Take a Deep Breath: a Stealthy, Resilient and Cost-Effective Botnet Using Skype

Antonio Nappa[1], Aristide Fattori[1], Marco Balduzzi[2]
Matteo Dell'Amico[2], and Lorenzo Cavallaro[3]

[1] DICo, Università degli Studi di Milano, Italy
{nappa, joystick}@security.dico.unimi.it
[2] Eurecom, Sophia-Antipolis, France
{marco.balduzzi,matteo.dell-amico}@eurecom.fr
[3] Faculty of Sciences, Vrije Universiteit Amsterdam, The Netherlands
sullivan@few.vu.nl

Abstract. Skype is one of the most used P2P applications on the Internet: VoIP calls, instant messaging, SMS and other features are provided at a low cost to millions of users. Although Skype is a closed source application, an API allows developers to build custom plugins which interact over the Skype network, taking advantage of its reliability and capability to easily bypass firewalls and NAT devices. Since the protocol is completely undocumented, Skype traffic is particularly hard to analyze and to reverse engineer. We propose a novel botnet model that exploits an overlay network such as Skype to build a *parasitic overlay*, making it extremely difficult to track the botmaster and disrupt the botnet without damaging legitimate Skype users. While Skype is particularly valid for this purpose due to its abundance of features and its widespread installed base, our model is generically applicable to distributed applications that employ overlay networks to send direct messages between nodes (e.g., peer-to-peer software with messaging capabilities). We are convinced that similar botnet models are likely to appear into the wild in the near future and that the threats they pose should not be underestimated. Our contribution strives to provide the tools to correctly evaluate and understand the possible evolution and deployment of this phenomenon.

1 Introduction

Botnets are a major plague of the Internet: miscreants have the possibility to hire and control armies of several thousands of infected PCs to fulfill their malicious intents. Theft of sensitive information, sending unsolicited commercial email (SPAM), launching distributed denial of service (DDoS) attacks, and scanning activities are among the most nefarious actions attributable to botnets.

The threat posed by bots and their constant and relevant position in the underground economy made them one of the most discussed topics by security experts and researchers [27, 20]. A plethora of techniques have been proposed and deployed to address such a phenomenon [24, 37, 9, 50, 51, 31]. Some of them aim to understand botnets' *modus operandi* [45, 29, 44], while others aim to detect patterns typically exhibited by bot-infected machines [24, 9, 50, 51].

Unfortunately, despite the efforts spent by the research community in fighting botnets, they remain an omnipresent menace to the safety, confidentiality, and integrity of Internet users' data. On the one hand, bots authors, increasingly motivated by financial profits [12, 45], are constantly looking for the most appealing features a botnet should have: stealthiness (i.e., non-noisy, low-pace, encrypted and distributed communications), resiliency (to nodes shutdown), and cost-effectiveness (i.e., easy to infect/spread to new machines). On the other hand, defense strategies must cope with such ever evolving malware, while being, at the same time, easy-to-deploy exhibiting a contained rate of false positives.

Skype is the *de-facto* standard when it comes to VoIP and related communications. It has a number of ancillary features that make it the ideal platform for a solid communication infrastructure. In fact, it protects the confidentiality of its users by encrypting all their communications, it is fault-tolerant by adopting a de-centralized communication infrastructure, and it is firewall- and NAT-agnostic, meaning that it generally works transparently with no particular network configuration. Therefore, despite its closed-source nature, it is not surprising how Skype has rapidly gained a huge consensus among the millions of Internet users that use it on a daily basis.

Despite some efforts tailored to understanding Skype's code and network patterns [6, 17], such a closed infrastructure remains almost obscure, nowadays. Skype-generated network traffic is thus extremely difficult to filter and arduous to analyze with common network-based intrusion detection systems [3]. As an unavoidable consequence, criminals have soon realized how Skype's encrypted communications could then protect the confidentiality of their (illegal) business, hampering the activities of law enforcement agencies [1, 2, 4, 33]. Even worse, the whole Skype infrastructure meets perfectly all the aforementioned features for a stealth, resilient, and cost-effective botnet. A plethora of Skype users can potentially become unwitting victims of a powerful skype-based botnet: the year 2009 alone counted for 443 millions of active Skype accounts, with an average number of 42.2 millions of active users per day [18]. These numbers would certainly attract cyber-criminals soon, and such network and communication characteristics would definitely make the traffic generated by a Skype-based botnet an especially difficult needle to find within the haystack of regular Skype traffic.

In this paper, we show how to take advantage of the features and protection mechanisms offered by Skype to create a botnet that is, at the same time, resilient, easy to deploy and difficult to analyze. Using existing infrastructures for botnet command and control is not a new idea, since many botnets nowadays adopt IRC servers; however, a *parasitic* peer-to-peer overlay built on top of another decentralized overlay allows for many new and interesting features[4]:

– it is hard to set bots and regular Skype traffic apart, as our parasitic overlay network sits on top of the regular Skype network;
– the malicious network so obtained has no bottlenecks nor single point of failure, and this is achieved by deploying an unstructured network with no hierarchical differences among the nodes;
– the lack of a hierarchical structure allows also to use any controlled node as an entry point for the botmaster;
– our parasitic overlay network tolerates the loss of bots: each node/bot contains only a small set of neighbors used for message passing and no information about the botmaster;
– dynamic transparent routing strategies are in charge of routing messages through alternative routes, should one or more bots become unavailable (e.g., shut down);
– the policy adopted for registering new nodes makes it cost-unattractive to obtain a comprehensive list of all the bots.

Simulation experiments we performed show that our model is indeed practical and guarantees a strong resilience to the botnet, ensuring that messages are delivered even when many bots are offline.

It is worth noting that exploring emergent threats is a problem with important practical consequences, as already acknowledged by the research community [49]: emerging botnets have already begun to adopt stealthy communications [45] and de-centralized network infrastructures [29, 44] as a mean to become more resilient and hard to track down to keep contributing to the flourishing cyber-criminal business. Thus, a necessary first step for developing robust defenses is that to study about emergent threats. This is the motivation of our work: to show that it is easy, feasible, and practical to deploy a stealth, resilient, and cost-effective botnet. We finally conclude the paper by sketching the design and implementation of a host-based countermeasure that aims to classify Skype plugins as good or malicious by monitoring their interactions with the core application. Although our results are preliminary, they look promising to tackle such a nefarious threat at the end-host systems.

2 Skype Overview

Skype is a widely used application, which features VoIP and calls to land-line phones, audio and video conferencing, SMS and instant messaging, and more. It is organized as a hybrid peer-to-peer (P2P) network with central servers, supernodes, and ordinary clients [3].

[4] We point out that Skype is not the only network that can be exploited by a parasitic overlay: any de-centralized overlay network providing direct messaging capabilities between the nodes can be a suitable target.

Supernodes play an important role in the whole network. In fact, a set of supernodes is responsible for bootstrapping the network. Thus, they act as the point of entrance in the overlay infrastructure, and messages sent by a node are routed through them. They are elected by considering different criteria, such as the availability of a public IP, and the bandwidth/computational resources available on the client. Thus, every host with such features can become a supernode.

The security of Skype is so strong that Governments of some Countries have reported that criminals and mobs started using Skype for their communications in order to avoid eavesdropping by the police forces [1, 4, 33]. The only effective countermeasures seem to be devoted at attacking the end-hosts system [41].

On the one hand, software that comes with valid security and privacy policies creates positive sensations of trust and safeness to its users, encouraging the enlargement of the installed base. On the other hand, the presence of possible weaknesses in its architecture gives to attackers a means to take advantage of its features for purposes beyond the original design of the application. Indeed, miscreants have already started to misuse the API to deploy malware. For instance, Peskyspy[5] is a trojan that records Skype conversations and provides a backdoor to download them, while Pykspa[5] and Chatosky[5] are two worms that spread using the Skype chat. Skype malware benefit from the fact of being deployed as Skype plugins. In fact, whenever a plugin issues a command (e.g., to create a chat or to send a message), Skype behaves exactly as if the command was invoked by the real user. For example, all the Skype plugins' traffic that leave an host is automatically encrypted. Malware can take advantage of these features to hide themselves and their actions, thus becoming very hard to detect. All these features make the API appetizing to miscreants that look for new and powerful means to create and control a botnet [7, 15].

Fortunately, to the best of our knowledge, Skype-based botnets are still exclusively theoretical. Nonetheless, it is interesting to ask ourselves whether such an emerging threat could potentially be a serious menace to the (Internet) society, in the near future. In the following, we show that it is indeed practical and feasible to build a cost-effective botnet by exploiting the features offered by a pre-existent overlay networks such as, in our case, Skype.

2.1 The Skype API

The Skype API allows developers to write applications using features such as sending chat or SMS messages, starting or redirecting calls, or searching for friends. Unfortunately, this API mechanism is far from being as secure as the core of Skype is [17].

A weakness of the API is that there is no control over the number of messages that a plugin is allowed to send. Basically, all the possible activities that a human user can perform through the client (calls, SMSs and chats) can be automated by a plugin, without any flooding control that would limit spamming.

[5] Symantec names.

For instance, Pykspa, is a Skype-based malware that spreads by spamming messages with links throughout the Skype chat without any rate-limiting threshold. Furthermore, a Skype plugin can directly have access to the search routine and easily harvest many addresses of Skype-registered contacts. This information is fundamental since the access to the search routine gives the possibility to find formerly unknown peers.

Every time a third party application wants to interact with Skype, a check is performed to determine if this software is allowed to access the API. The mechanism used by Skype to accomplish this control is based on white/blacklisting. There are two levels of white/blacklists: *local* and *global*. The former is stored in the Skype configuration file using a hash value: when a new plugin wants to interact with Skype, at the first execution, Skype comes up with a dialog box that prompts for the user acknowledgment. When the user selects to authorize or block the plugin, her decision is stored in the configuration file along with its hash value. This hash value is the checksum of the plugin binary and is used by Skype to check for changes in an already authorized plugin. Instead, the latter list is determined centrally by Skype authorities and then propagated to its users throughout the P2P network.

Unfortunately, there are two inherent limitations of such an approach. On the one hand, the existence of API bindings for interpreted languages, make it hard to white/blacklist a single plugin as the interpreter program is the one that interacts with the core. On the other hand, hackers were able to reverse engineer the hash function used to calculate the plugin signatures, and published their results on the Internet giving to malware authors the possibility to develop their own extensions [17]. As such a vulnerability can be fixed in the next releases of Skype, we opt for a different strategy: silently waiting for the authorization dialog to appear, and then performing a fake click to authorize the malware without user consent. We implemented this technique in our bot prototype (see Section 3.2).

3 System Description

In this Section we present our parasitic overlay, a network of non-structured peers that exchange messages over a pre-existent P2P overlay network such as, in our case, Skype.

In our botnet, messages exchanged between bots and the master flow through the network exactly as legitimate messages of the application. This makes the botnet traffic unrecognizable with respect to the legitimate Skype traffic: parasitic overlay nodes behave as ordinary peers of the underlying "host" overlay network. Fig. 1 shows how nodes communicate in the parasitic overlay: botnet nodes (in black, on the above layer, linked to the corresponding infected skype node through a dashed line) send messages to each other directly, without being aware of how routing is performed in the underlying Skype network.

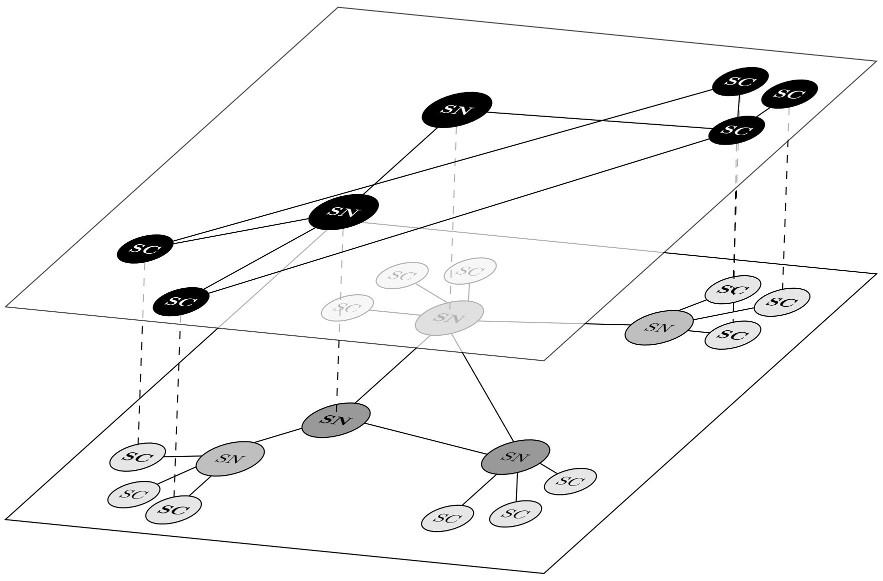


Fig. 1. The parasitic overlay network over Skype. While all the messages are routed from Skype clients (SCs) through supernodes (SNs), the parasitic overlay network makes no hierarchical difference between supernodes and regular clients.

3.1 Botnet Protocol

The communication between the bots and the master is protected by using an ad-hoc encryption scheme in addition to the encryption already performed by Skype. This preserves the confidentiality of the messages exchanged by the bots, even if Skype disclosed, to law enforcement authorities, the encryption key of a particular bot-engaged communication. Messages are sent through the Skype chat as common conversations between users. To accurately replicate the behavior of botnets present in the wild, we designed the architecture in order to provide unicast, multicast and broadcast communication between the master and the bots.

Message Encryption. Bots can receive single commands, group commands and global commands, respectively useful for well targeted attacks, botnet rental or for updates or massive attacks. This is possible by using different encryption mechanisms between the master and the bots. Each node owns a set of symmetric keys: a *node key*, used to receive unicast messages from the master, and any number of *group keys*, to receive multicast messages. Group keys are sent to nodes via new messages from the master. All messages from nodes to the master are encrypted using the master's public key (shipped with the malware binary), while messages from the master to nodes are encrypted with the appropriate symmetric key and signed by the master. All nodes try to decrypt the messages they receive with the keys they possess. All encrypted messages are prepended by a random string to avoid that messages containing the same clear-text result in the same ciphertext.

Message Passing. The message-passing procedure broadcasts every message to all participating peers in the network, using a flooding algorithm similar to the one used in unstructured peer-to-peer networks such as Gnutella [22]: when a peer receives a new message, it forwards it to all neighbors. By doing so, no routing information to reach the botmaster is disclosed. A set of hashes of all received messages is locally kept by nodes to avoid forwarding again old messages. Algorithm 1 shows the details of a bot behavior upon reception of a message. If the received message has not been processed yet, and if the bot is able to decrypt the message, it means the new message is directed to him, thus it verifies if the master's signature is valid. If this condition holds, the bot executes the command received in the message. In the end, regardless whether the message is directed to the bot or not, it stores the hash of the message and forwards it to its neighbors. Overhead and performances of the flooding mechanism will be discussed in Section 4.

Botnet Bootstrap. When new nodes join the botnet, they bootstrap their connection by generating a node key and by connecting to a set of pre-defined *gate nodes* (GNs), shipped with the binary, that serve as temporary neighbors for the network bootstrap. Gate nodes are ordinary nodes connected to the network, and they are used to reach the botmaster via the message passing protocol described in Section 3.1. The new node announcement contains its Skype username, the newly-generated node key and, as any communications sent from nodes to the botmaster, it is encrypted with the botmaster public key. Again using the message passing protocol, the botmaster responds with a list of l nodes that will be, from that moment on, the neighbors of the new node. This message is encrypted with the node's symmetric key that was sent to the botmaster. Appropriate values for the l parameter will be discussed in Section 4.1. Since the GNs set guarantees to new infected bots the possibility of joining the network, they may appear as a particularly vulnerable point: if the GNs are excluded from the Skype network or the malware gets uninstalled from them, no new nodes can join the botnet. However, it is important to point out that these gate nodes are ordinary nodes, exactly as the other nodes in the network, and therefore the list of GNs shipped with the binary can be updated at will by the botmaster if the GNs are unreachable. Moreover, the gate nodes do not have any special routing information, and therefore they will not disclose any information about the identity of the botmaster even if they fall under the control of an authority and are inspected.

Bootstrap Fail-Over. In the unlikely case the GNs are not available, because they have been dismissed already, the bot cannot receive any bootstrap list

```
Input:
  – Received message $M$
  – List of neighbors $N$
  – Set of received message hashes $H$
  – List of symmetric keys (node key and group keys) $K$
Output:
  – Commands to execute $execute(C)$
  – Messages to forward $forward(F, N)$
foreach message $M$ do
    if $hash(M) \in H$ then
        $drop(M)$;
    end
    else
        if $M$ can be decrypted with a key $k \in K$ then
            $C \leftarrow decrypt(M, k)$;
            if signature of $M$ is verified then
                $execute(C)$;
            end
        end
        $add(hash(M), H)$;
        $forward(M, N)$;
    end
end
```

Algorithm 1: Message-passing algorithm.

from the master. As a fallback measure, the bot issues a Skype search based on a criterion generated from a seed S that is common to all bots. This seed is obtained by an external source that is completely independent and easily accessible by every bot, e.g., by following a strategy similar to the Twitter-seeded domain-flux adopted recently by the Torpig botnet [45]. The master registers one or more Skype users with usernames generated starting from S and sets in their public fields, e.g. the status message, the list of the active GNs that the new bots have to use for their bootstrap phase. As the approach relies on dynamic and daily updated external sources, it seems unfeasible, for a defense mechanism, to predict and shut all the soon-to-be-registered users off in a timely, effective, and cost-effective manner. Moreover, this fallback measure does not expose more information about the parasitic overlay than the normal bootstrap phase.

3.2 Implementation

The proof-of-concept bot has been entirely developed in Python, exploiting the capabilities of the `Skype4Py` library. The library is cross-platform and the bot developed on Linux runs also on Windows and Mac OS X operating systems.

As discussed earlier, Skype comes with an access control mechanism that prevents to load a plugin without an explicit user acknowledgment. At the first execution of a plugin, Skype prompts the user with an authorization request that blocks the program execution until an answer is given. Subsequently, Skype calculates a signature for the plugin and updates its configuration file. There are basically two ways to circumvent this Skype-enforced access control mechanism. One would require to reverse the underlying hashing algorithm, while the other would require to mimic users' interactions. The first approach, described in [17], suffers from updates of the hashing algorithm that would require to be understood and reversed again. On the other hand, the second approach is more generic and is the one we have thus implemented.

From a malware author's perspective, such an access control poses a problem to the automated registration of malicious applications and can decrease the success rate of an infection. To cope with this issue, we integrated in our bot an X11 tool [42] that, simulating keyboard and mouse inputs, automatically validates our plugin registration. We have instructed the prototype to listen and to react immediately at the Skype authorization dialog, authorizing the plugin in a concealed fashion. Our implementation bypasses the Skype security protection mechanism and allows an automated and hidden registration of the bot. The same approach is practical on Microsoft Windows through the use of standard libraries (e.g., `FindWindow()`, `GetCurPos()`, and `MouseEvent()` functions [36]), without the need of external tools.

4 Experiments

We performed two different sets of experiments on our botnet prototype. In Section 4.1, we describe a simulation of our message-passing algorithm that aims at evaluating the trade-offs between the overhead due to the message-passing algorithm and the percentage of bots that are reached by every message. In Section 4.2, we show the empirical observations made while deploying and running our bot prototypes on real systems.

4.1 Network Traffic Simulation

To test the effectiveness of message delivery and the overhead it involves, we wrote a custom simulator that emulates the dynamics of message spreading of Algorithm 1. Since our message-passing protocol floods messages to all nodes, relying on cryptography to ensure that only the intended recipients can decrypt it, we measure the effectiveness of the algorithm with two quantities: coverage, that is the percentage of nodes that are reached by a message sent from a given starting node, and the overhead, expressed as the ration between number of messages sent in the whole system and the number of nodes in it. A perfect algorithm would have a coverage of 100% (all bots are reached) and an overhead of 1 (each peer receives exactly one message).

Parameter	**Description**	**Default**
n	number of nodes	10,000
l	links added by each new peer	100
m	number of messages sent in the simulation	100
a	probability that a node is online	0.05

Table 1. Parameters for the network traffic simulation.

In our simulator, we generate a network topology with n nodes and l links per node. We start with a completely connected topology of l nodes, and then we iteratively add new nodes connecting them with a random subset of l pre-existing ones, until we reach the desired size of n nodes. Before simulating the propagation of each message, we consider each node and randomly shut it off or leave it on according to a uniform probability a, representing the average peer online availability. Table 1 gives an overview of the simulation parameters.

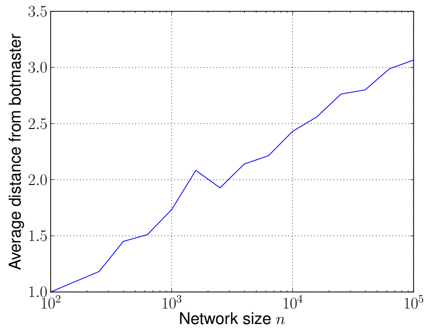
If we consider our network with the tools of graph theory, it is important to evaluate the size of connected components in the subgraph that we obtain by considering only online nodes. In fact, when a message gets propagated starting from a given node, it will reach all nodes that belong to the same connected component. For Erdös-Rényi (ER) random graphs, a key value is the number of edges in the network, which in our case–considering the probability that nodes are online–corresponds to the value of $l \cdot a \cdot \overline{n}$, where $\overline{n} \simeq n \cdot a$ is the number of online nodes. In an ER graph [8], for a large number of nodes $\overline{n}$, a giant component (i.e., a connected component having size proportional to $\overline{n}$) appears when number of edges is greater than $0.5\overline{n}$, and the whole graph becomes connected with high probability when this value reaches $(\ln \overline{n}/2)\overline{n}$.

While in our case, due to the way we build the network, we do not have a perfect ER graph, we experimentally observe in Table 2 a similar behavior with respect to coverage, and a clear trade-off between network coverage and overhead due to sending redundant messages. In particular, a choice of $l = 40$, entailing $l \cdot a \cdot \overline{n} = 2\overline{n}$ reaches around 90% of the nodes imposing a cost of 1.88 messages per node, while a value of $l = 92$ resulting in $l \cdot a \cdot \overline{n} \simeq (\ln \overline{n}/2)\overline{n}$ reaches 99.84% of the nodes that are online, but it involves sending 4.57 messages per node.

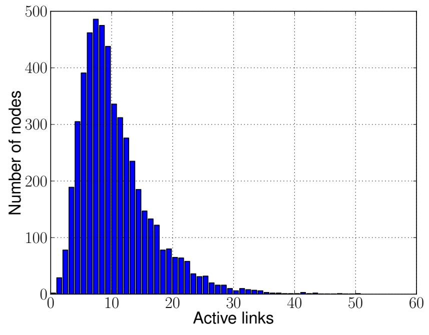
Links per node (l)	**Number of edges ($l \cdot a \cdot \overline{n}$)**	**Coverage**	**Overhead**
10	$0.5 \cdot \overline{n}$	8.69%	0.08
20	$1 \cdot \overline{n}$	55.23%	0.71
40	$2 \cdot \overline{n}$	90.29%	1.88
60	$3 \cdot \overline{n}$	96.76%	2.88
92	$4.60 \cdot \overline{n}\,(\simeq (\ln \overline{n}/2)\overline{n})$	99.84%	4.57

Table 2. Coverage and overhead for various values of the number of links l.

Figure 2(a) shows the number of hops separating, on average, each node from the botmaster with growing network size n. This number of steps, and therefore



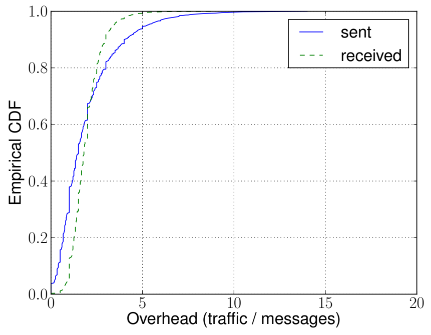
(a) Number of hops.



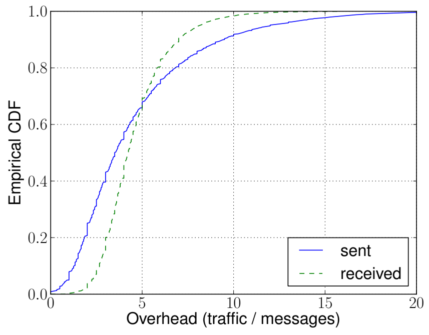
(b) Number of online neighbors.

Fig. 2. Informations on network topology.

latencies in message delivery, have a slow logarithmic growth with respect to the network size. Instead, Figure 2(b) shows the number of online neighbors per node ($n = 100,000$ in this case). Nodes that joined the network earlier are more likely to have more connections, since they had more opportunities of getting chosen as new neighbors of nodes that just joined the network.



(a) $l = 40$.



(b) $l = 92$.

Fig. 3. Number of sent and received messages per node.

Another important issue is load balancing between peers: in Figure 3, we analyze the empirical CDF with the average number of sent/received messages per node. We observe that there is no strong deviation from the average for received messages, while this phenomenon is more strongly perceivable for sent messages. We attribute this to the fact that the older nodes, being more well-connected and closer to the "center" of the network, receive their messages generally earlier than their neighbors and they are thus more often responsible for propagating them.

4.2 Bot Deployment

We tested our Skype botnet infrastructure on a testbed network of several hosts that we “infected” with our bot prototype. We simulate the infection by injecting the bot execution code in the start-up scripts of the infected machine’s users. In a real scenario, attackers can infect their victims in several different ways, e.g. by setting-up a drive-by-download site that exploits the visitors’ browser vulnerabilities or by sending SPAM email embedding the malicious code.

We deployed our bot on 37 hosts, geographically distributed between France and Italy. In one of them we included the code to support the botmaster operations, such as listing the botnet’s bots, managing the network, and sending commands.

In a first experiment, we confirmed the capacity of the bot to discover an active Skype session and to silently register itself as a trusted plugin without explicit user authorization, as outlined in Section 3.2. Then, once given to each bot a single gate node (chosen among the botnet nodes), we verified that all bots correctly joined the botnet, by registering with $l = 2$ neighbors nodes. By doing so, we validated the implementation of the botnet bootstrap protocol formulated in Section 3.1. Table 3 details the timings of bootstrapping steps; bots were able to complete their bootstrapping procedure, on average, within 12 seconds.

Bootstrap action	**Time (s)**
Attaching to Skype	1.55
Contacting the Botmaster	4.40
Linking to the Network	6.03
Total	11.98

Table 3. Bootstrap timing.

In a second experiment, we booted every bot and made the botnet run for about 14 hours. We used an ad-hoc *fuzzer* to instruct commands to the bots at random time intervals, registering 1,373 total issued orders. The average time for the master to obtain an answer from a bot that executed a command ranged from 5.25 to 15.75 seconds. We noticed that the variability in this measure depends mainly on the network topology of our parasitic overlay: older nodes, which are usually placed closer to the botmaster, receive their messages first regardless of Internet-level proximity with their neighbors.

With this botnet topology, the bots’ answers were reaching the botmaster with an average hop count of 3.62 (see Figure 4). This count is higher than the numbers presented in Figure 2(a), due to a lower value $l = 2$ chosen in this setting. However, only a slow logarithmic growth of this value (and of message delivery latencies) is to be expected with the growth of the network size.

To estimate the amount of traffic generated from our overlay botnet to keep the botmaster concealed, we picked two random nodes, where we measured and classified the number of incoming and outgoing messages. The number of dupli-

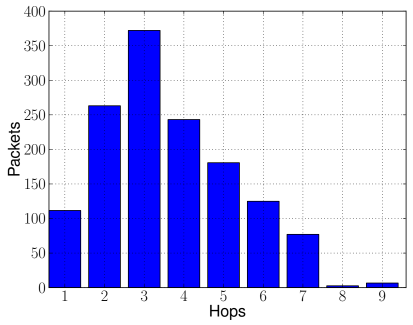


Fig. 4. Number of hops.

cates between all received messages ranged between 74% and 83%. This added cost in network traffic is the price to pay in order to obtain the resilience properties described in Section 4.1.

Finally, to evaluate the resources consumed by our prototype, we installed the *HotSanic* analysis tool [38] on all bots to monitor network, CPU, and memory used by the prototype. Our bandwidth consumption was below 1KB/s even if during the bootstrap phase it was possible to notice some peaks around 6KB/s caused by the messages employed for bootstrapping. We can easily assume that the bandwidth consumption is very low. We did not observe noticeable variations on the use of CPU and memory.

5 Security Analysis

In this Section, we discuss possible attacks and countermeasures against our botnet model, focusing on each different part of the botnet lifecycle. In the following we refer to "attacker" as a security analyst that is trying to compromise the botnet model. We assume that the attacker is able to reverse engineer the malware and collect traffic dumps of known bots.

First, it is extremely difficult to obtain information about the network topology by observing the traffic sent and received by bots: all traffic undergoes two levels of encryption (one provided by Skype, the other by our scheme); the botmaster can manage the network using any infected node as entry point and change it at will; the routing behavior adopted by the botmaster's node is exactly the same as any other node. To make the job of the attacker even more difficult, nodes can be instructed to add random delays to message forwarding and to randomly generate "garbage" messages that will be delivered to the whole network even if they cannot be decrypted by any peer.

Second, an attacker that takes control of an infected machine gains access to the list of neighbors' Skype usernames and to the messages addressed from the master to that bot. This data can be used to detect what the botnet, or part of it, is currently up to (e.g. which SPAM campaigns it is currently perpetrating), but not very much can be said about the overall botnet infrastructure. Nodes can change Skype identifiers if the botmaster instructs them to do so, making information about neighbors short-lived. The botmaster is still very difficult to track since, when seen from a neighbor's point of view, it is not distinguishable from any other infected host.

Third, if an attacker can successfully reverse-engineer the malware, she is able to discover the hard-coded GNs and to collect the announce sent during bootstrap. With this information, she can perpetrate a replay attack on the botnet. This attack is done by repeatedly delivering announce messages to progressively gather neighbor nodes received during the bootstrap phase. In order to mitigate this attack, it is possible to limit the number of neighbor nodes sent to new bots within any defined temporal window.

Finally, since in our model messages are flooded to the whole network, an attacker can try to overburden the botnet nodes by sending a large number of meaningless messages to the network; to mitigate this effect, we propose to adopt a rate-limiting approach on incoming messages [28]: in this way, only a given maximum number of messages from each neighbor is routed to the rest of the network; thus, only the closest neighbors of each attacker will be likely to suffer from the attack.

6 A Host-Based Skype Malware Detector

We have so far discussed how a hideous Skype plugin can infect a victim to make it part of a botnet. In the following, we describe an approach that allows for a deep analysis of Skype plugins, possibly leading to the detection of the evil ones.

As mentioned earlier, Skype's network traffic is encrypted with strong cryptography algorithms and its binary is protected by anti-debugging and obfuscation techniques [3, 17]. While these functionalities create a very good protection system for common users, they also constitute a limit as it is almost impossible for an external entity to investigate Skype's internals and its network traffic. Our idea leverages the fact that, while all the network traffic is encrypted, the messages exchanged on the API communication channel established between Skype and a plugin, as seen in Section 2.1, are completely in clear text. It is therefore possible to analyze the actions performed by a plugin before they are delivered to the Skype core to infer a model that describe the plugin's behaviors at best.

We propose a behavior-based analysis of the *command protocol layer* (CPL) of the Skype API, for the purpose of detecting whether an application is performing malicious actions through Skype. The set of Skype's API commands is quite small and therefore behavior-based analysis can be very effective when applied to the CPL.

The command protocol layer API is based on messaging between Skype and a plugin. This messaging is performed by leveraging the system's standard functions. In a first setup phase, known as *attach* phase, a plugin establish a channel between itself and Skype, known as the *communication layer*. Messages are then exchanged over this layer between the two applications, using a plain-text command protocol.

To perform our analysis, we hijack the attach phase. The hijacking is done through a two-part system: the first component is a Skype plugin of its own, known as *proxy*, while the second one is `WUSSTrace`, a Windows user-space system call tracer [34]. When the proxy receives messages from the target plugin, it simulates the corresponding Skype behavior and replies, thus establishing a communication layer between itself and the plugin. From now on, the proxy acts as a relay and forwards commands sent over the channel from the target plugin to Skype, and viceversa. The plugin is unaware of not being communicating directly with Skype and it consequently behaves normally.

The proxy component includes an analysis engine and several models of malicious behaviors that we created observing the API calls issued by existing Skype malware. By matching the behavior of the attached plugins with the malicious models at our disposal, it is possible to give an preliminary evaluation of the plugin behavior.

Our behavior-based approach is similar to the ones proposed by other malware detection and analysis systems [10, 35]. The main difference lies in the fact that we apply it to a different (and higher) level, i.e. the API CPL. By keeping our analysis at the higher CPL level, we avoid all the fine-grained details that other techniques must cope with. These details include, for example, syscall analysis and system API analysis, that are often greatly complex and costly. At the same time, we are able to extract the behavioral semantic of a Skype plugin, exactly as similar techniques.

The first set of results we obtained shows a high rate of false-positive during analyses performed on a certain temporal window. We plan to overcome this limitation by appropriately throttling the temporal window size and inserting an API message rate limiting. This limit should be tailored on the number of interesting API actions performed by plugin in a certain amount of time. We have defined a small set of "interesting" actions and we plan, as a future work, to refine this set by observing the behavior of benign and malign plugins. Through these observations and experiments, we also plan to better refine the temporal window parameter used so far. Although the classification technique is still in an early development stage we are convinced that our approach is a good starting point.

7 Related Work

The botnet phenomenon has quickly become a major security concern for Internet users. As such, it has rapidly gained popularity among the mass media and the attention of the research community interested in understanding, analyzing, and detecting bot-infected machines [29, 44, 45]. Even worse, such a threat is nowadays exacerbated by the fact that malware authors are willing to give up their fame for an economic profit [12, 45]: a reason that motivates by itself miscreants, more than ever, to constantly work towards stealth, high-resilient, and low-cost botnets.

Storm [29] and Waledac [44] are probably the two most famous P2P-based botnets present in the wild. Although these botnets are hard to track down due to their decentralized infrastructure, researchers have shown how it is possible to infiltrate them, disrupt their communications, and acquire detailed information about the botnets' *modus operandi* (e.g., spreading/infection mechanisms, type of threats, corpus of infected hosts).

To overcome such drawbacks, Starnberger *et al.* presented Overbot in [43]. Overbot uses an existing P2P protocol, Kademlia, to provide a stealth command and control channel while guaranteeing the anonymity of the participating nodes and the integrity of the messages exchanged among them. Overbot is the closest work to ours; although we share a similar underlying decentralized structure, there are a number of salient properties that set the two approaches apart. In our approach, the communication bootstrap of a node starts by sending messages using a set of pre-defined nodes–gate nodes (GNs)–that are shipped with the malware. Gate nodes are ordinary bot-infected nodes, and, as such, they perform message routing exactly as any other node. They are just used during the initial bootstrap phase every time a new infected node wants to join the network, but, after that, they do not need to continuously receive communication. On the other hand, sensor nodes in Overbot are a resident part of the botnet: an observer may perform statistical analysis and inferences on the traffic that such nodes generate and receive. Furthermore, Overbot's sensor nodes are equipped with the private key of the botmaster. This means that once a node is compromised, it becomes possible to decrypt *all* the traffic sent to the botmaster. On the other hand, a compromised node in our approach exposes only its symmetric key, which gives the chance to disclose the traffic sent only by that node.

Research in the analysis and detection of bot-infected machines has been quite prolific in the past years [14, 21, 39, 5, 23, 25, 26, 24, 31, 51, 46, 50, 9].

Original signature-based systems, focused on the detection of syntactic artifacts of the malware, are vulnerable to obfuscation techniques and have thus been superseded by approaches that aim to characterize the behavior of a malicious samples on end-user systems [32, 35, 52, 19, 10]. These approaches are usually effective in analyzing, detecting and describing malware behaviors. Unfortunately, the ability of the malware to mimic legitimate process behaviors may trick such systems to produce too many false alarms. Moreover, the computational resources required to perform the analysis are non-negligible, and, even worse, users are required to install the analysis platform on their machines. Therefore, it is desirable to have complementary solutions that monitor network events to spot malware-infected machines.

From a different perspective, research in the detection of bots based on the analysis of network events has proceeded by following two main directions. One line of research revolves around the concept of *vertical correlation*. Basically, network events and traffic are inspected, looking for typical evidence of bot infections (such as scanning) or command and control communications [25, 23, 5]. Unfortunately, some of these techniques are tailored to a specific botnet structure [23, 5], while others rely on the presence of a specific bot-infection lifecycle [25]. Others are not immune to encoding or encryption schema as they require to analyze the packets' payload [50], and others again are sensible to perturbation in the network or timing features of the observed connections [9].

The second line of botnet detection research, instead, focuses mainly on *horizontal correlation*, where network events are correlated to identify cases in which two or more hosts are involved in similar, malicious communication. Interesting approaches are represented by BotSniffer [26], BotMiner [24], TAMD [51], and the work proposed in [46]. Except for [46], which detects IRC-based botnets by analyzing aggregated flows, the main strength of the these systems is that they are independent on the underlying botnet structure, and thus, they have shown to be effective in detecting pull-, push-, and P2P-based botnets. On the other hand, correlating actions performed by different hosts requires that at least two hosts in the monitored network are infected by the same bot. As a consequence, these techniques cannot detect single bot-infected hosts. This is a significant limitation, especially when considering the trend toward smaller botnets [14].

Even worse, state-of-the-art techniques [24] are generally triggered upon the observation of malicious and noisy behavioral patterns, where scan, SPAM, and DDoS activities are probably the most representative actions. Unfortunately, in their quest to an ever increasing illegal financial gain [20, 27] and to avoid being easily detected by making *much ado for nothing*, bots engage in low-pace, legitimate-resembling activities [45]. Spotting such communications becomes then a very hard task, which, in the end, hampers the detection of the infected machines.

The parasitic overlay network presented in this paper has all the features required to thwart the current state-of-the-art botnet detection approaches. Message encryption hampers the creation of content-based network signatures, while unknown routing strategies make it difficult to track down IP addresses. In addition, Skype itself makes the network highly resilient to failure and provide a massive user corpus, which gives the chance to rely on a non-negligible number of bots. It is worth noting that speculations on using Skype as a vehicle to build a powerful botnet infrastructure have been around for a while [7, 15, 30, 47, 48, 13]. Fortunately, to the best of our knowledge, such rumors have never evolved into a full-fledged Skype-based botnet in the wild. We have nonetheless shown that such a botnet can be easily designed and implemented. In addition, our simulation and deployment experiments have shown that building a stealthy, resilient and low-cost botnet is indeed possible and practical. Research in botnet detection must thus be refined to deal with the threats posed by such advanced malicious networks that are likely to appear in the near future.

8 Conclusion

In this paper we have described that the design and implementation of a stealth, resilient, and cost-effective botnet based on a general overlay network, such as the one offered by Skype, is not a chimera. It is indeed a practical and realistic threat that may emerge in the near future. The unstructured *parasitic overlay* network proposed, effectively propagates messages leaving to each node only a limited knowledge of the whole network topology, making the botmaster difficult to track down and making the network difficult to map.

In the *parasitic overlay*, messages are flooded through the network to avoid propagating information about how to reach the botmaster, relying on cryptography to ensure that the messages can be only be read by the intended recipients. In future work, taking inspiration from routing strategies in anonymous peer-to-peer networks [40, 11, 16], we intend to explore more efficient routing strategies for messages, making sure that the information given to each node makes it still difficult to track down the botmaster.

Since we believe that the menace posed by the model of botnet presented in this paper will soon emerge, our future works will focus also on the improvement of the host-based detection technique we briefly outlined.

9 Acknowledgments

This work has been supported by the European Commission through project FP7-ICT-216026-WOMBAT, by the French ANR through the VAMPIRE project and through the MECANOS project by the POLE de Competitivite SCS (France).

References

1. ADNKRONOS INTERNATIONAL. Italy: Govt probes suspected mafia use of Skype, February 2009. `http://www.adnkronos.com/AKI/English/Security/?id=3.0.3031811578`.
2. ANDERSON, N. Is Skype a haven for criminals?, February 2006. `http://arstechnica.com/old/content/2006/02/6206.ars`.
3. BASET, S., AND SCHULZRINNE, H. An analysis of the Skype peer-to-peer internet telephony protocol. In *CoRR* (2004).
4. BBC. Italy police warn of Skype threat, February 2009. `http://news.bbc.co.uk/2/hi/europe/7890443.stm`.
5. BINKLEY, J. R. An algorithm for anomaly-based botnet detection. In *SRUTI '06* (2006).
6. BIONDI, P., AND DESCLAUX, F. Silver Needle in the Skype, March 2006.
7. BLANCHER, C. Fire in the Skype–Skype powered botnets..., October 2006. `http://sid.rstack.org/pres/0606_Recon_Skype_Botnet.pdf`.
8. BOLLOBÁS, B. *Random Graphs*. Cambridge University Press, January 2001.
9. CAVALLARO, L., KRUEGEL, C., AND VIGNA, G. Mining the network behavior of bots. Tech. Rep. 2009-12, Department of Computer Science, University of California at Santa Barbara (UCSB), CA, USA, July 2009.
10. CHRISTODORESCU, M., JHA, S., SESHIA, S. A., SONG, D., AND BRYANT, R. E. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005)* (2005).
11. CIACCIO, G. Improving sender anonymity in a structured overlay with imprecise routing. In *Lecture Notes in Computer Science* (2006).

12. CNET News. Hacking for dollars, July 2005. `http://news.cnet.com/Hacking-for-dollars/2100-7349_3-5772238.html`.
13. CNET News. Skype could provide botnet controls, January 2006. `http://news.cnet.com/2100-7349_3-6031306.html`.
14. Cooke, E., Jahanian, F., and McPherson, D. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet* (2005).
15. Danchev, D. Skype to control botnets?!, January 2006. `http://ddanchev.blogspot.com/2006/01/skype-to-control-botnets.html`.
16. Dell'Amico, M. Mapping small worlds. In *IEEE P2P 2007* (2007).
17. Desclaux, F., and Kortchinsky, K. Vanilla Skype part 2, June 2006.
18. Ebay. Ebay, Paypak, Skype 2009 Q1 financial report. `http://ebayinkblog.com/wp-content/uploads/2009/04/ebay-q1-09-earnings-release.pdf`.
19. Egele, M., Kruegel, C., Kirda, E., and Yin, H. Dynamic Spyware Analysis. In *Proceedings of the 2007 Usenix Annual Conference (Usenix 07)* (2007).
20. Franklin, J., Paxson, V., Perrig, A., and Savage, S. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security* (2007).
21. Freiling, F. C., Holz, T., and Wicherski, G. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *In Proceedings of 10 th European Symposium on Research in Computer Security, ESORICS* (2005).
22. Gnutella Development Forum. Gnutella protocol specification. `http://wiki.limewire.org/index.php?title=GDF`.
23. Goebel, J., and Holz, T. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets* (2007).
24. Gu, G., Perdisci, R., Zhang, J., and Lee, W. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium* (2008).
25. Gu, G., Porras, P., Yegneswaran, V., Fong, M., and Lee, W. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of the 16th USENIX Security Symposium* (2007).
26. Gu, G., Zhang, J., and Lee, W. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)* (2008).
27. Gutmann, P. The Commercial Malware Industry. In *Proceedings of the DEFCON conference* (2007).
28. He, Q., and Ammar, M. Congestion control and message loss in Gnutella networks. In *Proceedings of SPIE* (2003).
29. Holz, T., Steiner, M., Dahl, F., Biersack, E., and Freiling, F. Measurements and Mitigation of Peer-to-Peer-based Botnets:A Case study on Storm Worm. In *USENIX Workshop on Large Scale Exploits and Emerging Threats* (2008).
30. IT World. Making a PBX 'botnet' out of Skype or Google Voice?, April 2009. `http://www.itworld.com/internet/66280/making-pbx-botnet-out-skype-or-google-voice`.
31. Karasaridis, A., Rexroad, B., and Hoeflin, D. Wide-scale Botnet Detection and Characterization. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets* (2007).

32. Lanzi, A., Sharif, M., and Lee, W. K-Tracer: A System for Extracting Kernel Malware Behavior. In *the 16th Annual Network and Distributed System Security Symposium (NDSS'09)* (2009).
33. Leiden, J. Anti-mafia cops want Skype tapping, Feburary 2009. `http://www.theregister.co.uk/2009/02/24/eurojust_voip_wiretap_probe/`.
34. Martignoni, L., and Paleari, R. WUSSTrace - a user-space syscall tracer for Microsoft Windows. `http://security.dico.unimi.it/projects.shtml`.
35. Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., and Mitchell, J. C. A Layered Architecture for Detecting Malicious Behaviors. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection, RAID, Cambridge, Massachusetts, USA* (Sept. 2008).
36. Microsoft. MSDN Library on developing Windows User Interfaces. `http://msdn.microsoft.com/en-us/library/ms632587(VS.85).aspx`.
37. Passerini, E., Paleari, R., Martignoni, L., and Bruschi, D. FLuXOR: Detecting and Monitoring Fast-Flux Service Networks. In *Lecture Notes in Computer Science* (2008).
38. Pissny, B. HotSanic, HTML overview to System and Network Information Center, July 2004. `http://hotsanic.sourceforge.net`.
39. Rajab, M. A., Zarfoss, J., Monrose, F., and Terzis, A. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement* (2006).
40. Sandberg, O. Distributed routing in small-world networks. In *ALENEX 2006* (2006).
41. Schneier, B. Bavarian government wants to intercept Skype calls. `http://www.schneier.com/blog/archives/2008/02/bavarian_govern.html`.
42. Sissel, J. xdotool. `http://www.semicomplete.com/projects/xdotool/`.
43. Starnberger, G., Kruegel, C., and Kirda, E. Overbot - A botnet protocol based on Kademlia. In *Proceedings of the International on Security and Privacy in Communication Networks, SecureComm, Istambul, Turkey* (2008).
44. Stock, B., Goebel, J., Engelberth, M., Freiling, F., and Holz, T. Walowdac - Analysis of a Peer-to-Peer Botnet. In *European Conference on Computer Network Defense (EC2ND)* (November 2009).
45. Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., and Vigna, G. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS'09)* (2009).
46. Strayer, W. T., Walsh, R., Livadas, C., and Lapsley, D. Detecting botnets with tight command and control. In *Proceedings of the 31st IEEE Conference on Local Computer Networks* (2006).
47. TechWorld. Cambridge prof warns of Skype botnet threat. VoIP traffic can cover a multitude of sins., January 2006. `http://news.techworld.com/security/5232/cambridge-prof-warns-of-skype-botnet-threat/`.
48. TechWorld. How bad is the Skype botnet threat? Skype's sneakiness leads to a security risk., January 2006. `http://features.techworld.com/security/2199/how-bad-is-the-skype-botnet-threat/`.
49. EU Forward. Forward: Managing Emerging Threats in ICT Infrastructures, 2008. `www.ict-forward.eu`.
50. Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., and Kirda, E. Automatically Generating Models for Botnet Detection. In *14th European Symposium on Research in Computer Security (ESORICS), Lecture Notes in Computer Science, Springer Verlag* (2009).

51. Yen, T.-F., and Reiter, M. K. Traffic Aggregation for Malware Detection. In *DIMVA '08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2008).
52. Yin, H., Song, D., Egele, D. M., Kruegel, C., and Kirda, E. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security* (2007).