fluxDetect – A Heuristic Approach to Detecting Fast-Flux Service Networks

Ashu Goel – G024 Summer Intern

8/5/11

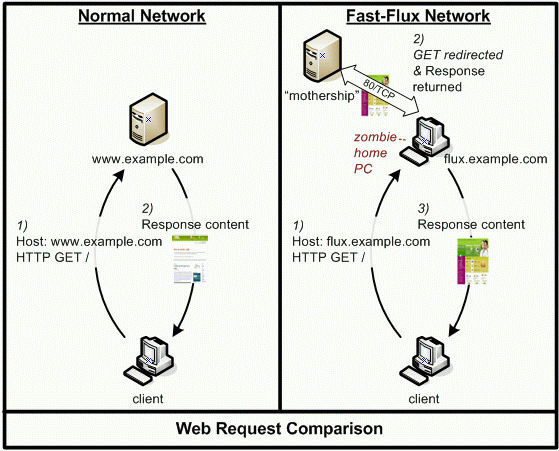
***Background***

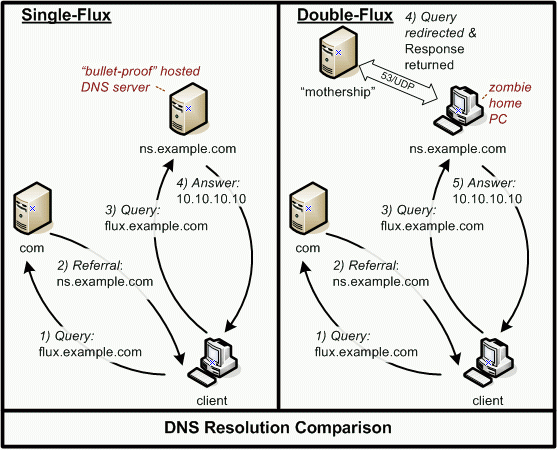
A fast flux botnet registers fully qualified domain names ([www.example.com](http://www.example.com)) to multiple IP addresses, which are then fluxed with extreme frequency, using both round-robin IP addresses and a very short time to live (TTL) for the given domain. For example, if the TTL of [www.example.com](http://www.example.com) was 3 minutes, then a browser connecting every 3 minutes would be connecting to a different infected computer each time. As a result, it makes it very hard to track and detect these fast-flux service networks.

The second key defining feature of fast-flux botnets is blind proxy redirection. When a computer attempts to connect to a certain IP address, the compromised system merely acts as a proxy to a back-end server that actually serves the content. The client trying to connect has no idea that it is simply connecting to a “dummy” bot, of sorts. This technique has been used for quite a while in legitimate webserver operations, to spread the load and maintain high availability, but obviously it is now being widely used by cyber criminals.

The back-end servers are known as fast-flux “motherships”, and are the controlling elements behind the entire network. They are quite similar to the C2 systems found in most common botnets, with one major exception: while most of the C2 servers use IRC to communicate, fast-flux service networks **have almost always been found to use HTTP methods** to communicate. There are two main types of fast-flux networks: single-flux and double-flux.

The network described above is a single-flux network: the client connects to http://flux.example.com , which connects to the “dummy” (usually home PC), which then forwards the request to the mothership, and returns the response to the client. The IP addresses for the front-end nodes are changed as often as every 3-10 minutes. An example is shown below:



 Double-flux domains add another layer of flux in that the in addition the A records being flux, the authoritative NS records are also fluxed. Below is an example illustrating the difference between single and double flux service networks. In the single-flux network on the left, the authoritative nameserver that is queried returns the IP address of the queried domain. By contrast, in the double-flux network on the right, when the domain name is queried to the “dummy” nameserver, the request is forwarded to the mothership, who then returns the IP address, which is then forwarded to the client.

There are 3 main advantages for attackers using fast-flux networks. One of the main advantages is the simplicity of the network: there is only one mothership that can control everything, which makes it easy for the attacker in terms of logistics. Second, the front-end “dummy” nodes are essentially disposable assets. They are also very hard to track down because they are so disposable and widespread (can span all 7 continents). Finally, the existence of the dummy nodes prolongs the operational lifespan of the motherships, as they are protected by these proxies.

There are also a number of other considerations to take into account when developing detection methods for fast-flux botnets. For example, there is the issue of consumer-level hardware: most botnets consist of residential PCs. Since residential PCs are usually connected to the Internet via relatively low speed network links, most bots have low computation power and network bandwidth. As a result of the low computation power, a bot’s message forwarding to either the client or mothership can be slow. Furthermore, due to the low network bandwidth, significant network queuing will occur, which leads to a greater variation in network delays. Furthermore, attackers have to deal with uncontrollable foreground applications on the host PC. PC users can unknowingly be slowing down bot operations by running foreground applications, such as games, movies, etc. Therefore, it is safe to assume that if we can measure the network operations of the connection between a client and suspect server, we can use those as heuristics.

***Detection Mechanisms (Heuristics) – 15 Total***

1. *Number of A records ­(****from Holz, et. al)*** *–* since fast flux networks are known to point to multiple IP addresses, the number of A records returned in a simple dig lookup proves to be a good heuristic
2. *Number of Distinct Autonomous Service Numbers (ASNs)* ***(from Holz, et. al)****–* By doing a whois lookup of all the IP addresses in the A records, we can find the number of distinct ASNs of the domain, with the assumption that since fast flux networks are so widely distributed, they will have a greater number of distinct ASNs than benign servers
3. *Time to Live (****from Passerini et. al)***– the time to live of fast flux servers is usually very small compared to that of benign servers
4. *Number of Distinct Network Prefixes (****from Passerini et. al)***– Again using the whois lookups of all the IP addresses in the A records, we can find the number of distinct network prefixes associated with the domain
5. *Number of Nameserver Records (****from Holz et. al)****–* Do a simple NS lookup and get the number of nameserver records; this is helpful for recognizing double-flux networks, although fluxDetect does not distinguish between single and double flux botnets
6. *Spatial Uniform Distribution of A records (A Entropy)* ***(from Huang et. al)***
   * By performing geolocation (obtaining the latitude/longitude) of each IP address in the A record, we can “map” out the domain into a set
   * For each spatial mapping C(Q), where C(Q) = <C1 (latitude), C2 (longitude)>, of a set of IP addresses, the Time Zone Entropy (TZE) can be determined by:

, where Nt(C(Q)) is the number of IP addresses located in the *t*th timezone (using GMT).

1. *NS Entropy* ***–*** same as A Entropy, but with NS records instead ***(from Huang et. al)***
2. *Spatial Service Relationship Estimator (Average Minimal Service Distance*) ***(from Huang)***
   * based on actual latitude and longitude of IP address
   * define *dmm’* as the spatial service distance and a 2-norm distance in Euclidean space from the *m*th IP address *qm* in QA (A record) to the *m’*th IP address *qm’* in QNS (Nameserver record):

* + Therefore, the MSD for each IP address is defined as:
  + The Average MSD and Standard Deviation across all IP addresses are then computed and returned

1. *HTTP Delays* ***(from Hsu et. al)***
   * Three delay metrics used as heuristics (active probing)
   * **Network Delay (ND)**: difference between the time the client sends the first TCP SYN packet to the server and the time the client receives the TCP SYN+ACK packet from the server
   * **Document Fetch Delay (DFD)** – time required for the server to fetch a webpage. Let RD be the difference between the time the client sends out the HTTP GET request and the time the client receives the HTTP response (hopefully code 200). Then the DFD = RD – ND (network delay)
   * **Processing Delay (PD)** – time required for the server to process a dummy request (fluxDetect uses an HTTP HI method) that does not require any additional computation and I/O operation. Let AD be the difference between the time the client sends out the HTTP request and the time the client receives the HTTP response (hopefully code 400 or 405). Then PD = AD-ND. ***NOTE***: not all servers will return a 400 or 405 error code when the HTTP HI method will sent. Some servers, after not recognizing it, will treat it as a HTTP GET method and return a code 200 instead.
   * In order to make sure our delay metrics were accurate, the HTTP 1.1 persistent connection option was turned off (*no keep alive*), and the delays were repeated 4 more times for a total of 5 connections. The persistent connection option had to be turned off in order to ensure that a new network delay was measured each time.
   * **The average NDs, DFDs, and PDs, as well as their respective standard deviations, were returned (in seconds) as heuristics**
   * **A

***Classification Algorithm***

fluxDetect uses Weka, a Data Mining tool, to perform classification of URLs. It currently uses the **AdaBoostM1** algorithm developed by Freund and Schapire. The AdaBoost algorithm boosts other learning algorithms by running them multiple times and assigning a weight to each instance in which it is run. If the “error” of a certain instance goes above 0.5, the booster will stop running. The training set consisted of 100 benign URLs and 60 known fast-flux URLs. The benign URLs were taken from the Alexa Top 100, with some sites removed and others added, based on connection. The 60 known fast-flux URLs were taken from the ATLAS Global Fast Flux Database and the FastFlux Tracker at **abuse.ch (which is currently shut down)**. The following “Weak Learners” were used with 30 iterations of the boosting algorithm. Each learner was trained on the set and then tested using a 10-fold cross validation:

1. **J48 Classification Tree**

* **True Positive Rate:** 96.7%
* **False Positive Rate:** 1%

1. **Multilayer Perceptron**
   * Learning Rate: 0.1
   * Momentum: 0.5
   * Uses 15% of training set as a validation set, and stops after 20 successive epochs of rising error
   * **True Positive Rate:** 95%
   * **False Positive Rate:** 0%
2. **Sequential Minimal Optimization (an SVM)**
   * **True Positive Rate:** 95%
   * **False Positive Rate:** 2%

***Errors and Further Development***

* *Error Handling*: sometimes the client fails to resolve or connect to a certain domain name. What values should be returned for each respective heuristic? If training, should the URL be thrown out? If testing, should the URL be immediately flagged as fast-flux?
* *Re-querying* – Since many fast-flux domains use Round Robin DNS fluxing, we would get more results by re-doing a lookup after the TTL has expired. For example, if a suspected domain has a TTL of 300 seconds, then the program could re-do the dig lookup after 300 seconds have expired to see if any new IP addresses are returned
* *Heuristics based on alphanumeric characteristics of domain names*: **Yadav, et. al** published a recent report on detecting fast flux networks (specifically those with algorithmically generated domain names, such as Kraken/Bobax, Conficker, and Torpig) based on the alphanumeric characteristics of the dataset. However, many of these algorithmically generated domains do not follow traditional fast flux networks in that many only map to 1 IP address, which could create false negatives
* *Better training data*: very few active fast flux domain names are publicly available. Although preliminary results look promising, it is hard to be sure with only 60 fast flux domains
* *Runtime*: Since the client is forced to connect 5 times to the domain in order to get delay metrics, the runtime is sub-optimal (depending on the server). Is there a way to improve this?
* ***Proxy Connections:*** When connected to the MITRE proxy, not all domain names will be resolved. Therefore, it is more useful to connect to unmanaged internet and use a public DNS server, such as the Google Public DNS

***Next Steps****:* Apply methods used here to passively detect C2 botnets (in progress).

**References**

Alexa Top Sites in the United States. Visited: 5 Aug 2011.

ATLAS: Arbor Networks, Inc. <http://atlas.arbor.net/>

dnsbl.abuse.ch: abuse.ch fastflux tracker (2010), http://dnsbl.abuse.ch/fastfluxtracker.php

Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

T. Holz, C. Gorecki, K. Rieck, and F.C. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Proceedings of the Network and Distributed System Security Symposium*, 2008.

C. Hsu, C. Huang, and K. Chen. Fast-Flux Botnet Detection in Real Time. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection,* 2010.

S. Huang, C. Mao, and H. Lee. Fast-Flux Service Network Detection Based on Spatial Snapshot Mechanism for Delay-Free Detection. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security,* 2010.

The Honeynet Project. Know your enemy: Fast-flux service networks, 2007.

Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*.Morgan Kaufmann Publishers Inc., San Francisco,CA, USA, 2005.

E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. FluXOR: Detecting and Monitoring Fast-Flux Service Networks. In *Proceedings of the 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, 2008.

R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Fast Flux Service Networks Through Passive Analysis of Recursive DNS Traces. In *Proceedings of the 2009 Annual Computer Security Applications Conference,* 2009.

S. Yadav, A.K.K. Reddy, A.L.N. Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. In *Proceedings of the 10th Annual Conference on Internet Measurement,* 2010.