**Fast Flux Watch: A mechanism for online detection of fast flux networks**

Basheer N. Al-Duwairi *, Ahmad T. Al-Hammouri

CyberSecurity Research Laboratory, Department of Network Engineering and Security, Jordan University of Science and Technology, Irbid 22110, Jordan

**ABSTRACT**

Fast flux networks represent a special type of botnets that are used to provide highly available web services to a backend server, which usually hosts malicious content. Detection of fast flux networks continues to be a challenging issue because of the similar behavior between these networks and other legitimate infrastructures, such as CDNs and server farms. This paper proposes Fast Flux Watch (FF-Watch), a mechanism for online detection of fast flux agents. FF-Watch is envisioned to exist as a software agent at leaf routers that connect stub networks to the Internet. The core mechanism of FF-Watch is based on the inherent feature of fast flux networks: flux agents within stub networks take the role of relaying client requests to point-of-sale websites of spam campaigns. The main idea of FF-Watch is to correlate incoming TCP connection requests to flux agents within a stub network with outgoing TCP connection requests from the same agents to the point-of-sale website. Theoretical and traffic trace driven analysis shows that the proposed mechanism can be utilized to efficiently detect fast flux agents within a stub network.

© 2014 Production and hosting by Elsevier B.V. on behalf of Cairo University.

**Introduction**

Botnets, networks of compromised machines under an attacker’s control, are the source of so many security threats including distributed denial-of-service (DDoS) attacks, spam, and identity theft [1–8]. Fast flux networks (FFNs) represent a special type of botnets that are being used by cybercriminals—in a way similar to that used in Content Distribution Networks (CDNs) and Round Robin Domain Name System (RRDNS)—to provide high availability and dynamicity for their malicious websites (usually online scam websites). The main idea of fast flux networks is to use bot machines as proxies that relay user requests to backend servers (i.e., the content servers). A frequent and fast change of proxies (known as flux agents) is required to evade detection and blocking, and to ensure high availability at the same time because these bots are often typical PCs that go online and offline at different times.

Fast flux networks represent a new trend in the operation and management of online spam campaigns. In these campaigns, spammers flood email inboxes of thousands of email users with advertisements about different products or services (e.g., pharmaceutical, adult content, phishing, etc.). These advertisements usually include hyperlinks to websites that represent the point-of-sale for the campaigns. Until recently, each point-of-sale website is used to map to a single IP address that...
remains static for considerable amount of time, and thus giving defenders the opportunity to block access to the corresponding website, or even track it for the sake of legal pursuits. With FFNs, the domain name of the point-of-sale website maps to several IP addresses that keep changing at a fast rate. Fast flux domains are characterized by the very short TTL values for their A records, and by the frequent change-of-mapping to multiple IP addresses that usually belong to different autonomous systems [9,10].

Previous work in the area of FFNs has mainly focused on detecting and characterizing FFNs by analyzing Domain Name System (DNS) records of suspicious domain names. In this context, DNS records could be collected by actively querying the DNS system for domain names found in spam email messages (this approach was followed by Holz et al. [9]). Alternatively, the DNS records can be collected through passive monitoring of DNS traffic of an Internet Service Provider (ISP) network (an approach that was followed by Perdisci et al. [11]). Both approaches require collecting massive amount of information for analysis, and they do not provide a real-time detection of fast flux agents.

In this paper, we propose a novel mechanism for real-time detection of flux agents within an organizational network without requiring the collection of DNS traffic information. The proposed mechanism, called Fast Flux Watch (FF-Watch), is envisioned to exist as a software agent at leaf routers that connect stub networks to the Internet. The core mechanism of FF-Watch is based on the inherent features of fast flux networks where flux agents within stub networks take the role of relaying/redirecting client requests to point-of-sale websites of spam campaigns. Therefore, the basic idea is to correlate incoming TCP connection requests to flux agents within a stub network with outgoing TCP connection requests from the same agents to the point-of-sale website.

The rest of this paper is organized as follows. In ‘Fast flux networks’, we provide the relevant background about fast flux networks and their role in hosting online scam. ‘Methodology’ section describes the proposed FF-Watch mechanism. Then we present the evaluation of the proposed FF-Watch mechanism and discuss the results. Finally, conclusions and future research directions are outlined.

Fast flux networks

The issue of fast flux networks was reported for the first time by the Honeynet project [12] in 2007. However, Holz et al. [9] were the first to study this phenomenon systematically in 2008. Basically, FFNs can be considered as a special type of botnets that are used by botmasters to provide high availability to their malicious websites (known as mothership servers) while hiding their location and identity (i.e., IP addresses) to avoid blacklisting. These networks consist of compromised nodes called flux agents that serve as proxies to the mothership servers. A request to a fast flux domain will go through one of the flux agents before being forwarded to the mothership server. The flux agent will relay the response back to the client as shown in Fig. 1.

There have been considerable research efforts focusing mainly on detecting and characterizing FFNs. Previous work mainly relied on collecting domain names from e-mail’s spam traps as the primary source of information, with the main goal is to classify domains into fast flux domains and non-fast flux domains based on certain features and characteristics that distinguish fast flux domains using different machine-learning algorithms. Generally, the research done in this area can be categorized, based on the approach followed in identifying flux agents, as follows.

- **Active detection**: In this approach, domain names of scam websites are extracted from spam archives, which were obtained from various spam traps. For each domain name, several DNS queries are performed (e.g., using the dig tool) to collect information about the set of resolved IP addresses. DNS answers for these queries are then examined to determine whether the domain name is being either legitimate or fast flux. The decision is based on observing certain features that characterize FFNs, and is usually done using artificial intelligence algorithms. This approach was adopted by most of the previous work in this field [13].

- **Passive detection**: This approach was proposed Perdisci et al. [11]. In this approach, live traces of DNS traffic (queries and answers) are collected by placing monitors at various strategic locations in an ISP network. The traffic is then analyzed searching for FFNs’ footprints. The premise here is that it is possible to capture DNS information of domain names not only present in spam emails, but also in any other online applications, such as chat rooms, and malicious websites. The advantage of this approach is that it does not pose additional load on network resources to make active DNS lookups, as in the active approach. Additionally, it cannot be detected by botmasters who may suspect high DNS lookup rates on their infrastructure.

- **Real-time detection**: Recently, Hsu et al. [14] presented a system to detect FFNs in real-time with the goal to cut the detection time to few seconds without affecting the detection accuracy. The idea relies on the observation of the longer delays for HTTP responses as a result of relaying the requests via fast flux agents. Relaying requests through fast flux nodes typically requires additional time because of the
relatively limited computation power and bandwidth associated with the agents. Another real-time fast flux detection approach was proposed by Martinez-Bea et al. [15].

The main problem with the passive approach is the need to deal with huge amount of DNS traffic traces that correspond to legitimate and non-legitimate domain names. In contrast, the active-detection-based approach deals with fewer amounts of DNS traffic traces that correspond to non-legitimate domain names in most cases. On the other hand, real-time detection approach may incur high false positive and false negative rates due to the possibility of misclassifying legitimate Web servers having both limited bandwidth and limited computing power as fast flux domains, while missing FFNs possessing high bandwidth and computational capacity machines.

The empirical measurements of fast flux networks performed by Holz et al. [9] revealed very interesting facts about the nature of these networks, such as geographical distribution of flux agents, sharing of flux agents, and sharing of scam web pages. Subsequent research studies have confirmed these findings and added new knowledge to the field. In particular, the study performed by Konte et al. [10], focused on the dynamics and roles of fast-flux networks in mounting scam campaigns. The study considered the rate of change in fast-flux networks, the change of locations in the DNS hierarchy, and the extent to which the fast-flux network infrastructure is shared across different campaigns. Other studies (e.g., [16,17]) focused on botnet detection through fast flux identification.

Our proposed mechanism, FF-Watch, differs completely from the previous work in the sense that it does not require collection and analyzing huge amounts of DNS traffic actively or passively. The key feature of FF-Watch is to utilize the inherent feature of fast flux networks that flux agents within stub networks take the role of relaying client requests to the point-of-sale websites of scam campaigns. In this context, FF-Watch can exist as a software agent at leaf routers that connect end hosts to the Internet.

Methodology

In this section, we discuss the design and architecture of the proposed FF-Watch mechanism. First, we explain the basic idea of this mechanism, then we provide details about its different aspects.

The basic idea of FF-Watch

The basic idea of the proposed fast flux detection mechanism is to correlate incoming TCP connection requests (i.e., incoming SYN packets) to machines within a stub network with outgoing TCP connection (i.e., outgoing SYN packets) requests from the same internal machines to an external server within a certain time window. The intuition here is that such machines are likely to act as flux agents that are part of a fast flux network. Typically, flux agents within a stub network act as proxies that relay traffic between web clients and a backend server that hosts a malicious content. This means that monitoring and correlating incoming and outgoing TCP connection establishment requests at the leaf router of a stub network would allow the identification of flux agents within that stub network.

For illustration, we consider the scenario shown in Fig. 2a. In this scenario, the point-of-sale website represents the content server of a spam campaign (e.g., www.anyprod.com) that employs fast flux mechanisms. Machines A, B, and C shown in the stub network are assumed to be flux agents for that domain. When a client visits www.anyprod.com, then he/she will be directed via DNS to one of the agents in the stub network (e.g., machine A). That agent then connects to the point-of-sale server and relays content back to the client. After the connection establishment with machine A, the client issues HTTP specific commands (e.g., GET) and waits for the server’s response. However, since machine A is acting as a flux agent that relays requests to a backend server, then the agent itself establishes a TCP connection with that server and starts relaying clients’ requests. A typical messages exchange between the client, the flux agent, and the mothership server (i.e., the point-of-sale server) is shown in Fig. 2b.

Based on this example, it is clear that the leaf router of the stub network is in the best position to monitor and detect flux agents within the associated network. As just mentioned, this can be done by correlating incoming TCP connection requests to machines inside the stub network with TCP connection requests originating from the same machines to outside. To achieve this, it is sufficient to record the destination IP address of an incoming SYN packet and the time the packet passes by the router. An outgoing SYN packet with a source IP address that matches one of the already recorded addresses (and within a certain time window) is thus triggered as a strong indication that such request is originating from a flux agent within that stub network. Fig. 3 shows the FF-Watch algorithm to be performed at the leaf router of a stub network.

A Bloom filter-based implementation of FF-Watch (BFFF)

Bloom filters [18] represent a typical choice for efficient implementation of the proposed FF-Watch algorithm because of its ability to record SYN packets’ thumbprints with low storage requirements and an adjustable false positive rate. In addition, it offers fast way to check whether a packet is in the table or not. Here, we provide a brief description of Bloom filters.

Next, we describe how to implement the proposed FF-Watch mechanism using Bloom filters, and we provide theoretical analysis of the efficiency of this implementation.

Bloom filters

A Bloom filter is a data structure for representing a set of elements (also called keys) to support membership queries [18]. The idea (illustrated in Fig. 4) is to allocate a vector $R$ of $m$ bits, initially all set to 0, and then choose $k$ independent hash functions, each with range $\{1, \ldots, m\}$. For each given key, $A$, the bits at positions $H_1(A), H_2(A), \ldots, H_k(A)$ in $R$ are set to 1. (Note that, a particular bit position might be set to 1 multiple times.) Given a query for a key $B$, we first check the bits at positions $H_1(B), H_2(B), \ldots, H_k(B)$. If any of them is 0, then certainly $B$ was not previously inserted in the filter. Otherwise (i.e., all $H_i(B)$ are 1’s), we conjecture that $B$ was inserted in the filter although there is a given probability that this was not the case, i.e., a false positive. The two parameters $k$ and $m$ should be chosen such that the probability of a false positive
is acceptably low. The false positive rate, $P_{FP}$, of a Bloom filter is given by:

$$P_{FP} = \left(1 - \left(1 - \frac{1}{m}\right)^k\right)^n \approx \left(1 - e^{-kn/m}\right)^k. \quad (1)$$

**Modified FF-Watch algorithm**

The original FF-Watch algorithm (Fig. 3) represents a basic and naïve way to correlate incoming and outgoing SYN packets. The main challenge of using a Bloom filter to implement the basic algorithm is the fact that we cannot keep track of SYN packets’ arrival times. Therefore, it is not possible to map incoming and outgoing SYN packets (to and from the same source address) within a certain time window. To overcome this problem, we propose to convert SYN packet’s arrival time to a coarse time in a way similar to that used in SYN cookies [19]. As such, the incoming SYN packets table (IST) can be implemented as a Bloom filter with $k$ hash functions. Inserting inbound SYN packets’ thumbprints in the IST can be achieved by calculating the Bloom filter’s hash functions of the packet’s destination IP address and its coarse time. On the other hand, membership testing for outbound SYN packets can be achieved by calculating the Bloom filter’s $k$ hash functions of the packet’s source IP address and its coarse time. In the latter, step, if any one of the hashed IST bits is zero, the source address of the packet was not previously stored in the table, and so the connection is originating from a benign machine. If, however, all the bits in the second step are one, it is highly likely the exact source IP address of the packet was previously stored in IST. However, it is also possible to have a false positive due to the fact that some other insertions of different IP addresses resulted in setting the same bits to one. Since a Bloom filter has limited capacity, it is important to point out that once the full capacity of the IST is reached, it becomes necessary to swap to another empty one. Fig. 5 shows the Bloom-filter-based implementation of the BFFF-Watch algorithm.

In SYN cookies, the coarse time, $t$, is a 32-bit time counter that increases every 64 s [19]. It is possible to adapt the same
approach for setting the coarse time in the BFFF-Watch algorithm. However, $64$ s is considered a large value for mapping incoming and outgoing SYN packets from and to a flux agent within a stub network. In fact, the time counter increment represents an interesting parameter that affects the performance of the algorithm. Selecting a large increment value would result in high false positive rate because the algorithm would then correlate SYN packets originating from sources inside the stub network with those seen coming from outside which is not necessarily true. On the other hand, selecting a small increment value would result in high false negative rate because many SYN packets originating from flux agents within the stub network might be missed.

**Results and discussion**

Ideally, evaluating the proposed FF-Watch mechanism requires access to an enterprise traffic traces (incoming and outgoing) that is confirmed to contain fast flux behavior. Because such traffic trace is difficult to obtain, we used traffic traces that we believe do not contain fast flux behavior since it dates back to a time long enough before fast flux had been employed by botmasters. Analyzing such traffic traces focusing on incoming and outgoing SYN packets in a way similar to that described in FF-Watch would provide guidance for selecting the appropriate coarse time increments and the typical amount of time for which packet digests need to be stored in the Bloom filter. This time amount is necessary to estimate the memory requirement of BFFF-Watch.

**Trace-driven evaluation**

In this subsection, we validate the proposed FF-Watch mechanism using Internet packet-level traffic traces. The data consist of $11$ GB of anonymized packet header traces that were originally collected at Lawrence Berkeley National Laboratory (LBNL); see [20]. The traces were gathered at two core routers inside LBNL’s network during the following times

- 10 min on October 4, 2004.
- One hour on December 15, 2004.
- Once hour on December 16, 2004.
- One hour on January 6, 2005.
- One hour on January 7, 2005.

We use these data for evaluation as follows. We first preprocess the trace files to extract only the SYN packets. Overall, the trace files contained $550,226$ SYN packets. For each resultant record associated with a SYN packet, we take the destination IP address and search for matching source IP address starting from the very next record in trace file. We report the occurrences along with the time difference of the two SYN packets (i.e., the ones having common IP addresses as a destination and as a source). We found that $97,713$ such events that enjoy the general behavior of FFNs. Fig. 6 shows the cumulative distribution function (CDF) of the time interval length between an incoming and an outgoing SYN packet to and from a same host observed at either router. Almost $90\%$ of such behavior (i.e., the suspected FFNs behavior) occurs in $10$ min or less. (Note that in the figure, we have truncated the results for interval lengths larger than $10$ min.)

![Fig. 6](image)

**Fig. 6** The cumulative distribution of time interval between an incoming and outgoing SYN packets to and from the same machine for the range $[0,600]$ seconds.

Given that the whole idea of FFNs is to serve Web requests via redirection, it is safe to assume that the time interval between an incoming and outgoing SYN needs to be minimal because of the interactivity characteristic of web requests and responses. Hence, we then zoom into those intervals that are shorter than $200$ ms; the results are shown in Fig. 7. Approximately, $6\%$ of the intervals are $100$ ms or less. Therefore, a question arises why such a behavior (i.e., the FFNs’ behavior) exists in the traffic.

First, it is due to the nature of the traffic that contains both intra-organizational traffic and wide-area network traffic. Due to the anonymization process, we cannot separate these two types of traffic based on the IP addresses. However, closely looking into the traffic reveals that most of the suspected behavior stems from e-mail, network management, host name resolution, etc.; see [21] for a through categorization of the different traffic types. Consequently, we then proceed to focus only on the HTTP traffic. The top part of Fig. 8 shows the CDF of intervals for the HTTP traffic only, that is, the traffic for which the incoming and outgoing requests are both HTTP. Now, about $10\%$ of the HTTP traffic exhibits the FFNs behavior with intervals of $100$ ms or less; see the bottom part of Fig. 8. However, $78\%$ ($434$ out of $557$) of all FFNs’ behavior instances (with the HTTP) comes from only two machines that we believe they work as HTTP proxies. After excluding the traffic associated with these two machines, we obtain the results in Fig. 9. It is clear that all intervals are now higher than $1.4$ s, a value that will never be appropriate for Web interactivity necessary for FFNs. As one conclusion, the results in Fig. 9
suggest that we can set the coarse time increment in the BFFF-Watch algorithm to be within this range, say, 1 s.

A concern might arise of how for a router to then differentiate between legitimate web traffic redirection (e.g., via open proxies) and FFNs traffic. The answer is to whitelist all such legitimate services that are located inside an organization perimeter, or within an ISP boundary.

Memory requirements of (BFFF-Watch)

The amount of memory required to store incoming SYN packet digests for long time in the IST is not necessary in BFFF-Watch because of the online nature of this algorithm. In theory, it is obvious that packet digests need to be stored in the IST for an amount of time that is slightly larger than the time duration between incoming SYN packets destined to certain nodes within the stub network and outgoing SYN packets originating from the same nodes. Since this value varies, we will assume that $T$ seconds is an appropriate amount of time that can be used after which IST can discard stored digests to make room for new SYN packets.

Based on the results indicated in Fig. 9, we can say that duration of 1.4 s represents a suitable value of $T$. However, this value may be conservative and requires swapping the IST frequently, so setting $T$ to a larger value (e.g., 60 s) can be a better choice.

The amount of memory required by an IST depends on several factors that include the number of incoming SYN packets during the observation interval $T$, and the targeted false positive rate expressed in Eq. (1). For example, using a bloom filter with three hash functions ($k = 3$), and a memory efficiency ($n/m$) of 0.2, the effective false positive rate of 0.092 can be achieved for full Bloom filter [22]. Based on these calculations, a Bloom filter with size 1 M bits is sufficient to store the digests of 200,000 incoming SYN packets during an observation interval of 60 s.

Conclusions

Fast flux networks continue to be one of the major techniques used by botnets to provide highly available malicious web services without revealing the identity of a backend server. This paper presented FF-Watch, a mechanism for online detection of fast flux agents. FF-Watch is proposed as a software agent to exist at leaf routers that connect stub networks to the Internet. The core mechanism of FF-Watch is based on the inherent features of fast flux networks where flux agents within stub networks take the role of relaying client requests to point-of-sale websites of spam campaigns. The main idea of FF-Watch is to correlate incoming TCP connection requests to flux agents within a stub network with outgoing TCP connection requests from the same agents to the point-of-sale website. An efficient Bloom filter-based implementation of FF-Watch was proposed. Theoretical and traffic trace driven analyses show that the proposed mechanism can be deployed to efficiently detect fast flux agents within a stub network.

Future research directions include exploring collaborative approach for fast flux detection and identification (localization) of the mothership server(s), and evaluating the proposed mechanism using recent traffic traces that contain the fast flux behavior.

Conflict of interest

The authors have declared no conflict of interest.

Acknowledgment

This research was funded by the Deanship of Research at Jordan University of Science and Technology under Grant Number 288-2011. We would like to thank the anonymous reviewers for their helpful comments.
References

[1] Dagon D, Guofei Gu, Lee CP, Wenke Lee. A taxonomy of botnet structures. In: Proceedings of the 23rd computer security applications conference; 2007 December 10–14. p. 325–39.

[2] Rajab M, Zarfoss J, Monrose F, Terzis A. A multifaceted approach to understanding the botnet phenomenon. In: Proceedings of the 6th ACM SIGCOMM on Internet measurement; 2006. p. 41–52.

[3] Rajab M, Zarfoss J, Monrose F, Terzis A. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In: Proceedings of the 1st workshop on hot topics in understanding botnets; 2007.

[4] Karasaridis A, Rexroad B, Hoeflin D. Wide-scale botnet detection and characterization. In: Proceedings of the 1st workshop on hot topics in understanding botnets; 2007.

[5] Barford P, Yegneswaran V. An inside look at botnets. In: Christodorescu M, Jha S, Maughan D, Song D, Wang C, editors. Advances in information security. USA: Springer; 2007. p. 171–91.

[6] Grizzard JB, Sharma V, Nunnery C, Kang BB, Dagon D. Peer-to-peer botnets: overview and case study. In: Proceedings of the 1st workshop on hot topics in understanding botnets; 2007.

[7] Holz T. A short visit to the bot zoo. IEEE Sec Privacy 2005;3(3):76–9.

[8] Guofei Gu, Perdisci R, Zhang J, Wenke Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. USENIX Security; 2008. p. 139–54.

[9] Holz T, Gorecki C, Rieck K, Freiling FC. Measuring and detecting fast-flux service networks. In: Proceedings of the network and distributed system security symposium; 2008.

[10] Konte M, Feamster N, Jung J. Dynamics of online scam hosting infrastructure. In: Proceedings of the 10th international conference on passive and active network measurement; 2009. p. 219–28.

[11] Perdisci R, Corona I, Dagon D, Wenke Lee. Detecting malicious flux service networks through passive analysis of recursive DNS traces. In: Proceedings of the computer security applications conference; 2009. p. 311–20.

[12] The Honeynet project, know your enemy: fast-flux service networks [Internet]; 2007 [cited 27.08.13]. <http://www.honeynet.org/book/export/html/130>.

[13] Hu X, Kaysz M, Shin KG. Measurement and analysis of global IP-usage patterns of fast flux botnets. In: Proceedings of IEEE INFOCOM, Shanghai, China: 2011 April 10–15. p. 2633–41.

[14] Hsu C-H, Huang C-Y, Chen K-T. Fast-flux bot detection in real time. In: Proceedings of the 13th international conference on Recent advances in intrusion detection; 2010. p. 464–83.

[15] Martinez-Bea S, Castillo-Perez S, Garcia-Alfaro J. Real-time malicious fast-flux detection using DNS and bot related features. In: 2013 Eleventh annual international conference on privacy, security and trust (PST); 2013. p. 369–72.

[16] Yadav S, Reddy AKK, Reddy ALN, Ranjan S. Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. IEEE/ACM Trans Netw 2012;20(5):1663–77.

[17] Zhao D, Traore I. P2P botnet detection through malicious fast flux network identification. In: 2012 Seventh international conference on P2P, parallel, grid, cloud and Internet computing (3PGCIC); 2012. p. 170–5.

[18] Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM 1970;13(7):422–6.

[19] Bernstein DJ. Syn cookies [Internet] [cited 27.08.13]. <http://cr.yp.to/syncookies.html>.

[20] LBNL/ICSI enterprise tracing project [Internet] [cited 27.08.13]. <http://www.icir.org/enterprise-tracing/download.html>.

[21] Pang R, Allman M, Bennett M, Lee J, Paxson V, Tierney B. A first look at modern enterprise traffic. In: Proceedings of the 5th ACM SIGCOMM on Internet measurement; 2005.

[22] Snoeren AC, Partridge C, Sanchez LA, Jones CE, Tchakountio F, Kent ST, et al. Hash-based IP traceback. In: Proceedings of the 2001 SIGCOMM conference; 2001 August. p. 3–14.