**Process-level C2 Communication Thwarter for DGA-Based Malware**

Anjali Menon, *University of Illinois at Urbana-Champaign*

Abstract

Since the advent of Domain Generation Algorithm (DGA) -based malware, the tools to prevent the malware from communicating to its C2 (Command and Control) server have been centered around either detecting Algorithmically Generated Domain Names or isolating entire devices in the network which receive a high volume of NXDomain responses. These techniques either do not provide any immediate fix for the affected system or block all network traffic to and from the device. In this paper, we propose a tool [1] for thwarting communication to its C2 by a DGA-based malware by performing process-level tracking of DNS responses.

1. Introduction

Domain Generation Algorithms are algorithms used, particularly by some malwares, to generate a large number of domain names. A subset of the domains generated by these algorithms, called Algorithmically Generated Domains or AGDs, directly point to the command and control (C2) servers of the respective malwares. Since the domain names are dynamically generated and are often hard to predict unless the malware itself is reverse engineered, traditional methods such as creating blacklisted domains and sinkholes cannot be effectively used to prevent the infected device from contacting the C2 server.

Current research around DGA-based malware focus on predicting the AGDs ahead of time by lexically analyzing domain names, often by employing machine learning techniques. However, these methods require a training phase, and if a new DGA is discovered it would take time before AGDs can be accurately identified. Additionally, the effectiveness of these techniques depends on the accuracy of the machine learning model.

Other works [2] have combined the AGD detection approach with the observation that DGAs tend to generate queries that result in a high volume of NXDomain responses. This is due to the fact that only a small fraction of the many Algorithmically Generated Domains resolves to the C2 server, as it is not feasible for the botmasters to register all possible values of the AGDs. The technique of monitoring the number of NXDomain responses on the edge of the network has also been used in some tools to isolate potentially infected devices from the network.

In this paper, we extend this concept of tracking the cardinality of DNS resolution failures to isolate potentially infected processes in a device as opposed to isolating the entire device from the network. The advantage of this approach is that the infected device can still continue functioning normally while ensuring that only communication of the device with C2 will be hampered.

This paper makes the following contributions:

* Provides a band-aid to stop any processes in a device that are suspected to be DGA-based malwares from contacting their C2s, while allowing other non-suspicious processes to continue making regular network calls.
* Identifies potential C2 IP addresses and configure device firewall to permanently blacklist them.

2. Related Work

To be filled.

3. Design

As observed in Antonakakis et al. [2], most of the domains generated by a DGA would result in Non-Existent Domain (NXDomain) responses. Our DGA-C2 Communication Thwarter is built on the fact that non-DGA processes would not generate a large number of DNS queries that result in NXDomain responses. Therefore, a process that receives domain resolution failures greater than a pre-defined threshold must very likely be running a Domain Generation Algorithm and attempting to receive data from such dynamically generated domains.

**3.1 Routing All Incoming DNS Responses to a Listener**

We setup the firewall of the device to route all incoming DNS responses to our listener. DNS responses are identified by their source port number 53.

**3.2 Process-level NXDomain Tracking**

For each DNS response packet that is routed to our listener, we perform the following actions:

**3.2.1 Find the process to which the DNS response belongs**

We do this by finding the process id of the open UDP connection where the local listening port is equal to the destination port in the DNS response packet. This provides an accurate method for finding the process id corresponding to a DNS packet because the port on which a DNS request is sent out will be the same port that will be listening to the corresponding response.

**3.2.2 Record the process and DNS response information**

If the response is NXDomain (identified by DNS rcode field 3), then we record the process id, process name and the DNS response information for the particular packet in our database.

**3.2.3 Blacklisting suspicious processes**

If the number of NXDomain responses previously received by a process id, as recorded in our database, is greater than a pre-defined threshold, then the process is added to a blacklist.

**3.3 Spoofing Future Successful Domain Resolutions for Blacklisted Processes**

For any successful domain resolution for a blacklisted process (identified by DNS rcode field 0), we assume that this successful resolution must give the IP address of the actual C2 server. Therefore, we spoof the DNS response packet to be an NXDomain response and forwards it to the target process.

The information about the packet pre-spoofing is also stored in the database.

All other packets and all packets of non-blacklisted processes are forwarded to the respective processes as is.

**3.4 Block All Traffic from Suspicious IP Addresses**

Since response IPs in the successful domain resolutions for blacklisted processes must likely be the addresses of the C2 servers, we mark these addresses as suspicious. We then add firewall rules to block all further traffic from these suspicious IP addresses.

**3.5 Cleaning Stale Data**

We should only be tracking DNS responses for active processes. Since it is possible that a process id of a dead process can be re-used by a future process, we periodically run a job to clear our database of information about dead processes.

4. Experimental Setup

5. Results

6. Discussion and Limitations

7. Conclusion

Acknowledgments

I would like to thank Professor John Bambenek for his guidance and helpful feedback.

References

[1] <https://github.com/anjalim3/DGA-C2-Communication-Thwarter>

[2] https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final127.pdf

[3] https://onlinelibrary.wiley.com/doi/full/10.1002/sec.1495