infoblox

DECOY DOG IS NO ORDINARY PUPY:

Separating a Sly DNS Malware from the Pack

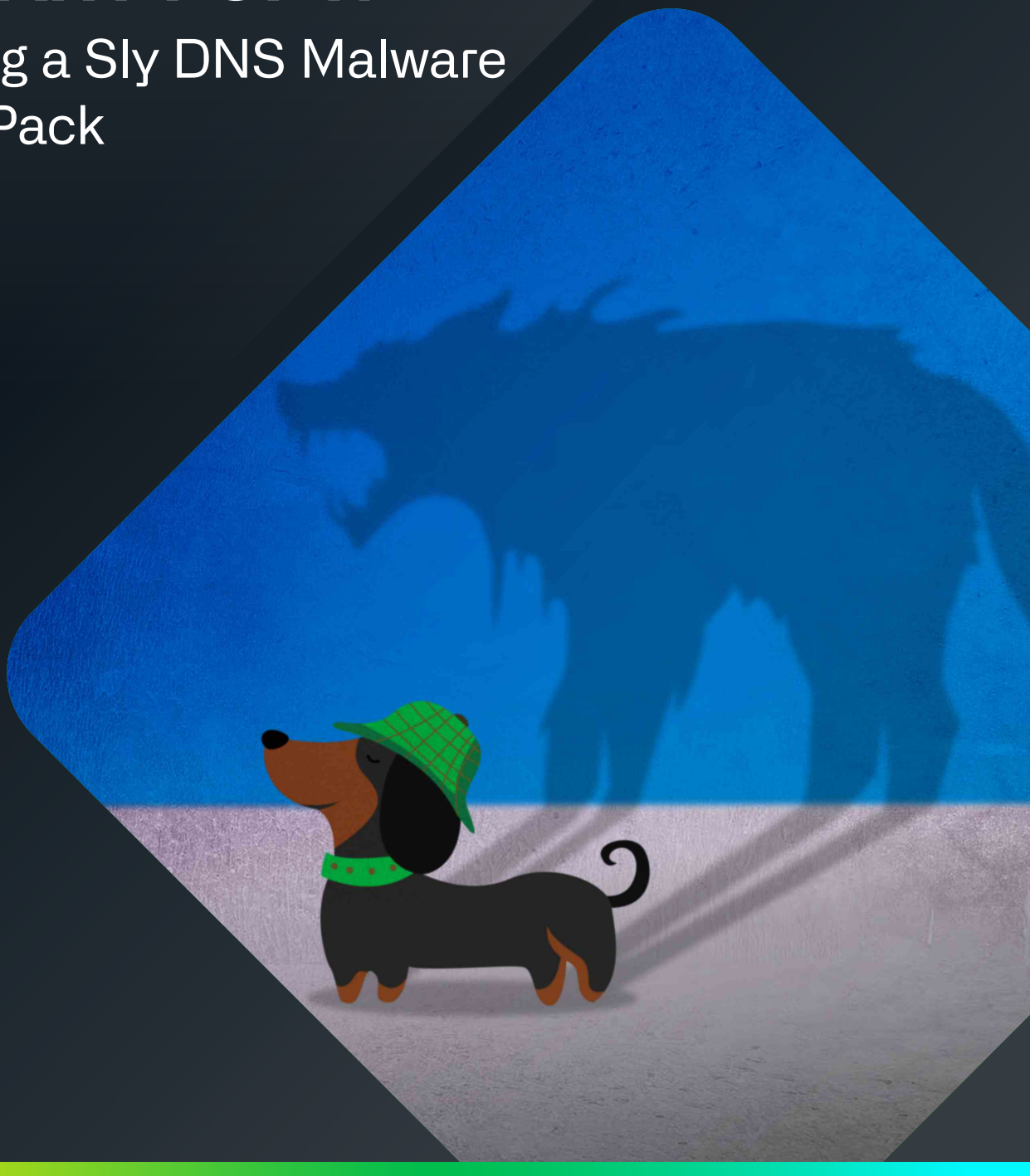

TABLE OF CONTENT

Executive Summary

Decoy Dog is a malware toolkit discovered by Infoblox that uses the domain name system (DNS) to perform command and control (C2). A compromised client communicates with, and receives direction from, a controller via DNS queries. That controller is integrated into a DNS name server to which queries are transmitted through the normal resolution process. We disclosed Decoy Dog's existence in April 2023 and released a detailed report of our initial findings on April 23rd. The discovery was based on monitoring of DNS data. Analysis at the time confirmed that the toolkit was built based on a remote access trojan (RAT) known as Pupy, but it wasn't known what systems were being exploited, how the toolkit was deployed, or whether Pupy had been modified.[1] We expected that, with the details we provided, others in the community would locate the compromised machines and the full story would become known. However, the mystery surrounding Decoy Dog has only grown.

Since April, Infoblox has conducted further research into Decoy Dog and Pupy. This report is the result of that research. We have learned that Decoy Dog is a major upgrade to Pupy that uses commands and configurations that are not in the public repo. We developed algorithms to separate Decoy Dog client communications and infer a number of other properties about each controller. This allows us to conclude with high confidence that the toolkit has spread and is under the control of at least three actors. While the activity we have observed remains confined to Russia and Eastern Europe, there are distinct groupings of techniques, tactics, and procedures (TTPs) within the controllers consistent with multiple actors.

Every Decoy Dog actor responded to our April disclosures in some fashion, and the variations support our assessment of multiple operators. Directly following the first announcement on social media, some of the name servers were taken down. All remaining were modified to remove behavior we highlighted in our first paper, though this was accomplished in different ways depending on the controller. One set of controllers began restricting responses to queries depending on the country of origin, a technique called geofencing, while others altered their response to queries for the ping subdomain.

One actor responded so quickly to our disclosure on LinkedIn that we initially thought the new domains were copycat registrations by security researchers. Further analysis, however, showed that those were replacement domains. Rather than shutting down their operation, the actor transferred existing compromised clients to the new controllers. This is an extraordinary response demonstrating the actor felt it necessary to maintain access to their existing victims. It created a clear separation between the TTPs of one set of the Decoy Dog domains and all others.

In the weeks following our announcement, we were surprised that no one came forward to identify the underlying malware and vulnerability that gave Decoy Dog its foothold to operate. But as our research progressed, it became clear why the communications went undetected for over a year. Attacks using Decoy Dog have been highly targeted and each controller has a small number of active clients. Some servers have consistently maintained four to eight active clients for months at a time. While others saw increased numbers of simultaneously active clients over time, the entire number of impacted devices observed at any one time has been less than 100. A small victim set generally rules out financially motivated actors, and the need to persist on a device over a long period of time is consistent with highly advanced actors.

We were able to reconstruct portions of the Decoy Dog communications by identifying signatures from our own Pupy traffic. We established a Pupy server on the Internet, which, when combined with selective reverse engineering of the code, allowed us to correlate DNS queries and responses to specific Pupy commands. From this we were able to a) determine that Decoy Dog contains commands not found in Pupy, and b) characterize most of the

1 https://github.com/n1nj4sec/pupy

communications. Additionally, Decoy Dog actors appear to leverage Pupy to utilize other transport layers outside of DNS for functions such as key exchange. Threat actors likely consider this one of Pupy's advantages as a remote access trojan (RAT).

The first known deployment of the Decoy Dog toolkit took place in late-March or early-April 2022. It was sold or stolen shortly thereafter, as indicated by the emergence of a second controller, with different TTPs, that was active by mid-May. A third domain was registered in July 2022 and strategically aged until September. It is possible that these latter two controllers are owned by the same actor as they share many characteristics, including hosting in Russian IP space. However, they do have some differences. A few months later, two more domains were registered, again with distinct characteristics from the previous controllers. The actor who registered these domains migrated clients immediately following our disclosure to new domains. In total, Infoblox is currently monitoring 21 Decoy Dog domains, some of which were registered and deployed within the last month.

Having determined that Decoy Dog differed significantly from Pupy through our analysis of DNS logs, we examined related binary samples available on VirusTotal to see if the differences were apparent in the executables. Reverse engineering these samples showed that although they were detected as Pupy, they are far more advanced than the open source version. The samples include a) the ability to execute arbitrary Java code on the client, b) several new transport mechanisms, and c) new DNS mechanisms to ensure persistence. One mechanism is similar to a traditional DNS domain generation algorithm (DGA) and uses free dynamic DNS providers to connect to so-called emergency controllers. All of the samples share the same fundamental updates, although one of the samples has unique capabilities not seen in the others, related to the use of streaming transports.

For reasons that remain unclear, Decoy Dog violates core principles of covert communications, which generally aim to avoid detection and recovery of the content by an adversary. While normal Pupy servers reject communication queries repeated from compromised clients, Decoy Dog servers not only respond to replayed DNS queries but will respond to any well-crafted query. This behavior is similar to wildcard configurations in DNS and was a significant factor in the detection of Decoy Dog by Infoblox. Given the sophistication of Decoy Dog, we speculate that the replay and wildcard behavior are by design; whatever the intention, widespread replay of the DNS was partially responsible for the industry's inability to see Decoy Dog as a new malware.

Aggressive internet scanning by a security vendor led to the retransmission of millions of Decoy Dog communications through global networks, including several of our customers. This, in turn, led to our discovery of the toolkit. The vendor's inability to identify the traffic as malware, in order to avoid replaying the queries, triggered DNS connections from uninfected networks to the Decoy Dog controllers. We are confident that no Infoblox customer was infected and that the queries to our resolvers were all a result of anomalous vendor scanning. In spite of the lack of immediate threat to our customer networks, Decoy Dog remains a sophisticated toolkit with uncertain origins and it may continue to spread.

Not only is Decoy Dog newly observed in the wild, but to our knowledge, it is the first use of the DNS C2 component of Pupy in a malicious operation. In part, this is likely due to the difficulty of establishing a Pupy name server, which requires modifying the software in the repo and properly configuring the DNS. The lack of exposure makes it harder for the security industry to detect and defend against both Pupy and Decoy Dog. To help disrupt operations that use these C2 systems, we are providing the community with a research dataset containing Pupy DNS traffic captured from our own server and details of the inner workings of the software. This documentation is the first of its kind and will allow others to build detection algorithms, as well as reproduce our findings.

The story of Decoy Dog reveals the power of DNS as a source of threat detection and response. It also reveals an inherent weakness of the malware-centric intelligence ecosystem that dominates the security industry. The toolkit was discovered by DNS threat

detection algorithms, and the only defense against it today is DNS. Moreover, we had flagged several controller domains as suspicious and were blocking them at our resolvers prior to realizing they were all using a common malware. This type of protection, which thwarts malicious activity before it is identified, and often before it is operationalized, is unique to DNS detection and response systems.

In this paper we provide defenders with the knowledge to identify Pupy and Decoy Dog. While we will describe the DNS C2 in-depth, we will not provide information that helps bad actors deploy Pupy, nor will we disclose the full Decoy Dog DNS signature. We explain some behavior we identified in our original paper and highlight how Decoy Dog is distinct from Pupy. Further, we will describe our analysis of large volumes of Decoy Dog DNS traffic that allowed us to estimate the number of clients and the command traffic without possessing the malware itself or controlling the name server. We describe how the Decoy Dog samples differ from Pupy. Finally, we discuss how the Decoy Dog operators reacted to our disclosures and demonstrate common traits across subgroups of the controllers. The appendices contain additional supporting technical information.

BACKGROUND

Infoblox discovered Decoy Dog, a command and control (C2) toolkit using the domain name system (DNS) in early April 2023. It is based on an open source remote access trojan (RAT) called Pupy[2] and transports encrypted communication between clients and servers, or controllers, via domain name queries and IP address responses. The discovery arose from algorithms monitoring passive DNS queries to Infoblox resolvers for anomalous behavior. Queries for Decoy Dog domains had been made from security appliances in a small number of customer networks. These queries created a signature consistent with persistent, low-profile malware beaconing. Human review of the activity was alarming because although DNS was clearly being used as a confidential communication channel, the domains were not identified as C2 in any publicly available intelligence data. In fact, some were labeled "reputable" in online reputation checkers. We released a set of domains on April 13th to help the community block the traffic and identify the nature of the compromise.

During our original research, Infoblox identified a unique DNS signature that was independent of the Pupy software. The actors had deployed and operated their C2 system in a very specific way; for this reason, we identified Decoy Dog as a distinct toolkit. Only a small number of domains worldwide shared this signature, all of which were Decoy Dog name servers.

On April 23rd, we published part of the signature, initial analysis of the passive DNS, and a subset of the controller domains in our report "Dog Hunt: Finding Decoy Dog Toolkit in Anomalous DNS Traffic."[3] This paper highlighted a specific behavior of Pupy, in which it returned a series of localhost responses to queries for specific subdomains containing 'ping'. It also described a number of trends within the DNS communications that we could not fully explain at the time. In particular, we identified surprising patterns in the IP addresses that were returned in responses and the fact that the servers responded to replayed queries, which is unexpected for a covert communication system.

Following the announcements, a wide range of members of the security community, including vendors and other organizations, contacted us. Many of them had seen related traffic in their own networks, or in their customer networks, but no one had identified compromised devices or recognized the scope of activity. Some of those organizations provided information that led us to isolate and confirm how the DNS was generated in our

2 https://malpedia.caad.fkie.fraunhofer.de/details/win.pupy

3 https://blogs.infoblox.com/cyber-threat-intelligence/cyber-threat-advisory/dog-hunt-finding-decoy-dog-toolkit-via-anomalous-dns-traffic/

own networks. Others helped confirm the breadth of activity and test hypotheses. This informal collaboration was very useful and we are grateful.

For simplicity, we use the term Pupy in this paper to refer specifically to the Pupy DNS C2 and not to Pupy in general.

Pupy

A RARE BREED

Pupy is an open source post-exploitation remote access trojan (RAT) that features a complex modular transport system.[4] While the primary Pupy codebase was made available in GitHub in 2015, the DNS C2 mechanism was not added until 2019. This paper is the first public documentation of Pupy C2. Additionally, we are providing a dataset in GitHub for others to both reproduce our work and create defenses for the future.

While Pupy is open source, the use of the DNS C2 protocol is rare; we have been unable to identify its use outside of Decoy Dog in the wild.[5] From our own resolvers, which serve enterprises and organizations across the globe, we have found no evidence of Pupy DNS C2 use historically. Within global pDNS for the first six months of 2023, using DNS detectors we have developed for Pupy, we found no use of the software outside of Decoy Dog. Finally, we have privately queried a wide range of vendors; none of whom had seen it used either. Where the use of Pupy by advanced persistent threat (APT) actors was reported, apparently the DNS C2 components were not employed.[6]

The rare use of Pupy is likely due, at least in part, to the difficulty in operating the system. Establishing Pupy communications over the global DNS is not easy. It requires correctly configuring the name server and modifying the code in the GitHub repo. Additionally, there are complexities in DNS that vary across recursive resolvers that the Pupy software does not handle correctly. These challenges have likely hindered its adoption both by red teams and hackers alike, in contrast to popular tools like Cobalt Strike, which we see fairly frequently.[7]

Although Pupy DNS C2 is rare today, the use of Decoy Dog is spreading and the likelihood that defenders will face Pupy in some form is growing. In order to help prepare the community, Infoblox performed significant research on both Decoy Dog and Pupy. Infoblox deployed a Pupy server on the Internet to compare its behavior with Decoy Dog. We then captured packet data (pcap) and passive DNS logs from Infoblox resolvers. We used our Pupy deployment in conjunction with selectively reverse engineering of the code to better understand the unique nature of Decoy Dog. In this section, we explain the components of Pupy that are relevant to our research. For simplicity, we limit this paper to communications using IPv4 (A record) responses, although when available, Pupy will use IPv6 (AAAA) responses. The query encoding described in the paper is the current default for Pupy, version 2 (unless otherwise specified).[8]

---

4 https://github.com/n1nj4sec/pupy

5 The phrase “in the wild” is used in cyber security vernacular to mean deployed operationally and not part of isolated penetration testing or research.

6 https://www.volexity.com/blog/2022/06/15/driftingcloud-zero-day-sophos-firewall-exploitation-and-an-insidious-breach/

7 https://www.esecurityplanet.com/threats/how-cobalt-strike-became-a-favorite-tool-of-hackers/

8 An earlier version of Pupy C2 did not include host information in each query. We now know that Decoy Dog is version 3 of the client, but the query encoding appears to be the same as version 2.

HOW PUPY OPERATES

In our previous paper, we gave an overview of Pupy and highlighted some unusual characteristics of Decoy Dog.[9] In this paper, we will delve further into the Pupy communication protocol in order to demonstrate its connections to Decoy Dog and how to exploit passively collected Pupy DNS in order to understand an ongoing operation.

Pupy is designed to provide continuous communications between infected clients and the server so that when the actor wants to remotely access the client, the connection is already established. The actor is able to monitor connected clients and selectively command them to provide a wide range of actions. The DNS is used only for C2 communications. Any significant data exfiltrated from the client is sent over one of the many other transport options offered by Pupy. As a result, the Pupy DNS client is restricted to checking in with the controller, acknowledging commands, providing system information, and a handful of other duties. Between handling commands from the server, the client sleeps.

DNS communications are initiated and maintained by the client. The client sends queries through its normal DNS resolution path or through DNS over HTTPS (DoH) when it is enabled and available.[10] The controller sends commands in response to client requests in the form of encrypted IP addresses. Every query-response is a complete communication, meaning that neither the client nor server can split data for a single command across two DNS queries. This protocol is distinguished from common DNS tunneling systems, e.g., Iodine,[11] where the client establishes a session over DNS that may include the reconstruction of several packets at either end in order to process the communication. The client is obliged to acknowledge most commands, and the server responds to every valid client query with either commands or acknowledgement. The client vocabulary is extremely limited. It has nine types of queries by which it manages sessions, acknowledges commands, sends system information, and establishes keys. Custom commands can be added by writing additional functions, but require a full understanding of the software.

Upon waking, the client queries the server in one of two different ways, depending on whether a shared key has been established. This query provides the server with current information about the system and state of the Pupy client, or makes a simple query that serves to initiate a new encrypted session. While it is possible to disable encrypted sessions, this is not the default and it has not been observed in Decoy Dog. In response, the controller either acknowledges the request, requires the client to perform a key exchange, or sends new commands. Once the full set of commands is complete, the client will sleep for the established interval, by default 60 seconds. This process is repeated while the client is running. A high-level overview of the Pupy client-server communications is shown in Figure 1, and a more detailed view of the client process can be found in Appendix A.

9 https://blogs.infoblox.com/cyber-threat-intelligence/cyber-threat-advisory/dog-hunt-finding-decoy-dog-toolkit-via-anomalous-dns-traffic/

10 Pupy uses Quad9 servers by default for DoH.

11 https://github.com/yarrick/iodine

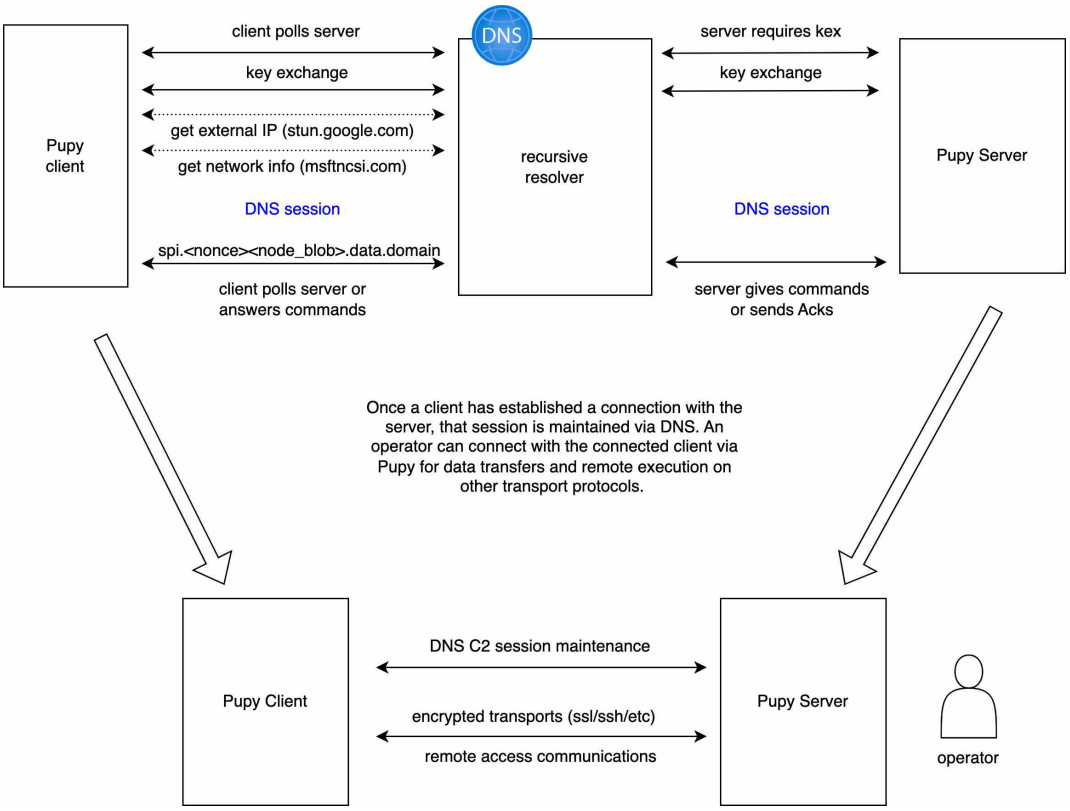

Figure 1. A high-level overview of Pupy communications.

The Pupy actor interacts with clients from the controller command line utility. When the client contacts the controller, any queued commands will be encoded in the DNS response. The operator establishes a connection on a client open port and specifies the transport layer to be used for exfiltration. The server DNS communications are still fairly restricted, although more extensive than the client. There are a wide range of commands and these can be concatenated into a single response to the client. Whereas the client initiates the communication exchange, the server is responsible for ensuring the security of the communications. It does this by enforcing so-called sessions with each client, which serve to rotate encryption keys. This is described in the next section.

SESSION INITIATION

Pupy requires an encrypted session to be established between the client and controller before transmitting the actor's commands. This session expires when client communications timeout and may be forced to renew for other reasons, including errors in the DNS query decoding or a restart of the client. Sessions are identified by the presence of a security parameter index (SPI) label in the query and are encrypted using an ephemeral shared key. Since communication details depend on a variety of factors, including whether the client has previously connected to the server, the exact protocol for session initialization can differ, creating variance in the observed DNS exchanges. However, the typical exchange is as follows:

- The client checks in to the server either without an established session or with an expired session (query 1).
- The server responds with a command requiring a key exchange and client system information.[12]

12 This is typically in the form of two commands referred to as Policy and Poll on the server side.

- The client acknowledges the requirement for system information (query 2).
- The client generates a random, private-public key pair using an elliptic curve algorithm and sends this to the server; the server does the same and responds with their new key (query 3).
- The client and server use this exchange to establish a new shared session key, which is used to encrypt packets with AES encryption, and also create the SPI to identify the session.
- The client gathers information about its network, including its external IP address, using additional DNS queries to other services.
- The client transmits this information using the shared encryption key and signaling the presence of an active session with the inclusion of the SPI in the query (query 4).
- The client sends additional system status information (query 5).

The shared key and SPI are typically established after three queries, although the key exchange is technically a single query and response. During a session, each query and response will be encrypted using this shared key. The encryption also uses a 32-bit nonce generated by the client that changes for every query. When a new session is established, the keys are regenerated, but the client nonce value continues while the client is operating. This is discussed further in the section below.

QUERY ENCODING

The client generates queries that contain encrypted communications to the server. These may include key exchange information or a response to commands from the server. There is a maximum of 52 bytes of transmitted data that can be communicated in each query. In addition to the transmitted data, each query includes:

- nonce, a 4-byte incrementing value generated by the client
- version, a 1-byte value indicating the Pupy DNS C2 version
- cid, a 4-byte value from the client's configuration, which is randomly generated when creating the client
- iid, a 2-byte value containing the bottom 16 bits of the Pupy client process
- node id, a 6-byte value from the client, typically the MAC address of the device
- optionally, SPI, a 4-byte value generated during the key exchange and present in queries representing a session on the server for a given client.

Every client query includes these 13 bytes of client information as well as a 4-byte checksum over the underlying payload. The underlying payload is encrypted and consists of a series of commands and related data.

The client encrypts and encodes data to be transmitted to the server as a fully qualified domain name (FQDN), called a query name (qname) in the DNS protocol. The entire process, shown in Figure 2 below, includes encrypting, arranging, and encoding both the transmitted data and additional information needed by the server. It works as follows:

- The data to be transmitted is appended with host-specific information.
- This composite byte string is encrypted using a shared symmetric key and the current nonce.
- The first encrypted bytes of the transmitted data, up to 35 bytes, is encoded and used for the first, or right-most, label of the qname.

- The remainder of the encrypted bytes, which may contain up to 17 bytes of the transmitted data, is prepended with the current nonce value, and encoded to create the second label of the qname.
- If the security parameter index (SPI) exists in the client, it is encoded and used in the third, or left-most, label of the qname; this value is set following a key exchange with the server.
- The nonce is incremented by the length of encrypted data within the client to be used for the next query.

The encoding from encrypted bytes to a domain name label was described in our previous paper. It uses a custom map in combination with 32-bit encoding to ensure that the final result is a valid domain name. The underlying data payload structure is described in Appendix B.

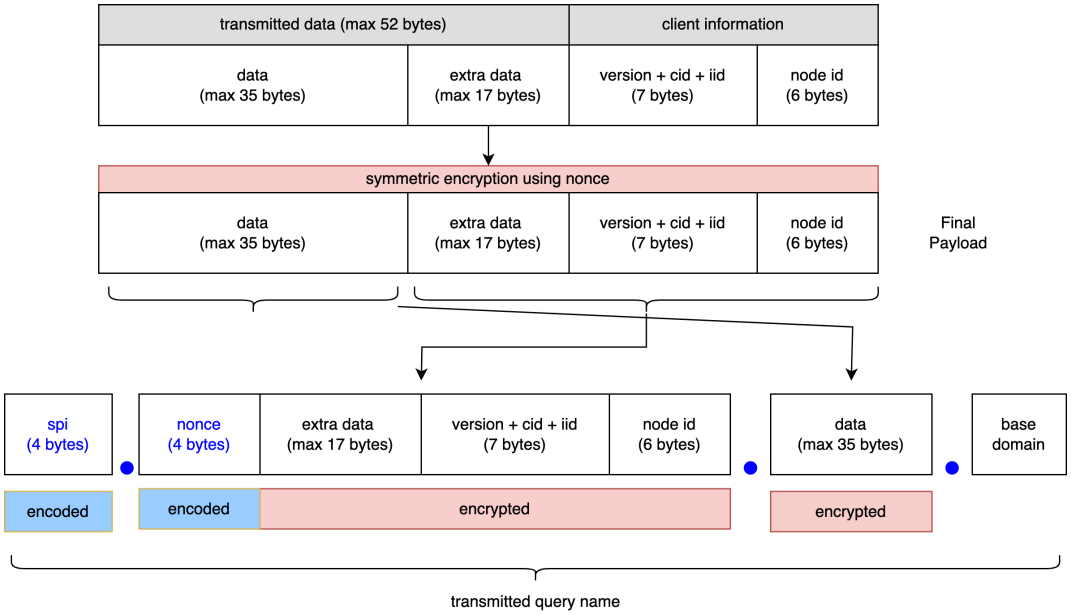

Figure 2. The process for converting data from the client to a qname for a DNS query. The base domain is the Pupy server's domain name.

Pupy uses AES by default when encrypting DNS queries. If a shared key has been established with the client, it uses this to encrypt the complete byte string symmetrically, otherwise it uses the established public key. In either case, the current nonce is also used in the encryption to ensure that the encoded query is unique, even if the underlying transmitted data remains the same across multiple queries. This is a standard mechanism to protect against cryptographic attacks. As a result, the queried domain name can be decoded to reveal the encrypted data, but the encrypted data cannot be decrypted without the key. The nonce value is initialized with a random 32-bit value and is incremented by the length of the payload on every query.

When the Pupy name server receives a query, it decodes the domain name to reveal the SPI value, the nonce, and the encrypted payload. To ensure it is receiving valid client communications, the server checks if the SPI is valid when present, and that the nonce is larger than the previous one recorded for the client. It makes several other checks on the data, including a check on the version number, which is encrypted in the payload. If any of these checks fails, it will return an error to the client.

In particular, Pupy does not answer the same query twice and any unmodified Pupy server will respond to a query it has already received in the past with an NXDOMAIN (no such

domain) response. We have validated this behavior with our own Pupy server by attempting to query a previously queried domain name. This is important because a characteristic of Decoy Dog is that it responds to replayed DNS queries with answers consistent with the Pupy C2 protocol.

Because the DNS query contains a reversible encoding of the nonce, and the nonce increments by the payload length in every query, we can reconstruct threads of queries associated with a single client. As we will see later in this paper, given passive DNS collection for a Pupy or Decoy Dog domain, we can use this reconstruction to estimate the number of clients, as well as the nature of the communication in certain cases.

SPECIAL DOMAIN NAME HANDLING

On receipt of a query, the server dissects the query name and determines whether it matches the appropriate structure for an encrypted packet from a client. There are a few special cases that have unique processing. Other than those special cases, it will reject any request that does not meet the expected format. One of those special cases is ping requests, which we described in our previous paper. A query for a subdomain pingN, where N is an integer, will return a sequence of localhost responses that is N long. A query for ping itself returns 15 such answers, and a query for the base domain returns a single localhost response, i.e. 127.0.0.1.

Outside of the ping requests, the server can be configured to respond to single label queries with a single IP address. The purpose of this functionality is unknown and it does not appear to be used in the client; it is referred to in the source code as a DNS activation request. This capability is not documented and to utilize it an actor would need to understand how the server software functions.

The special handling for single label subdomains is accomplished by configuring 'activation' entries, which are key-value pairs of strings. The value is then used in conjunction with the server's private key to create a response IP address. This response is created using a one-way hash function and cannot be inverted. The hash is case sensitive and is defined as

MD5(subdomain_label + activation_value + private_key)

RESPONSE ENCODING

When the server receives a query from a client it will decode, decrypt, check the results, and process the client data. In particular, a properly formatted client communication must contain two or three labels, as described earlier in the section on query encoding. The server will then assemble a response to the client containing one or more commands. While it can return either IPv4 (A) or IPv6 (AAAA) queries, we will limit our description to IPv4 (A) queries for simplicity.

The server response is an encrypted binary string that is then encoded into one or more A records.[13] The process for this encoding is shown in Figure 3 below. The maximum number of bytes in the response is 64, which are encoded in 3-byte segments, resulting in a maximum of 22 IPv4 addresses in the response.

- In the first step, the server calculates the length of the response and prepends this to the response data. It then appends random bytes to create a composite string that is a multiple of 3 bytes in length.[14] We call this composite string the payload.

13 A series of commands is assembled and then encrypted using a shared key and the current nonce prior to encoding, if a key exchange has been completed. Otherwise the server's private key is used, along with the nonce, to encrypt the data with a public key elliptic curve algorithm.

14 In the code, this process is more convoluted but has the same result.

- In the second step, IPv4 addresses are iteratively created from 3-byte segments of the payload. Each IPv4 address is represented by a 32-bit value where bit 0 is the high bit.
- The top 3 bits of each address are random.
- Each segment has an index, which allows the data to be ordered by the client on receipt; this is represented by 5 bits. This index is in bits 3-7 of the result.
- The payload segment is in bits 8-30, which forces the high bit of the payload segment to be the bottom bit of the first octet in the IPv4 address.
- Finally, the least significant bit, bit 31, is a check bit generated on the payload segment. Because of the nature of this checksum, this bit is 1 in 75% of the IPv4 addresses.
- The resulting 32-bit string is interpreted as an IPv4 address and appended to the response.

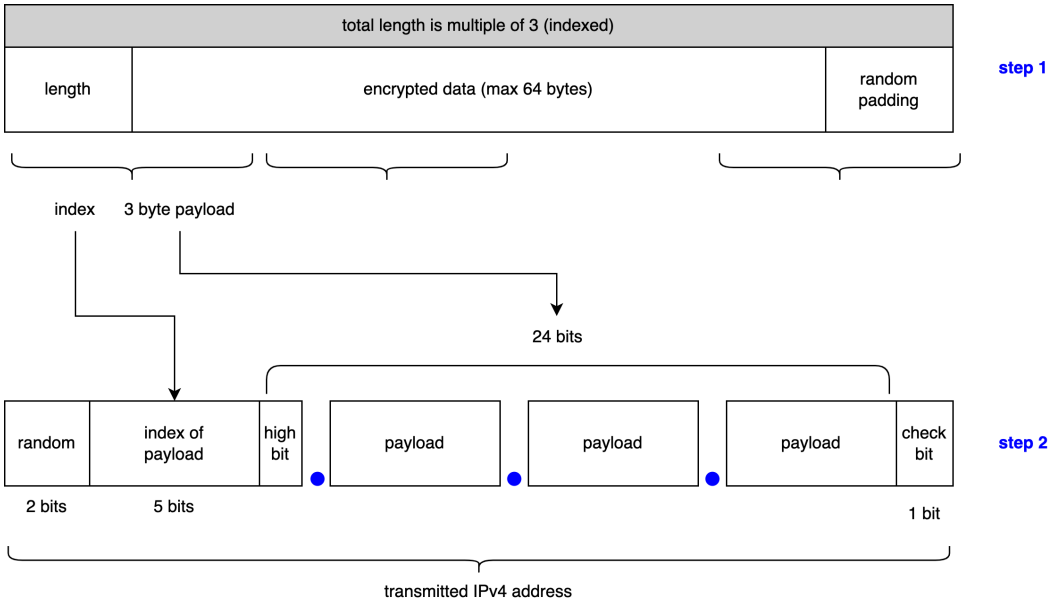

Figure 3. Pupy server encoding of an IPv4 response to a client query. The data is encoded into a series of IPv4 addresses using 3 bytes of the payload in each address.

In our previous paper, we noted that Decoy Dog has a surprising distribution of IPv4 responses. We now know that this was an artifact of Pupy response encoding. The use of three random bits and an incrementing index as the top 7 bits of the first octet in every response, guarantees the resulting IPv4 addresses will be in specific ranges and that those ranges are directly correlated to the number of answers in the response, which itself is determined by the size of the data being transmitted to the client. In particular, the first IP address will always be in the range 64.0.0.0/8, 128.0.0.0/8, or 192.0.0.0/8.

Each time the index is increased, the choices for the first octet of the IP address are shifted by two. Specifically:

- The first IP address will start with 64, 128, or 192 because the index is 0 and the length is a maximum of 64. As a result, only the top 3 bits are set in the first IP address of the response.
- The second IP address will start with 66, 67, 130, 131, 194, or 195 because the index is 1, which adds 2 to the randomly generated 3 top bits, and the top bit of the data payload can be a 0 or a 1.
- The third IP address will start with 68, 69, 132, 133, 196, or 197, etc.

We can see the result of this algorithm for an increasing number of responses in Figure 4 below. In particular, we use a Hilbert map to demonstrate how the first octet of the IP addresses are correlated to the total number of responses for 3, 12, and 15 answers.

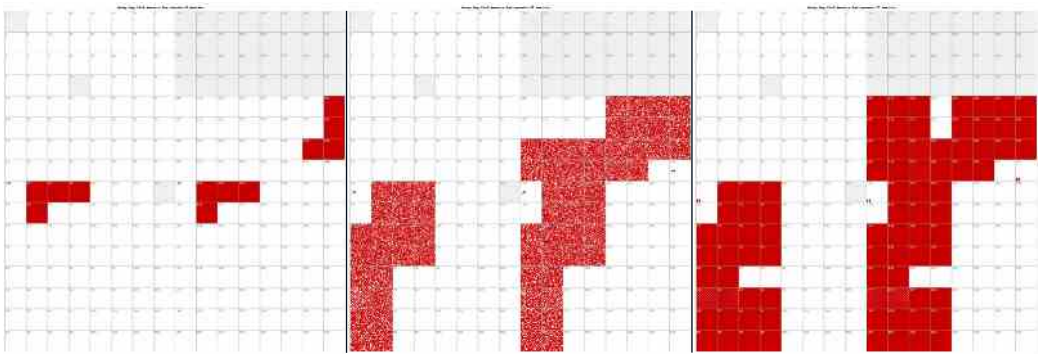

Figure 4. Hilbert maps demonstrating the distribution of IPv4 addresses in Pupy responses containing 3, 12, and 15 answers, respectively.

The structure of the IPv4 addresses allows anyone who observes the full response to reconstruct the transmitted data. While this data is encrypted, the responses can be profiled using the length and time series analysis. This type of analysis can reveal information about the communications as we will see later in this paper.

PASSIVE DATA ANALYSIS

While Pupy communications are strongly encrypted, information necessary to decrypt and track the packets is encoded in a reversible manner. If the DNS queries and responses are collected, they can be analyzed in aggregate to derive information about the Pupy deployment and clients. The passive collection of DNS data, referred to commonly as passive DNS (also as pDNS), happens at many places in the Internet, including corporate resolvers, public recursive resolvers, as well as root and TLD servers. In the sections that follow, we show how passive DNS collection of Pupy queries can be exploited to gain information about the communications.

We can recover a great deal of information about a Pupy controller and its clients from passive DNS. In particular, we can recover

- the approximate number of active clients at any one time,
- the types of exchanges occurring between the server and clients,
- signatures of the deployment, such as the client sleep interval, and
- a timeline of client key exchanges and overall activity.

We used these techniques to analyze traffic from our own server as well as Decoy Dog servers. This allowed us to gain an understanding of how similar Decoy Dog is to Pupy, and how similar the servers are to each other. Ultimately these techniques allowed us to profile every Decoy Dog deployment. Technical details of the methods used are further covered in Appendix C.

PUPY PAYLOAD SIGNATURES

The nature of communications between a client and server can be inferred to a certain degree using passive data analysis. The client vocabulary, meaning the distinct payloads that it can make, is highly restricted: there are only nine types of client communications. Two types share the same payload length, while another type can have multiple lengths. An actor can create custom events in Pupy, potentially creating additional payload length diversity.

The server has a more flexible vocabulary and is able to convey multiple commands in a single DNS response, making it more challenging to profile. However, the vast majority of communications in a Pupy system relate to session initialization, key exchange, and client heartbeats to the server. The server communications are dominated by acknowledgements of client requests, error messages, including the need to establish a new session, and key exchanges.

As a result, signatures for different types of communications can be created using the lengths of underlying payloads of DNS queries and responses. These signatures allow us to separate common maintenance activity from meaningful commands from the server, and isolate the use of custom event types. These can be used to profile the overall behavior of a passively observed Pupy client and server, including Decoy Dog.

In Figure 5 below, we show a heatmap of the payload lengths observed in client queries and server responses in our own Pupy data. While the server lengths have more variation due to command arguments and concatenated commands, the client communications are well defined. For profiling communications, we use the length of the underlying payload, including checksums and node information. As a result, for example, the client acknowledgment (Ack) is 19 bytes long and the server Ack is 6 bytes long. Appendix D contains tables for common client and server payload lengths and their relationship to commands.

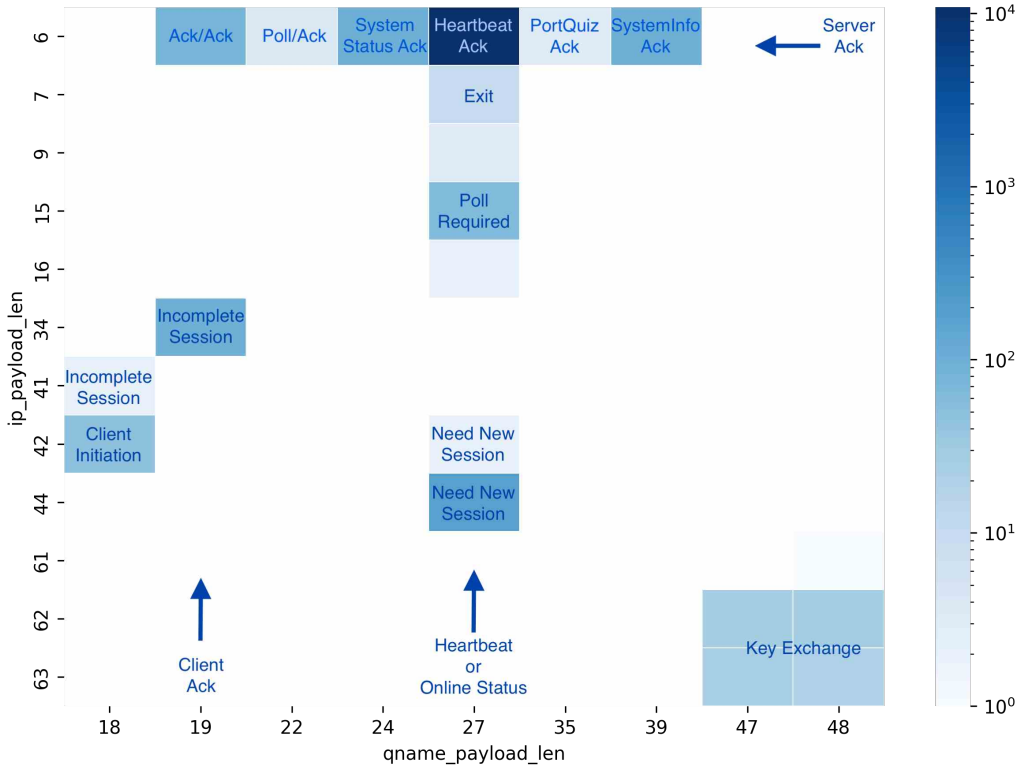

Figure 5. An annotated distribution of common payload length pairs observed in Pupy traffic. The payload is the encrypted data transmitted in either the query or the response. This chart does not include complex DNS C2 commands from the server and cells without annotation are not fully identified. The length is in bytes.

Decoy Dog

Decoy Dog communications were observed not only at Infoblox resolvers, but at many public and commercial resolvers. To better understand Decoy Dog operations and how the toolkit differs from Pupy, we used other passive DNS collections to augment our own. In aggregate, our analysis covers over 15 million DNS events during the time period March 29th, 2022

through June 16th, 2023. Additionally, we actively probed the name servers and compared the passively collected DNS traffic with that generated by our own Pupy client and server.

We used a range of techniques to better understand Decoy Dog and its operations. We also reverse engineered samples found in the public repository VirusTotal, which validated our DNS findings and unveiled other capabilities. In the sections that follow, we will describe our analysis in detail and show the results. The highlights of this work are:

- Decoy Dog is not Pupy, but a large refactor that significantly extends the malware's capabilities and helps ensure persistence on a compromised device.
- It is operated by a handful of actors, who employ distinct TTPs and have responded differently to our April 2023 revelation of the toolkit.
- The number of overall impacted devices is small, with as few as four on a single controller.
- New controllers registered since April 2023 have adapted to mitigate characteristics outlined in our original paper; this includes geofencing mechanisms to limit responses to client IP addresses to certain locations.
- DNS analysis proved a powerful tool not just to detect Decoy Dog but to understand its use and separate it from Pupy, which, combined with selective reverse engineering gives a robust picture of Decoy Dog and the threat it poses.

KEY EXCHANGES

As previously described, a session starts when the key exchange is completed and the SPI value is set. In theory, a single encrypted session can continue indefinitely, but in practice, there are a number of conditions under which the controller will require a new session to be established. Thus, a single running instance of the client typically can have many sessions. Using Pupy payload signatures, we can determine when shared keys were generated between a client and server, and make rough estimates of the number of client initializations, either from a new compromise or a restart of the client, for each controller over time.

Figure 6 below shows the timeline of key exchanges for several Decoy Dog controllers. There are gaps in observed key exchanges for some controllers. The last key exchange for claudfront[.]net was observed in December 2022, although client activity not only continued, but increased in 2023; over 70% of all unique SPI values were first observed in 2023. Similarly, the controller allowlisted[.]net had no key exchanges from December 2022 until after our disclosure in April 2023. Finally, cbox4[.]ignorelist[.]com also shows a long period of time without key exchanges, with a small number occurring directly before the domain stopped operating. We suspect the actors reconfigured the clients to perform the key exchange over a different transport than DNS.

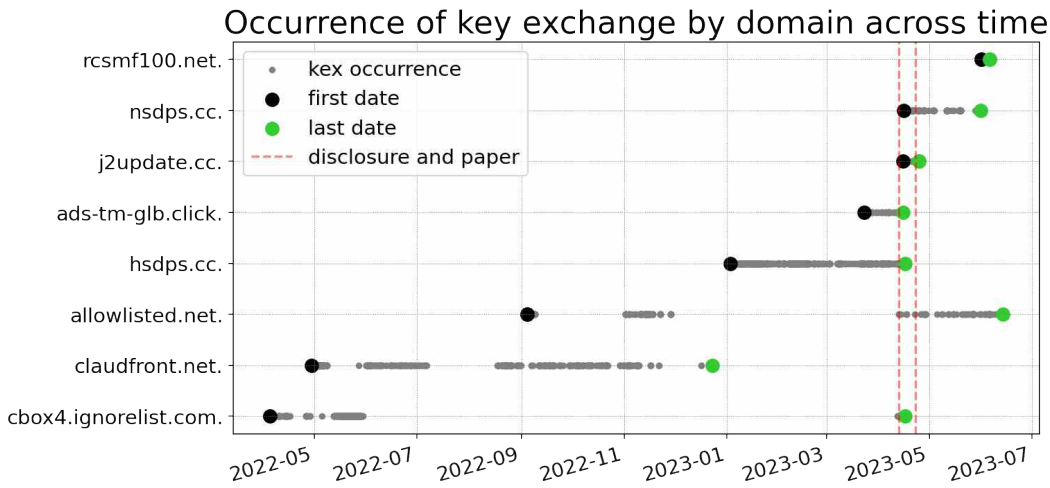

Figure 6. Timeline of observed key exchanges for select Decoy Dog domains.

CLIENT TIMELINES

In addition to the number of overall clients, we wanted to determine how many active clients each controller maintained at a time and how long clients were actively communicating with the server. We used the method of grouping nonce values described in Appendix C. This analysis resulted in key insights into Decoy Dog operations over a long period of time, as demonstrated in graphics that follow. In particular:

- All the controllers are managing a small number of clients at any one time, with some controlling as few as four and all likely under fifty.
- The original domain, cbox4[.]ignorelist[.]com, is one of the larger controllers and exhibits a jump in clients at multiple points in time. It also maintains a small number of very long running clients.
- The second controller to be observed, claudfront[.]net, has a dramatic increase in activity in February 2023.
- The third controller to be observed, allowlisted[.]net, has consistently maintained a small number of simultaneous clients.
- The controllers ads-tm-glb[.]click and hsdps[.]cc transferred clients to new controllers following our disclosure.
- Claudfront[.]net and allowlisted[.]net did not modify operations in response to our disclosure, cbox4[.]ignorelist[.]com ceased operations, and both hsdps[.]cc and ads-tm-glb[.]click transferred clients to new domains.

While it is challenging to estimate the total number of clients over all time, the small number of simultaneously active clients indicates that these operations are highly targeted. It also explains why security vendors did not detect the activity and have not yet found infected devices. The infected clients are present in a very small number of networks, apparently ones that are not able to identify and block the C2 communications in DNS.

In the line chart diagrams that follow, we represent the activity of a single client as a line, and we call this a client thread. The y-axis shows distinct client threads identified by a nonce chain. When a Pupy client is restarted, through a reboot or some other means, a new nonce will be generated and a new thread will be observed. In some diagrams, there are clear breaks in activity that likely indicate client restarts. The x-axis indicates time.

Figure 7 shows client activity for the initial Decoy Dog domain cbox4[.]ignorelist[.]com. The first client thread starts in late-March 2022 and the longest thread lasted nearly a year. We can see that this controller initially had only a few clients, but a change occurred in mid-May 2022 resulting in nearly 40 concurrently active clients. Similar increases in client threads occurred periodically, with the largest monthly increase in August 2022; however, as new client threads began, others ended. Over the entire year of activity, the number of concurrent clients appears to be under 50 at all times. We can also see from Figure 7 that a quarter of the client threads persisted for six months or more, consistent with a sustained operation. All communications ceased following the LinkedIn post and have not been observed again.

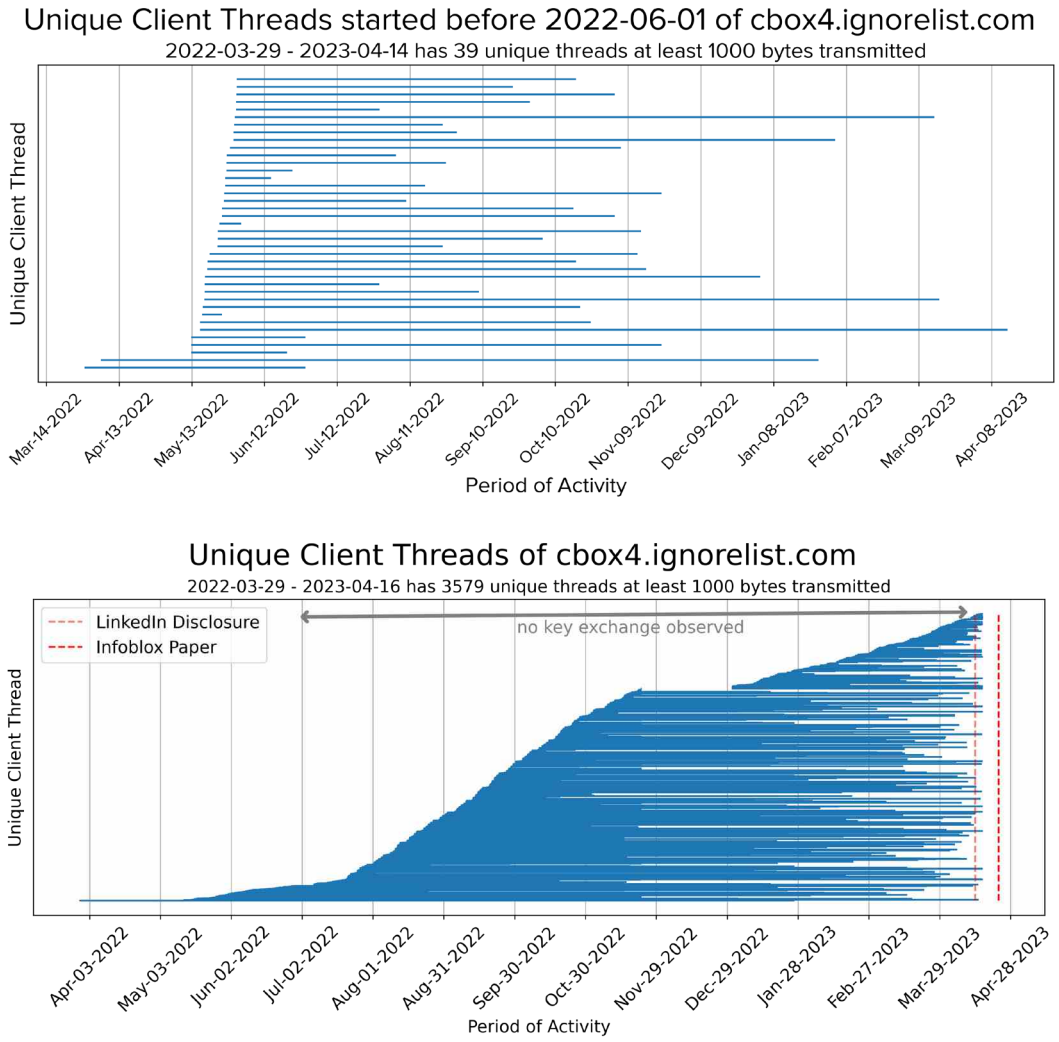

Figure 7. The top figure shows clients that were present prior to June 1, 2022, and the bottom figure shows client threads over time.

The DNS activity for claudfront[.]net, chronologically the second Decoy Dog domain to appear, is quite different from cbox4. As shown in Figure 8 below, there were less than ten simultaneously active clients to this controller until early February 2023. After that time, the number of clients increased substantially, although not to the extent one would expect with a widespread infection. The timing of this increase is shortly before the submission of a binary sample containing the controller domain to VirusTotal on February 13th.[15] Unlike cbox4[.]

15 0375f4b3fe011b35e6575133539441009d015ebecbee78b578c3ed04e0f22568, first submitted 2023-

ignorelist[.]com, there was no notable change to claudfront[.]net queries following our disclosure.

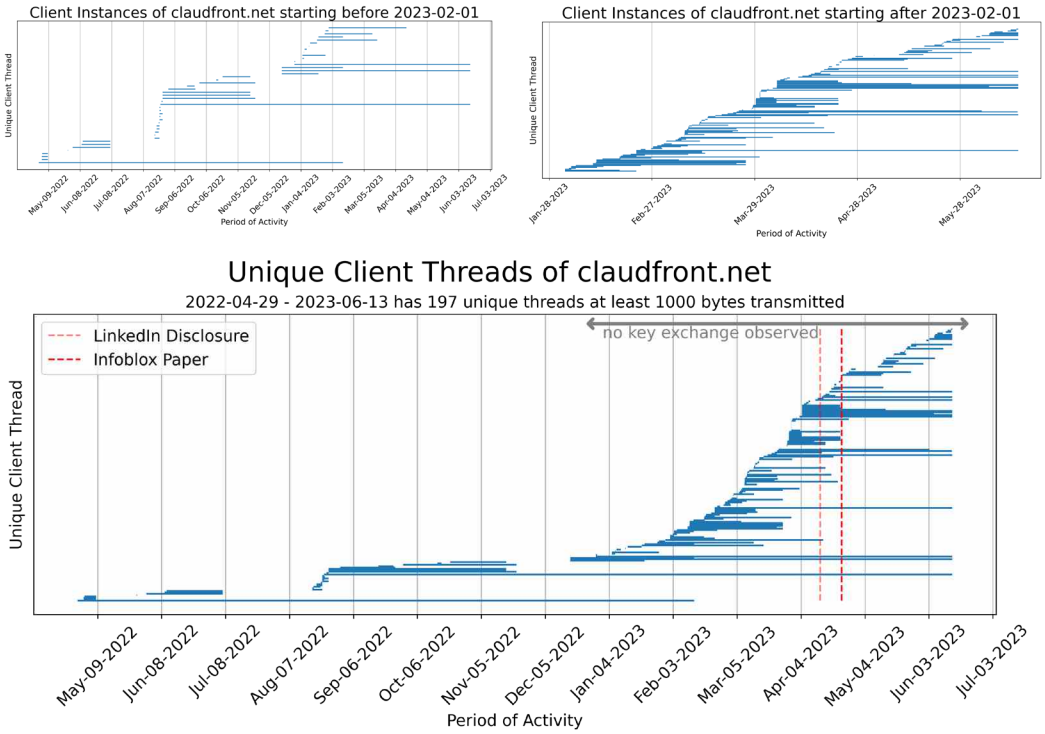

Figure 8. Client threads for claudfront[.]net based on nonce threads over time. There is a significant change in early February 2023, which is zoomed in with separate images showing distinct time periods.

The third domain, allowlisted[.]net, shows another variation in behavior. In this case, the number of clients is consistently small: under ten at any given time. Unlike claudfront[.]net there is no change in February 2023 and there is no known binary sample containing allowlisted[.]net available. There are no key exchanges seen from mid-November 2022 until shortly after our disclosure, which coincides with the sharp ending of client activity and restart of several threads in April 2023, as shown in Figure 9.

02-13 07:39:55 UTC

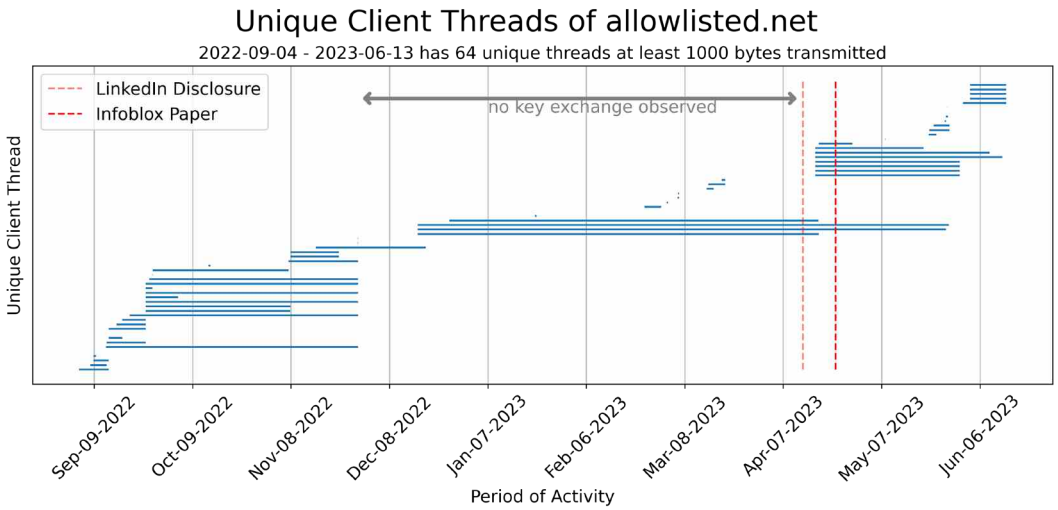

Figure 9. Client threads for allowlisted[.]net. There are a very small number of clients on allowlisted[.]net historically and this has not changed since the disclosure.

Finally, we observed related activity by hsdps[.]cc, nsdps[.]cc, ads-tm-glb[.]click, and j2update[.]cc. The domains hsdps[.]cc and ads-tm-glb[.]click ceased operating after our disclosure on social media, but several of their clients were transferred to nsdps[.]cc and j2update[.]cc, respectively. We discovered this by creating nonce chains across all of the domains over time and identifying threads that began communicating with one controller and ended with another.[16]

The new domains, nsdps[.]cc and j2update[.]cc, were registered less than 48 hours after our social media announcements. We can see from the client thread diagrams that one set of domains ceases activity while others start. The controllers began actively communicating with clients almost immediately thereafter. Following the discovery of client transfer via DNS analysis, we found evidence in binary samples of a command to make this change, as we will describe later.

Figure 10. A timeline comparison of four Decoy Dog controller domains. The controllers hspds[.]cc and ads-tm-glb[.] click cease communications following the Infoblox disclosure and the domains nsdps[.]cc and j2update[.]cc begin communications. We also observed client transfers between these domains.

16 The probability of this occurring at random with a 32-bit random nonce is extremely low, and the number of nonce 'transfers' from one controller to another for these domains was high.

Since our original paper, we have seen additional controllers become active, each with a very small number of clients. The client behavior shown here, in conjunction with the response to our announcement, indicates that the Decoy Dog toolkit is being used by multiple actors.

DECOY DOG PAYLOAD SIGNATURES

We decoded the client and server payload lengths of 15.5 million query responses observed in global pDNS over a 13-month period. We then compared Pupy signatures for client-server payloads with the Decoy Dog observed data to understand the behavior of the servers. While we found that the overall traffic distributions aligned with Pupy, there were definite differences. Decoy Dog clients utilize a larger set of requests, or vocabulary, than is found in default Pupy.

Figure 11 shows the relative distributions of payload length pairs across all Decoy Dog systems. Using our Pupy signatures, as detailed in Appendix D, we can draw a few immediate conclusions:

- More than the nine expected client payloads were present.
- There were server payload lengths we had not observed in our lab.
- The majority of communications related to session maintenance and key exchanges.
- A large percentage of the queries to Decoy Dog servers received an error response and showed variation consistent with scanning by a third party rather than a true client. Most of these occurred after our announcements.

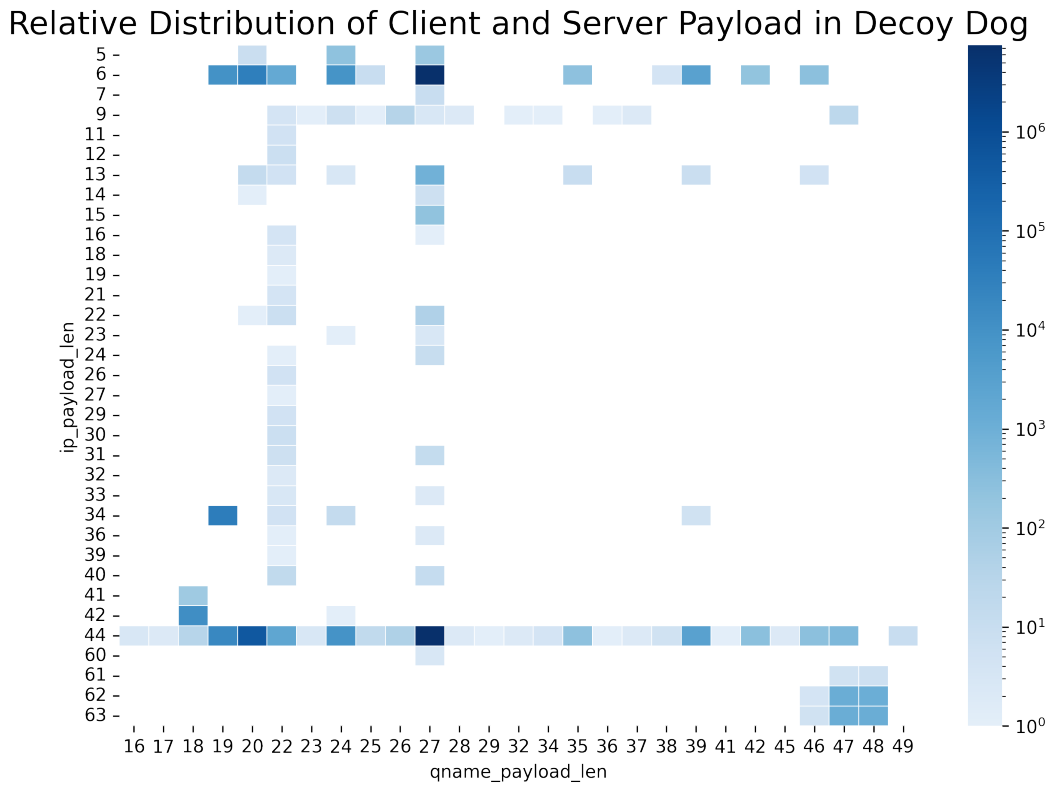

Figure 11. The relative distribution of client and server payload lengths as observed in Decoy Dog communications.

The unique client payloads included lengths 20, 25, 38, 42, and 46. Some of these may be associated with a different key configuration or a change to polling parameters; we can't determine what the communication was, but the variation exists. Additionally, there were additional response payload lengths beyond those observed in Pupy. Most notably, Decoy

Dog has a server payload of 13 bytes which is seen over time in spurts of activity. We are unable to determine what this payload is, but it is consistent with a single command requiring 8 bytes of data to be transmitted to the client. We also saw a number of server responses containing a payload of 5 bytes, another length not observed in our Pupy data, and indicative of a single command requiring no data transfer to the client. Figure 12 below summarizes the unique payload pairs found in Decoy Dog and not seen in our Pupy experiments.

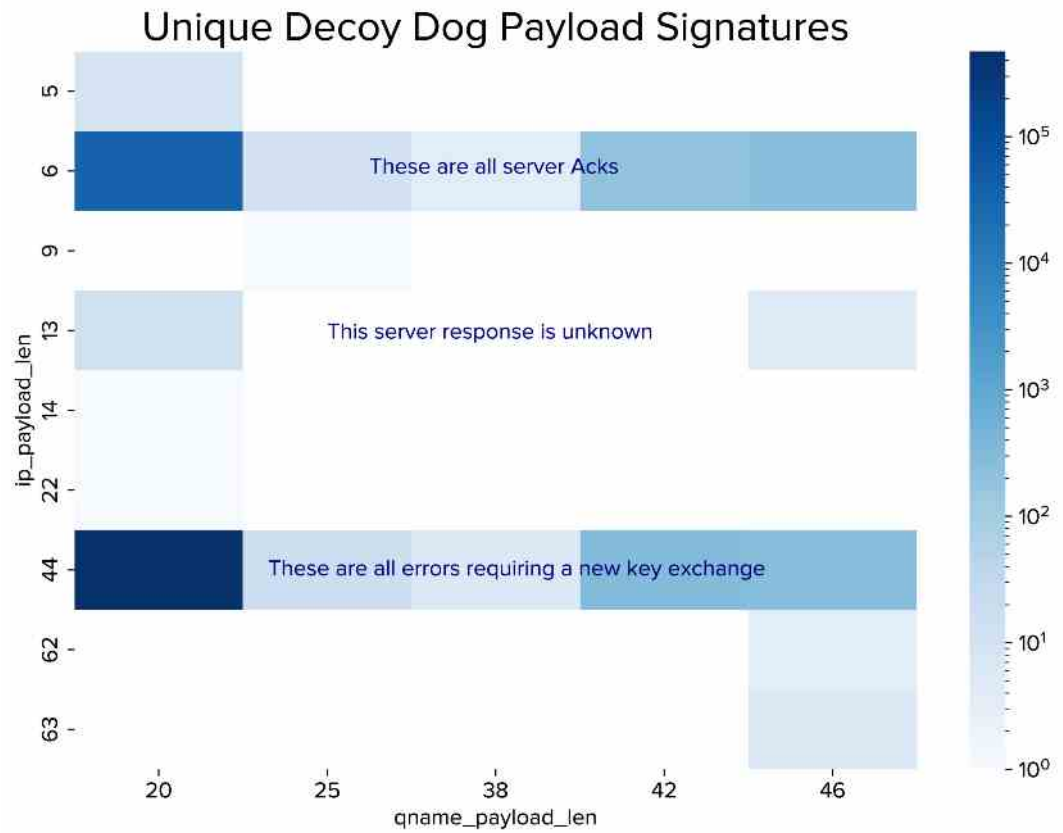

Figure 12. A summary of client-server payload length pairs observed in Decoy Dog and not found in default Pupy communications.

We also used time series to identify changes to the default configurations. Under an established Pupy session, the client will check in every 30 seconds. Using statistical analysis on the variation of client heartbeat queries, we found heartbeat intervals of 2 minutes and 30 minutes in addition to the default 30 seconds.

As a result of this analysis, we were able to understand the nature of communications for each Decoy Dog domain, separating routine maintenance from remote access commands. We were also able to isolate likely customizations of Pupy used across and within subsets of Decoy Dog servers. We found that the vast majority of Decoy Dog traffic is routine acknowledgements and errors, and that the error communications were disproportionate to what we expect to see based on observations of Pupy. We share the results of our investigation into this phenomenon of error responses in the next section.

WILDCARD AND GEOFENCING BEHAVIOR

We reported in our original technical paper that Decoy Dog servers answered replayed DNS queries. This remains perplexing. As we tried to understand when, and how, Decoy Dog would respond to a query that had originally been made days or weeks prior, we uncovered an even more surprising behavior. Several of the Decoy Dog servers not only respond to replays, but they respond to any query that is consistent with Pupy encoding. In DNS, we call this a wildcard response. Where a normal Pupy server would return an NXDOMAIN or SERVFAIL response, the Decoy Dog server typically returns 15 IP addresses.

Figure 13 below shows responses to randomized queries. In this case, we have placed the phrase 'wild' and 'wildcard' within the query name and received 15 answers in response from two different Decoy Dog servers. The responses are different to each query and conform to the Pupy encoding scheme. Through our research, we learned that Decoy Dog is handling almost all errors this way instead of returning the expected NXDOMAIN responses. See Appendix E for additional information on error handling.

```
; <<>> DiG diggui.com <<>> @ns1.rtupdates.net wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22151
;; flags: qr aa rd; QUERY: 1, ANSWER: 15, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. IN A

;; ANSWER SECTION:
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 64.88.80.242
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 131.163.188.250
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 68.221.203.220
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 198.206.187.196
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 200.37.65.250
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 75.195.241.234
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 141.67.92.44
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 142.153.85.81
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 209.92.80.161
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 147.26.100.52
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 213.83.7.105
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 150.143.51.118
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 153.171.88.194
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 219.226.5.44
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.rtupdates.net. 60 IN A 157.111.237.108

;; Query time: 150 msec
;; SERVER: 5.252.179.232#53(5.252.179.232)
;; WHEN: Sat Jun 03 15:29:11 UTC 2023
;; MSG SIZE  rcvd: 321
```

```
; <<>> DiG diggui.com <<>> @ns1.allowlisted.net wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33023
;; flags: qr aa rd; QUERY: 1, ANSWER: 15, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. IN A

;; ANSWER SECTION:
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 64.88.161.73
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 67.179.145.230
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 69.153.193.38
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 71.14.146.226
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 73.22.176.2
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 138.151.231.153
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 141.232.226.212
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 79.241.118.178
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 209.158.29.150
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 147.248.180.89
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 148.158.234.156
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 215.63.12.236
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 153.141.240.250
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 219.18.219.74
wildcard5yt1j46ra2gwossxhw1q9999.2vzxfwild3999999.allowlisted.net. 60 IN A 156.250.150.9

;; Query time: 151 msec
;; SERVER: 83.166.240.52#53(83.166.240.52)
;; WHEN: Sat Jun 03 15:31:15 UTC 2023
;; MSG SIZE  rcvd: 323
```

Figure 13. Wildcard response behavior from two Decoy Dog authoritative servers. In both cases the servers responded with 15 IP addresses consistent with Pupy encoding to the same randomized query containing the strings 'wild' and 'wildcard'.

Even more surprising, some of the Decoy Dog servers also respond differently depending on the IP address of the recursive resolver making the query on behalf of the client. In Figure 14, we show the replay of a query to the Decoy Dog domain nsdps[.]cc, which originally occurred several weeks prior. When making the query via the Yandex public resolvers, we received a response containing 15 IP addresses. We also received 15 IP addresses from the Russian TimeWeb public resolvers. However, of the over thirty public resolvers we tried, no others returned a response. This type of behavior is consistent with geofencing, wherein a server responds to DNS queries based on the geolocation of the IP address. We discovered this behavior in June 2023, and found that some of the servers responded only when we routed DNS queries through Russian IP addresses, while others would respond to any well-formed query from any location. This type of selective response ensures that the controller is only communicating with clients appearing to be in Russia. We know this functionality was added post-disclosure because the controllers had previously resolved queries from the Infoblox recursive resolvers.

```
; <<>> DiG diggui.com <<>> @77.88.8.8 qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42579
;; flags: qr rd ra; QUERY: 1, ANSWER: 15, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. IN A

;; ANSWER SECTION:
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 46 IN A 72.11.125.198
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 36 IN A 203.92.202.218
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 45 IN A 76.74.229.130
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 44 IN A 207.26.86.188
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 31 IN A 80.154.112.164
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 43 IN A 146.160.113.9
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 52 IN A 148.235.159.60
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 41 IN A 151.103.182.130
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 54 IN A 89.76.7.130
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 45 IN A 218.111.60.250
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 42 IN A 93.43.159.18
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 49 IN A 128.88.84.164
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 45 IN A 195.161.207.129
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 51 IN A 68.172.178.156
qnomwvxags5u1pawkzmkllkmuoda9999.abp11i4yk5y1fqyd4tpzm1q9.nsdps.cc. 49 IN A 199.24.240.30

;; Query time: 459 msec
;; SERVER: 77.88.8.8#53(77.88.8.8)
;; WHEN: Tue Jun 20 14:31:11 UTC 2023
;; MSG SIZE  rcvd: 335
```

```
; <<>> DiG diggui.com <<>> @74.82.42.42 hoxlgxq9.yopzgoha3rlp4pdcclosfb63yodq9999.enueh2eluu6uqnjtjpid4lq9.nsdps.cc A
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

```
; <<>> DiG diggui.com <<>> @ns2.nsdps.ns2.name hoxlgxq9.yopzgoha3rlp4pdcclosfb63yodq9999.enueh2eluu6uqnjtjpid4lq9.nsdps.c
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

Figure 14. A comparison of responses to a replayed Decoy Dog query from Yandex public resolvers, Hurricane Electric public resolvers, and the authoritative resolver. These queries were made in succession via a Tor browser. Only the query via Yandex received a response.

When a query is made for a domain name that cannot be decoded using the Pupy default encoding (we added extra characters for this test), the nsdps[.]cc servers return an IP address that essentially is a sinkhole. As shown in Figure 15 below, we altered the query slightly so that it cannot be correctly decoded. In this case, a random IP address within the range 172.0.0.0/8 was returned. Normally Pupy would return an NXDOMAIN response.

```
; <<>> DiG diggui.com <<>> @77.88.8.1 hoxlgxq9.yopzgoha3rlp4pdcclosfb63yodq9999.wildenueh2eluu6uqnjtjpid4lq9.nsdps.cc A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2019
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;hoxlgxq9.yopzgoha3rlp4pdcclosfb63yodq9999.wildenueh2eluu6uqnjtjpid4lq9.nsdps.cc. IN A

;; ANSWER SECTION:
hoxlgxq9.yopzgoha3rlp4pdcclosfb63yodq9999.wildenueh2eluu6uqnjtjpid4lq9.nsdps.cc. 32 IN A 172.67.132.113

;; Query time: 1695 msec
;; SERVER: 77.88.8.1#53(77.88.8.1)
;; WHEN: Tue Jun 20 23:44:46 UTC 2023
;; MSG SIZE  rcvd: 124
```

Figure 15. A query for an invalid Pupy domain name for controller nsdps[.]cc will return a random IP address in the 172.0.0.0/8 range instead of the expected NXDOMAIN response.

Some of this behavior may be explained as an artifact of DNS resolution from the client. When a host is queried in the DNS, some resolvers will attempt to resolve potentially related domain names in order to prepare for potential future queries. For example, a recursive resolver that receives a query for www[.]baddomain[.]com, may attempt to resolve baddomain[.]com in addition to www[.]baddomain[.]com. We saw this behavior at our own Pupy server when routing client queries through some public resolvers.

SINGLE LABEL RESPONSES

By default, Pupy rejects inbound requests to labels that do not match the structure of a client communication or an established ping query. However, as we explained in the "Special Domain Name Handling" section above, the DNS activation request feature allows an actor to configure the Pupy server so that it responds to queries for custom resources. In the global pDNS logs, we identified queries with a single label subdomain. The only such subdomain was 'm' and we hypothesized that resolution of these domains was possible via the activation function. By the nature of the activator hash function, a single static IP address should be returned for these queries. We found this behavior in 4 domains: hsdps[.]cc, nsdps[.]cc, j2update[.]cc, and ads-tm-glb[.]click, and it is another shared characteristic of this set of domains that is not seen in any other controllers. Each of these returned a single IP address; however, instead of the expected static IP address, we found 104 unique addresses in the responses. This seems to indicate a difference in the feature from the default Pupy, but we do not know the purpose.

BINARY SAMPLE ANALYSIS

Following our DNS discoveries, we looked at binary samples available in VirusTotal to determine whether the source of the differences from Pupy was readily apparent. By analyzing the imports and function tables of two Decoy Dog samples, we identified a unique signature specific to Decoy Dog implants that allowed us to discover additional Decoy Dog samples. Reverse engineering these samples further confirmed our findings that Decoy Dog is substantially different from Pupy, and that the most mature code may have been created by a second developer. The client is upgraded to Python 3.8 and includes a number of new transports, upgraded encryption, custom commands, and new DNS functionality. The sample related to one controller, claudfront[.]net, contains features not found in the others. This section describes some of the key findings and the process; more technical details are available in Appendix F. Analytic data related to the binaries will also be added to our GitHub repo.

The first sample was uploaded in September 2022 and the others were uploaded in 2023; three of them following our disclosure. We extracted and compared the configurations of the different Decoy Dog samples, which showed that the encryption keys differ between servers. All the samples that communicated with cbox4[.]ignorelist[.]com contain the same RSA and SSL keys, indicating that the existence of different samples is not related to server key changes. A full list of the decrypted keys can be found in the Github repository detailed in Appendix I. The earliest SSL certificate in the samples was generated on December 26th, 2021, and belongs to cbox4[.]ignorelist[.]com, the first observed controller.

One significant discovery was that Decoy Dog includes custom code in its Pupy client that allows the attackers to send and execute Java modules at runtime by injecting them into a JVM (Java Virtual Machine) thread. This capability does not exist in standard versions of Pupy. This code has been found in all Decoy Dog samples and is identical across all instances. The remaining binary functions in all known Decoy Dog client samples are identical to the functions in base Pupy clients.

The inclusion of Java modules raises more questions than answers. By default, Pupy is already highly capable and supports the use of Python modules out of the box. Expanding these capabilities and writing Python modules is a straightforward process that does not require modifications on the server-side or changes to the client binary. One could easily

create a Python module to execute and run Java modules. In contrast, injecting Java modules at runtime without using jni.h (or the rest of the standard Java/C API) is not a trivial task and requires specialized knowledge. Hence, it is likely that the addition of these Java modules allows attackers to target systems that do not run Python, systems running a privileged or unmonitored Java virtual machine, or scenarios where the attackers aim to avoid leaving evidence on the machine by not creating files.

The clients also have new functionality, which matured over time. The client software is created by marshaling a Python configuration file into a given binary. The configuration file includes settings, all the keys necessary for communications (RSA, SSL certificates, passwords, etc..), and client Python modules. The modules found in the samples, which are unpacked and run by the compromised devices, are vastly different from the publicly available Pupy code.

Extracting and analyzing embedded modules paints a fascinating story of Decoy Dog developments and custom changes. First, a considerable number of Pupy modules have simply been removed from Decoy Dog, possibly because the attackers deemed them useless. Secondly, similar samples exhibit a large number of differences in modules, sometimes with very different capabilities. Third, the large number of changes and the complexity added by new functionalities show considerable development time and resource fine tuning of Pupy. Furthermore, the Pupy codebase and modules were ported from Python 2.7 to Python 3.8, which improved the quality of the code, the stability of memory operations and the compatibility with Windows. The samples include a client version which changes from 3 to 4 over time; the most recent Pupy client available is version 2. A timeline summarizing the submission dates in comparison to code maturity and key features is found in Figure 16 below.

By analyzing the nature and number of changed modules, we were able to identify that from a code maturity perspective, the sample with the hash ad186df91282cf78394ef3bd60f04d859bcacccbcdcbfb620cc73f19ec0cec64 is the earliest publicly available Decoy Dog binary. It communicates with the cbox4[.]ignorelist[.]com name server. Although it shares the most code with Pupy, this sample was not uploaded to VirusTotal until April 27, 2023, several days after our paper was released. However, based on the included SSL certificate, this sample could date as far back as December 2021. The developer added specific polling functionality, an XOR function, new transports, and full support for multithreaded network communications. Interestingly, a number of new modules specifically target Win32, even though all samples so far are Linux libraries. In this executable, the code responsible for handling DNS communications is the same as the default Pupy.

As time went by, samples communicating with cbox4[.]ignorelist[.]com became more complex. Over a series of three samples, an increasing number of communications modules were added, including an entire module to communicate using bidirectional-streams over synchronous HTTP (BOSH), as well as complete rewrites of the SSL, TCP and UDP modules. The actors behind Decoy Dog also added in a number of scripts to port the existing exploit and communication modules to Windows platforms, rewrote the picocmd client responsible for DNS communications, and implemented a number of quality of life and stability improvements to the old code. References to Windows in the code hint toward the existence of an updated Windows client that includes the new Decoy Dog capabilities, although all of the current samples are targeting Linux.

The later versions also include an emergency module that enables a compromised machine to contact a third party DNS server if the malware is being prevented from communicating with the C2 server over an extended period of time. This module uses a DGA to select domains for the client to query within free dynamic DNS services. These versions also allow for bootstrapping to locate the C2 controller, the establishment of beacon domains, and incorporate CNAME queries into the emergency service. Extensive persistence mechanisms, found starting with client version 3, are capabilities most often associated with intelligence operations rather than those conducted by financially motivated actors or red teams.

The most mature code, connecting to the controller claudfront[.]net, includes two new commands called AlterDnsCncDomain and CompromisedNode. As described earlier, we determined via analysis of client nonce values that some of the Decoy Dog actors had transitioned clients to new controllers following our disclosure. Based on the publicly available Pupy source code, we did not see how this was possible without the use of custom commands. It appears likely that the AlterDnsCnCDomain command is the source of those client transitions and therefore the controllers associated with nsdps[.]cc may be using the most advanced code. The large departure of this code from the rest may indicate that a new developer was involved. The code includes version 4 of the client.

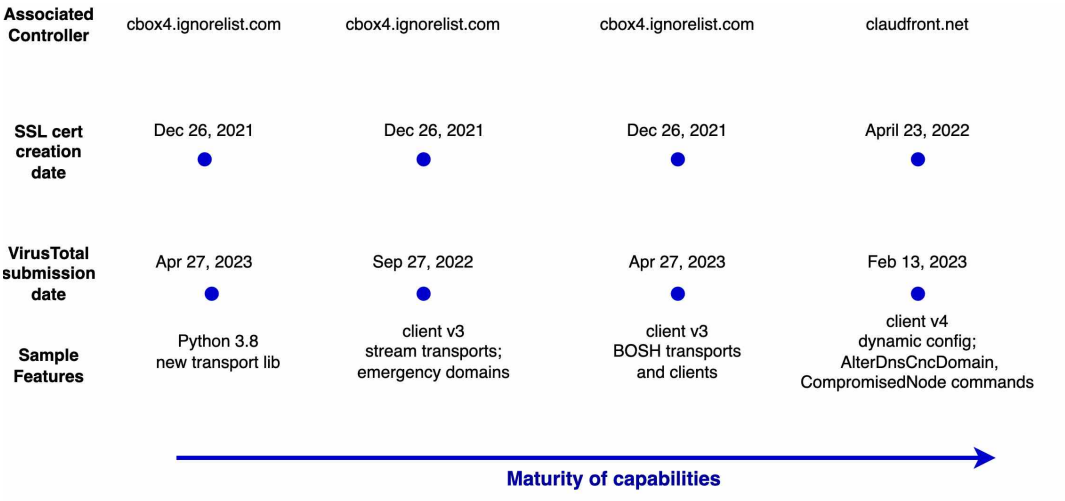

Figure 16. A timeline of VirusTotal Decoy Dog related submissions and maturity of code.

It is worth noting that despite all of Decoy Dog's enhancements, YARA rules developed for more basic versions of Pupy still manage to detect the malware. However, they are unable to detect that the samples deviate substantially from the known code and capabilities. This may lead malware researchers to the false assumption that Decoy Dog samples are just basic Pupy since both types of malware are flagged by the same rule. For this reason, we have included a new YARA rule for Decoy Dog in Appendix G.

COMPARING CONTROLLERS

Infoblox is currently tracking 21 Decoy Dog domains. A number of these have had little or no observable C2 activity and we are not disclosing them at this time. Some controllers changed following our initial disclosure on social media, and the rest changed after we released our first paper. They all responded by either ceasing operations, moving clients to new controllers, or modifying the "ping" behavior we had described in the paper. Some even added geofencing. These responses, in conjunction with other TTPs used, allow us to conclude that there are at least three actors utilizing the toolkit at this time. In Table 1 below, we've grouped a subset of controller domains based on their behavior and similar characteristics.

Group of Domains	Characteristics
cbox4.ignorelist[.]com	• first active domain and likely source of Decoy Dog toolkit • deactivated after disclosure • use of Afraid dynamic DNS • heartbeat interval 30 seconds • not geofenced • at least three distinct client software iterations • first observed by us in late-March 2022, but may have been present as early as December 2021 • client v2 and v3
claudfront[.]net allowlisted[.]net maxpatrol[.]net atlas-upd[.]com	• second set of active controllers, starting in May 2022 • continued operations after disclosure • registered with Namecheap • queries to ping12.<domain> before remote encrypted communication was first seen • changed ping response to a NODATA response • Russian IP hosting • heartbeat interval of 30 seconds • not geofenced • client v3 and v4 • there are some differences between allowlisted[.]net and claudfront[.]net that may indicate different actors
hsps[.]cc nsdps[.]cc j2update[.]cc ads-tm-glb[.]click	• third set of active controllers, starting in December 2022 • moved clients between controllers after disclosure • parked original controllers • heartbeat intervals of 2 minutes and 30 minutes • geofenced after disclosure • changed ping response to a single non-local loopback IP address • use of a single domain label: m • possibly client v4
rcmsf100[.]net	• first observed in June 2023 • shares hosting with allowlisted[.]net • ping response of NODATA • geofenced

Table 1. A comparison of several Decoy Dog controllers.

DECOY DOG IN INFOBLOX NETWORKS

Infoblox has determined that our resolvers were triggered by a security vendor scanner replaying Decoy Dog queries. A combination of the scanner's behavior and Decoy Dog's behavior created the detected signal. Internet scanning has become a prominent business, and scanning now accounts for a large amount of Internet traffic. It is performed by both legitimate and malicious actors. A recent study used a darknet telescope to understand the impact of these scans.[17] While most scanning is limited to port scans, which attempt to identify open ports across the global IP space, there is a wide range of other scanning activities in the environment. For example, there are scanners searching for open directories and open DNS resolvers. Some organizations fully document their scanning activity, but many do not.

"Aggressive scanning" is unauthorized or high volume scanning activity that potentially degrades the performance of a network. It can create a denial of service to a network, or as in the case of Decoy Dog, create false security events.[18] Aggressive scanning benefits the operator at the expense of networks whose owners have not agreed to the activity. In April 2023, security teams for networks with Decoy Dog detections spent significant resources attempting to find the root cause of these DNS queries to ensure their systems were not compromised. These queries were particularly alarming as they originated predominantly from firewalls, and the firewall industry has expressed heightened concerns about attacks on firewalls in recent months.[19]

The way Decoy Dog queries arrived at our resolvers and why they caused a signal akin to a targeted malware C2 beacon is complicated. In order to support defenders' recognition of similar activity, we will provide a brief explanation and an illustration in Figure 17.

For Infoblox to receive Decoy Dog DNS queries, a customer network must have Infoblox as their DNS provider. Additionally, the customer must have security appliances such as firewalls, that have both inbound URL filtering configured and DNS forwarding from that device to our resolvers. These criteria alone are restrictive. When they are met, the following sequence occurs:

- The scanner attempts to retrieve content for the malware C2 directly from an IP address within the network. It does this even though these DNS C2 communications are not web content.
- The security appliance intercepts the request and attempts to resolve the domain name.
- The DNS request is forwarded to Infoblox, which resolves the query and returns the response. If the domain is in a DNS blocklist configured by the customer, it will not return results.
- If the domain being scanned by the vendor is not Decoy Dog or other malware, it will be resolved and, depending on the firewall rules, the content of the website will be returned to the scanner.

17 Aggressive Internet Wide Scanners: Network Impact and Longitudinal Characterization, May 2023, Anand, Dainotti, Sippe, Kallitsis. https://arxiv.org/pdf/2305.07193.pdf

18 https://live.paloaltonetworks.com/t5/general-topics/spurious-hits-from-the-expanse-webcrawler/td-p/447239 , last accessed 2023-06-11

19 https://blog.talosintelligence.com/state-sponsored-campaigns-target-global-network-infrastructure/, last accessed 2023-06-11

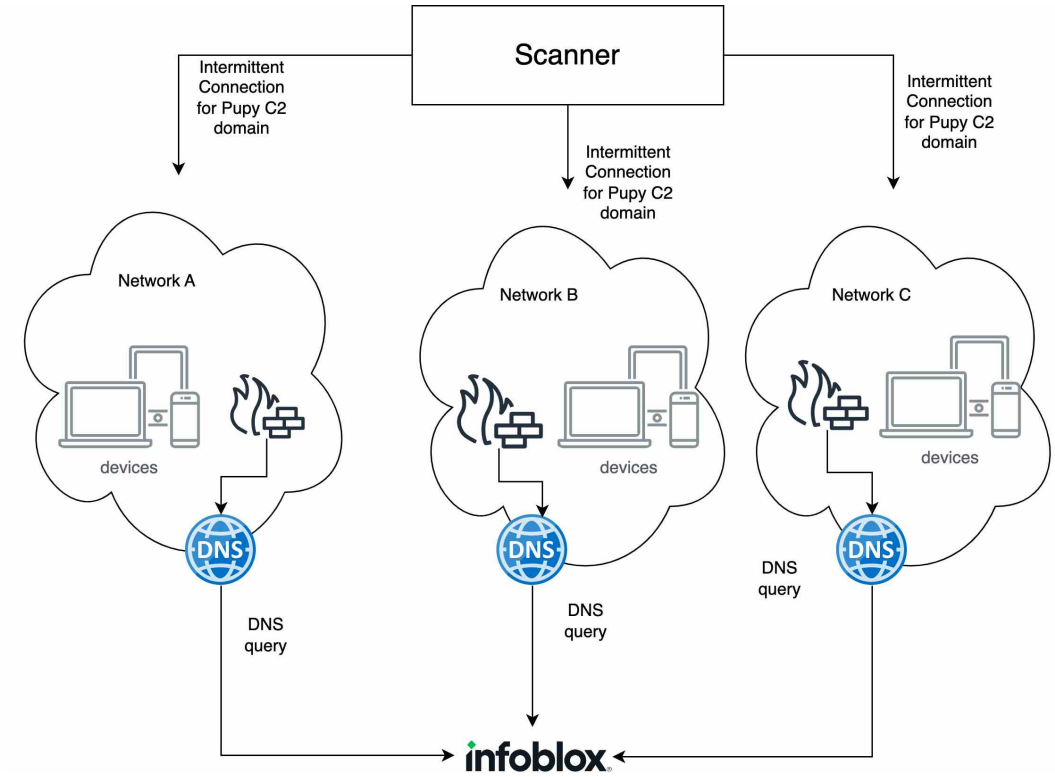

Figure 17. Queries for Decoy Dog DNS C2 domains were made to Infoblox resolvers from devices inside of different networks. These were caused by a commercial scanner and were triggered intermittently.

Infoblox has determined that the vendor performs scans even if the IP address has no known open ports and that it will utilize rare ports in addition to common ports. We do not know how the vendor decides which IP addresses and ports to use. The consequence of indiscriminate aggressive scanning of this nature is that very sensitive devices can appear compromised when they are not. While the vendor appears to broadly and constantly scan for content, Infoblox only observed DNS queries when the above criteria were met. As a result, while the number of scans made by the vendor was very large, which is consistent with aggressive scanning, we only resolved a small number of queries, intermittently over time. This type of configuration also introduces the ability for an actor to perform reconnaissance on certain networks; we describe this in Appendix H.

Infoblox Intelligence keeps historical records of all DNS activity and uses them to create and maintain aggregated statistics of domain activity in our networks and in global DNS. We use these aggregations to identify a wide range of threats, including anomalous behavior that is consistent with malware C2 beacons. In particular, we are looking for domains for which queries, over time, occur in an abnormal number of customer networks, have subdomains consistent with data exfiltration, and that have a low number of queries relative to their expected behavior. To accomplish this, we use statistics of every domain we have observed over multiple years and trillions of DNS queries.

Once discovered, Decoy Dog and other malware C2 beacons look highly suspicious, but detecting them is very challenging. By its nature, DNS traffic is highly variable and contains a large percentage of outliers, meaning domains that are rarely seen and have a domain name structure consistent with data exfiltration. However, DNS exfiltration and beaconing is very rare outside of established pen testing activities. Furthermore, the DNS signature of pen testing is quite distinct from malware C2 beacons. While Decoy Dog proved to be DNS C2 from a variant of the Pupy RAT, a high-volume system, it appeared to be a low-profile beacon because the traffic was injected into the networks by the security vendor.

While the Decoy Dog queries to our resolvers were initiated by the scanner, they were detected because of the unusual behavior of Decoy Dog name servers. As revealed in our previous paper, Decoy Dog name servers responded to repeated queries, although sometimes intermittently. This is inconsistent with Pupy and other encrypted communication protocols. We've now further learned that the controllers respond to any well-formed query. The combined behavior caused our systems to detect an intermittent low-volume beacon. This type of scanning and open DNS forwarding behavior within a network poses additional security risks to an enterprise. By allowing an external party to trigger DNS queries from inside of a network, an attacker can perform reconnaissance against a network. We describe this vulnerability further in Appendix H.

Conclusion

Decoy Dog is clearly a serious threat. A handful of threat actors have been using the toolkit for over a year with the only documented detections resulting from monitoring of DNS data. It is used in operations that are highly targeted and we have only observed its controllers interacting with a very limited number of active clients. Although we have been able to learn much about Decoy Dog, it will remain a serious threat until the vulnerabilities used to establish its foothold are identified and mitigated.

After our initial disclosure of Decoy Dog, threat actors responded in a variety of ways to ensure continued access to victim systems. These responses included changing the DNS response behavior of controllers, adding geofencing restrictions to controllers, and moving clients to new controllers. Despite these adaptations, Infoblox has continued to track them and learn more about Decoy Dog and how it differs from Pupy RAT.

The changes made to Pupy to create Decoy Dog are considerable and are indicative of a sophisticated threat actor. These changes include:

- Pupy was written in Python 2.7. Decoy Dog requires Python 3.8 and includes numerous improvements including Windows compatibility and improved memory operations.
- Pupy has a very limited communications vocabulary. Decoy Dog significantly expands that vocabulary through the addition of multiple new communications modules.
- Decoy Dog responds to replays of previous DNS queries where Pupy does not.
- Pupy does not respond to wildcard DNS requests, but Decoy Dog does. This essentially doubles the number of resolutions seen in passive DNS. In fact, Decoy Dog responds to DNS requests that don't match the structure of valid communication with a client.
- Decoy Dog adds the ability to run arbitrary Java code by injecting it into a JVM thread and adds a number of new methods to maintain persistence on a victim's device.

The sophistication of these changes make the choice for Decoy Dog to respond to any well-crafted query even more curious. Although this decision would appear to be a mistake at first glance, there is likely some yet unknown rationale for it. At present, it is just another mystery of Decoy Dog.

In the future as these mysteries surrounding Decoy Dog are further investigated, defenders should be mindful of the following:

- IPs in both Pupy and Decoy Dog are encrypted data. They don't represent real IPs used for communication. Any connections to real IPs associated with malware are spurious.
- Although the IPs returned in DNS responses aren't meaningful, the DNS queries and responses themselves have meaningful information that can be used for tracking. However, the communication volume is low, meaning a long log history is needed to track detected communications.

- The wildcard responses of the toolkit combined with aggressive scanning by the security vendor may give the appearance of compromise where there is none.
- A YARA rule is available that can detect the Decoy Dog client on a victim machine. It is able to differentiate Decoy Dog from the publicly available version of Pupy.

Decoy Dog was detected solely using DNS threat detection algorithms. To date, there is no public disclosure describing detections of the malware itself and the full scope of its capabilities is not yet known. The fact that it has operated undetected for so long highlights a weakness that occurs when the industry overly relies on malware-based detection. DNS detection and response is currently the only way of defending against Decoy Dog and may be the best option even after victim vulnerabilities and Decoy Dog itself are fully understood.

Indicators

The Decoy Dog indicators related to the controllers and samples described in this report are listed below and available in our open Github repository.[20]

Group of Domains	Characteristics
ads-tm-glb[.]click	Decoy Dog C2 domain
allowlisted[.]net	Decoy Dog C2 domain
atlas-upd[.]com	Decoy Dog C2 domain
cbox4[.]ignorelist[.]com	Decoy Dog C2 domain
claudfront[.]net	Decoy Dog C2 domain
hsdps[.]cc	Decoy Dog C2 domain
j2update[.]cc	Decoy Dog C2 domain
maxpatrol[.]net	Decoy Dog C2 domain
nsdps[.]cc	Decoy Dog C2 domain
rcmsf100[.]net	Decoy Dog C2 domain
13[.]248[.]169[.]48	Decoy Dog C2 name server IP
156[.]154[.]132[.]200	Decoy Dog C2 name server IP
194[.]31[.]55[.]85	Decoy Dog C2 name server IP
5[.]199[.]173[.]4	Decoy Dog C2 name server IP
5[.]252[.]176[.]63	Decoy Dog C2 name server IP
5[.]252[.]176[.]22	Decoy Dog C2 name server IP
5[.]252[.]179[.]18	Decoy Dog C2 name server IP

20 https://github.com/infobloxopen/threat-intelligence/tree/main/cta_indicators

67[.]220[.]81[.]190	Decoy Dog C2 name server IP
69[.]65[.]50[.]194	Decoy Dog C2 name server IP
69[.]65[.]50[.]223	Decoy Dog C2 name server IP
70[.]39[.]97[.]253	Decoy Dog C2 name server IP
83[.]166[.]240[.]52	Decoy Dog C2 name server IP
4996180b2fa1045aab5d36f46983e91dadeebf d4f765d69fa50eba4edf310acf	Decoy Dog binary SHA256
ab8e333ef9bc5c5a7d1ed4cab08335861e150 b0639d3d0ca4c30b7def5cdccde	Decoy Dog binary SHA256
ad186df91282cf78394ef3bd60f04d859bcaccc bcdcbfb620cc73f19ec0cec64	Decoy Dog binary SHA256
6c8f413111f1abfee788dad4ee7cca37e0c259 7cca66d155af958c535faf55cc	Decoy Dog binary SHA256
0375f4b3fe011b35e65751335394410 09d015 ebecbee78b578c3ed04e0f22568	Decoy Dog binary SHA256
6c8f413111f1abfee788dad4ee7cca37e0c259 7cca66d155af958c535faf55cc	Decoy Dog binary SHA256
t1fde0f101c9395f39ecd16430b41041a59107 c73c904087309fb8d0e8d87e0077129f3f	Decoy Dog Telfhash signature[21]

21 https://github.com/trendmicro/telfhash

APPENDIX A: CLIENT COMMAND PROCESSING

Figure 18 illustrates the operating cycle of the client described in the paper. The client repeatedly transitions between sleeping, polling the server, and responding to commands.

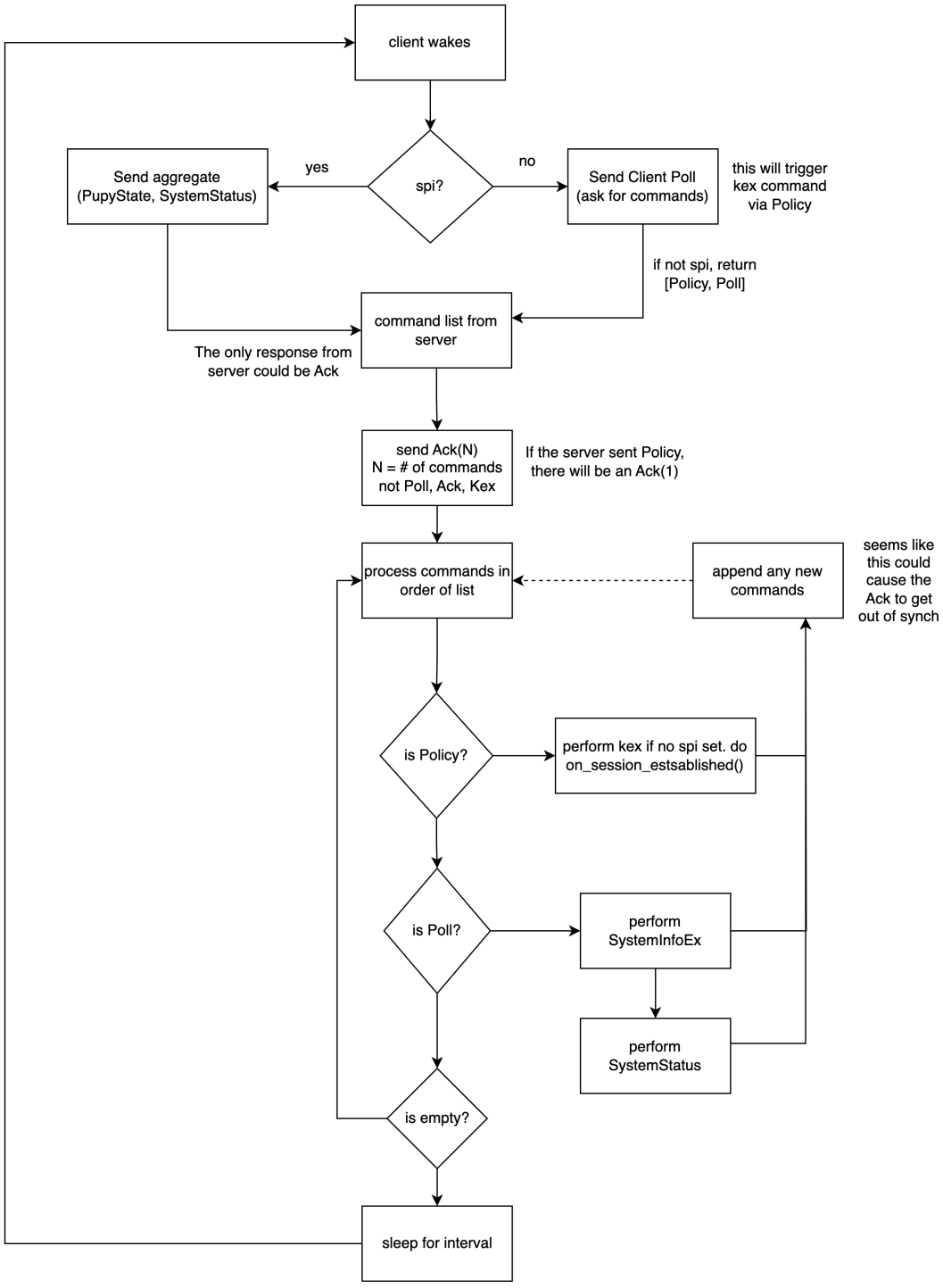

Figure 18. Client workflow.

APPENDIX B: COMMUNICATION PAYLOAD STRUCTURE

The structure of the encrypted payload for client and server is identical, but there are differences in their processing. In particular, the client includes 13 bytes of client information in every query along with the data payload as described earlier.

The client and server both use the term command for the type of information they are transmitting to the receiver. Thus, when the client contacts the server upon waking, it is considered a client command. The commands are registered so that the client or server can apply specific processing to the data. There can be more than one command in a single communication, though from the client this is rare.

The payload that is sent for encoding and transmission has the following form:

- a 4-byte checksum,
- concatenated command packages, containing a 1-byte command identification and a variable command-dependent data part.

The total length of the payload cannot exceed 52 bytes.

APPENDIX C: RECONSTRUCTING CLIENTS FROM PASSIVE DATA

As described earlier, Pupy queries include encrypted data and two encoded values - the nonce and SPI - that provide some security and allow the server to order client communications. The SPI value is specifically used to identify an ongoing session within the server and is present in queries following a successful key exchange. As a result, queries that contain the same SPI and that occur close in time are almost guaranteed to be from the same client. On the other hand, a single client will have many sessions and many SPI values over time, so the SPI alone cannot distinguish clients. Instead, we use the nonce values to separate client communications.

When the client is initialized, it randomly generates a 32-bit nonce value to serve as a starting point. With each packet, this nonce is incremented by the length of the data being transmitted. The server uses the nonce as a minor security check, ensuring that it increases with each query received, but its primary use is to correctly decrypt and interpret the underlying communication. From a series of observed Pupy queries, we can decode these nonce values and compute the next nonce in the series, as shown in Figure 19 below.

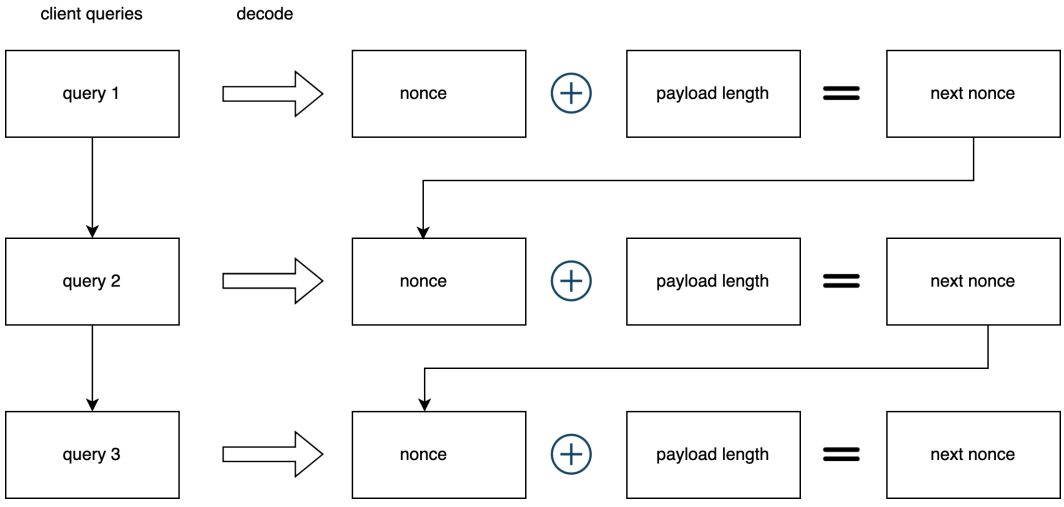

Figure 19. The relationship of nonce values within a series of Pupy queries.

As a result, we can both order queries from a single client and confirm that a series of queries belong to a single client. In passive collection of a Pupy deployment, the queries may originate from many clients and overlap in time. However, we can still separate these observations into separate client activity with a high degree of confidence due to the construction of the nonce. Because the nonce is used to encrypt the payload, the developer used a strong random number generator to create it. This ensures that each client will generate unique starting nonce values.[22] The nonce is recreated each time the client is restarted.

The extra security for the encryption also provides a mechanism to distinguish clients in aggregate observations. To do this, we compute both the encoded nonce and the next nonce value for every query. We then chain the queries together using the sequential nonce values as shown in Figure 20 below. While the underlying data remains encrypted, we can estimate the number of clients and make observations about the length of their activity. Further, we can infer information about the communication itself using the payload lengths and comparing time series across clients.

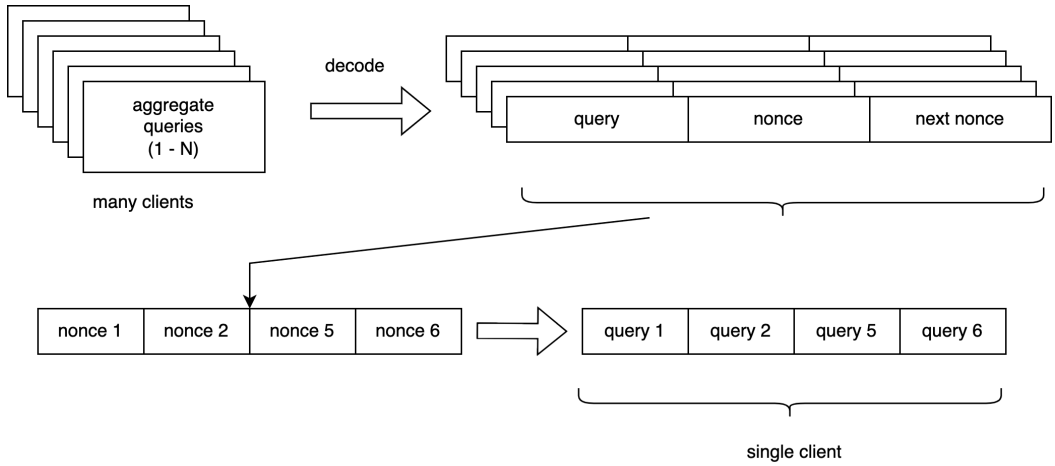

Figure 20. Separating a client thread of queries from an aggregate set of observations using the nonce values.

There are two challenges with this type of exploitation: changes in the DNS resolver of the infected client and packet drops. By default, Pupy uses the client's default DNS resolver and the choice of resolver may not be under the actor's control. If the client roams, it may utilize different recursive resolvers depending on the local environment. In enterprise networks, they may use DNS infrastructure from vendors like Infoblox, in which the DNS queries will be forced over the enterprise recursive resolvers regardless of the client's settings.[23] In addition, when DNS is transported over UDP, packet loss is inevitable. The result is that we are unlikely to observe every query in passive DNS alone, thus creating gaps in the recovered nonce chain that might be significant in size.

We can still reconstruct the client threads however, by taking advantage of the fact that the nonce is a randomly-generated value. The developer used a strong number generator that ensures independent Pupy clients are extremely unlikely to share a nonce value. Moreover, since only 52 bytes of data can be transmitted at a time, and the nonce value increments by the payload, two independently-generated nonce chains are unlikely to overlap. As a result, clients can be separated by ordering nonce values and grouping those that are statistically

22 There are rare probabilities that the same nonce could be generated at the same time by two different clients.

23 Who is Answering My Queries: Understanding and Characterizing Interception of the DNS Resolution Path, Baojun Liu, et al. 2018, https://www.usenix.org/conference/usenixsecurity18/presentation/liu-baojun

similar. A single client has only one nonce at a time, allowing us to estimate the number of active clients at any given time. As we show in the main body of the paper, we found this technique to be very effective in recovering Decoy Dog client query chains.

APPENDIX D: PAYLOAD SIGNATURES

The tables in this section include the payload lengths for specific commands that are commonly observed in Pupy communications. In particular, it provides the encrypted payload length for every standard client command. Server payloads are more flexible than those of the clients; the most common are shown below.

Client Command	Payload Length
Client check-in (initial)	18
Ack	19
Client check-in (rare variant)	22
System status	24
Online status	27
Client check-in (in session)	27
Port quiz	35
System information extended	39
Key exchange	47, 48

Table 2. Client commands and payload lengths.

Server Command	Payload Length
Ack	6
Need session: policy, poll	42
Session incomplete: ack, policy	34
Error: message, policy, poll	44
Need system info: poll	15
Key exchange	62, 63
Exit	7

Table 3. Common server command and payload lengths.

APPENDIX E: ERROR HANDLING

Pupy contains custom handling for a variety of errors that the server may encounter. A domain that does not properly decode or is replayed will result in an NXDOMAIN response from the server. The code snippet below shows the server query processing. If no answer is returned, it will return an NXDOMAIN response.

```
answers = self.process(qtype, qname.stripSuffix(self.domain).idna()[:-1])
klass = SUPPORTED_METHODS[qtype]


if answers:
    for answer in answers:
        reply.add_answer(RR(qname, qtype, rdata=klass(answer), ttl=600))


    if self.edns:
        reply.add_ar(EDNS0(udp_len=512))
else:
    reply.header.rcode = RCODE.NXDOMAIN
```

Figure 21. Pupy server source code that processes client queries.

In Decoy Dog, many client queries that should result in an NXDOMAIN from the server instead return a response, typically 15 IP addresses. This appears to be due to a change in code, wherein Decoy Dog responds to a large variety of possible errors with a DnsCommandServerException internally. The DnsCommandServerException will result in a response to the client, specifying the type of error encountered, and instructing the client to perform a new key exchange followed by transmitting system information. The code block for this error handling is shown below.

```
except DnsCommandServerException as e:
    nonce = e.nonce
    version = e.version
    responses = [e.error, Policy(self.interval, self.kex), Poll()]
    emsg = 'Server Error: {} (v={})'.format(e, version)
    logger.debug(emsg)
    if node:
        node.warning = emsg
```

Figure 22. Pupy server source code that returns an error to the client.

Under normal communications between a Pupy server and client, this type of exception will be raised when there is no active session for a known client. It is also used when the client payload is invalid or has an incorrect checksum. In all other cases, the result is an NXDOMAIN.

APPENDIX F: BINARY SAMPLE ANALYSIS

Pupy Client Binaries

When the Pupy server is first set up, it compiles Pupy library files and creates a static template file for each architecture. These template files are compressed, heavily obfuscated, and stripped of all symbols.

Client binaries can then be manually created using pupygen.py on the server. The script creates C2-specific binaries by marshaling specific configuration bytes (remote host, transport type, debug flag, etc.) into the static template corresponding to the target architecture and file type.

The Pupy client binaries offer a variety of advanced functionalities and are able to target virtually every platform, including Windows, macOS, Linux, Solaris, and Android. In particular, they are able to stay resident in memory, interact with the server, offer full reverse shell capabilities, create fileless copies, etc. When the binary is executed, it will create copies of itself in memory in an effort to avoid detection and make itself more resilient to process-killing techniques.

Example Java Injection Function

Decoy Dog binaries include a number of new functions related to Java injection. This is an example of one of those functions.

```
undefined8 FUN_00105903(void)
{
  int iVar1;
  long lVar2;
  long lVar3;
  long lVar4;
  undefined8 uVar5;
  char *pcVar6;
  undefined local_20 [8];
  undefined8 local_18;

  local_18 = 0;
  if (DAT_005fbda0 == 0) {
     pcVar6 = "JVM was not loaded yet";
  }
  else {
       jvm_address = check_jvm_is_running(0);

       if (jvm_address == 0) {
       return 0;
       }
       classloader_address = find_classloader(lVar2);
       if (classloader_address == 0) {
       pcVar6 = "Preferred classloader was not found";
       }
       else {
       thread_class_address = find_jv_thread(lVar2);
       if (thread_class_address == 0) {
       pcVar6 = "Could not find Thread class";
       }
       else {
       iVar1 =
inject_in_thread(jvm_address,thread_class_address,"currentThread","()Ljava/lang/Thread;",&lo
cal_18);
       if (iVar1 == 0) {
       iVar1 = inject_in_class(jvm_address,local_18,"setContextClassLoader","(Ljava/lang/ClassLoader;)V",
                                          local_20,classloader_address);
       if (iVar1 == 0) {
          uVar5 = (*DAT_005fb748)(1);
          return uVar5;
       }
      pcVar6 = "Iteration failed";
      }
      else {
      pcVar6 = "Could not find current JVM Thread";
      }
  return 0;
```

Figure 23. Partially disassembled Decoy Dog function, trying to find the current running JVM thread for injection.

APPENDIX G: YARA RULE FOR DECOY DOG

The following YARA rule can be used to detect the Decoy Dog samples that we have observed as of July 2023.

```
/*
This rule only detects Decoy Dog. It was adapted from Florian Roth's Pupy Rule
original author : Florian Roth / @neo23x0
original link : https://github.com/Neo23x0/signature-base/blob/master/yara/gen_pupy_rat.yar
*/

/* Rule Set ----------------------------------------------------------------- */
import "elf"
import "pe"

rule DecoyDog_Backdoor {
   meta:
     description = "Detects Decoy Dog backdoor"
     license = "Detection Rule License 1.1 https://github.com/Neo23x0/signature-
base/blob/master/LICENSE"
     author = "Infoblox Inc."
     reference = "https://github.com/n1nj4sec/pupy-binaries"
     date = "2023-07-11"

   strings:
     $x1 = "reflectively inject a dll into a process." fullword ascii
     $x2 = "ld_preload_inject_dll(cmdline, dll_buffer, hook_exit) -> pid" fullword ascii
     $x3 = "LD_PRELOAD=%s HOOK_EXIT=%d CLEANUP=%d exec %s 1>/dev/null 2>/dev/null" fullword ascii
     $x4 = "reflective_inject_dll" fullword ascii
     $x5 = "ld_preload_inject_dll" fullword ascii
     $x6 = "get_pupy_config() -> string" fullword ascii
     $x7 = "[INJECT] inject_dll. OpenProcess failed." fullword ascii
     $x8 = "reflective_inject_dll" fullword ascii
     $x9 = "reflective_inject_dll(pid, dll_buffer, isRemoteProcess64bits)" fullword ascii
     $x10 = "linux_inject_main" fullword ascii
     $x11 = "jvm.PreferredClassLoader" fullword ascii
     $x12 = "jvm.JNIEnv capsule is invalid" fullword ascii

   condition:
     (3 of them and $x11 ) or (3 of them and $x12)
     or (uint16(0) == 0x5a4d and pe.imphash() == "84a69bce2ff6d9f866b7ae63bd70b163" and
$x11) or (elf.telfhash() ==
"t1fde0f101c9395f39ecd16430b41041a59107c73c904087309fb8d0e8d87e0077129f3f")
}
```

Figure 24. YARA rule for detecting Decoy Dog samples.

APPENDIX H: SECURITY VULNERABILITIES EXPOSED

When a device is configured to perform a DNS query on an inbound connection, they allow an external entity to partially control their behavior and resources.[24] In particular, this configuration can provide threat actors a means for reconnaissance, open resolution, and potential participation in a denial of service attack. Because DNS is complex, both vendors and network operators may not understand these risks. While the security appliances that transmitted the queries we detected were intended to have novel features, the use of DNS in those features exposes the network to reconnaissance and potentially other threats.

A device within a network that serves DNS queries to any external entity is known as an open resolver. In some cases, a device may return responses but not fully resolve external DNS queries due to a wide range of circumstances. In either case, such devices pose a risk to the network itself and to the use of the network to amplify distributed denial of service (DDOS)

24 https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA10g000000PLRaCAO, last accessed 2023-06-11

attacks. The risks of open DNS resolvers have been well documented and open resolvers are forbidden under many service contracts, including those of Infoblox, due to these risks.

In the case of Decoy Dog queries, the security appliances were not open resolvers, but still allowed an external party to trigger DNS queries. This kind of configuration cannot be used for an amplification attack, but it can be used by a threat actor for other purposes. For example, a threat actor can perform reconnaissance against a network; shown in Figure 25 below. The actor creates a domain and configures the corresponding name server to log incoming queries. The actor then uses a scanning mechanism to send tailored domain names to connect with the network. In the case of an open resolver search, these might be DNS queries. In the case of Decoy Dog, they were HTTPS connections. In either event, the internal device generates a DNS query that is sent to the actor-controlled name server. The actor is then able to tie the domain name and original IP address to the query it received. While these types of attacks obtain a limited amount of information in each attempt, they are well-established mechanisms to map internal networks for later attack.

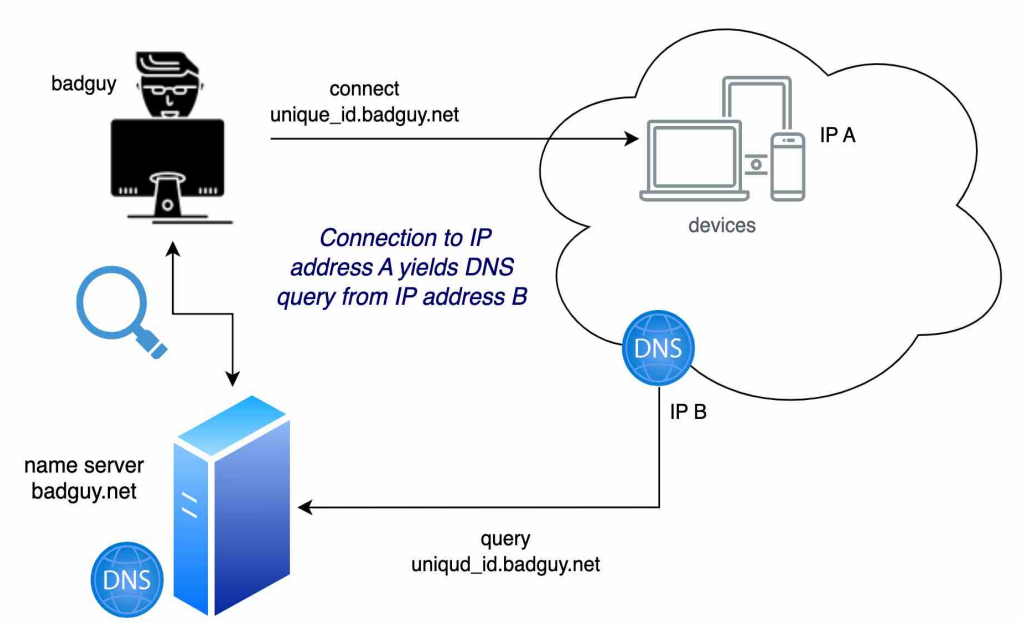

Figure 25. An actor performs reconnaissance on a network by crafting unique domain names that create DNS queries to their name server.

APPENDIX I: RESEARCH DATA

For our research, we established a Pupy server and routed communications between the server and clients through our recursive resolvers. We collected those DNS query logs for our analysis and are making the logs available for research. The data covers several days of varying activity. Most of the time, we controlled the clients by establishing a reverse proxy and commands were sent through SSL. We suspect this is the case for Decoy Dog as well. However, we did exercise all of the available commands via DNS responses from the server. Additionally, there are time periods with multiple clients active simultaneously and numerous client restarts. The scope of activity included should allow for the results described here to be recreated.

The data is available in our public GitHub repository infobloxopen: threat-intelligence.[25] The query-response logs contain A record results and are packaged in a csv file that contains the following fields:

- timestamp: the time of the query in Unix epoch seconds
- query: the fully qualified domain name transmitted in the client query
- response: the set of IP addresses returned by the server
- client_payload_len: the number of payload bytes within the query, including the host information
- server_payload_len: the number of payload bytes within the response

The repo also includes the indicators in this paper; additional indicators are available to defenders upon request as TLP:RED information. Further, we are providing data that resulted from reverse engineering binary samples available on VirusTotal. This includes:

- Embedded configuration parameters for each sample
- Embedded cryptographic keys and password for each sample
  - BIND_PAYLOADS_PASSWORD
  - DCONFIG_PUBLIC_KEY (only for client v4)
  - DNSCNC_PUB_KEY_V2
  - ECPV_RC4_PRIVATE_KEY
  - ECPV_RC4_PUBLIC_KEY
  - SCRAMBLESUIT_PASSWD
  - SIMPLE_RSA_PUB_KEY
  - SIMPLE_RSA_PRIV_KEY
  - SSL_BIND_CERT
  - SSL_BIND_KEY
  - SSL_CA_CERT
  - SSL_CLIENT_CERT
  - SSL_CLIENT_KEY
- A YARA rule and a TELF hash that can detect Decoy Dog binaries

25 https://github.com/infobloxopen/threat-intelligence

RP-072023 v1