Insights from the Inside: A View of Botnet Management from Infiltration

Chia Yuan Cho[§], Juan Caballero[†§], Chris Grier[§], Vern Paxson[‡§], and Dawn Song[§]
[§]UC Berkeley [†]Carnegie Mellon University [‡]ICSI

Abstract

Recent work has leveraged *botnet infiltration* techniques to track the activities of bots over time, particularly with regard to spam campaigns. Building on our previous success in reverse-engineering C&C protocols, we have conducted a 4-month infiltration of the *MegaD* botnet, beginning in October 2009. Our infiltration provides us with constant feeds on MegaD's complex and evolving C&C architecture as well as its spam operations, and provides an opportunity to analyze the botmasters' operations. In particular, we collect significant evidence on the MegaD infrastructure being managed by multiple botmasters. Further, FireEye's attempt to shutdown MegaD on Nov. 6, 2009, which occurred during our infiltration, allows us to gain an inside view on the takedown and how MegaD not only survived it but bounced back with significantly greater vigor.

In addition, we present new techniques for mining information about botnet C&C architecture: "Google hacking" to dig out MegaD C&C servers and "milking" C&C servers to extract not only the spectrum of commands sent to bots but the C&C's overall structure. The resulting overall picture then gives us insight into MegaD's management structure, its complex and evolving C&C architecture, and its ability to withstand takedown.

1 Introduction

Researchers have recently gained new, detailed insights into the operation of botnets via *infiltration*: running either live bots in controlled environments [9,14], or custom programs that mimic bot command-and-control (C&C) activity [8,13,16]. Such work has primarily aimed at monitoring the instructions issued to bots in order to investigate how botmasters employ their botnets and to assess botnet population dynamics. Less studied has been the issue of *botnet management*: the dynamics of how botmasters *change* their botnets, either in terms of altering elements of a current "campaign" or reconstructing the botnet itself in the face of significant disruption, and the question of whether a botnet infrastructure may have multiple managers (botmasters).

In this work we undertake such an analysis based on a 4-month infiltration of the *MegaD* botnet begun in Oct. 2009. While much of our measurement drew upon our earlier work in reverse-engineering MegaD's C&C protocol [11] and the cryptographic routines that obfuscate it [12], we also developed additional methods for gathering information about the botnet. We discovered that we could use "Google hacking" to locate additional C&C servers based on fingerprinting the web pages they supply when non-bots visit them. Once found, we can build *milkers* that probe the different C&C components to extract not only the spectrum of commands sent to bots, but also the C&C's overall structure. We call them *milkers* because they "milk" MegaD C&C components for extensive information regarding MegaD's operations, providing an opportunity to analyze the botmasters' ongoing activities.

In particular, by monitoring MegaD's complex and evolving C&C architecture and its spam operations, we collect significant evidence of the MegaD infrastructure being managed by multiple botmasters. MegaD's most recent C&C architecture comprises multiple, disjoint groups of C&C components with very few elements shared across groups. Each group exhibits different server replacement patterns and notable differences in spam operations. Further, our monitoring period spans an attempted *takedown* of MegaD that resulted in decapitation of significant C&C components, providing us with an inside view on the takedown and how MegaD not only survived it but bounced back with significantly greater vigor.

We begin in § 2 with an overview of MegaD and a description of our infiltration methods in § 3. In § 4 we analyze the architecture and evolution of the botnet's C&C infrastructure, and in § 5 we look at how the botmaster varies the makeup of the templates used to control MegaD's spam operations. We conclude in § 6.

2 MegaD Overview

MegaD is a mass spamming botnet first observed in 2007, credited at its peak with responsibility for sending a third of the world's spam [3]. MegaD is striking for its resilience, having survived two major attempts at disruption: the McColo shutdown of Dec. 2008 [4], and a *take-*

down effort coordinated by FireEye in Nov. 2009 [2], during the period of our study. In both cases MegaD initially ground to a halt [5], but then quickly bounced back with greater vigor. In the week prior to FireEye's Nov. 6 takedown effort, MegaD contributed about 4% of spam, and the botnet initially showed little signs of life after the attack. However, its activity soon increased, exceeding pre-takedown levels by Nov. 22, and constituting 17% of worldwide spam by Dec. 13 [6].

MegaD C&C Servers. A MegaD bot interacts during its lifetime with four types of C&C servers: *Master Servers (MS)*, *Drop Servers (DS)*, *Template Servers (TS)*, and *SMTP Servers (SS)*.

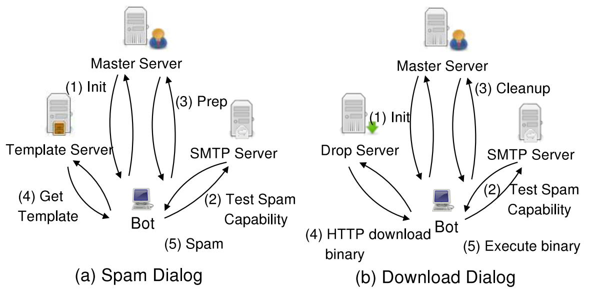
The botmaster uses the master servers to distribute commands to the bots. Bots locate a master server using a rendezvous algorithm, based on domain names hardcoded in the bot binaries. Upon locating a master server, a bot employs pull-based communication using MegaD's custom C&C protocol. The bot periodically probes the master via request messages to which the server replies with both authentication information and a general command. The bot performs the requested action and returns the results to the master server.

Drop servers distribute new binaries. A bot locates a drop server by receiving a special command from its master server containing a URL specifying a file to download through regular HTTP.

Template servers distribute the spam templates that bots use to construct spam. A bot locates a template server via a directive from the master server specifying the address and port to contact. Again, communication proceeds in a pull-based fashion using MegaD's custom C&C protocol.

SMTP servers play two distinct roles. First, bots check their spam-sending capabilities by sending a test email to them. A bot locates the server for this testing via a command from the master server specifying the server's hostname. Second, bots notify an SMTP server after downloading a new spam template and prior to commencing to spam. A bot locates the SMTP server used for template download notification via a PTR_RSL control parameter in the spam template, which specifies the SMTP server's hostname. The notification protocol is non-standard SMTP. Instead of sending the usual SMTP "HELO <hostname>" message, the bot sends a special "HELO 1" message and closes the connection.

MegaD dialogs. Throughout our monitoring, we have observed only two different sequences of commands (i.e., *dialogs*) issued by master servers, as depicted in Figure 1. In the *spam* dialog, bots are first ordered to test their ability to send spam using a given SMTP server. If the test succeeds, the master server engages the bot in an elaborate preparation phase to obtain information about the infected host, followed by sending a *GetTemplate* command that reveals the identity of a template server from which the bot fetches a spam template. The bot acknowledges reception to both the template server and the SMTP server, after which it starts sending spam. When it finishes, it re-initiates the spam dialog with the master server.



(a) Spam Dialog
(b) Download Dialog

Figure 1: MegaD Dialogs: (a) Spam dialog and (b) Binary download dialog

The *download* dialog also starts with the master server ordering a test of the bot's ability to send spam. However, rather than then proceeding to the preparation phase, the master server orders the bot to download a new binary from a drop server and execute it.

We empirically determined that master servers specialize in supervising either spam or download dialogs, but not both. However, both operations are conducted using the same C&C protocol.

3 Infiltrating MegaD

We aim for our MegaD infiltration to extract information about both the *complete C&C architecture* as well as the *malicious activities* of the botnet as these evolve through time. In this section we present the main two techniques we use for our infiltration: 1) creating *milkers*, i.e., programs that mimic a bot's network interactions but without the malicious side effects (e.g., without sending spam) and Google hacking techniques to discover C&C servers. In addition, we also run MegaD binaries in a controlled environment to monitor their externally visible activity.

To create the milkers we leverage our previous results in extracting MegaD's protocol grammar [11] and the encryption/decryption functions MegaD uses to obfuscate its C&C protocol [12]. Using milkers has several advantages over running bots in a controlled environment. First, the milkers are more lightweight and do not require containment. More generally, they allow us to readily modify the bot interaction behavior to more aggressively probe the C&C servers at higher rates; impersonate multiple bots by changing bot communication identifiers; and directly interact with template, download or SMTP servers without having first received a corresponding command from a master server. This latter allows us to probe servers in situations for which a live binary would not. In particular, this capability enabled us to continue probing the C&C architecture during FireEye's disruption of MegaD, as detailed in § 4.

3.1 Monitoring MegaD's Spam Operations

A MegaD bot only carries out the spam operations. It is the botmaster that decides all details by creating a spam template that describes the structure of the spam message, the data to use for the different fields, and the parameters that control the bot's spam engine. Thus, templates fully describe the botnet's spam operations.

To obtain information about MegaD's spam operations over time we constructed a *template milker*, i.e., a program that periodically queries a template server for templates. We run one milker per known server. The milkers directly and continually extract templates, giving us over time access to the full set of templates employed by the botnet. Such a technique was previously employed for the Storm botnet by the authors of *Stormdrain* [10].

Our template milker probes a template server at a configurable average rate, with some added jitter to help mask its artificial nature. A single milker can impersonate multiple bots by changing the bot identifiers in the payload of the messages it sends. In addition we operate our milkers through Tor [7], which provides some IP address diversity. Thus, if the template server distributes templates based on the bot's identifier or its IP addresses, we can still collect a comprehensive set of templates.

Ignoring the master server. When communicating with a template server, bots include a 16-byte bot identifier issued from the master server. Template servers *could* validate the identifier as indeed having been previously issued by a master server, but they do not do so: templates are still served on any 16-byte identifier, indicating that either master servers do not communicate the bot identifiers to template servers or template servers do not check them. Thus, our milkers can communicate directly with a given template server, completely bypassing the master server.

Rate-limiting bypass. We discovered that template servers rate limit the communication of templates to bots based on the 16-byte bot identifier, sending a given bot no more than 16 templates per half-hour period. Thus, by selecting random identifiers, our milker can bypass this limit and harvest templates at an arbitrary rate.

3.2 MegaD's Complete C&C Architecture

A MegaD bot obtains only a partial view of the whole C&C architecture. The bot communicates with a single master server that it discovers through its rendezvous algorithm and thus only interacts with other C&C servers as directed by that master server. To understand the complete architecture we need to go beyond the perspective as seen by a single bot. For example, probing a master server can lead us to previously unknown template and drop servers. In addition, the dropped binary from a new drop server can in turn lead us to additional master servers, from which we can repeat the process.

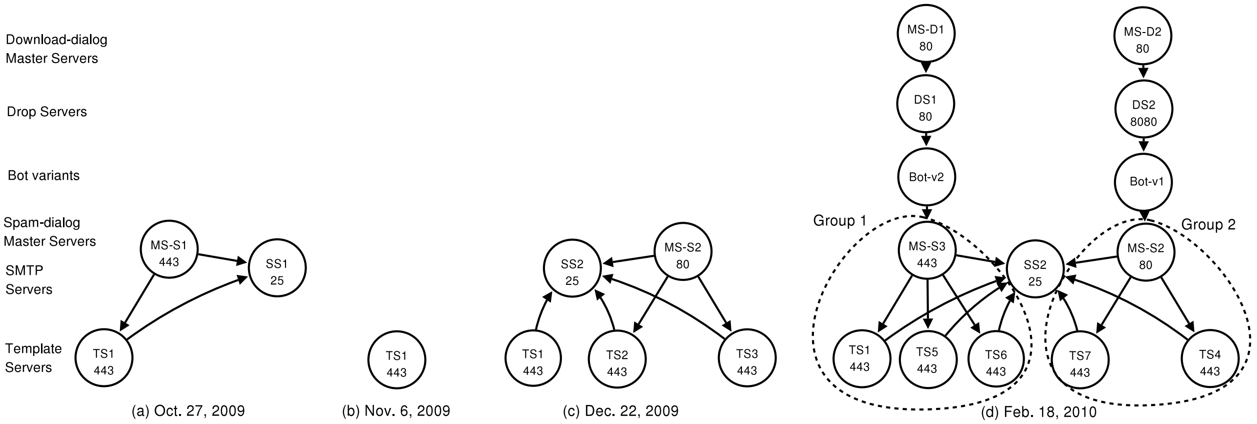
To this end, we construct a C&C milker that periodically queries a master server for commands. Every time the C&C milker receives the identity of a new template server, we start running a new template milker for that template server. Similar to template milkers, because our C&C milkers are not actual MegaD bots, we can increase their probing rate and impersonate multiple bots by changing IP addresses and bot identifiers.

Google hacking for locating master servers. One way to locate master servers is to analyze the connections of live MegaD binaries. We can find such binaries by rummaging through online repositories or requesting specimens from other researchers. However, this proves to be a painful process, requiring many hours to locate and then set up the binaries to run properly in the contained environment.

To locate more master servers, we devise a trick that leverages the ubiquity of search engines locating web servers around the Internet. Each MegaD master server runs the (TCP) C&C protocol on either port 80 or 443. Master servers running on port 80 expect to receive queries using regular HTTP, and if so camouflage themselves as follows to appear as normal web servers: for a request beginning with "GET", rather than replying with a response from the encrypted C&C protocol, they fake an HTTP "success" response with a small, crafted HTML content. The content renders in a browser as an innocuous-looking "Microsoft test page".

MegaD master servers do not appear to check whether the request comes from a search engine. Thus, their camouflage content gets added to the search engine's database. However, we discovered that the camouflage content contains sufficiently distinguishable elements to enable *fingerprinting* it as distinct from other Internet HTTP servers. We therefore can construct a Google query that returns only pages provided by MegaD's master servers that run on port 80, a technique commonly known as "Google hacking". We can in addition verify which of the returned results correspond to MegaD servers by attempting to contact the server using the MegaD C&C protocol rather than HTTP.

Our query returns exactly 4 results on 4 unique hostnames with no false positives. Two of those servers are already known to us, and another we verify by connecting to it using our C&C milker. To date the last one has been unreachable, but independent reports confirm that it was



(a) Oct. 27, 2009 (b) Nov. 6, 2009 (c) Dec. 22, 2009 (d) Feb. 18, 2010

Figure 2: Our evolving view of MegaD's C&C architecture over the infiltration. Each node represents an element of MegaD's infrastructure and C&C servers are labeled with their role and port numbers. Directed arcs represent the points-to relationships between members. The final achitecture reconstructed by the infiltration is shown in (d), where we note the two distinct C&C server groups, and the SMTP server that provides a central point for monitoring.

indeed a MegaD C&C server in the past [3].

4 Discovery of C&C Architecture

We now turn to our findings on MegaD's C&C architecture and how our 4-month infiltration using milkers and Google hacking techniques throws light on MegaD's complex and evolving C&C architecture. Our goal to obtain the overall picture of the C&C architecture is complicated by two problems. First, we start with a partial view of the C&C architecture and we need to evolve our view over time by discovering new C&C elements using our C&C milkers and Google hacking techniques. Second, our infiltration coincides with FireEye's attempt to shut down MegaD, which removed part of MegaD's C&C architecture. Thus, both the C&C architecture and our view of it evolves over time.

4.1 Takedown and Reconstruction

In this section we first present our insights from inside FireEye's takedown and then show how our C&C milkers and Google hacking techniques enabled us to evolve our view of MegaD's C&C architecture. Figure 2 shows our evolving view of MegaD's C&C architecture over the infiltration period. Figure 3 shows the chronology and significant events of our MegaD infiltration.

Our infiltration begins on Oct. 27, 2009 with the knowledge of 3 C&C servers: one master server (MS-S1), one template server (TS1), and one SMTP server (SS1), which we had identified a priori by running a MegaD bot in our contained environment. Our view of MegaD's C&C architecture on that day is shown in Figure 2(a) and we start our infiltration by running a template milker on TS1.

An inside view of FireEye's takedown. On Nov. 6, 2009, FireEye launched a coordinated effort to take down MegaD. The takedown was widely lauded as successful since MegaD's spam trickled to a halt. However, 16 days later its share of the world's spam exceeded its 4% pre-takedown level and by Dec. 13 it had climbed to 17% [6].

The takedown included both MS-S1 and SS1, thus our captive MegaD binary stopped working that day. However, the takedown did not affect TS1, which surprisingly continued handing out spam templates to our milker like nothing had happened. Our template feeds reveal that the templates distributed by TS1 remained unchanged for one week after the takedown. The first sign of a recovery was on Nov. 13, when the PTR_RSL control parameter in the template was updated to point to a new SMTP server SS2. That update also allowed us to discover a new master server (MS-S2) on the same domain.

From our privileged vantage point we can establish two key findings about the botmaster's takedown response. First, the botmaster did not have backup domains and ISPs ready when the takedown happened, since for a whole week the templates in TS1 pointed to a dead domain. Second, it took the botmaster a week to find a new ISP to host their infrastructure and set up the new C&C servers.

Interestingly, the update of the PTR_RSL control parameter in the spam templates on Nov. 13 was not used for recovery. We tested this by replicating the takedown and template changes on our captive bot, and found that it did not recover from the takedown. Similarly, FireEye's bots did not recover after the takedown [15]. Thus, we believe that MegaD did not bounce back using resilience mechanisms in the bot binary but by pushing out fresh bi-

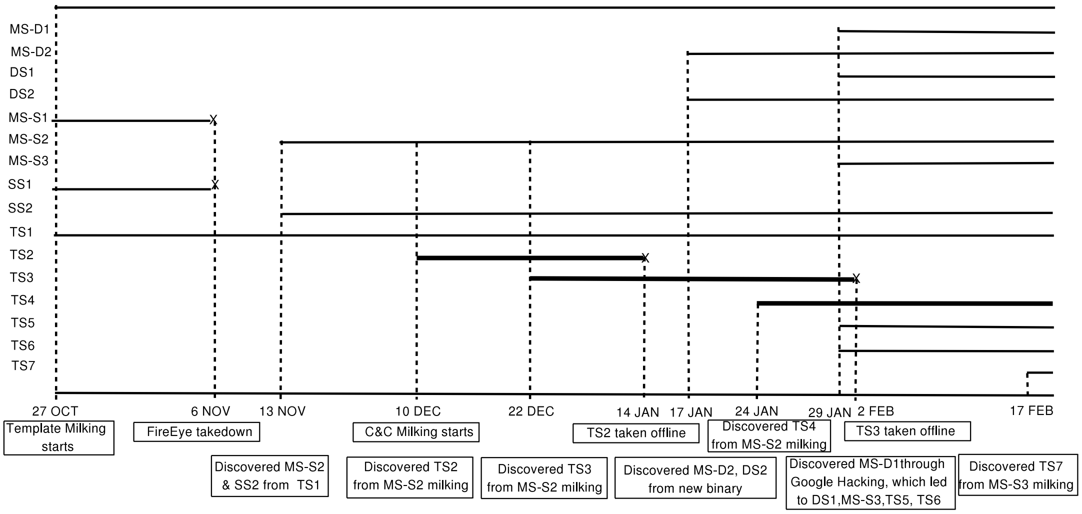


Figure 3: Timeline of significant infiltration events. The bold lines highlight a subtle pattern where template servers emerge and go offline with overlapping periods of availability to support spam operations.

naries.

MegaD is known to participate in a Pay-Per-Installation (PPI) service in which a downloader like *Piptea* drops other malware (e.g., a MegaD bot) on compromised hosts for a fee [1]. We believe that the MegaD botmasters paid the PPI operators to push out fresh bot binaries. What remains unanswered is how much overlap there is between the pre-takedown and post-takedown MegaD population and to what extent the downloader was the resilience mechanism used for bouncing back.

Evolving our view of the C&C architecture. Due to the takedown, our view of MegaD's C&C was reduced to a single node on Nov. 6 as shown in Figure 2(b). From Dec. 10, we began the process of evolving our view of the C&C architecture by starting a C&C milker for MS-S2, and added our Google hacking technique on Jan. 29. We leave the interested reader to examine the timeline of events in Figure 3 for details. Using our techniques we discovered 6 additional template servers (TS2 to TS7), 2 download dialog master servers (MS-D1 and MS-D2), an additional spam dialog master server (MS-S3), 2 drop servers (DS1 and DS2), and different binary variants distributed by each drop server. The C&C discovery process culminated in a rich architectural view of MegaD by Feb. 18 as shown in Figure 2(d).

4.2 C&C Groups

Figure 2(d) shows the final MegaD C&C architecture that we have reconstructed through our 4-month infiltration using our milkers and Google hacking techniques. There exists two master servers in charge of spam operations (MS-S2 and MS-S3) and two master servers in charge of drop/update operations (MS-D1 and MS-D2). Each master server that specializes in spam operations is supported by a set of template servers. The fact that the sets of template servers are *disjoint* signals the presence of two separate groups in the architecture: Group 1 with MS-S3 supported by a backend of TS1, TS5, and TS6, and Group 2 with MS-S2 supported by TS4 and TS7 (and previously TS2 and TS3). Each group is supported by one master server specialized in update operations as well as one drop server. The binaries dropped by a drop server connect only to the spam master server that the drop server supports. Both groups share a single SMTP server (SS2), which all bots use to test their spam-sending capabilities and to report template downloads. The fact that all bots are aware of SS2 makes it well suited as a central point for monitoring the spam operations across both groups.

Completeness of discovered architecture. The caveat is the completeness of our final view on MegaD's architecture. However, we can compare our final architectural view with those from other researchers, the most complete of which is FireEye's list of 15 active MegaD C&C IPs prior to the takedown compiled by monitoring near to a hundred MegaD binaries [2, 15]. We note that the list is not ideal as it does not differentiate between server types. Further, FireEye's snapshot was made *before* the takedown, and is itself incomplete, e.g., it does not include the surviving template server TS1. Nevertheless, if we assume FireEye's set of 15 C&C servers plus the missing TS1 as MegaD's full architecture at any time, then the set of 12 active servers in our final view constitutes 75% of the overall architecture.

Server	Days	# Tmpl.	# Addr. (% uniq)	Addr./ tmpl.	# Uniq subj.	# Uniq URLs
TS1	115	141K	283M (75%)	2,000	156	252
TS2	36	28K	27M (70%)	969	107	9
TS3	42	47K	68M (84%)	1,465	96	41
TS4	26	19K	32M (93%)	1,710	103	38
TS5	21	22K	44M (97%)	2,000	30	43
TS6	15	14K	28M (96%)	2,000	15	25
TS7	2	713	1.3M (97%)	1,859	17	4
Total	257	271K	483M (70%)	1,782	355	330

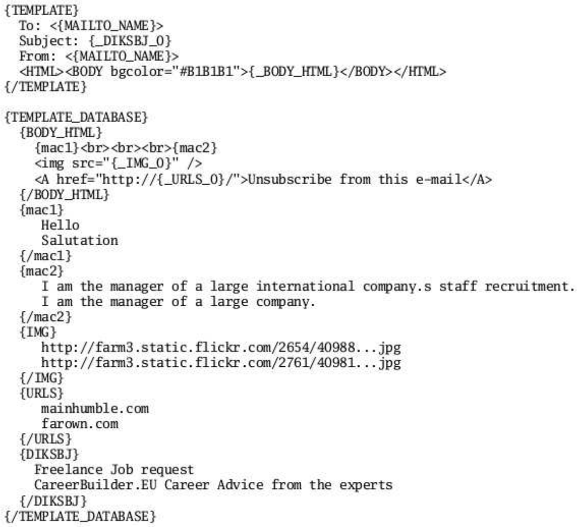
Table 1: Aggregate Statistics of Template Dataset

Differences in group management. One pattern that emerged during our infiltration is that for Group 2, a new template server would be added to the backend of MS-S2 and a few days later the older template server would be taken offline. For example, TS3 was added on Dec. 22 and on Jan. 14 TS2 went offline. This trend is highlighted with bold lines in Figure 3. This pattern looks like a server replacement that guarantees availability of the spam operations during the replacement. Although we do not know the reasons behind such replacements, this pattern does not appear in Group 1, which indicates that both groups are managed differently.

Although the fact that there is a central SMTP server to which all bots connect points to the infrastructure being shared or centrally managed, the question that arises is whether those differences in group management are due to multiple botmasters sharing the same botnet infrastructure, or to other reasons such as Group 2 rebuilding its infrastructure having incurred greater damage from the takedown. We address this question in Section 5 by analyzing the differences in spam operations between the 2 groups. Our results show significant evidence on the spam operations of each group being under separate management.

5 Template Milking

In this section we report our analysis of the templates obtained by the operation of our template milkers from Oct. 27, 2009 through Feb. 18, 2010. During this period we collected 271 K spam templates containing 483 M email addresses. Table 1 shows aggregate statistics for the monitored servers. We began our milking operations with TS1, adding additional milkers every time we discovered a new template server, for a total of seven, two of which are no longer active.



```
{TEMPLATE}
  To: <{MAILTO_NAME}>
  Subject: {_DIKSBJ_0}
  From: <{MAILTO_NAME}>
  <HTML><BODY bgcolor="#B1B1B1">{_BODY_HTML}</BODY></HTML>
{/TEMPLATE}

{TEMPLATE_DATABASE}
  {BODY_HTML}
    {mac1}<br><br><br>{mac2}
    <img src="{_IMG_0}" />
    <A href="http://{_URLS_0}/">Unsubscribe from this e-mail</A>
  {/BODY_HTML}
  {mac1}
     Hello
     Salutation
  {/mac1}
  {mac2}
     I am the manager of a large international company.s staff recruitment.
     I am the manager of a large company.
  {/mac2}
  {IMG}
     http://farm3.static.flickr.com/2654/40988...jpg
     http://farm3.static.flickr.com/2761/40981...jpg
  {/IMG}
  {URLS}
     mainhumble.com
     farown.com
  {/URLS}
  {DIKSBJ}
    Freelance Job request
    CareerBuilder.EU Career Advice from the experts
  {/DIKSBJ}
{/TEMPLATE_DATABASE}
```

Figure 4: MegaD spam template sample: structure, polymorphic elements, and how polymorphic elements are combined to form an email body

Template structure. The MegaD spam template is a structured document comprising two sections: the template and the element database. Figure 4 shows a simplified spam template where the template section is delimited by the {TEMPLATE} and {/TEMPLATE} tags and the element database by the {TEMPLATE_DATABASE} and {/TEMPLATE_DATABASE} tags. The template section captures the structure of the email headers and the body of the message. It contains static text as well as element tags, which the bot replaces with values from the element database. The element database contains named *data elements* and a single *control element*. The data elements contain the values that the bot replaces into the tags in the template section, while the control element specifies the behavior of the bot's spam engine using 29 configurable parameter-value pairs.

Each data element has a set of values in the template. We call elements with more than one value in their set *polymorphic*. Polymorphic elements are fundamental for the success of the spam operation because the body of any two spam messages generated from the same template will differ as different values are selected for each tag. Such polymorphism is designed to evade spam filters that look at the body of the email messages.

Changes in template structure. The template structure is the ordered list of data elements in a spam template. Throughout the infiltration we observe how the template structure changes as the botmaster introduces and experiments with new types of elements.

Most notably, the templates from TS1 before the FireEye takedown used sets of static strings for subjects, as well as a mostly static HTML body for the spam message where the only polymorphic data element was the URL to spamadvertise. However, after the takedown the botmaster started experimenting with the template structure. First, it combined multiple data elements to dynamically construct a coherent subject and HTML body,

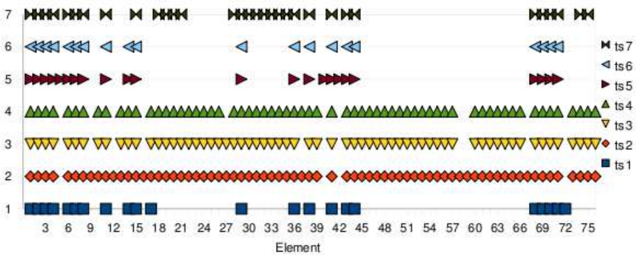


Figure 5: Template structure commonalities across servers.

which makes it more difficult to build spam filters based on these elements. An example of this highly dynamic HTML body is shown in Figure 4 using the {mac1} and {mac2} tags. In addition, it experimented with using image spam, by inserting {IMG} tags in the body of the message. The {IMG} tags point to images hosted in Flickr. These experiments were carried out using TS1 and TS2. They started on Nov. 18 and lasted less than two weeks. After this experimentation phase, the botmasters finally deployed some of those changes.

Evidence of separate management. Interestingly, the final deployment falls under the two different management groups that we introduced in § 4. One month after the experimentation phase, the {IMG} became a permanent data element in Group 1, which comprises TS1, TS5, and TS6. Similarly, the {mac*} and {sb*} elements were deployed in Group 2, which comprised TS2 and TS3 at that time and were later replaced by TS4 and TS7. Thus, only Group 1 does image spam and although both groups polymorph the subject by using sets of values, only Group 2 has the extra polymorphism provided by constructing the subject and HTML body using {mac*} and {sb*} tags.

Figure 5 summarizes the commonalities in template structure by plotting occurrences of unique data elements across all template servers, starting on Dec. 9, once the experimentation phase on TS1 and TS2 was over. It shows that {TS1, TS5, TS6} (Group 1) and {TS2, TS3, TS4} (Group 2) have similar template structures. We only have two days of template data for TS7, which explains why some unique elements from the group have not appeared.

Changes in polymorphic data elements. Here we analyze how often the botmaster changes the set of values associated with a polymorphic data element. There are three cases. A *single-set-polymorphic* data element uses a set of values that never changes, that is, all templates have the same set of values over time. For example, MegaD's spam messages fake Outlook Express (OE) email signatures using an element that captures the OE version. The set of OE version values stays constant across templates over time.

A *multi-set-polymorphic* data element uses the same set of values across all templates at any point of time. The set of values are updated by the botmaster and stay constant across all templates until the next update. Multi-set-polymorphic elements are updated by the botmaster to evade spam filters. For example, the {URLS} element, representing the spamvertised sites, is multi-set-polymorphic because the botmaster refreshes the set of URL values at a low average rate (once every 2 days) in reponse to URL blacklisting and new spam campaigns being launched. Other multi-set-polymorphic elements in Figure 4 are the subject ({DIKSBJ}), the HTML body ({BODY_HTML}), and the tags used to form the HTML body ({mac1}, {mac2}).

A *every-set-polymorphic* data element uses a dynamically generated set of values on each template sent by the template server. The set of values is chosen as a subset of a larger value set only known to the template server. For example, the {DOMAINS} element, which contains the set of target email addresses to spam, is an every-set-polymorphic element because the template server selects a small subset of the complete email list for each requested template. Other every-set-polymorphic element are {LINK} and {IMG}.

Every-set-polymorphic elements are most convenient for the botmaster because they provide automated polymorphism without requiring manual intervention. One would expect skilled botmasters to convert as many multi-set-polymorphic elements as possible into every-set-polymorphic elements to improve the efficiency of their spam operations.

Figure 6 illustrates *multi-set-polymorphic* and *every-set-polymorphic* elements in the TS1 templates and how they are updated over time. We see differences in update rates between every-set-polymorphic elements such as {IMG} and {LINK}, whose value sets are dynamically generated for each template, compared to manually updated *multi-set-polymorphic* elements such as {DIKSBJ}, {BODY_HTML}, and {URLS}.

Evidence of separate management. We find the update rates for multi-set-polymorphic elements particularly interesting because they require sustained effort from the botmaster on continual updates, which provides important information about how the botmaster manages the template servers. We examine the update rates across template servers, focusing on the subject ({DIKSBJ}) due to its use across all template servers. The results are shown in Table 2. We observe the template servers in Group 1 having similar update rates, i.e., approximately every 3 days, and the template servers in Group 2 significantly slower update rates. This provides more evidence on the fact that both C&C server groups are managed separately.

Implications of separate management. The clear distinction in update rates between the two C&C server

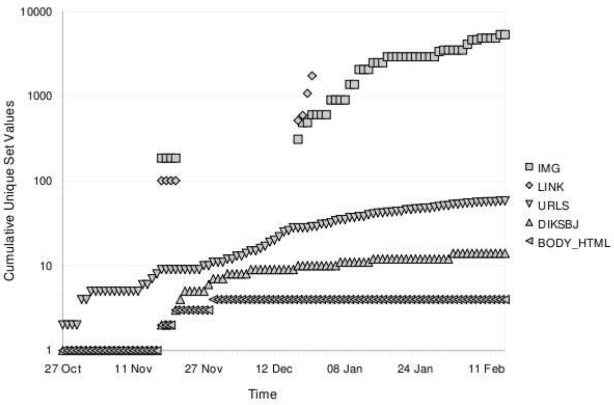


Figure 6: Changes in multi-set and every-set polymorphic data elements in TS1. The vertical axis is the cumulative unique set identifier for each element.

Group	1	1	1	2	2	2	2
Server	TS1	TS5	TS6	TS2	TS3	TS4	TS7
Ave. # Days	6.8	5.3	15.0	2.0	2.8	1.7	2.0

Table 2: Average number of days between updates of subject elements ({DIKSBJ}) across template servers, indicating significant faster updates in template servers from Group 2.

groups aligns with the evidence we previously collected from the C&C architecture and the changes to template structure. A final piece of evidence comes from the differences between the groups in terms of spam campaigns. We find that Group 1 focuses exclusively on Viagra campaigns, while Group 2 runs multiple and diverse campaigns, including Viagra, job scams and money mule recruitment. One might attribute the differences in update rates and spam campaigns between the two groups to profitability, i.e. the spam campaigns in Group 2 may simply be more profitable to justify the more frequent updates. However, this does not explain the need for greater architectural changes as seen in § 4.2. We therefore arrive at the conclusion that the day-to-day activities of MegaD falls under two separate management groups (botmasters). In addition, the subject update rates, as well as the server replacement pattern shown in Section 4 also indicate that the managers of Group 2 are significantly more dynamic than the managers of Group 1 in its operations.

6 Conclusion

We have presented a 4-month longitudinal assessment of the control architecture and management of the MegaD botnet by employing our C&C milking, template milking and Google hacking techniques. Our infiltration has culminated in a rich architectural view of MegaD's C&C and provided significant evidence on the MegaD infrastructure being managed by multiple botmasters. Our inside view of the attempted takedown offers new insights on how botnets actually recover from takedowns: instead of relying on reslience mechanisms to recover bots, the botmasters simply push out new binaries.

7 Acknowledgements

We would like to thank FireEye and in particular Atif Mushtaq for help in understanding the events around the MegaD takedown. We also thank Matt Williamson and the anonymous reviewers for their insightful comments.

This material is based upon work partially supported by the NSF under Grants 0311808, 0448452, 0627511, 0433702, CNS-0905631, CNS-0831535, and CCF-0424422, by the USAF Office of Scientific Research under grant 22178970-4170, by ARO under grant DAAD19-02-1-0389, and by ONR under MURI Grant N000140911081. Opinions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Botnetweb. `http://blog.fireeye.com/research/2009/04/botnetweb.htm`.

[2] Killing the beast...part 4 (ozdok). `http://blog.fireeye.com/research/2009/11/killing-the-beastpart-4.html`.

[3] Mega-d. `http://www.m86security.com/trace/i/Mega-D,spambot.896.asp`.

[4] Mega-d botnet returns after mccolo shutdown. `http://www.darkreading.com/security/attacks/showArticle.jhtml?articleID=212300170`.

[5] Mega-d botnet takes a hit. `http://www.m86security.com/labs/traceitem.asp?article=1161`.

[6] Tracking spam botnets. `http://www.m86security.com/labs/bot_statistics.asp`.

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proc. USENIX Security*, 2004.

[8] Chris Kanich et al. The Heisenbot uncertainty problem: Challenges in separating bots from chaff. In *Proc. LEET*, 2008.

[9] Chris Kanich et al. Spamalytics: An empirical analysis of spam marketing conversion. In *ACM CCS*, October 2008.

[10] Christian Kreibich et al. Spamcraft: An inside look at spam campaign orchestration. In *Proc. LEET*, 2009.

[11] Juan Caballero et al. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *ACM CCS*, 2009.

[12] Juan Caballero et al. Binary code extraction and interface identification for security applications. In *Proc. NDSS*, February 2010.

[13] Thorsten Holz et al. Measurements and mitigation of peer-to-peer-based botnets: A case study on Storm worm. In *Proc. LEET*, 2008.

[14] John P. John, Alexander Moshchuk, Steven D. Gribble, and Arvind Krishnamurthy. Studying spamming botnets using Botlab. In *Proc. USENIX NSDI*, 2009.

[15] Atif Mushtaq. Personal communication, March 2010.

[16] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Internet Measurement Conference*, October 2006.