Tor `.onion` addresses. Since the address space is intractable (to craft an address with specific first 8 letters, it takes about 25 days [50]); Randomly querying all possible `.onion` addresses, requires probing an address space of size 32^{16}.
- *Out-of-band communication*: the peer list can be transmitted through another infrastructure. For example, by using a peer-to-peer network such as BitTorrent and Mainline DHT to store and retrieve peer lists, or by

using social networks, such as Twitter, Facebook or YouTube.

We envision that OnionBots would use a customized approach based on hardcoded peer list and hotlists. As mentioned earlier, in OnionBots the blacklisting of nodes is not practical, since their addresses change periodically. In the following section we describe how OnionBots address the bootstrapping and recruitment during network formation and maintenance.

C. Maintenance of the OnionBot Communication Graph

OnionBots form a peer-to-peer, self-healing network that maintains a low degree and a low diameter with other bots to relay messages. Peer-to-peer networks are broadly categorized as structured and unstructured [51], where both categories are used by botnets, and are studied in previous work [31], [30], [52]. However, the already existing peer-to-peer networks are generic in terms of their operations. Therefore, their design and resiliency is based on different assumptions and requirements. In the following, we propose a Dynamic Distributed Self Repairing (DDSR) graph, a new peer-to-peer construction that is simple, stealthy and resilient. Furthermore, it is an overlay network, formed over a privacy infrastructure such as Tor.

Neighbors of Neighbor Graph: In this section, we introduce DDSR graph construct that is used in the network formation of OnionBots. The proposed construct is inspired by the knowledge of Neighbors-of-Neighbor (NoN). Previous work [53] studied the NoN greedy routing in peer-to-peer networks, where it can diminish route-lengths, and is asymptotically optimal. In this work we discuss how NoN concepts can be used to create a self-healing network.

Consider graph G with n nodes (V), where each node $u_i \in V$, $0 \leq i < n$, is connected to a set of nodes. The neighbors of u_i, are denoted as $N(u_i)$. Furthermore, u_i has the knowledge of nodes that are connected to $N(u_i)$. Meaning that each node also knows the identity of its neighbor's neighbors. In the context of our work the identity is the `.onion` address.

Repairing: When a node u_i is deleted, each pair of its neighbors u_j, u_k will form an edge (u_j, u_k) if $(u_j, u_k) \notin E$, where E is the set of existing edges. Figure 3 depicts the node removal and the self repairing process in a 3-*regular* graph with 12 nodes. The dashed red lines indicate the newly established links between the nodes. For example, as we can see if we remove one of the nodes (7), its neighbors (0, 1, 4) start to find a substitute for the deleted node (7), to maintain the aforementioned requirements. In this case the following edges are created: (0, 1), (1, 4), and (1, 4).

The basic DDSR graph outlined in the previous paragraph does not deal with the growth in the connectivity degree of each node, denoted by $d(u)$; after multiple deletions the degree of some nodes can increase significantly. Such increase of the nodes' degree is not desirable for the resiliency and the stealthy operation of the botnet. Therefore, we introduce a pruning mechanism to keep the nodes' degree in the range $[d_{min}, d_{max}]$. Note that d_{min} is only applicable as long as there are enough surviving nodes in the network.

Pruning: Consider the graph G, when a node u_i is deleted, each one of its neighbors, starts the repairing process. However this scheme causes the degree of the neighbors of node u_i, to increase significantly after t steps (deletions). To maintain the degree in the aforementioned range($[d_{min}, d_{max}]$), each neighboring node of the deleted node (u_i), deletes the highest degree node from its peer list. If there is more than one such candidate, it randomly selects one among those for deletion, until its degree is in the desired range. Removing the nodes with the highest degree, maintains the reachability of all nodes, and the connectivity of the graph.

Forgetting: In the proposed OnionBot, nodes forget the `.onion` address of the pruned nodes. Additionally, to avoid discovery, mapping and further blocking, each bot can periodically change his `.onion` address and announce the new address to his current peer list. The new `.onion` address is generated based on a secret key and time. This periodic change is possible because of the decoupling between IP address and the bots, which is provided by Tor. Later, in section IV-D we explain how the C&C is able to directly reach each bot, even after they change their address.

D. Command and Control Communication

In this section, we show how Tor enables a stealthy, and resilient communication channel with C&C. As mentioned before, OnionBot is a non IP address based construction, therefore current techniques to detect and disable the C&C (e.g., IP blacklisting and DNS traffic monitoring) are ineffective against it.

In OnionBot we assume two classes of messages: 1) messages from C&C to the bots and 2) messages from bots to C&C. The messages from C&C can be either directed to an individual node(s) (e.g., a maintenance message telling a bot to change it peers) or directed to all bots (e.g., DDoS attack on *example.com*). Furthermore, the botmaster can setup group keys to send encrypted messages to a group of bots. While a bot can tell the difference between a broadcast message and messages directed to an individual bot(s), it is not able to identify the source, the destination and the nature of these messages. Therefore the authorities are not able to detect different messages and drop harmful message and only allow the maintenance message to pass through. As a result they can not create the illusion that the botnet is operational, when it is actually neutralized.

In OnionBot, the bots report their address to C&C, and establish a unique key to be shared with the botmaster at the infection/rally stage. This allows C&C to have direct access to the bots, even after they change their `.onion` address. Each bot generates a symmetric key, K_B, and reports it to the C&C. K_B is encrypted with the hard coded public key of the C&C ($\{K_B\}_{PK_{CC}}$). After establishing the key, bots can periodically change their `.onion` address based on a new private key generated using the following recipe, `generateKey(`PK_{CC}`, H(`K_B`, `i_p`))`. Where, `H` is a hash function, and i_p is the index of period (e.g., day). All messages are of the same fixed size, as they are in Tor. Furthermore, to achieve indistinguishability between all messages, we use constructions that generate indistinguishable encodings from uniform random strings, such as Elligator [54]. As a result no information is leaked to the relaying bots.

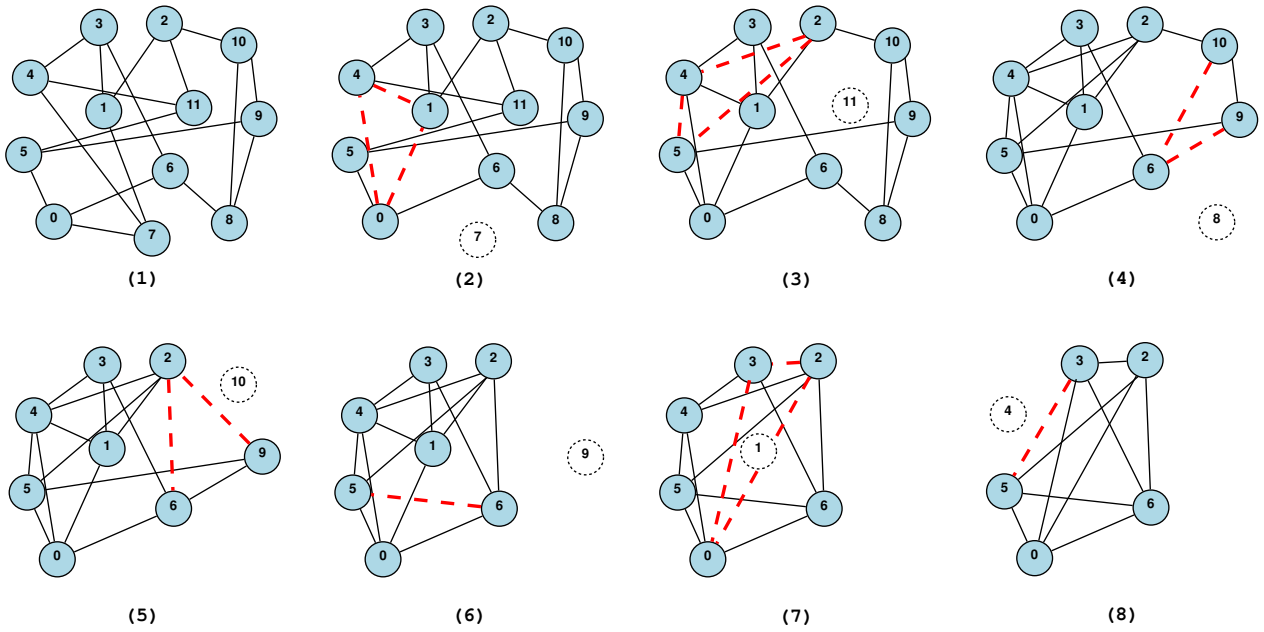


Fig. 3: Node removal and the self repairing process in a 3-*regular* graph with 12 nodes. The dashed red lines, indicate the newly established links between the nodes.

Botnet	Crypto	Signing	Replay
Miner	none	none	yes
Storm	XOR	none	yes
ZeroAccess v1	RC4	RSA 512	yes
Zeus	chained XOR	RSA 2048	yes

TABLE I: Cryptographic use in different botnets.

E. Operation

While many current botnets lack adequate secure communications [55] (e.g., sending messages in plaintext) that leaves them open to hijacking, the OnionBot's communication is completely encrypted since it uses Tor and SSL. Note that the the encryption keys are unique to each link. Table I summarizes a number of botnet families and their lack of adequate crypto blocks [55]. Furthermore, we introduce new cryptographic blocks that enable the OnionBot to offer new services, such as a botnet-for-rent [48] and a distributed computation platform for rent.

To achieve the aforementioned services, we need to account for three aspects of the messages: 1) authenticity, 2) expiration time, and 3) legitimacy. Public key encryption and certificates that are based on the chain of trust, are suitable candidates to solve the authenticity and the legitimacy of the messages, and the expiration of the message (rental contract term) can be addressed by using timestamps. In the following we describe the details of such operation.

Imagine Trudy wishes to rent the botnet from Mallory, and every bot has the public key of Mallory hardcoded. Trudy sends her public key PK_T to Mallory, to be signed by the private key of Mallory SK_M. The signed message (T_T) acts as a token containing PK_T, an expiration time, and a list of whitelisted commands. When Trudy wants to issue a command to her rented bots, she signs her command by using her private key SK_T and includes T_T. This way, the bots are able to verify the legitimacy of such commands, by looking at the token and the signature of the message.

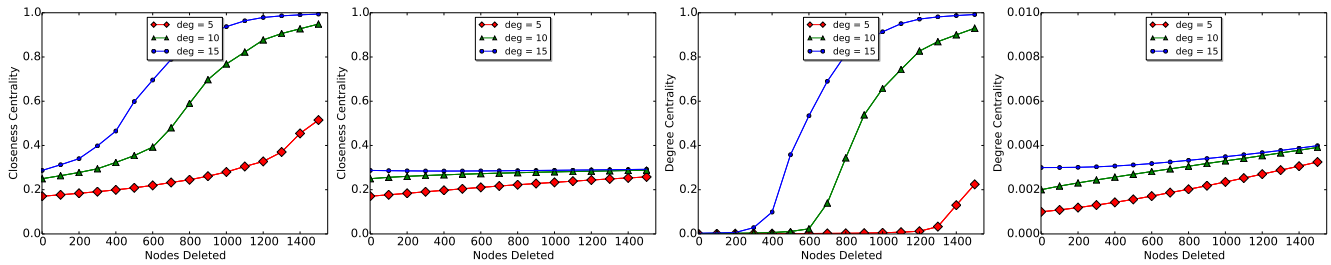
As a bussiness operation, Trudy pays Mallory using Bitcoin, where the whole transaction can be carried out over Silk Road 2.0. Furthermore Mallory can instruct her bots to install computation platforms such as Java Virtual Machine (JVM). By doing so, she can also offer a cross-platform distributed computation infrastructure to carry out CPU intensive tasks, such as bitcoin mining or password cracking.

V. OnionBots Evaluation

To evaluate the envisioned OnionBots, we look at two aspects: the self-healing network formation resilience performance and the resilience to analysis techniques, such as botnet mapping, hijacking, or even assessing the size of the botnet. The NoN look-ahead routing is proven to be asymptotically optimal, however such claims have not been studied in the context of self-healing networks. Although it is desirable to rigorously prove properties of such networks, in this work we use empirical data and simulations for evaluation.

A. Mapping OnionBot

OnionBots provide a more resilient structure by using features available in the Tor network that previous botnets lack. All OnionBot nodes are directly accessible, even those running behind NAT, compared to previous work [56]. If a bot is captured and the address (`.onion`) of other bots is revealed, it is still not practical to block the bots. Additionally, bots can periodically change their `.onion` addresses, and share it only with their operational peers. Therefore, limiting the exposure



(a) without pruning (b) with pruning (c) without pruning (d) with pruning

Fig. 4: The average closeness centrality (4a, 4b) and degree centrality (4c, 4d) of nodes in a *k-regular* graph, ($k = 5, 10, 15$) with 5000 nodes after 30% node deletions, with and without pruning.

of the bot's address. As a result, one single `.onion` could be blocked, as we later discuss in section VI, but it is not feasible to block all of them.

B. OnionBot Network Resilience

To evaluate the resiliency and performance of the self repairing construction we use some of the metrics that are used in graph theory, such as the changes in graph centrality after node deletions. Centrality metrics examined in previous studies [57], [58] include closeness centrality and degree centrality.

The *closeness centrality* of node u, is the inverse of sum of the shortest paths between node u to all $n-1$ other nodes. Since the sum of distances depends on the number of nodes, it is normalized.

$$C(u) = \frac{n-1}{\sum_{v \neq u} d(u,v)}$$

Where n is the number of nodes, and $d(u,v)$ is the shortest path between nodes u and v. It is an indication of how fast messages can propagate in the network, from a node, v, to all other nodes sequentially.

The *degree centrality* of a node u is the number of its neighbors. The degree centrality values are normalized by the maximum possible degree in graph G. Therefore it is the fraction of nodes that u is connected to. It is an indication of immediate chance of receiving whatever is flowing through the network (e.g., messages).

Another metric that we explore and is overlooked by previous work, is the *diameter* of the graph. The diameter of a graph is defined as the longest shortest path (geodesic distance) in the graph. It is formally defined as the maximum of $d(u,v)$, $\forall u, v$, and provides a lower bound on worst case delay.

While a theoretical analysis is more desirable, it is also much harder. In the following, we resort to simulation to get a good sense of the properties of OnionBot. We simulate the node deletion process in a *k-regular* graph, ($k = 5, 10, 15$) of 5000 nodes, with up to 30% (1500) node deletions. Figure 4 illustrates the average closeness centrality with pruning (Figure 4b) and without pruning (Figure 4a). As we can see in Figure 4a, closeness centrality of the nodes is stable, and even after node deletion, it does not decrease. Furthermore we measure the degree centrality of the nodes in the aforementioned graph, with pruning (Figure 4d) and without pruning (Figure 4c). As we can see, the degree of nodes increases significantly after node deletions without pruning. Low degree centrality is desirable in advanced persistent attacks (APT). Additionally, it decreases the chances of detection and take down, because of maintaining a low profile and avoiding to raise the alarm. For example, Stuxnet only infected maximum of three other nodes [59], to slow down its spread and avoiding detection.

To better understand the effect of size, we simulate a small botnet of size 5000 [57] and a medium botnet of size 15000. Figure 5 depicts the aforementioned metrics. As we can see in Figures 5a and 5b, the self-repairing graph remains connected even when a large portion (90%-95%) of the nodes are deleted, compared to a normal graph (a graph with no self-repairing mechanism). Note that, in a normal graph after 60% node deletion, the number of partitions increases sharply. As we can see in Figures 5c and 5d, the degree centrality slightly increases in the DDSR compared to a normal graph, since the healing process ensures that the degree of the nodes stays within a specified range. However, as we remove the nodes in a normal graph, the diameter increases until the graph is partitioned, where the diameter is infinite. In OnionBot, as the nodes are deleted and the number of nodes decreases, the diameter of the graph also decreases accordingly (Figures 5e and 5f).

VI. Mitigation of Basic OnionBots

In this section, we look at different mitigation strategies against OnionBots. Mitigation and detection can take place at different levels, such as host level or network level. Host level remediation techniques include the use of anti-virus software and frameworks such as the Security Behavior Observatory (SBO) [60]. Because of the scaling limitation of such techniques, and the fact that the compromised hosts are rarely updated or patched, we focus on the network level strategies.

Many of the current detection and mitigation mechanisms are IP-based, and rely on the network traffic patterns or DNS queries to distinguish legitimate traffic from malicious traffic. However, current solutions do not work with OnionBots, since

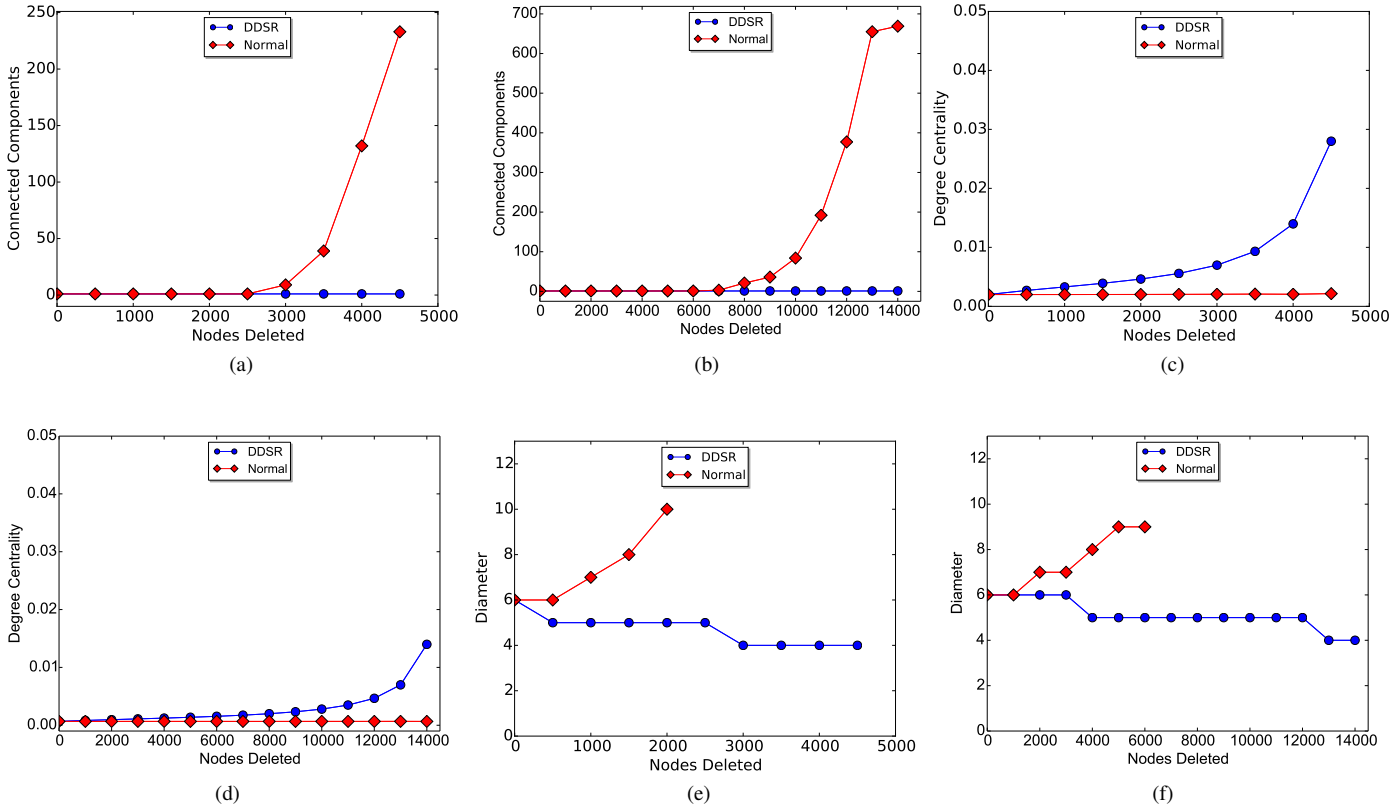


Fig. 5: Graphs depicting the number of connected components, average degree centrality, and graph diameter, after incremental node deletions, in a 10-*regular* graph of 5000 (left side) and 15000 (right side) nodes.

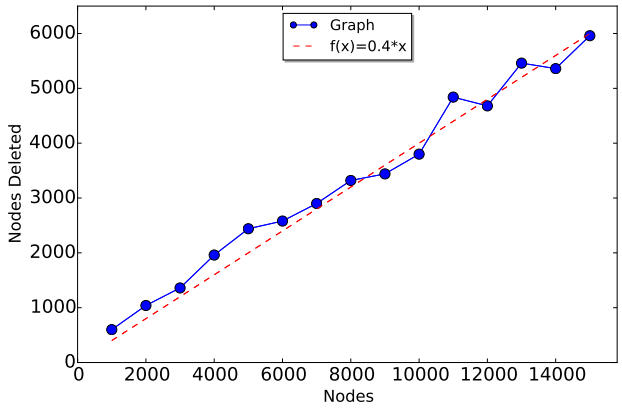


Fig. 6: Partitioning of graphs after removing nodes. The network becomes partitioned after removing on average about 40% of the nodes, in 10-*regular* graphs of size n=1000, to n=15000.

the Tor traffic is encrypted, non IP-based, and there are no conventional DNS queries to resolve the `.onion` addresses. Furthermore, even if an adversary captures a bot sample (e.g., by using honeypots or other similar techniques), and recovers the `.onion` address of its peers, he is still unable to directly map these addressees to their corresponding IP addresses and take down the infected hosts. Since the proposed construction offers a self-repairing low-degree, low-diameter network, even after taking over a large portion of the bots, the botnet remains functional. As Figure 6 shows, an adversary needs to take down about 40% of the bots simultaneously, to even partition the network into two subgraphs. Note that, it means there is not enough time for the graph to self-repair. As we can see, the conventional solutions that ignore the privacy infrastructure construction of OnionBots are not effective. Therefore, we need to adapt our detection and mitigation methods, and integrate them into the foundation of such infrastructures. In this section we divide the network level mitigations into two categories; techniques that are generic to Tor, and schemes that are specific to OnionBots. In particular, we propose a new OnionBot specific mitigation method, called Sybil Onion Attack Protocol (SOAP).

A. *Targeting OnionBots Through Privacy Infrastructures*

Generic mitigations targeting Tor are based on denying access to the bots through the `HSDirs`. As described before, the list of `HSDirs` can be calculated by any entity who knows the `.onion` address (in case there is no `descriptor-cookie`). Hence, an adversary can inject her relay into the Tor network such that it becomes the relay responsible for storing the bot's descriptors. Since the fingerprint of relays is calculated from their public keys, this translates

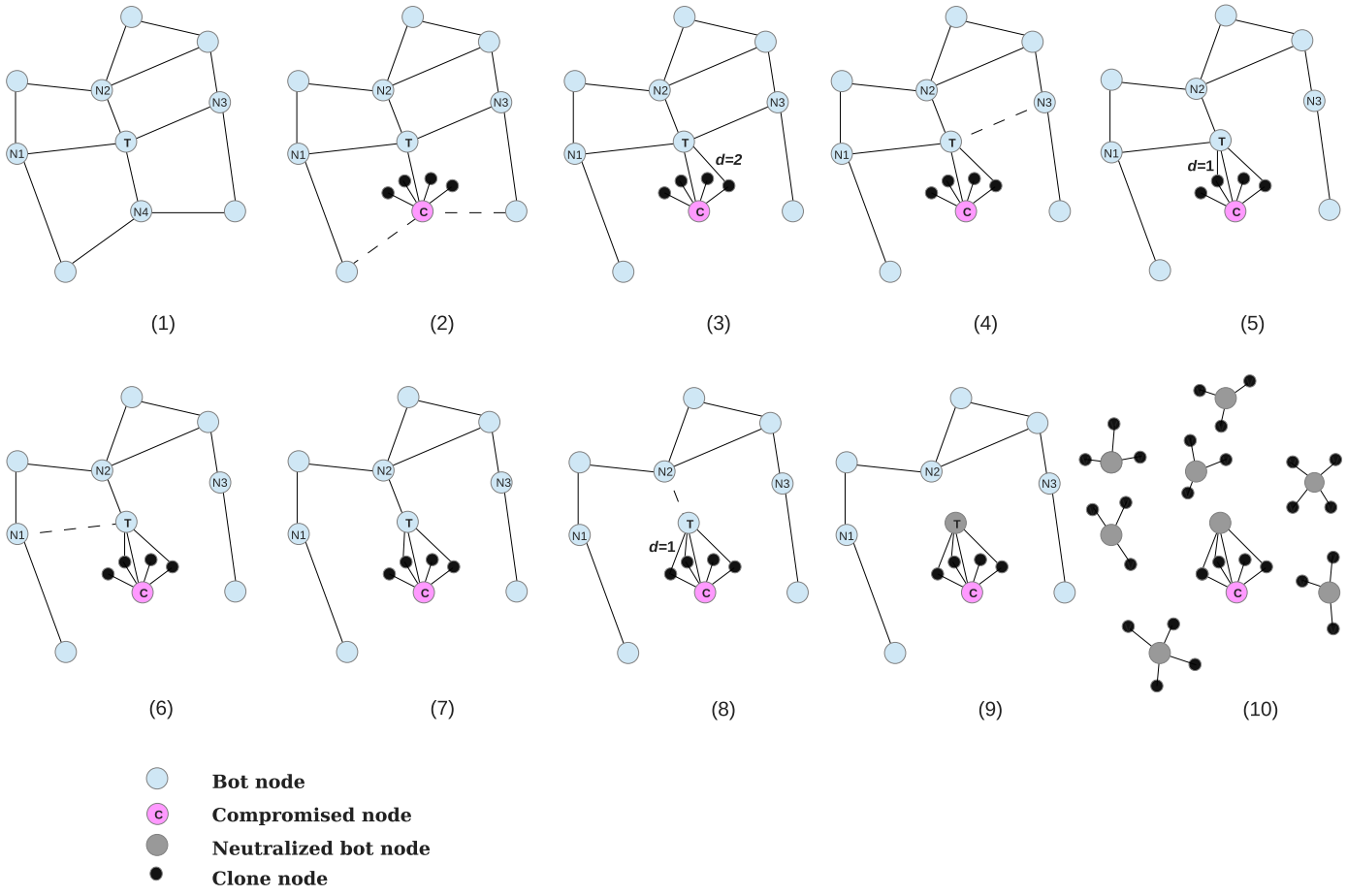


Fig. 7: Soaping attack: node T is under attack by the compromised node C and its clones. In each step one of the clones initiates the peering process with the T, until it is contained. After several iterations, the network is partitioned, and the botnet is neutralized.

into finding the right public key [8]. Nevertheless, it should be noted that an adversary needs to position herself at the right position in the ring at least 25 hours before (it takes 25 hours to get the `HSDir` flag). It is difficult to mitigate against many bots, since the adversary requires the computation power, and a prior knowledge of the `.onion` addresses. Furthermore, it disrupts the operation and user experience of the Tor network.

A more long term approach involves making changes to the Tor, such as use of CAPTACHAs, throttling entry guards and reusing failed partial circuits, as described in [61]. Having said that, these mitigations are limited in their preventive power, open the door to censorship, degrade Tor's user experience, and are not effective against advanced botnets such as OnionBot.

B. Sybil Onion Attack Protocol (SOAP)

We devised a mitigation mechanism that uses OnionBots' very own capabilities (e.g., the decoupling of IP address and the host) against them. We first overview the attack here, and then provide a step by step explanation as depicted in Figure 7. To attack the botnet and neutralize it, we first need to find the bots' `.onion` addresses. This can be done either by detecting and reverse engineering an already infected host, or by using a set of honeypots. Although this is not a trivial task, and requires a significant amount of effort, it allows us to infiltrate the botnet, traverse its network, and identify the other bots. After identifying the bots' `.onion` address, we run many hidden services, disclosing a subset of these as neighbors to each peer we encounter, so gradually over time our clone nodes dominate the neighborhood of each bot and contain it. Note that, we can run all of the clones on the same machine because of the decoupling between the IP address and the host.

Figure 7 depicts the soaping attack in different steps. Node T is the target of the soaping attack, nodes N_i, are its neighboring bot nodes, and nodes C are the adversary (e.g., the authorities), and his clones, which are represented with small black circles. In step 1, the botnet is operating normally, and none of T's neighbors are compromised. In step 2, one of its peers, N_4, is compromised. Then, N_4 (now depicted as C), makes a set of clones (the small black circles). In step 3, a subset of C's clones, start the peering process with T, and declare their degree to be a small random number, which changes to avoid detection (e.g., d=2). Doing so increases the chances of being accepted as a new peer, and replacing an existing peer of T. In step 4, T forgets about one of its neighbors with the highest degree, N_3, and peers with one the

clones. The clones repeat this process until T has no more benign neighbors (steps 5-8). As a result, T is surrounded by clones and is contained (step 9). As we can see after many iterations the adversary can partition the network into a set of contained nodes, and neutralizing the botnet.

VII. Related Work

In this section, we look at other work that examine alternative botnet constructions. However, they still rely on traditional models, which makes them vulnerable to the current detection and mitigation techniques, once their design is known.

Kartaltepe et al. [62], investigate a new generation of botnets that use online social networks, such as Twitter as their C&C infrastructure. An instance of such malware, Naz, gets its commands by making GET requests to the RSS feed of its botmaster on Twitter. The tweets contain the base64 encoding of shortened URLs (e.g., bit.ly) that redirect the bot to the compressed malicious payload.

Nagaraja et al. [63], propose Stegobot, a botnet that communicates over probabilistic communication channels. It spreads via social networks, and steals information from its victims.

Stranerger et al. [52], introduce a botnet communication protocol, called Overbot. Their design leverages Kademila peer-to-peer protocol, a distributed hash table (DHT) used by many peer-to-peer applications. They investigate the possibilities of using the existing protocol to design stealth C&C channels. The bot uses the 160-bit hash values in a search request to announce it's sequence number, which is encrypted with the public key of the botmaster. Later, this sequence number is used to send commands to the bot.

Nappa et al. [64], propose a parasitic botnet protocol that exploits Skype's overlay network. Skype provides a widespread resilient network with a large install base for C&C infrastructure. The communications between the master and the bots are encrypted using adhoc schemes. The protocol broadcasts messages to all peers in the network, similar to the algorithms used in Gnutella. Once each peer receives a new message it passes it to all of its neighbors.

Vogt et al. [65], examine the possibility of creating a super-botnet by splitting a large botnet into many smaller botnets. Each smaller botnet in the super-botnet network, stores some routing information on how to communicate with the rest of the network. They use a tree-structured infection process, where each new zombie learns how many additional host it should infect and add to its botnet. This design results in a connected graph, with many densely connected cliques.

Lee and Kim [66], explore the design and mitigation of botnets that use URL shortening services (USSes) for alias fluxing. A botmaster uses the USSes to hide and obfuscate IP address of the C&C by using a dictionary of 256 words for each part of an IPv4. For example, 10.15.43.89 can be mapped to "Brown.Fox.Jumps.Over." Then this expression is transformed into a search query, such as google.com/q?=Brown+Fox+Jumps+Over. Using the URL shortening service, bots can find the corresponding IP address by using the same dictionary.

Wang et al. [56], design a hybrid peer-to-peer botnet, which is composed of servant and client bots. Their botnet communicates with a fixed number of peers contained in each bot to limit the node exposure. The botmaster can control, monitor and update the bots by sending the messages through servant bots, and getting the reports from a sensor host. These messages are encrypted using individualized predefined or dynamically generated keys.

Xu et al. [67] study the use of DNS for C&C using two communication modes to piggyback messages over the DNS messages, codeword and tunneled. In the codeword mode, the bot makes a query (e.g., codeword.example.com) and the server replies with an appropriate answer (e.g., the IP address of a victim for DoS attack). In the tunneled mode the client encodes its data using a base32 encoding and sends a `CNAME` query. After receiving the query, the server uses base32 encoding to construct the corresponding `CNAME` reply.

VIII. Conclusion

Privacy infrastructures such as Tor had a tremendous impact on society, protecting users anonymity and rights to access information in the face of censorship. It also opened the door to abuse and illegal activities, such as ransomware [7], and a marketplace for drugs and contraband [68], [6]. In this work we envisioned OnionBots, and investigated the potential of subverting privacy infrastructures (e.g., Tor hidden services) for cyber attacks. We presented the design of a robust and stealthy botnet that lives symbiotically within these infrastructures to evade detection, measurement, scale estimation and observation. It is impossible for Internet Service Providers (ISP) to effectively detect and mitigate such botnet, without blocking all Tor access. Additionally, OnionBots rely on a resilient self-healing network formation that is simple to implement, yet it has desirable features such as low diameter and low degree. Such botnets are robust to partitioning, even if a large fraction of the bots are simultaneously taken down. In the scenario of a gradual take down of nodes, the network is also able to self-repair, even after up to 90% node deletions. More importantly, we developed soaping, a novel mitigation attack that neutralizes the OnionBots. We also suggested mitigations that act at the Tor level. There are still many challenges that need to be preemptively addressed by the security community, such as the byzantine behavior of OnionBots [69], [70]. We hope that this work ignites new ideas to proactively design mitigations against the new generations of crypto-based botnets.

References

[1] R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and taxonomy of botnet research through life-cycle," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, August 2013.

[2] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: A survey," *Computer Networks*, vol. 57, 2013.

[3] "Know your enemy: Fast-flux service networks," http://www.honeynet.org/papers/ff/, July 2007.

[4] "The inevitable move - 64-bit zeus enhanced with tor," http://securelist.com/blog/events/58184/the-inevitable-move-64-bit-zeus-enhanced-with-tor/, December 2013.

[5] "Chewbacca - a new episode of tor-based malware," http://securelist.com/blog/incidents/58192/chewbacca-a-new-episode-of-tor-based-malware/, December 2013.

[6] N. Christin, "Traveling the silk road: A measurement analysis of a large anonymous online marketplace," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW, 2013.

[7] "Cryptolocker ransomware," http://www.secureworks.com/cyber-threat-intelligence/threats/cryptolocker-ransomware/, December 2013.

[8] A. Biryukov, I. Pustogarov, and R. Weinmann, "Trawling for tor hidden services: Detection, measurement, deanonymization," in *Security and Privacy (SP), 2013 IEEE Symposium on*, 2013.

[9] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses," in *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, ser. CATCH, 2009.

[10] J. Liu, Y. Xiao, K. Ghaboosi, H. Deng, and J. Zhang, "Botnet: Classification, attacks, detection, tracing, and preventive measures," *EURASIP J. Wirel. Commun. Netw.*, 2009.

[11] N. Daswani, T. G. C. Quality, S. Teams, and G. Inc, "The anatomy of clickbot.a," in *In USENIX Hotbots*, 2007.

[12] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection." in *USENIX Security Symposium*, 2008.

[13] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," in *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, ser. SRUTI, 2005.

[14] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *Proceedings of 10 th European Symposium on Research in Computer Security, ESORICS*, 2005.

[15] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC, 2006.

[16] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy, "Studying spamming botnets using botlab," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI, 2009.

[17] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI, 2010.

[18] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS, 2009.

[19] P. Porras, H. Saïdi, and V. Yegneswaran, "A foray into conficker's logic and rendezvous points," in *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, ser. LEET, 2009.

[20] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security)*, 2012.

[21] S. Yadav, A. K. K. Reddy, A. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.

[22] S. Yadav, A. Reddy, A. Reddy, and S. Ranjan, "Detecting algorithmically generated domain-flux attacks with DNS traffic analysis," *Networking, IEEE/ACM Transactions on*, 2012.

[23] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, II, and D. Dagon, "Detecting malware domains at the upper DNS hierarchy," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC, 2011.

[24] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *Proceedings of the 19th USENIX Conference on Security*, 2010.

[25] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE : Finding malicious domains using passive DNS analysis," in *NDSS, 18th Annual Network and Distributed System Security Symposium*, 2011.

[26] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding bots in network traffic without deep packet inspection," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT, 2012.

[27] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC, 2012.

[28] A. Mohaisen, O. Alrawi, A. G. West, and A. Mankin, "Babble: Identifying malware by its dialects," in *IEEE Communications and Network Security*, 2013.

[29] A. G. West and A. Mohaisen, "Metadata-driven threat classification of network endpoints appearing in malware," *DIMVA: Proceedings of the 11th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, 2014.

[30] 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*, ser. LEET, 2008.

[31] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the storm and nugache trojans: P2P is here," in *;login*, 2007.

[32] B. Stock, J. Göbel, M. Engelberth, F. C. Freiling, and T. Holz, "Walowdac - analysis of a peer-to-peer botnet," in *Proceedings of the European Conference on Computer Network Defense*, ser. EC2ND, 2009.

[33] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus," in *Malicious and Unwanted Software: "The Americas" (MALWARE), 8th International Conference on*, 2013.

[34] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding P2P bots with structured graph analysis," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security, 2010.

[35] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy P2P botnets using statistical traffic fingerprints," in *Dependable Systems Networks (DSN), IEEE/IFIP 41st International Conference on*, 2011.

[36] T.-F. Yen and M. Reiter, "Are your hosts trading or plotting? telling P2P file-sharing and bots apart," in *Distributed Computing Systems (ICDCS), IEEE 30th International Conference on*, 2010.

[37] B. Coskun, S. Dietrich, and N. Memon, "Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC, 2010.

[38] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium*, 2004.

[39] P. Narang, S. Ray, C. Hota, and V. Venkatakrishnan, "Peershark: Detecting peer-to-peer botnets by tracking conversations," in *International Workshop on Cyber Crime (IWCC)*, 2014.

[40] "2012 data breach investigation report," http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2012-ebk_en_xg.pdf, December 2012.

[41] C. Rossow, C. Dietrich, and H. Bos, "Large-scale analysis of malware downloaders," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer Berlin Heidelberg, 2013.

[42] R. Potharaju, E. Hoque, C. Nita-Rotaru, S. Sarkar, and S. Venkatesh, "Closing the pandora's box: Defenses for thwarting epidemic outbreaks in mobile adhoc networks," in *IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2012.

[43] A. Sanatinia, S. Narain, and G. Noubir, "Wireless spreading of wifi aps infections using wps flaws: an epidemiological and experimental study," *Communications and Network Security (CNS)*, 2013.

[44] O. Thonnard and M. Dacier, "A strategic analysis of spam botnets operations," in *Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*, ser. CEAS, 2011.

[45] M. Marchetti, M. Colajanni, M. Messori, L. Aniello, and Y. Vigfusson, "Cyber attacks on financial critical infrastructures," in *Collaborative financial infrastructure protection: Tools, abstractions and middleware*, R. Baldoni and G. Chockler, Eds. Springer, January 2012, ch. 3, pp. 53–81.

[46] W. Chang, A. Mohaisen, A. Wang, and S. Chen, "Measuring botnets in the wild: Some new trends," in *ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS, 2015.

[47] "Microsoft: Kelihos ring sold 'botnet-as-a-service'," http://www.darkreading.com/risk-management/microsoft-kelihos-ring-sold-botnet-as-a-service/d/d-id/1100470, September 2011.

[48] R. Hund, M. Hamann, and T. Holz, "Towards next-generation botnets," in *European Conference on Computer Network Defense, EC2ND*, 2008.

[49] R. Bless, O. P. Waldhorst, C. P. Mayer, and H. Wippel, "Decentralized and autonomous bootstrapping for ipv6-based peer-to-peer networks," in *Winning Entry of the IPv6 Contest 2009 by IPv6 Council*, 2009.

[50] "Shallot," https://github.com/katmagic/Shallot, June 2012.

[51] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys Tutorials, IEEE*, 2005.

[52] G. Starnberger, C. Kruegel, and E. Kirda, "Overbot: A botnet protocol based on kademlia," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, ser. SecureComm, 2008.

[53] G. Singh Manku, M. Naor, and U. Wieder, "Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks," in *Proceedings of the thirty-sixth annual ACM symposium on Theory (STOC)*, 2004.

[54] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS, 2013.

[55] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP, 2013.

[56] P. Wang, S. Sparks, and C. Zou, "An advanced hybrid peer-to-peer botnet," *IEEE Transactions on Dependable and Secure Computing (DSN)*, 2010.

[57] T.-F. Yen and M. K. Reiter, "Revisiting botnet models and their implications for takedown strategies," in *Proceedings of the First International Conference on Principles of Security and Trust*, ser. POST, 2012.

[58] C. Davis, S. Neville, J. Fernandez, J.-M. Robert, and J. McHugh, "Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures?" in *Computer Security - ESORICS 2008*, ser. Lecture Notes in Computer Science, 2008.

[59] "W32.stuxnet," http://www.symantec.com/security_response/writeup.jsp?docid=2010-071400-3123-99, February 2013.

[60] A. Forget, S. Komanduri, A. Acquisti, N. Christin, L. F. Cranor, and R. Telang, "Security behavior observatory: Infrastructure for long-term monitoring of client machines," 2014.

[61] N. Hopper, "Challenges in protecting tor hidden services from botnet abuse," in *Proceedings of the Financial Cryptography and Data Security*, ser. FC, 2014.

[62] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu, "Social network-based botnet command-and-control: Emerging threats and countermeasures," in *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ser. ACNS, 2010.

[63] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov, "Stegobot: a covert social network botnet," in *Information Hiding*. Springer, 2011.

[64] A. Nappa, A. Fattori, M. Balduzzi, M. Dell'Amico, and L. Cavallaro, "Take a deep breath: A stealthy, resilient and cost-effective botnet using skype," in *Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA, 2010.

[65] R. Vogt, J. Aycock, and M. J. Jacobson, "Army of botnets," in *Proceedings of Annual Network and Distributed System Security Symposium (NDSS)*, 2007.

[66] S. Lee and J. Kim, "Fluxing botnet command and control channels with url shortening services," *Comput. Commun.*, 2013.

[67] K. Xu, P. Butler, S. Saha, and D. Yao, "DNS for massive-scale command and control," *Dependable and Secure Computing, IEEE Transactions on*, 2013.

[68] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Content and popularity analysis of tor hidden services," *arXiv preprint arXiv:1308.6768*, 2013.

[69] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *Dependable and Secure Computing, IEEE Transactions on*, vol. 8, no. 4, pp. 564–577, July 2011.

[70] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *Proceedings of the 1st ACM Workshop on Wireless Security*, ser. WiSE, 2002.