DHT Blind Rendezvous for Session Establishment in Network Layer Moving Target Defenses

Christopher Morrell
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
morrell@vt.edu

Reese Moore
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
ram@vt.edu

Randy Marchany
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
marchany@vt.edu

Joseph G. Tront
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
jgtront@vt.edu

ABSTRACT

This paper introduces a new method of securely exchanging information through a moving blind rendezvous by leveraging the size and distributed nature of BitTorrent Mainline Distributed Hash Table (DHT) in order to bootstrap a connection between nodes in a network layer moving target defense (MTD) system. Specifically we demonstrate an implementation of this scheme integrated with an existing MTD implemented in the IPv6 space: the Moving Target IPv6 Defense (MT6D). We show how MT6D peers can use this protocol to exchange configuration information, allowing them to locate other nodes as they move around the Internet, and how they can securely establish connections and related association parameters with no prior knowledge of the other party's network state. We require a minimal amount of pre-shared information between nodes; only that peers have access to public key information. This scheme enables mobility for peers within the MT6D protocol, allows dynamically changing configurations, and allows an MT6D server to scale to supporting many clients without a quadratic explosion in the number of secret keys which need to be maintained.

Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems—*Security and Protection*; E.3 [**Data**]: Data Encryption; C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*

Keywords

Moving Target Defense; Key Agreement; Session Establishment; Mobile Security; Mobile Privacy; IPv6

1. INTRODUCTION

Network based moving target defense systems rely on their ability to logically move around the network by changing addresses in order to avoid detection by malicious parties. The ability to securely exchange keying information is one of the primary weaknesses of these network based moving target defense systems since their inception. When utilizing a moving target defense, two endpoints should not communicate directly with each other using static addresses for any reason. The use of static addresses could allow an attacker an opportunity to monitor the node as it is attempting to hide, thus defeating the security benefits provided by a moving target defense. In past moving target defense system designs, it was required that two hosts exchange some shared key and their predicted subnet manually in order to establish a connection. Exchanging configuration information in this manner caused a number of problems, including a limitation on network mobility, increasing the inherent risk in keeping configuration information static, and the impracticality of attempting to scale this method to a large number of hosts. In this paper, we introduce a scheme that solves these each of these problems.

In today's Internet, it is normal for machines to move between subnets on a fairly regular basis. Due to the hierarchical nature of Internet routing, your moving target defense peer would need to know each subnet that you would be connected to throughout the day in order to establish a connection to you. Statically defined and manually distributed configurations preclude the ability to modify configurations on the fly based on potential compromises that may be detected. In an ideal scenario, the moving target defense configuration could be modified on mid-session due to the movement of an endpoint through the network or in order to work around a compromise. Finally, the manual distribution of configurations does not scale. Not only is there the risk of one of the nodes accidentally sharing their configuration information, but there is also the logistical challenge of distribution and configuration every node that wishes to participate in the moving target network. If it is desired that a moving target defense scenario involve a large number of peers, some automated means of distribution must be created.

There are a number of methods that could be used to dynamically distribute configuration information. An example of this would be to use some form of escrow server. This escrow server would communicate directly with each of the endpoints, thus avoiding the need for the endpoints to communicate directly. Unfortunately, an escrow server would be a single point of failure for the network. Additionally this escrow server would be a single point that some attacker could target, thus making it much easier to find the moving target nodes that wish to hide from said attacker. In this paper we leverage a scheme that uses the distributed nature of the BitTorrent Distributed Hash Table to serve as a form of escrow without becoming a single point through which all moving target endpoints must pass.

Moving Target IPv6 Defense (MT6D) [5] is a specific implementation of a network based moving target defense that relies on statically defined and manually distributed configuration files. We push MT6D beyond its static configuration limitations by applying a recently described blind rendezvous methodology which provides mobility, dynamic configuration, and scaling without the need to share anything beyond a public key. This new scheme relies on the size, resilience, and anonymous nature of the BitTorrent Distributed Hash Table to improve security, privacy, and anonymity.

We begin this paper by providing requisite background information in order to help the reader understand the technologies upon which we build our improvements to MT6D and follow with a brief section that discusses related work. We then describe the modifications that were made to the MT6D protocol in order to support our key distribution scheme and discuss the results of our implementation of this scheme. Finally, we conclude with a discussion of our future work before concluding.

2. BACKGROUND

2.1 Internet Protocol version 6

Internet Protocol version 6 (IPv6) is the 128-bit replacement to the aging 32-bit IPv4 standard. IPv6 was originally published as part of RFC 2460 [3] in December of 1998, when it was realized that IPv4 would not be able to support the number of IP addresses that would be required as the Internet continued to grow. The biggest benefit provided by the move to IPv6 is the immense growth in the number of addresses available as we move from 32 bits to 128 bits. As a simple demonstration of the vast quantity of addresses in IPv6, there are sufficient addresses that one could assign more that 667,000 addresses per square nanometer on the surface of Earth, including the oceans. To truly bring this number to scale, realize that the area of the tip of an average human hair is approximately 2 billion square nanometers. For direct comparison, we should see how this compares to the IPv4 addressing scheme that uses only 32 bits. Instead of addressing nanometers as with IPv6, we will address kilometers of the surface of the Earth. In IPv4, each square kilometer on the surface of the planet gets only 8 addresses. We use this extreme difference in size to try to communicate an understanding of the immense space gained by simply adding 96 bits to our IP addresses.

In IPv6, the address is generally split into three pieces, representing different parts or layers of the network. The first 48 bits are used to represent the routing prefix, the next 16 bits represent the subnet identification, and the last 64 bits, also known as the interface identifier (IID), represent the host. This means that on any subnet, there are approximately 2^{64} or $1.845 \cdot 10^{19}$ distinct IIDs available. In stark contrast, there are only a total of 2^{32} or approximately 4 billion addresses available in the entirety of the IPv4 address space. Due to the limited address space in IPv4, assigning addresses requires a great deal of planning and close control from network administrators in order to ensure efficient usage. In fact, the IPv4 addressed Internet relies very heavily on RFC 1918 [14] private addresses in order to function. These private addresses permit a large number of addresses to masquerade as a single address. In contrast, it is completely feasible in IPv6 to permit a client to simply select their own IID, since the probability of a collision is almost zero. In fact, according to the birthday paradox, it would take approximately $2^{64/2}$ or just over 4 billion attempts to have a greater than 50% likelihood of a collision inside of a 64 bit IPv6 subnet. This fact permits the majority of IPv6 networks to rely on IPv6 Stateless Address Autoconfiguration (SLAAC) in order to assign addresses to their clients.

SLAAC is an automatic addressing scheme that was formalized in RFC 2462 [17] and updated in RFC 4862 [18] that provides a machine with the ability to generate its own IPv6 address. In SLAAC, clients ask their local router for the routing prefix and subnet id portions of the address by sending a router solicitation message to the nearest router. The router responds with a router advertisement message which gives the client the first half of their IPv6 address. The host then generates its own IID and appends it to the prefix and subnet id that were received from the router. The host then assigns the newly calculated address to its network interface. IID generation algorithms vary, but are usually based on adding 16 fixed bits into the middle of the network interface card's 48-bit Media Access Control (MAC) address. Since a machine will likely use the same method each time it generates an IID, it is probable that the IID will be the same for a given interface on a machine. SLAAC is able to avoid collisions within a subnet by relying on the fact that there is a very small probability of two machines having the same MAC address on the same subnet at the same time.

2.2 Moving Target IPv6 Defense

The Moving Target IPv6 Defense (MT6D) was originally proposed by Dunlop, et al. in the 2011 paper, *MT6D: A Moving Target IPv6 Defense* [5]. MT6D is a network layer moving target defense that leverages the enormity of the IPv6 address space and the fact that nodes typically self-assign addresses within a subnet to allow nodes to rapidly and repeatedly change their network address without disrupting any ongoing network connections. As described in the previous section, SLAAC avoids collisions due to the number of addresses in the 48-bit MAC address space and the fact that MAC addresses were designed to be unique.

MT6D improves on this statistic by randomly generating all 64 bits of the IID, ensuring that there is an almost zero probability that MT6D generated addresses will collide with other nodes on the network.

MT6D algorithmically generates new IPv6 Interface Identifiers (IID) based on the node's SLAAC generated or statically assigned IID, a secret key, and the current time. Nodes must also be aware of the IPv6 subnet in which the peer with whom they wish to communicate is located. The process for generating that MT6D address follows:

$$\text{IID}'_{x(t)} = H(\text{IID}_x || K_s || t_i)_{0\to 63} \tag{1a}$$

$$\text{IPv6-Addr}_{x(t)} = \text{IPv6-Subnet}_x || \text{IID}'_{x(t)} \tag{1b}$$

The MT6D IID for time slot t_i is generated above in equation 1a by using a cryptographically secure one-way function such as SHA256 to hash the base IID, a shared secret key K_s, and the time slot. The first 64 bits of the digest are then used as the MT6D generated IID. The MT6D address for the time slot is then simply calculated by replacing the base IID with the MT6D IID and leaving the subnet intact, so that the network is able to route packets to the node.

The length of the time period is a network parameter which is prearranged before the system goes into use. The shorter the time period the more resources are utilized for binding an unbinding addresses but the system is more agile and more resilient to attackers monitoring the network traffic. Conversely, a longer rotation period utilizes fewer resources but does create a larger attack window. MT6D uses a default time period of three seconds between address changes. In order to increase the reliability of the network connection in the presence of higher latency and clock drift MT6D will maintain three address: the previous window address, the current window address, and the next window address.

MT6D encapsulates all traffic which is transiting its tunnel inside of a UDP datagram in addition to an MT6D addressed IPv6 packet. Many network firewalls and Internet middleboxes will reject protocols which they aren't able to recognize. Because of this, an outsider observing network traffic would see many different IPv6 nodes exchanging UDP packets, however the applications running on the MT6D nodes may be exchanging packets with some other arbitrary protocols such as ICMP or TCP. This encapsulation method allows unmodified applications to take advantage of the security advantages provided by the MT6D association while maintaining ongoing connections, oblivious to the changing network addresses.

The rapidly changing network addresses allow a machine to logically move throughout a subnetwork while maintaining connectivity with those "good" nodes that it wishes to communicate with. This logical movement presents a challenge for a would-be attacker, and forces a paradigm shift in the attacker's tactics. While traditional attack methods follow phases of reconnaissance, attack, exploit, and clean-up, MT6D forces the adversary to commit all of their resources to the reconnaissance phase. The enormity of the IPv6 address space, even within a single subnet, makes it incredibly unlikely that an adversary will be able to locate and continually communicate with the target machine simply through random chance. Additionally, MT6D is not intended to be deployed as the only line of defense, but instead as a layer of a defense-in-depth strategy with other, more traditional, layers providing additional security to the protected machine.

As described in [5], MT6D was designed as a peer-to-peer tunnel between two nodes on the network. Prior to deployment, these nodes must exchange configuration information including the pre-shared key, rotation period, and crucially the subnet that the other node is located in. In its original implementation, this configuration information was exchanged manually and stored in static configuration files on each host. The use of these static configuration files present several issues as the size of the MT6D network grows. As MT6D transitions from supporting only peer-to-peer networks to the client-server networks, the number of configuration files that must be distributed will increase linearly with n in a network of n clients. MT6D could also transition to support a multi-node paradigm, where supporting n clients requires a server to distribute $n^2 - n$ configuration files so that each of the clients can establish communication with each other in a mesh. It is easy to see how this distribution and configuration can quickly become an impossible task to scale to. Further, with a static configuration, clients are required to stay within their subnet or signal to the server that they will be moving between networks. This presents a problem for mobile nodes which may be rapidly moving between different networks, even during an ongoing session. In this work we show how a new technique based on securely exchanging information though a Distributed Hash Table (DHT) can be used to alleviate these problems and bootstrap connections for this and other moving target defenses.

2.3 Distributed Hash Table Blind Rendezvous

Utilizing the BitTorrent DHT for Blind Rendezvous and Information Exchange [11], by Moore, et al., presents a new method by which peers can securely exchange small (less than 256 bytes) amounts of information utilizing the BitTorrent Mainline Distributed Hash Table (DHT). The scheme uses a modification of the previously described MT6D algorithm to generate 160 bit DHT descriptors from a shared secret generated through elliptic curve Diffie-Hellman (ECDH) and the current time. Information can then be stored in the DHT though several means. As the current time is part of the descriptor generation step, the resulting logical location in the DHT is likely going to be very different between messages, providing a level of resilience to the messages exchanged through this scheme.

The BitTorrent Mainline DHT is a Kademlia [9] based DHT used by millions [10] of BitTorrent peers to locate other peers downloading the same torrent without the need for a centralized tracker [7]. The immense size of the Mainline DHT provides excellent resilience against attackers when users adopt a moving target defense approach to publishing information. By randomly changing the location to which data is published in the DHT, the user can avoid detection by becoming unpredictable in their use of the DHT. Additionally, the large size of the DHT provides some level of cover traffic to nodes which are using the network for non-BitTorrent information exchanges. Unlike the IPv6 network which uses a hierarchical addressing scheme, the DHT address space is flat, allowing the scheme to generate the full 160 bit descriptor using a modified MT6D algorithm.

The scheme requires that the peers who wish to exchange messages generate public/private keypairs (X, x) and (Y, y)

and securely distribute the public component of their keypair. This can be achieved through any key distribution mechanism, the public component may be shared between multiple other parties, and does not need to remain secret to an outsider. Once the keys are distributed, both parties may generate the same DHT descriptor d for the current time t as follows:

$$s = Y \cdot x = X \cdot y \tag{2a}$$
$$T = t - (t \mod n) \tag{2b}$$
$$d_T = H(s||T)_{0\to 159} \tag{2c}$$

The first step in the algorithm (equation 2a) is the ECDH step, where the peers generate s, a shared secret, based on their private key and the other party's public key. The second step (equation 2b) generates the current time period T from the current time t and the length of the time periods n, an agreed upon system parameter which defaults to 600 seconds. Finally, in equation 2c, the descriptor for the current time period d_T is generated by using a cryptographically strong one-way function such as SHA256 to hash the concatenation of the shared secret key and the time period. The first 160 bits of the digest are used as the descriptor for publications and retrievals in the DHT.

In this work, we leverage this scheme to allow for improved configuration distribution in MT6D. No longer do MT6D peers need to pre-arrange all information about their association, including their subnet prefix; instead they can simply exchange public key information and the MT6D client securely acquires the configuration information at run time. This allows for increased security by permitting session keys to be short lived as well as permits MT6D peers to move between subnets without requiring a reconfiguration of both nodes. This enables mobility in the MT6D system.

3. RELATED WORK

3.1 Scaling MT6D

In its current implementation, MT6D is designed to support only peer to peer communication. This paper is one step towards providing MT6D the capability to scale, while the work presented in [12] is another. In that paper, we focus on identifying the upper limits of the number of clients supportable by a single MT6D server based on hardware and operating system capabilities.

In order to understand what is possible to do with MT6D given the capabilities provided in the modern Linux kernel using standard commodity hardware, we must first determine the number of addresses that a machine can bind to its network interface. This helps us to determine how many clients we can support with a single server when the server is configured to use a separate set of MT6D addresses for each client. Additionally, we must determine how quickly we can send data to the server considering the number of addresses that it has currently bound. This will help us to determine the maximum possible rate of data as the number of active clients changes.

We use *netlink* messages to the kernel as the means of adding and removing IPv6 addresses from the interface, and we use *BSD sockets* in order to receive and transmit data across the network. This work continues to be pushed forward by exploring different ways to bind addresses, unbind addresses, and send and receive data. There is the possibility that we could bypass the utilities that are provided by the operating system and reach directly into the data structures that hold IP addresses and manipulate them manually. Of course, there is the potential for great risk in using this method, and more research is required to determine if it is feasible. We must determine what the risks are and if those risks provide us with some performance improvement over the methods that we are currently using. In regards to sending and receiving data, it may be possible to use zero copy networking [2, 15], where a server is permitted to bypass the kernel's network stack all the way to the device driver, reducing the reliance on the kernel's processing, reducing the need for multiple context switches, and reducing the number of copies that are required when a packet is moved from the hardware to the server.

The work presented in this paper parallels very closely the work towards scaling MT6D due to the fact that creating the ability for a server to support a large number of clients must also come with some means to securely distribute configuration data.

4. DESIGN

In order to establish an MT6D session, each end point must have the necessary configuration information. As described in [5], MT6D addresses are calculated by hashing a host's EUI-64 IID, a shared session key, and a timestamp with the SHA256 hashing algorithm. The values are concatenated and hashed using the form:

$$IID'_{x(i)} = H[IID_x||K_S||t_i]_{0\to 63} \tag{3}$$

In equation 3, $IID'_{x(i)}$ represents the MT6D IID for host x at time t_i, IID_x represents the EUI-64 generated or statically assigned IID from host x, K_S represents a shared session key, and t_i represents the time at instance i. These 64 bits are then concatenated with the 64 bit network address of host x, resulting in a complete 128 bit MT6D generated address.

While the original implementation of MT6D required a manual exchange of configuration information and the generation of a static configuration file to hold said information, we add the ability to dynamically generate this configuration information and share it over the network in a private but authenticated manner. We leverage the DHT based blind rendezvous scheme described in section 2.3 in order to share configurations. The move towards dynamically generated and automatically distributed configurations helps us to overcome limiting factors inherent in the initial design of MT6D. Our modification to MT6D includes a move from a statically defined symmetric key to a randomly generated key that changes at some pre-determined interval. This modification ensures that a captured key is only valid for a short period of time.

We were also required to make a number of additional modifications to the MT6D protocol in order to overcome several other issues. These modifications are slightly different depending on the role of the endpoint in the conversation. For our purposes, we define the server as the machine that is generating the connection information. We define the client as the machine that is consuming that configuration information and establishing a connection to the server. In the following subsections we describe our modifications to the MT6D scheme.

4.1 Server

It is the server's responsibility to ensure that configuration data is updated and published to the DHT on a periodic basis based on the DHT blind rendezvous scheme. The time period between updates should be based on the specific situation in which MT6D is being implemented. In an environment where there is greater risk to compromise or attack, the scheme should be tuned to rotate configuration information more often. We define this time period, T, as the beginning of the time window during which a specific configuration will be valid. T is of some pre-configured length n. In order to ensure configuration synchronization between hosts, we calculate T using $(now - (now\%n))$. This use of modulo arithmetic in combination with time forces time windows to begin on regular intervals, thus ensuring T is the same between hosts. Using T as a component within the DHT descriptor forces the configuration messages to expire periodically.

In order to publish information according to the DHT based blind rendezvous scheme described in section 2.3, the server must generate a descriptor for time period T, d_T, and a message m. d_t is calculated by the server using equation 4a, where H is a strong hashing algorithm such as SHA256, S_{pri} is the server's private key, C^i_{pub} is the public key for client C^i, and T defines the time period. The server then generates the message, m, by using equation 4b where Pfx_S is the server's IPv6 subnet, K is the MT6D symmetric key, Rot is the address rotation period, and $Nonce$ and MAC are generated by the encryption algorithms to prevent replay attacks and provide message authentication. Notice also that in the generation of the message, we encrypt the configuration data with both the server's private key as a digital signature and the client's public key to ensure that only the client can decrypt the message. The server then publishes message m using descriptor d_T into the DHT for future retrieval by the client.

$$d_T = H(S_{pri} \cdot C^i_{pub})||T)_{0\to 159} \tag{4a}$$

$$m = Nonce||MAC||E_{C^i_{pub}}(E_{S_{pri}}(E_{Nonce}(Pfx_S||K||Rot))) \tag{4b}$$

The original design of MT6D relies on the idea that both endpoints have knowledge of their peer's EUI-64 calculated IID, subnet, pre-shared key, and MT6D rotation time interval. Our goal was to improve the protocol to require that as little data as possible need be known by each endpoint prior to connection. By applying the DHT blind rendezvous scheme to MT6D, we require that no specific network data be known by either end point prior to connection. The blind rendezvous scheme allows us to require that only public keys be shared between end points. In order to use this scheme, a number of modifications to MT6D are required due to the lack of information that peers have before they intend to establish a connection.

Since we no longer require that the server has prior knowledge of the client's EUI-64 IID or subnet, and in fact could not know the specific subnet from which the client would connect, we cannot use the same methods applied in the original design of MT6D and instead must use some values for client subnet and IID that are known to both the server and the client. We solve this problem by using a technique similar to that described in section 2.3 for the generation of the descriptor. We generate the client's seed IID as shown in equation 5 where (S_{pub}, S_{pri}) is the public/private key pair of the server, (C_{pub}, C_{pri}) is the public/private key pair of the client, and K is a key that has been pseudo-randomly generated by the server. Using this method to generate the seed IID ensures that both endpoints of the conversation use the same seed information when calculating MT6D addresses.

$$C_{IID} = H((S_{pri} \cdot C_{pub})||K)_{0\to 63} \tag{5}$$

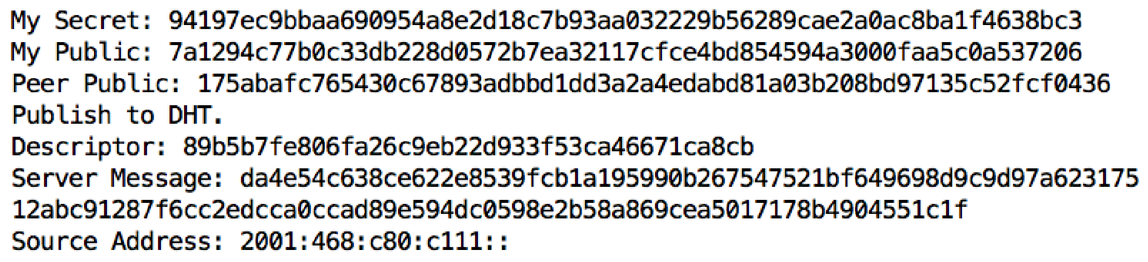
As has been discussed at length, the original design of MT6D required users to establish a symmetric key ahead of time and ensure that the key was in the static configuration file. Using a statically configured key brings with it a number of issues, including insecurity and the inability to scale. Should it happen that some adversary gains access to the key, there is no way to easily generate a new key. This provides some unauthorized party the ability to generate our MT6D addresses, thus allowing them to listen on those addresses, or worse impersonate one of the endpoints. Additionally, it is human nature to select a key that is memorable or pronounceable which increases the risk of potential dictionary or rainbow table attacks against our scheme. Through our modifications to MT6D, we transition from these statically defined, user selected keys to a randomly generated key that expires at fixed intervals and is shared via the BitTorrent DHT. With an established client IID, rotation period, and key, the server now has the ability to calculate MT6D IID's for the client using equation 1a.

4.2 Client

In this scheme, the client connects to the DHT, searches for the descriptor, acquires the message, and finally decrypts the message. The server generates that descriptor in the same was as the server, although it uses its private key, the server's public key, and the equation $H(S_{pub} \cdot C^i_{pri})||T)_{0\to 159}$. The client decrypts the message using its private key, the server's public key, and the nonce that was provided in the message. As long as there is a message somewhere in the DHT that is defined by the calculated descriptor, the client will retrieve it. In the case that the client requests a descriptor on the boundary between two different T windows, it may fail to find a descriptor or when decrypted the client may discover that it is now in time period $T + 1$. In these cases, the client simply searches for a descriptor with the new T value $(T + 1)$. The message includes almost all of the configuration information required to calculate its own MT6D addresses, namely the key K and the rotation period Rot. Additionally, the client calculates its own MT6D seed IID in the same way that the server calculated the client's seed IID with the exception of using its private key and the server's public key as seen below:

$$C_{IID} = H((S_{pub} \cdot C_{pri})||K)_{0\to 63} \tag{6}$$

The client must also use some value as the server's seed IID in the MT6D address generation algorithm. In order to keep the message described in equation 4b small, we choose not to send the server's IID through the DHT, rather we simply force the server's seed IID to :: or all 0's in binary. Since this is different from the client's seed IID, we are able to ensure that addresses generated by MT6D for both the server and the client are different. The use of the SHA256



```
My Secret: 94197ec9bbaa690954a8e2d18c7b93aa032229b56289cae2a0ac8ba1f4638bc3
My Public: 7a1294c77b0c33db228d0572b7ea32117cfce4bd854594a3000faa5c0a537206
Peer Public: 175abafc765430c67893adbbd1dd3a2a4edabd81a03b208bd97135c52fcf0436
Publish to DHT.
Descriptor: 89b5b7fe806fa26c9eb22d933f53ca46671ca8cb
Server Message: da4e54c638ce622e8539fcb1a195990b267547521bf649698d9c9d97a623175
12abc91287f6cc2edcca0ccad89e594dc0598e2b58a869cea5017178b4904551c1f
Source Address: 2001:468:c80:c111::
```

Figure 1: A snippet of server output shows key elements of our key exchange scheme.

[16] hashing algorithm and the fact that it abides by the avalanche effect ensures that addresses will be different at every rotation. The avalanche effect says that if even a single bit of the hashed message is modified, the resulting hash code will be unpredictably different from the original. Since we continue to use time in our address generation algorithm, there is much more than one bit of entropy, ensuring that each address that is generated by the MT6D algorithm is different. The client uses the same key, K, and rotation period, *Rot* for server MT6D IID generation as was used to generate the client's MT6D IID's.

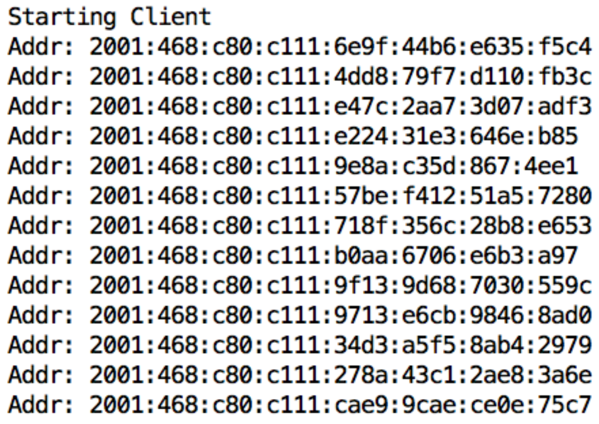
The intent of this scheme is that this MT6D configuration will only be used as a means of conducting an introduction between the two hosts. On the server, this set of addresses will be common among all clients that are known to it. This limitation is brought on in order to save resources on the server, requiring it to maintain only a single set of MT6D addresses for initial contact with clients. During the introduction, the hosts will authenticate each other using their public and private keys and agree on a new session key. This new session key will then result in a new set of MT6D addresses that will only be valid between the specific client and server. The server will then maintain a separate set of MT6D addresses for each of the clients that have established sessions with it and an additional set of addresses for introductions.

5. IMPLEMENTATION RESULTS

In the implementation of our proof of concept MT6D Key Exchange scheme, we leveraged the power of a number of C libraries in order to simplify our code. We pull heavily from libsodium [4], libev [6], and redis [13]. Sodium gives us all of our cryptological functions, including key pair generation, encryption, signing, and decryption. Libev is an event handler that manages our run loops, including address rotation, configuration rotation, and manages an exit handler so that we can exit gracefully regardless of the cause. Redis is a key/value server that we use as an analog to the BitTorrent DHT. Rather than potentially causing issues on the DHT, we chose to keep our data internal to our network for proof of concept and testing.

In figure 1, we show a snippet of output from our server's start up. This figure demonstrates the keys that are available to the server, the descriptor that is generated for publication to the DHT, the message that is published to the DHT, and the base addresses that are used for MT6D calculations.

Figures 2 and 3 show a sample execution of both our server and our client, demonstrating the effect that our configuration rotation has on our client's ability to find the server. For brevity, we only show the addresses that are assigned to the server. In reality, both the client and server are also calculating addresses for the client as well. In figure 3, time T is shown by the addresses in the top portion of the figure, while time $T+1$ is shown below the configuration rotation notification. $T+1$ represents the next time interval for purposes of configuration data aging.

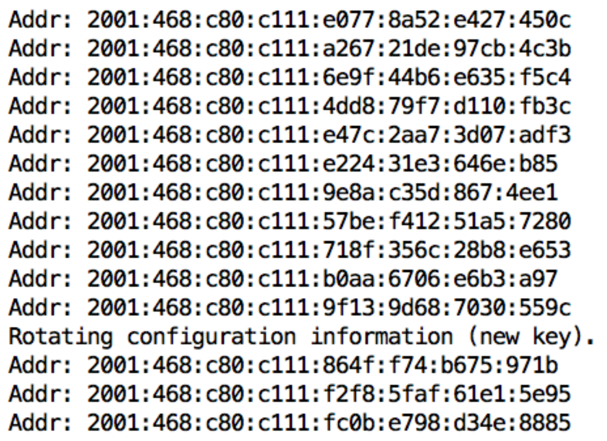


```
Starting Client
Addr: 2001:468:c80:c111:6e9f:44b6:e635:f5c4
Addr: 2001:468:c80:c111:4dd8:79f7:d110:fb3c
Addr: 2001:468:c80:c111:e47c:2aa7:3d07:adf3
Addr: 2001:468:c80:c111:e224:31e3:646e:b85
Addr: 2001:468:c80:c111:9e8a:c35d:867:4ee1
Addr: 2001:468:c80:c111:57be:f412:51a5:7280
Addr: 2001:468:c80:c111:718f:356c:28b8:e653
Addr: 2001:468:c80:c111:b0aa:6706:e6b3:a97
Addr: 2001:468:c80:c111:9f13:9d68:7030:559c
Addr: 2001:468:c80:c111:9713:e6cb:9846:8ad0
Addr: 2001:468:c80:c111:34d3:a5f5:8ab4:2979
Addr: 2001:468:c80:c111:278a:43c1:2ae8:3a6e
Addr: 2001:468:c80:c111:cae9:9cae:ce0e:75c7
```

Figure 2: A sample of the server's MT6D addresses as calculated by the client during time T and $T+1$.

The information in figure 3 becomes more meaningful when compared with the addresses that are displayed in figure 2. The client begins execution during time period T, and quickly pulls configuration information from the DHT and immediately has synchronized addresses with the server. The client's first calculated address matches the third address that the server has calculated. It is apparent that addresses are continually synchronized until the server rotates configuration. At this time, the server begins to use a new session key which the client does not have. After the address that ends with 559c, the client can no longer find the server. If the client simply requests new configuration from the DHT, it will acquire the new session key that the server published for time $T+1$, and will regain the ability to synchronize with the server.

By applying the Blind Rendezvous and Information Exchange in the BitTorrent DHT methodology to MT6D, we have enabled mobility for clients as well as alleviated the problem of key exchange and key distribution. This helps to allow the MT6D network to scale from a purely peer-to-peer tunnel architecture to one where there are many clients communicating with each other and with servers which provide services.



```
Addr: 2001:468:c80:c111:e077:8a52:e427:450c
Addr: 2001:468:c80:c111:a267:21de:97cb:4c3b
Addr: 2001:468:c80:c111:6e9f:44b6:e635:f5c4
Addr: 2001:468:c80:c111:4dd8:79f7:d110:fb3c
Addr: 2001:468:c80:c111:e47c:2aa7:3d07:adf3
Addr: 2001:468:c80:c111:e224:31e3:646e:b85
Addr: 2001:468:c80:c111:9e8a:c35d:867:4ee1
Addr: 2001:468:c80:c111:57be:f412:51a5:7280
Addr: 2001:468:c80:c111:718f:356c:28b8:e653
Addr: 2001:468:c80:c111:b0aa:6706:e6b3:a97
Addr: 2001:468:c80:c111:9f13:9d68:7030:559c
Rotating configuration information (new key).
Addr: 2001:468:c80:c111:864f:f74:b675:971b
Addr: 2001:468:c80:c111:f2f8:5faf:61e1:5e95
Addr: 2001:468:c80:c111:fc0b:e798:d34e:8885
```

Figure 3: A sample of the server's MT6D addresses as calculated by the server during time T and $T + 1$.

6. FUTURE WORK

Applying the Blind Rendezvous Information Exchange in the BitTorrent DHT methodology to MT6D, has enabled mobility for clients, improved the security of an MT6D session, and alleviated the logistical challenges inherent in large scale key exchange and distribution. This improvement helps to allow the MT6D network to scale from a purely peer-to-peer tunnel architecture to one where there are many clients communicating with each other and with servers which provide services. This work is one part of a larger problem that focuses on enabling MT6D to run as a server in a large scale client/server network.

As we discussed in section 5, we did not fully implement our scheme to utilize the BitTorrent DHT. Rather we used a small scale analog in our implementation of a redis key/value storage server. While this demonstrates that the basic structure of our scheme is valid, it does not provide us any information on how our scheme will react at scale. We plan to take the next steps by modifying our implementation to utilize the actual BitTorrent DHT, rather than an analog.

Benchmarking must be completed in order to demonstrate how easily the key exchange mechanism can scale, and what effect it has on the efficiency of the node publishing key exchange information to the DHT for hundreds or thousands of potential clients. In order to complete these benchmark, we will first complete a full scale implementation of our functioning proof of concept. Once fully functional, we intend to discover the limits of how many messages we can successfully publish and retrieve in a set time period. While it is unlikely that we will be able be able to push far enough to cause noticable impact on the DHT, we will begin to reach the limits of our machine that is publishing the messages and the local network that our machine uses to connect to the Internet. This bottleneck will allow us to define the upperbound as to either how many clients we can support, or how frequently we can update configuration information.

In order to scale past this bottleneck, we have considered several options that may help to improve our efficiency. One potential mechanism that may be considered would be to co-opt the two-stage message publications from the DP5 privacy preserving presence protocol [1], where instead of publishing a new DHT message for each client during each window, we would instead publish a short term DHT key to each client over some long window and use that key to publish a single introduction message to the DHT on the shorter window. For n clients and a long-term key length of w DHT publication windows, this reduces the number of DHT messages from nw to $n + w$, potentially saving a large amount of network traffic and other computational resources.

An implementation of an MT6D server making use of the DHT publication scheme introduced here must also be benchmarked in order to demonstrate how well it is able to scale to supporting hundreds or thousands of concurrent clients performing some representative computational task while also maintaining MT6D associations with those clients, many of which may be mobile. A further question for this is to determine whether it makes sense to negotiate distinct MT6D hopping keys for individual clients, or to generate a single hopping key that is used for the entire network and periodically rotated as clients join and leave the network. This question must consider not only performance improvement, but potential security issues that could arise due to sharing MT6D configuration information between multiple peers.

The BitTorrent DHT Blind Rendezvous and Information Exchange mechanism is not specific to MT6D either, and potential future work opportunities include integrating, testing, and analyzing its application to other protocols which could benefit from a secure and resilient mechanism of sending information to other parties without knowledge of their network address.

7. CONCLUSION

We have presented modifications to the Moving Target IPv6 Defense protocol which allow us to leverage a Distributed Hash Table based Blind Rendezvous scheme. By applying this scheme to MT6D, we now have a network based moving target defense system that is mobile, scalable, and increasingly secure. By reducing the amount of data shared between hosts to simply a public key, we simplify the distribution of moving target defense configuration information and reduce the risk of compromising said data. We also avoid a single point of failure that would occur if something similar to a single point escrow server were used to conduct this configuration information exchange.

We have additionally demonstrated an implementation of our ideas through the modification of MT6D, showing that our ideas are in fact feasible. Our proof of concept successfully allows a highly scalable yet secure exchange of moving target defense configuration information between peers. While we have limited our proof of concept in functionality, we believe that moving our implementation forward as discussed in section 6 is entirely possible. As we continue to push this work forward, we believe that we will discover additional methods that will improve the functionality of our moving target defense scheme.

It is important to understand that the scheme introduced in this paper is not to be seen as a replacement for the large number of route information coordination techniques that already leverage the BitTorrent DHT as described in [8]. We rely on those protocols to find information that exists within the DHT, but our focus here is to provide a means to bootstrap a moving target defense connection without requiring two endpoints to communicate directly or through some vulnerable single point of failure.

Network based moving target defense systems are not in the mainstream yet, as there are a significant number of issues to overcome in order to make them easy to deploy, use, and maintain. Our improvements to MT6D bring us one step closer to making moving target defense a reality across the Internet.

8. REFERENCES

[1] N. Borisov, G. Danezis, and I. Goldberg. Dp5: A private presence service. Technical report, Technical Report 2014-10, Centre for Applied Cryptographic Research (CACR), University of Waterloo, 2014.

[2] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed tcp. *Communications Magazine, IEEE*, 39(4):68–74, Apr 2001.

[3] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

[4] F. j. Denis. libsodium. `https://github.com/jedisct1/libsodium`. [Online; accessed 10 February 2015].

[5] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. MT6D: a moving target IPv6 defense. In *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*, pages 1321–1326, 2011.

[6] M. Lehmann. libev. `http://linux.die.net/man/3/ev`. [Online; accessed 4 September 2014].

[7] A. Loewenstern. Bittorrent dht protocol. *BitTorrent BEP*, 5, 2008.

[8] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 7(2):72–93, 2005.

[9] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[10] J. Mittag. Dsn research group-live monitoring. *History*, 9:06, 2014.

[11] R. Moore, C. Morrell, R. Marchany, and J. G. Tront. Utilizing the BitTorrent DHT for blind rendezvous and information exchange. In *Milcom 2015 Track 3 - Cyber Security and Trusted Computing (Milcom 2015 Track 3)*, Tampa, USA, Oct. 2015.

[12] C. Morrell, J. S. Ransbottom, R. Marchany, and J. G. Tront. Scaling ipv6 address bindings in support of a moving target defense. In *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for*, pages 440–445. IEEE, 2014.

[13] Pivotal. redis. `http://redis.io/`. [Online; accessed 10 February 2015].

[14] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), Feb. 1996. Updated by RFC 6761.

[15] J. Song and J. Alves-Foss. Performance review of zero copy techniques. *International Journal of Computer Science and Security (IJCSS)*, 6(4):256, 2012.

[16] N. S. H. Standard. Federal information processing standards publication fips 180-4, 2012.

[17] S. Thomson and T. Narten. IPv6 Stateless Address Autoconfiguration. RFC 2462 (Draft Standard), Dec. 1998. Obsoleted by RFC 4862.

[18] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.