BotMosaic: Collaborative Watermark for Botnet Traceback

Amir Houmansadr
Department of Electrical and Computer Engineering
University of Illinois at Urbana Champaign
Urbana, Illinois 61801–2307
Email: ahouman2@illinois.edu

Nikita Borisov
Department of Electrical and Computer Engineering
University of Illinois at Urbana Champaign
Urbana, Illinois 61801–2307
Email: nikita@illinois.edu

***Abstract*—We study the problem of finding a botmaster who is controlling a botnet. Botmasters will typically use several stepping stones to connect to the botnet to disguise their true location. While there has been a lot of work on detecting stepping stones, it has focused on interactive stepping stones that follow a different pattern of communication than botnets. Thus a different approach is necessary.**

We develop a new network flow watermarking scheme, called BotMosaic. The scheme works by capturing multiple bots within a honeynet and then inserting a watermark onto the communication stream of each bot towards the botmaster. By creating a *collaborative* watermark that is spread across several bot communication streams, it is possible to overcome the particular nature of botnet command-and-control communication that renders other watermarking techniques ineffective. Given a sufficient collection of trapped bots, the botmaster's connection can be detected with very low false error rates. BotMosaic detection is low-cost so it can be easily deployed by ISPs and enterprises to help address the botnet problem. It can also be used to locate internal hosts that are part of the botnet, i.e., bots, and also internal compromised machines being used as botnet stepping stones, providing an additional incentive for incremental deployment.

I. Introduction

A *botnet* is a network of compromised machines, *bots*, that is controlled by one or more *botmasters* to perform coordinated malicious activity. Botnets are among the most serious threats in cyberspace due to their large size. Reports in the media discuss botnets reaching a million nodes or more [1]. This enables the bots to carry out various attacks, such as distributed denial of service, spam, and identity theft, on a massive scale.

Botnets are controlled by the botmasters by means of a command-and-control (C&C) channel. This channel is used to both monitor the bots and issue them commands. In some cases, bots will also use this channel to send back information, such as stolen passwords. A common C&C structure is to use an Internet Relay Chat (IRC) server. All the bots in a network join a channel on the IRC server, and the botmaster uses the channel to broadcast commands, with responses being sent back via private messages. The IRC protocol is designed to support large groups of users and a network of servers to provide scalability and resilience, thus it forms a good fit for providing a C&C infrastructure. Some botnets use a more advanced structure, with bots communicating directly with each other in a peer-to-peer fashion, but many existing botnets still use the IRC model and will be the focus of our work.

Existing research on defending from botnets focuses largely on detecting and disabling individual bots, or the C&C servers. However, this does not eliminate the botnets, but rather only slows them down, as new computers are infected and replacement C&C servers are put online. An alternate approach is to locate the botmasters and hold them accountable. This is made difficult by the fact that botmasters will typically use a sequence of *stepping stones* [2] to connect to the IRC server, disguising their true location. There is a rich literature on detecting stepping stones [2], [3], [4], [5], [6], [7], [8], [9], but it focuses on interactive stepping stones that have different communication patterns than botmasters.

The two main difficulties to overcome are the short-lived nature of bot-to-botmaster communication and the large volume of traffic from other bots that acts as "chaff." To address both these issues, we design a new type of network flow watermark that is specialized to this task. Our approach is to capture a collection of bots inside a honeynet. We then insert a *collaborative* watermark that is spread across the communication stream of the entire collection of captured bots. Each stream is minimally modified, but as the streams are aggregated on the way to the botmaster, the combined watermark becomes visible. The watermark can be detected at a very low cost by detectors deployed at ISPs and enterprise border routers. Our watermark can also be used to locate stepping stones—compromised computers used by botmasters—as well as infected bot machines within a network. As such, there is added incentive for incremental deployment.

We analyze our scheme using simulations and experiments on PlanetLab [10]. We find that we can achieve a high rate of detection with few false positives using a watermark applied to small number of captured/imitated bots in the network, and the resulting detection time of about a minute.

The rest of the paper is organized as follows: Section II discusses the related work along with the general motivation for our watermark-based traceback scheme. In Section III we mention the detailed structure of the proposed collaborative watermarking scheme, while the main topology of the watermarking system is described in Section IV. Simulations and implementation results are mentioned in Sections V and the paper is concluded in Section VI along with some directions for the future research.

II. Related work and motivation

Botmaster traceback presents several difficulties. First of all, botmasters, to keep their identity secret, will typically not connect to a C&C server directly, but rather, use a series of (usually compromised) computers to relay their communications, via SOCKS proxies or SSH tunnels; this approach is generally known as using a *stepping stone* [2]. Stepping stone connections are often encrypted, making it difficult to recognize botnet control commands being relayed.

Additionally, a botmaster will connect to the control channel for short periods of time and communicate using short bursts of command-response pairs. This makes it difficult to correlate traffic to reveal the existence of a stepping stone; the existing techniques detect *interactive* stepping stones, where there is a long-lived stream of typing traffic, providing more data for correlation techniques. We aim to use *network flow watermarking* for botnet traceback problems.

Network flow watermarking is a technique that actively perturbs the traffic patterns of a network flow to insert a watermark inside them that can later be detected. It has been used to detect stepping stones, as well as compromise anonymous communication [4], [11], [5], [12], [13], [8]. Existing techniques, however, cannot be applied to the problem of botmaster traceback for two reasons. First, they are designed to work on long-lived flows; typically, hundreds of packets are necessary to detect the presence of a watermark. Botmaster communication, however, tends to be short-lived, with only a few packets sent from each bot. Furthermore, a watermark that is applied to a single bot-to-botmaster communication will be overwhelmed by traffic from other bots that will be aggregated along the same stepping stone connection. Although some of the existing watermarks are designed to resist a reasonable level of chaff, they do so by increasing the length of the watermark and thus cannot be used for botnet traceback in practice.

Ramsbrock et al. [14] designed a watermark specifically targeted to the task of botmaster traceback. Their watermark relied on adding extra whitespace at the end of IRC messages sent by bots in order to inject a pattern into packet sizes. They also adjusted the timings of packets in order to improve detection ability. Though an important first step, their whitespace watermarking approach has several important limitations. Whitespace watermarking only works well in the presence of low rates of chaff—less than 0.5 packets/second—whereas even in a small-size botnet, an aggregate response from all the bots would create a significantly higher chaff rate. Whitespace watermarking is also fragile to repacketization or retransmission of packets, as such events can cause it to lose timing synchronization. Finally, whitespace watermarking relies on modifying the contents of the messages sent by the bots, which can be easily detected by the botmaster.

This leads us to develop a new watermarking scheme, BotMosaic, for the task of botmaster traceback. BotMosaic uses a collaborative approach to distribute the watermark signal across multiple bot-to-botmaster connections. BotMosaic is specialized to the task of botmaster detection: just as generic stepping stone detection solutions do not work for botmaster traceback, BotMosaic cannot be used to detect other kinds of stepping stones.

III. BotMosaic scheme

Our aim is to design a watermark-based approach for the botnet trace back problems in an IRC-based botnet, i.e., tracing back a botmaster, detecting bots and botnet-related stepping stones. The work is different from similar watermarks in being collaborative by using *multiple* rogue bots for watermark insertion, which makes it specialized for the problem of botnet watermarking. *Rogue bots* are the bots collected/imitated and controlled by the watermarking system, as opposed to *real bots*. Multiple rogue bots allow us to spread the watermark power over a larger number of communications, compensating for the small amount of communication each individual bot performs with the botmaster. BotMosaic adopts an interval-based design used in other watermarks [5], [12], [13] to provide robustness to packet losses, delays, and reordering as well as chaff introduced by the traffic of the other bots in the botnet.

To obtain a large number of rogue bots, we propose using a darknet operation together with a honeynet framework [15], [16]. A network telescope can be used to route a large segment of the dark address space to a cluster of honeypot machines. The machines themselves can use virtualization to appear as a large number of hosts with various system configurations. Over time, we can expect that a number of the virtual honeypot machines will be compromised and start hosting bots that participate in a botnet. It may also be possible to deliberately infect other machines to increase the number of rogue bots.

We then insert a watermark in a coordinated fashion into the communications of *all* of the rogue bots. This allows us to insert a much stronger signal into the encrypted communication link back towards the botmaster/bots that is detectable even if these communications are mixed with traffic from other, real bots. The last component of our architecture is a series of detectors deployed around the Internet. The detectors maintain loose time synchronization with the watermarkers and use interval-based detection to account for packet delays, drops, or reordering. Section II describes how detectors are used for three different botnet problems, describing the incremental deployment of the scheme. Our overall architecture is shown in Figure 1.

A. Watermark insertion

We establish R rogue bots, joining and communicating with the botnet through the IRC C&C channel. By implementing rogue bots on the same machine using virtualization mechanisms like the one in [17], R can be made reasonably large without needing additional resources (other than real IP

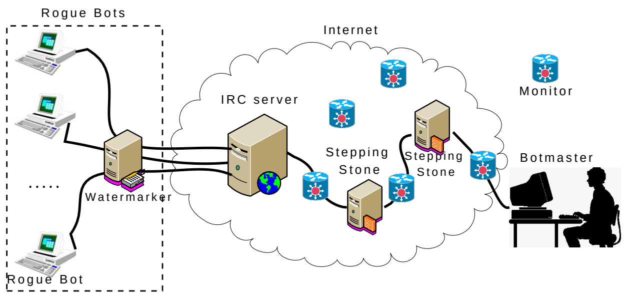


Fig. 1. BotMosaic watermarking traceback scheme.

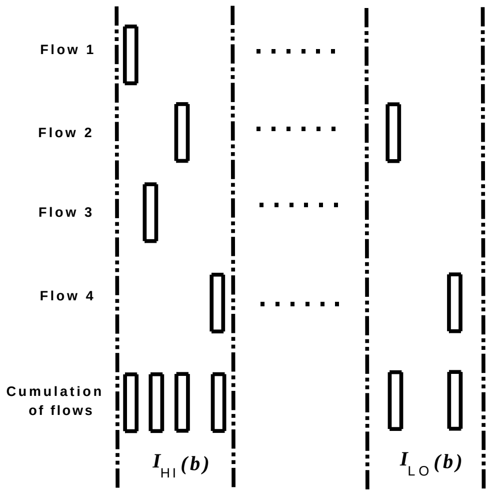


Fig. 2. Collaborative watermark insertion using multiple flows.

addresses). Letting B to be the number of *active* real bots connecting through the same C&C channel, increasing the ratio of the number of rogue bots to that of the real bots (R/B) improves watermark detection efficiency, as will be shown in the following sections.

We design a collaborative interval-based watermarking scheme to mark rogue bot traffic. To the best of our knowledge, this is the first collaborative network flow watermarking scheme. Figure 2 shows the main idea of the collaborative watermark insertion scheme. Each of the flows contain a share of the watermark so that mixture of the packets of these flows, performed in the botnet C&C channel, generates the watermark pattern while the watermark can not be detected from any of the single rogue bot flows.

The watermarker inserts an l-bit watermark sequence into the botnet flows. For any botnet, we assign two different watermark sequences; a *botmaster watermark* that is applied to the traffic being sent privately to the botmaster, i.e., PRIVMSG messages targeted to the botmaster, and a *bot watermark* applied to the traffic getting broadcast to the C&C channel, i.e., PRIVMSG messages targeting the channel (w_1 and w_2 in Figure 3, respectively). We also assign different pairs of watermarks for different botnets to be able to distinguish between detected entities related to different botnets.

We divide the time axis into $2l$ non-overlapping intervals with equal lengths T. Each watermark bit corresponds to two randomly selected intervals, with one of them is tagged as a HI interval and the other tagged as a LO interval.

The set of the HI-LO intervals of all the watermark bits is part of the watermarking key and is used for watermark detection. The general idea of watermarking is to send more packets in HI interval of each watermark bit compared to its corresponding LO interval in the mixture of all rogue flows. To send the b^{th} watermark bit we assign the timing of R rogue flows so that in the mixture of all R rogue flows the number of packets appearing in the HI interval of that bit is larger than the number of packets in the corresponding LO interval by some threshold η (see Figure 2). In other words, we should have

$$\sum_{f=1}^{R} N_f(I_{HI}(b)) - \sum_{f=1}^{R} N_f(I_{LO}(b)) \geq \eta \qquad b = 1, \dots, l \quad (1)$$

where $N_f(i)$ is the number of packets showing up in the i^{th} interval of f^{th} rogue flow, and $I_{HI}(b)$ and $I_{LO}(b)$ are the HI and LO intervals corresponding to b^{th} watermark bit, respectively.

For the generation of the rogue flows the watermarker first determines the total number of packets in each interval of the aggregated watermarked flow and then assigns the number of packets to each rogue flow, namely *watermark shares*. Based on IRC standards [18] the rate of the flows from IRC client to the server should not exceed 0.5 packets per second (exceeding this threshold causes the client to be penalized by extra delays on its packets). So, in an interval with length T seconds we expect to have at most $\frac{T \cdot R}{2}$ packets accumulated over all R rogue flows. For any watermark bit b, we randomly select the cumulative packet number for its HI interval to be:

$$N(I_{HI}(b)) \epsilon \left[\frac{T \cdot R}{4} \qquad \frac{T \cdot R}{2} \right] \quad (2)$$

We then assign the total number of packets in the $I_{LO}(b)$ interval to be:

$$N(I_{LO}(b)) = N(I_{HI}(b)) - \eta - \psi \quad (3)$$

where η is the *detection threshold* and ψ is the *confidence threshold*. Doing so for all of the watermark bits we find the cumulative number of packets $N(i)$ for any interval i. We then randomly distribute $N(i)$ within the i^{th} interval of all R rogue flows so that the overall rate of each flow does not exceed the 0.5 packets/sec constraint mentioned above.

Note that in this scenario, no information about the watermark can be inferred from a single watermarked flow; the difference of the packets in the corresponding HI -LO intervals of a single rogue flow might be either greater or smaller than the detection threshold. The watermark detection succeeds on the mixture of rogue flows.

B. Detection Scheme

A watermark monitor will intercept all traffic at a particular point on the Internet and attempt to detect the watermark in each flow that traverses the point. The watermarker shares a key with the detector, consisting of the interval size T and the location of the HI and LO intervals of the watermark:

$$Key = (T, \{\forall b = 1, \ldots, l : I_{HI}(b), I_{LO}(b)\}) \quad (4)$$

We assume that the target mixed-encrypted flow contains packets from R rogue bots and B real bots. A candidate flow is broken up into intervals, and then the watermarker computes $N_{HI}(i)$ and $N_{LO}(i)$ to be the number of packets in $I_{HI}(i)$ and $I_{LO}(i)$ intervals, respectively. Then we define:

$$\Delta(i) = N_{HI}(i) - N_{LO}(i) \quad (5)$$

We use two thresholds to decide whether a watermark is present. First, if $\Delta(i) > \eta$ (η is the detection threshold in (3)), we call that bit *detected*. The confidence threshold ψ ensures that natural variations in numbers of packets do not destroy a watermark bit.

Finally, we say that a watermark has been detected if the total number of detected bits n_c (out of l bits) is greater than or equal to some threshold θ, *hamming threshold*. It is easy to see that by increasing η and θ, we can decrease the number of false positives at the cost of creating more false negatives. We will discuss parameter choices in Section V.

Due to the delays applied to the mixed watermarked flows passing through the network, monitors need to do *synchronization* of the flows before detection. This is done using sliding windows, as will be discussed in section V-B1.

IV. Topology of the Traceback System

To locate botmasters, BotMosaic requires detectors to be deployed in other parts of the Internet. The chances of locating the botmaster increase with wider deployment of detectors. However, BotMosaic can be useful not only to find botmasters, but also to identify stepping stones inside a network and to locate compromised machines acting as bots. As such, there is an inherent benefit for an ISP or an enterprise to deploy a low-cost (see Section V-B2) detector within their network, something that we expect will eventually lead to widespread deployment

We next describe how BotMosaic can be used to perform each of the three above tasks; the deployment of watermarkers and detectors follows the topology depicted in Figure 3.

a) Tracking Down the Botmaster: If one of the border routers of *a* network deploying BotMosaic detects the botmaster-related watermark, w_1, on one of the flows entering the network, and none of its border routers find the same watermark simultaneously leaving the network, this indicates that the botmaster for that particular botnet resides within the network (network C in Figure 3). In this case, the destination address of the captured watermarked flow belongs to either the botmaster, or a stepping stone inside the network of botmaster residence.

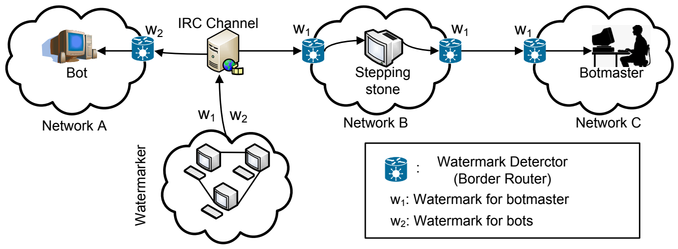


Fig. 3. Topology of the traceback system.

b) Detecting Stepping Stones: The border routers of an enterprise or ISP deploying BotMosaic may observe a watermark both entering its network and exiting at the same time (network B in Figure 3). This means that a computer inside the network is acting as a stepping stone and the administrators can take action to correct the compromise. Note that this is a variant of more general stepping stone detection techniques. However, unlike existing watermark-based stepping stone detection schemes [5], [12], [8], in this special case the ISP or enterprise does not need to insert watermarks on its own traffic. Thus the organization can avoid introducing a new inline device or module into its routers, which may unduly delay legitimate traffic and create a new point of failure—detection of BotMosaic watermarks can be performed on a mirrored port.

c) Detecting Bots: As mentioned before, to detect infected machines, bots, BotMosaic uses a watermark sequence, w_2, different than what intended for botmaster traceback and inserts it on the flows being sent to the C&C channel instead of the private communication to the botmaster. This would allow organizations, such as network A in Figure 3, to detect compromised machines inside their networks that participate in the botnet by observing the specified watermark sequences entering the network. This extra advantage of the detectors can serve as an added incentive for their widespread deployment.

V. Simulations and Experiments

The simulations and experiments in this section only consider the watermark applied on the botmaster traffic, due to space constraints, as the detection performance is the same for the watermark inserted in the bot traffic.

A. Simulations

We simulated the proposed scheme to evaluate its performance. We did the simulation in Matlab using the traces of botnets for *spybot* and *sdbot* botnets used originally in the BotMiner research [19].

1) sdbot trace: The sdbot trace belongs to a botnet with a botmaster, and four real bots. Since we needed a larger botnet to evaluate our scheme, we extended the trace to have 100 real bots.

Based on analyzing the trace, bots mainly are listening on the IRC channel for the commands, and upon receiving a command they respond to it appropriately. To extend the botnet trace from the existing 4-bot trace to a 100-bot trace, we sent a response to the channel on behalf of the newly added bots after a random delay whenever a command was issued and the existing bots responded to it.

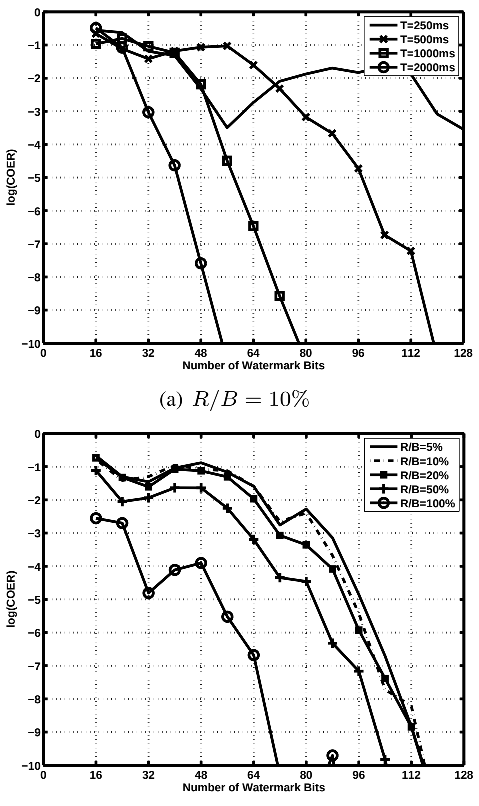
To simulate the watermarking scheme, we added watermarked flows generated by the rogue bots to the trace for different settings of watermark parameters. For each run of the simulations, we generated a new extended trace, selecting a different part from the real trace randomly and extending the trace for 100 bots as discussed above. In our scheme, *true positive bits* parameter is the number of detected bits by the detection scheme, when there *is* a watermark inserted, and *false positive bits* is the number of bits detected when the evaluated flow *is not* watermarked. As mentioned in section III, a threshold θ is used to decide whether a watermark is present. The *false positive* and *false negative* error rates are the probability of declaring a non-watermarked flow as containing a watermark and declaring a watermarked flow as not containing a watermark, respectively. The choice of the threshold θ represents a tradeoff between these two rates, as higher values of θ will have fewer false positives but more false negatives. We adjust θ to be such that the false positive and false negative error rates are the same; the resulting error rate is called the *crossover error rate* (COER).

We also add delay and jitter to botnet traffic, based on measurements we have performed on PlanetLab [10]. In Section V-B1 we will discuss how to synchronize our detector with the watermarker. However, non-uniform delay for different bots, as well as network jitter, will decrease the accuracy of our detection and thus we include it in our simulation.

Each experiment is run 100 times (each time with the same watermark parameters but different watermark bits and different bot traces) to get mean and variance of true positive and false positive parameters. Using these statistics we estimate the COER for each experiment. In our experiments, we set the number of (active) real bots to be $B = 100$, and vary the number of rogue bots, R. Figure 4(a) illustrates the estimated COER versus number of watermark bits, for different values of parameter T. The R/B value is fixed to 10%. In all of the simulations, the detection threshold η and the confidence threshold ψ are set to 1.

We also estimated the $COER$ for different values of R/B ratio. Figure 4(b) shows the $COER$ for different number of watermark bits having different ratios of R/B. As expected, increasing R/B improves the $COER$ at the expense of requiring more resources, e.g., real IP addresses.

Table I shows the results of the experiment for two different settings of the watermark parameters (each experiment is run 500 times). For the interval length of $T = 500msec$ and using 64 watermark bits and for a rogue to real ratio of $R/B = 10\%$, watermark can be inserted in a 64 second connection with the botmaster, and the resulting COER is on the order of 10^{-8}, which is very promising. Increasing the T parameter improves the COER, at the expense of needing more time for the botmaster to be online. The threshold that achieves the crossover error rate is about half of all the watermark bits.



(a) $R/B = 10\%$

(b) $T = 500msec$

Fig. 4. COER error of the detection scheme over sdbot botnet traces.

2) spybot trace: We performed similar experiments as above for the spybot traces from [19]. Similar to the sdbot trace, we extended the 5-bot botnet trace of spybot to a 100-bot trace. We then mixed the trace with the watermarked rogue flows and added appropriate network delays to the mixed flows. Then we evaluated the watermark detection similar to what we did for sdbot traces.

Figure 5(a) sketches the crossover error rate versus number of watermark bits, for different values of the interval length T. Again, R/B is fixed to 10%; i.e., $B = 100$ and $R = 10$, and each experiment is run 100 times with different randomly assigned watermark bits and randomly simulated real bot traffic. Once again, increasing T improves the COER at the expense of needing more time for the botmaster to be online.

We also evaluated the detection scheme for different ratios of R/B. Interval length T is fixed to $500msec$ and each experiment is run 100 times, each time with different watermark and randomly selected bot traffic. Results are shown in Figure 5(b).

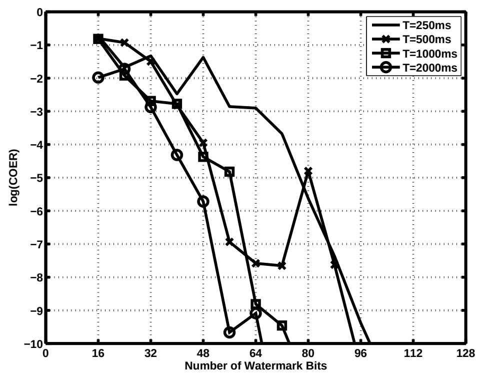
Table II shows the detection results for two sample sets of watermarking parameters. According to the first row of the table, the watermark can be detected in as few as 64 seconds with a COER of about 10^{-8}. Similar to sdbot simulations, we can trade elapsed time for COER, using different values of the watermarking parameters.

TABLE I
TWO SAMPLE RUNS OF THE DETECTION SCHEME OVER SDBOT TRACES, FOR $R/B = 10\%$ (AVERAGED OVER 100 RANDOM RUNS).

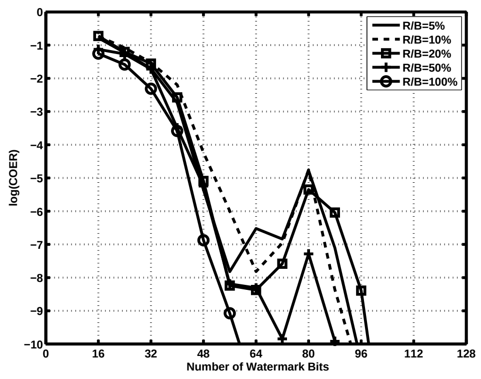
T	Watermark Bits	True Positive Bits	False Positive Bits	COER Hamming Threshold	COER	Elapsed Time(sec)
250	64	43.1900	22.3820	33	$2.8 * 10^{-3}$	32
2000	64	51.2880	12.8720	32	$3.524 * 10^{-13}$	256

TABLE II
TWO SAMPLE RUNS OF THE DETECTION SCHEME OVER SPYBOT TRACES, FOR $R/B = 10\%$ (AVERAGED OVER 100 RANDOM RUNS).

T	Watermark Bits	True Positive Bits	False Positive Bits	Optimum Hamming Threshold	COER	Elapsed Time(sec)
500	64	47.9960	16.1560	32	$2.3223 * 10^{-8}$	64
2000	64	50.3400	13.5100	32	$7.5534 * 10^{-11}$	256



(a) $R/B = 10\%$



(b) $T = 500msec$

Fig. 5. COER error of the detection scheme over spybot botnet traces.

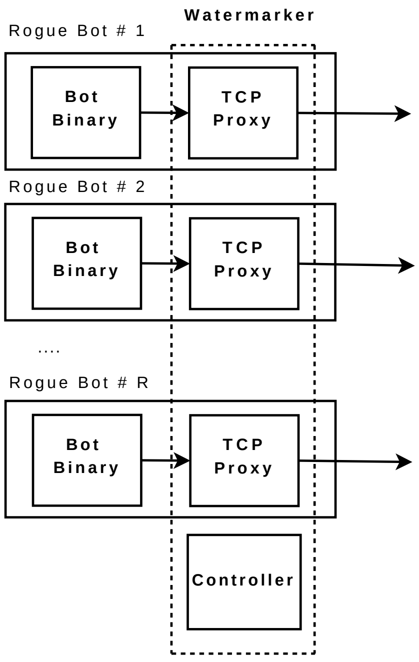


Fig. 6. Proxy-based implementation of the rogue bots.

B. Implementation

We tested BotMosaic on PlanetLab by creating synthetic bots that use an IRC channel to communicate with the botmaster. The rogue bots are implemented over physically separate hosts along the PlanetLab. Figure 6 shows the main structure of rogue bot implementation. Watermark proxies are installed over the rogue-bot hosts, and are controlled by a controller to insert the watermark. We route all the bot traffic (generally all the traffic for honeypot specific hosts) through the watermark proxy. Watermark proxies controlled by the controller are responsible for watermarking the bot traffic on all the rogue bot machines, so that the accumulated traffic makes the final watermarked traffic. By using a proxy, we avoid having to reverse engineer and modify the bot code to insert watermarks.

We implemented the BotMosaic watermarking scheme over PlanetLab infrastructure using randomly selected nodes as different entities in the experiment. Figure 7 shows the structure of our experiment. A botmaster is controlling botnet through the IRC C&C . To hinder detection, botmaster relays his traffic to the IRC server through 5 stepping stone nodes located in geographically different locations, and also encrypts the connections between stepping stones.

There are 100 real bots ($B = 100$) connected to the IRC C&C channel, listening for the commands from the botmaster, and sending appropriate responses to the channel. The real bots are chosen randomly, and are located in geographically different locations across the world. We set up $R = 10$ rogue bots to send watermarked flows to the C&C channel ($R/B = 10\%$). The rogue bots are also chosen randomly and

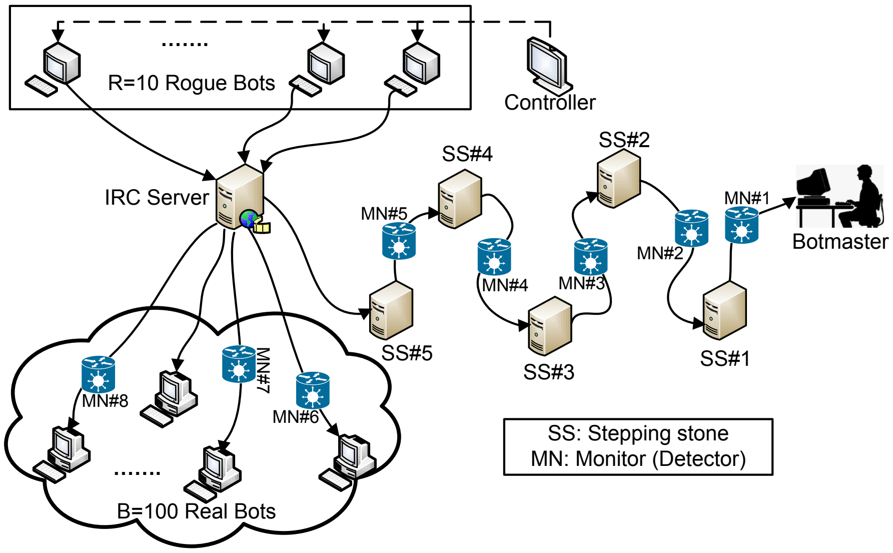


Fig. 7. Testbed structure of BotMosaic implementation over PlanetLab.

are located in different places. A controller node commands the rogue bots to join the C&C channel. Once all the rogue bots have joined the channel, controller commands all of them to start sending packets on the C&C channel containing corresponding shares of the watermark.

As for the simulations, due to space constraints we only provide the results for the watermark inserted on the botnet traffic to the botmaster, i.e., through PRIVMSG to the botmaster; we expect the detection performance to be the same for the watermark inserted into the traffic directed to the bots. In our experiments we used 10 distinct nodes to represent the rogue bots. In practice, it would be better to implement all of rogue bots on the same machine using virtualization mechanisms like the one used in the our implementation since we do not have access to that many free IP addresses.

We set up several monitors, i.e., watermark detectors, across the network to look up for the inserted watermark in network flows. The monitor deployment is described in Section IV. The monitors perform the watermark detection scheme as described in Section III. We set up 5 monitors on the way to the botmaster to check the true detection rate, and also 3 monitors on the paths not ending to the botmaster to evaluate the false detection rate.

Figure 8 shows detection results for different monitors. We set the T parameter of the watermarking system to be $T = 500msec$ and use watermarks with length l equal to 32, 64, and 128 bits. According to the simulations in the previous section, we set the hamming threshold to be half of the watermark bits in each case. Results are normalized by the total number of bits.

As results show, monitors on the path from IRC server to the botmaster ($MN\#1$ to $MN\#5$) are able to detect the watermark from the mixed-encrypted flows, as soon as only 32 seconds. On the other hand, monitors placed on the paths

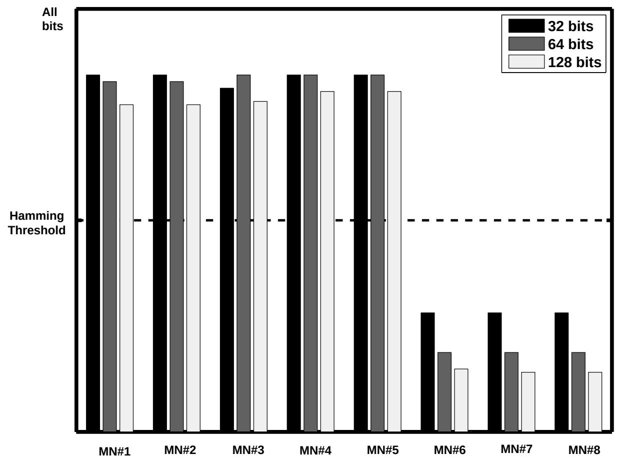


Fig. 8. Watermark bits detected by different monitors across the testbed (normalized by total number of bits).

not containing the botmaster watermark do not detect the watermark on the innocent flows.

Followed are other issues regarding BotMosaic detection scheme.

1) Synchronization: Watermark monitors need to synchronize the received watermarked flow with the watermark sequence, i.e., minimize the offset between intervals of watermarked flow and those of the watermark, in order to successfully detect the watermark. To find the right offset of the watermarked flow, we run the watermark detection scheme over the received flow applying different offset values from 0 to T in $T/100$ steps, and select the offset maximizing the number of detected bits as the right offset for that flow. Figure 9(a) illustrates the results of this sliding window approach for detection of watermark bits in a flow containing

TABLE III
RESOURCES REQUIRED FOR BOTMOSAIC.

Bits	Processing Time (μsec/flow)	Total Memory per flow (KB)	Total Memory (MB)
32	27.95	0.16	3.46
64	49.26	0.27	6.06
128	86.9	0.48	10.49

the watermark. Running the synchronization mechanism over an innocent flow, i.e., non-watermarked flow, the number of detected bits remains below the hamming threshold θ for different offset values, as shown in Figure 9(b). The use of this synchronization mechanism makes detection scheme tolerant to network delay (though variable network jitter still impacts the detection accuracy).

2) Resources: In order to study the processing and memory costs of BotMosaic detection scheme we implemented the watermark detector over a 21 GB network trace gathered from the routers of an anonymous US university. The utilized trace contains 21744 concurrent flows, with a total of 2.1 GB of timing information. Since the detection scheme should be implemented over border routers, this volume of traffic is representative of a highly loaded border router.

The experiment is done using a Unix system with 1.6 GHz of processing power and 2GB of memory. Table III illustrates the result of the experiment over the university traces. Even for 128 bits of watermark, which provides very small error rates as mentioned in previous sections, the total memory needed for watermark detection is about 11 MB, and the processing time for each flow is as small as 87 microseconds. The time and processing resources are even smaller for fewer watermark bits. Thus we expect that it may be possible to deploy BotMosaic even in high-performance routers used by large ISPs, to provide a better vantage point for botmaster traceback.

3) Other issues: Traditional flow watermarking schemes consider issues like chaff, repacketization, and packet addition/removal on the performance of watermark detection. Regarding BotMosaic, the mentioned perturbations are treated as the components of chaff, which is considered in the system design and evaluations.

Interval-based watermark schemes are subject to a multi-flow attack discovered by Kiyavash et al. [20]. To resist this attack, we can use the mechanisms proposed by Houmansadr et al. [21], for example, by changing the watermark key (the set of HI and LO intervals) over time.

Finally, it might be the case that the botmaster puts limitations on the number of packets each bot can send. In this case, watermarking will still be feasible by using more rogue bots to collaborate in the generation of BotMosaic watermarks.

VI. CONCLUSION

We have presented a new watermarking scheme, BotMosaic, to trace back botmasters who are controlling a botnet, which also helps in detecting infected machines and stepping stones. To do so, we present a different structure for the watermarking scheme, making it suitable for the botnet traceback problem, as existing watermarking schemes fail to do so. We showed through experiments that our watermark can be quite effective when 5%-10% of the bots are captured/imitated by a honeynet, and that our detection is simple enough to be able to handle large volumes of traffic. Any ISP/enterprise deploying the low-cost BotMosaic detectors over its border routers can take advantage of its benefits regardless of BotMosaic deployment in other parts of the Internet. We will research the use of similar watermarking techniques to trace back non-centralized botnets in the future.

REFERENCES

[1] J. Evers, "'bot herders' may have controlled 1.5 million pcs," *ZD-Net News*, October 2005, http://news.zdnet.com/2100-1009_22-145225.html.

[2] Y. Zhang and V. Paxson, "Detecting stepping stones," in *USENIX Security Symposium*, S. Bellovin and G. Rose, Eds. Berkeley, CA, USA: USENIX Association, Aug. 2000, pp. 171–184.

[3] X. Wang, D. Reeves, and S. F. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," in *European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, D. Gollmann, G. Karjoth, and M. Waidner, Eds., vol. 2502. Springer, Oct. 2002, pp. 244–263.

[4] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *ACM Conference on Computer and Communications Security*, V. Atluri, Ed. New York, NY, USA: ACM, 2003, pp. 20–29.

[5] Y. Pyun, Y. Park, X. Wang, D. S. Reeves, and P. Ning, "Tracing traffic through intermediate hosts that repacketize flows," in *IEEE Conference on Computer Communications (INFOCOM)*, G. Kesidis, E. Modiano, and R. Srikant, Eds., May 2007, pp. 634–642.

[6] T. He and L. Tong, "Detecting encrypted stepping-stone connections," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 1612–1623, May 2007.

[7] D. Donoho, A. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," in *International Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, A. Wespi, G. Vigna, and L. Deri, Eds., vol. 2516. Springer, Oct. 2002, pp. 16–18.

[8] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows," in *ndss*, Feb. 2009.

[9] A. Blum, D. X. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," in *International Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, E. Jonsson, A. Valdes, and M. Almgren, Eds., vol. 3224. Springer, Sep. 2004, pp. 258–277.

[10] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating systems support for planetary-scale network services," p. 253266.

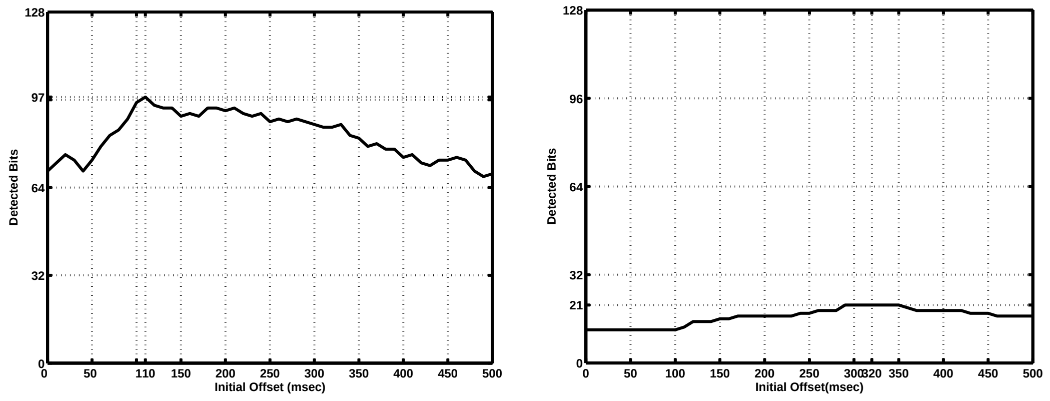
[11] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer VoIP calls on the Internet," in *ACM Conference on Computer and Communications Security*, C. Meadows, Ed. New York, NY, USA: ACM, Nov. 2005, pp. 81–91.

[12] ——, "Network flow watermarking attack on low-latency anonymous communication systems," in *IEEE Symposium on Security and Privacy*, B. Pfitzmann and P. McDaniel, Eds., May 2007, pp. 116–130.

[13] W. Yu, X. Fu, S. Graham, D.Xuan, and W. Zhao, "DSSS-based flow marking technique for invisible traceback," in *IEEE Symposium on Security and Privacy*, B. Pfitzmann and P. McDaniel, Eds., May 2007, pp. 18–32.

[14] D. Ramsbrock, X. Wang, and X. Jiang, "A first step towards live botmaster traceback," in *RAID*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds., vol. 5230. Springer, 2008, pp. 59–77.

[15] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, fidelity, and containment in the potemkin virtual honeyfarm," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 148–162, 2005.



(a) Watermarked flow (b) Non-watermarked flow

Fig. 9. Synchronization of the watermarking scheme. Maximum values are returned as the detection outcome.

[16] L. Spitzner, "The Honeynet Project: trapping the hackers," *Security & Privacy Magazine, IEEE*, vol. 1, no. 2, pp. 15–23, 2003.

[17] N. Provos, "A virtual honeypot framework," in *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2004.

[18] C. Kalt, "Internet Relay Chat: Server Protocol," RFC 2813 (Informational), Apr. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2813.txt

[19] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[20] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes," in *USENIX Security Symposium*, P. van Oorschot, Ed. Berkeley, CA, USA: USENIX Association, 2008.

[21] A. Houmansadr, N. Kiyavash, and N. Borisov, "Multi-flow attack resistant watermarks for network flows," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 1497 –1500.

[22] B. Pfitzmann and P. McDaniel, Eds., *IEEE Symposium on Security and Privacy*, May 2007.