Chhoyhopper: A Moving Target Defense with IPv6

A S M Rizvi* and John Heidemann†
University of Southern California / Information Sciences Institute
Email: *asmrizvi@usc.edu, †johnh@isi.edu

***Abstract*—Services on the public Internet are frequently scanned, then subject to brute-force password attempts and Denial-of-Service (DoS) attacks. We would like to run such services stealthily, where they are available to friends but hidden from adversaries. In this work, we propose a discovery-resistant moving target defense named "Chhoyhopper" that utilizes the vast IPv6 address space to conceal publicly available services. The client meets the server at an IPv6 address that changes in a pattern based on a shared, pre-distributed secret and the time of day. By hopping over a /64 prefix, services cannot be found by active scanners, and passively observed information is useless after two minutes. We demonstrate our system with the two important applications—SSH and HTTPS, and make our system publicly available.**

I. Introduction

Attackers frequently scan for services on the public Internet, then make brute-force password attempts and Denial-of-Service (DoS) attacks. IPv4 scanning has been possible for more than a decade [15] and recent tools allow scanning all of IPv4 in minutes [2], [13]. Mass scanning of IPv4 is done regularly by many parties [38]. Scanning is increasing in IPv6 as well, with evidence of IPv6 address space scanning [9] and development of public lists of responsive IPv6 addresses [23]. Once scanning detects an active service, attackers can carry out brute-force password attacks to get access [31], [4]. Services with static address can also be targeted by DoS attacks [8].

We would like to provide discovery-resistant *stealthy* services on the public Internet, available to friends but hidden from adversaries.

IPv6 adoption has been increasing over the years [6]. As of December 2021, 37% of Google accesses use IPv6 [12], and APNIC shows 29.4% of all global users are capable of using IPv6 [3]. From May 2018 to February 2020, Akamai reports 4x increase in the IPv6 traffic volume [28].

IPv6 provides a huge address space in which we can hide services. Even with clever scanning, when each LAN has 2^{64} addresses (or more), active discovery of services on intentionally obscure addresses is intractable (see §VI-A). With IPv6 prefixes of /48s as the recommended minimum size of publicly routable prefix, [36], and /56s recommended for homes [26], even with a million devices in a home, quintillions of addresses remain unused on every network.

Our insight is that only a discovery-resistant moving target can elude scanners. We describe *Chhoyhopper*[1], using the vast IPv6 address space to conceal publicly available services. The server hops to different IPv6 addresses in a pattern based on a shared, pre-distributed secret and the time-of-day. A client with the shared secret can match this pattern to find the server. As with SSH [33], we target services for small groups where out-of-band sharing of secrets (our hop key, or ssh's per-user keys) is viable; our approach can scale to support millions of such small groups. By hopping over a /64 prefix, any service cannot be found by active scanners, and passively observed information is useless after two minutes. We expect our system to be used by small organizations who want to protect their specific services used by their group from active scanners and brute-force attacks. Since the server hops over addresses, our system provides protection against DDoS attacks targeted to a fixed address.

We make three new contributions: first, we show that IPv6 address hopping can be used to protect existing services (§IV). Prior work suggested daily address changes for IoT devices with new services [19]. We instead propose changing addresses every minute, and show how to apply this approach to existing popular services like SSH and HTTPS. We provide a common hopping design that can be used by multiple services. To the best of our knowledge, this is the first design of a moving target defense for SSH and HTTPS utilizing IPv6. Second, we show how to support web security with TLS by adding support for DNS-based TLS certificates to our core hopping protocol (§IV-F). Finally, we propose a new approach to accommodate long-lived connections in the face of frequent address changes (§IV-D). We use ip6tables rules to retain the existing connections to a fixed internal address but changing NAT rules allow new connections only with the current IPv6 addresses. Our deployment is user friendly, and works similarly like the current client applications (§V).

Availability: Our implementation is freely available at https://ant.isi.edu/software/chhoyhopper/. We provide server module using a Python script for both SSH and HTTPS. Our client implementation has a Python script for SSH, and a browser extension for HTTPS.

II. Background

We next briefly review how IPv6 makes our solution possible. Full details about IPv6 are in its specification [16].

Network and Distributed Systems Security (NDSS) Symposium 2022
24 April - 28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/madweb.2022.23004

[1] Chhoy is the number "six" in Bengali, since we hop in IPv6.

The defining characteristic of IPv6 is its much larger address space relative to IPv4, with 128 bits per address instead of only 32. IPv6's larger address space was chosen to address the expected exhaustion of IPv4 addresses, realized in May 2014 [18]. Of the 128 bits, 64 are dedicated to LAN-specific information to supporter automatic address assignment [17]. We exploit these plentiful LAN addresses in our hopping mechanism.

Global IPv6 addresses contain a routing prefix (normally 48 bits or shorter), a subnet identifier (16 bits more), and an interface identifier (64 bits). The interface identifier can be static or can be generated by stateless auto configuration [25], or assigned using DHCPv6 [22]. In our work, the server uses a fixed /64 prefix (combining both routing prefix and subnet identifier), generates the interface identifier dynamically, and changing it every minute. A client needs to find out the interface identifier to get the service.

III. Related work

Our work is motivated by our desire to improve security using the unique properties of IPv6. As such, it augments existing IPv6 security and privacy, and is related to other moving target defenses.

There are several studies related to the dramatic growth in the IPv6 adoption, and suggest that IPv6 is no longer an "uninteresting rarity" [6], [5], [27], [12]. This widespread adoption of IPv6 implies that our use of IPv6 is viable and timely.

Though IPsec in IPv6 provides data integrity and confidentiality, it can expose the link-layer address, creating a new privacy risk [35]. To fix this, clients can choose random and ephemeral addresses using the IPv6 addressing privacy extension [24]. As an alternative way, providers utilize prefix rotation that changes the entire allocated prefix to improve address privacy [22], [32]. Our goal is the opposite; providing service in changing addresses, and clients need to find out the changing address.

We build on privacy-preserving IPv6 address assignment [10], [11], but while that work proposes updating addresses daily with a fixed pattern, we accelerate hopping each minute to service as an active defense against scanning. Our work is similar to port knocking [20], [7], but it hides in IPv6 rather than requiring "wake-up" packets. Closest to our work is IPv4-based port-hopping [21]; we take advantage of much larger IPv6 space (2^{64}) compared to the quite limited IPv4 port space (2^{16}). Work by Judmayer et al. uses a similar technique for IoT devices where they assume IoT devices use publicly routable IPv6 addresses [19]. Our solution does not interrupt running services, and is applicable for many other applications.

IV. Chhoyhopper Design

Our goal is to enable discovery-resistant public services. To accomplish this goal, clients will rendezvous with servers on a public, but temporary IPv6 address. By allocating the temporary address from a large space (2^{64} addresses), scanning is impractical, as we show in §VI-A. By changing the address frequently, reuse of a passively observed temporary address is only possible for a very brief window of time. The hopping pattern is cryptographically secure, so prior active addresses reveal nothing about future addresses.

A. Design Requirements

The Chhoyhopper design has a primary requirement of discovery resistance, and several secondary requirements:

Discovery resistance: Our primary goal is that services should be *discovery resistant*. An adversary should not be able to contact the service and carry out a brute-force or DoS attack, even if they know the network where the service is running.

Application support: Our moving target defense should support many existing, real-world applications. We describe a common hopping strategy as simple core defense, and then apply this design to applications such as SSH and HTTPS. Our design should be adaptive so that new applications can be added easily.

Transparency: Our system should be compatible with the current application clients without protocol changes. For example, an HTTPS client should be able to connect to a Chhoyhopper server using a web browser. Similarly, a client should have the SSH capability using a command line interface. Extensions to support hopping should be possible without changes to core programs.

Support for collateral services: Often IP addresses are exposed in collateral services, and we must ensure that a hopping IP address does not break other, related services. For example, HTTPS should support TLS authentication, and clients should be able to identify services by domain names, but both TLS and DNS must continue to function even if the underlying IP addresses hop. We integrate hopping in DNS lookup (§IV-B) and describe how TLS can support hopping addresses (§IV-D).

Uninterrupted connection while hopping: Our system should be able to hop over addresses seamlessly without breaking an already established connection. Our system should not require a restart to hop over to a different address so that no service interruption occurs. We meet this requirement by using ip6tables NAT rules.

Finally, a *non*-requirement is direct support for millions of clients. We depend a shared secret, but security of a secret over a large group is challenging. One could support large groups by splitting them into many small groups, each with a separate, revocable hopping secret.

B. Design Overview

A hopping IPv6 address must be understood at both the client and the server: the server will move service to a new address frequently, and a client must be able to find that server's current IPv6 address to start a new session. In addition, existing, long-lived sessions must continue even when the server moves to a new address.

Figure 1 shows the components of our system. The client and server must follow the same hopping pattern to rendezvous. We assume they share a pre-distributed secret key.

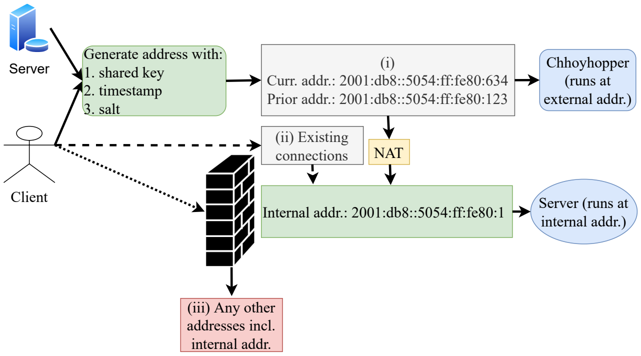

Server
Generate address with:
1. shared key
2. timestamp
3. salt
(i)
Curr. addr.: 2001:db8::5054:ff:fe80:634
Prior addr.: 2001:db8::5054:ff:fe80:123
Chhoyhopper (runs at external addr.)
Client
(ii) Existing connections
NAT
Internal addr.: 2001:db8::5054:ff:fe80:1
Server (runs at internal addr.)
(iii) Any other addresses incl. internal addr.

Fig. 1: Client and server interaction in Chhoyhopper.

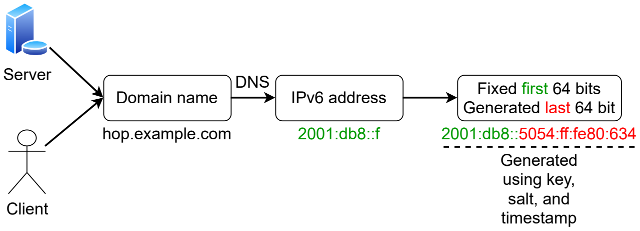


Fig. 2: Getting the rendezvous address.

We expect secret distribution to use common methods, such as out-of-band distribution of ssh keys today. Several methods are possible: including face-to-face sharing, secure interactive communication such as secure instant messaging, or other secure channels such as encrypted e-mail or an authenticated website. While we welcome new approaches to secret distribution, they are out-of-scope of this paper. Our requirement for this secret means Chhoyhopper cannot be used for anonymous clients to discover a server, since scanners could exploit any discovery process. It also means Chhoyhopper does not apply to very large groups where secret sharing becomes untenable. The design of the server ensures only the access of the legitimate clients who have the correct secret key and salt value. In this way, our service becomes discovery-resistant which is our first design requirement (§IV-A). Clients who lose the secret key will not have access to the service anymore and need to get the key again.

Next, we describe the selection and lifetime of the temporary address, hopping on the server, and hopping by the client.

C. Address Hopping Pattern

The server and the client compute the same temporary address by computing a cryptographic hash of the shared secret, a salt value, and the current time in minutes. We use the SHA-256 algorithm for hashing and the time in seconds since the Unix epoch. The salt value prevents rainbow attacks [29] and can vary by service or deployment.

We compute the IPv6 address in two parts. We take the DNS name of the service address and look up a full IPv6 address, but replace the low 64 bits of the address with the top 64 bits of the hash result. Figure 2 shows how a client and a server converge to a single rendezvous address. Our system gets an example IPv6 address (2001:db8::f) from the domain name using DNS. Then it keeps the first 64 bits (marked by green letters), and computes the last 64 bits (marked by red letters) using SHA-256 algorithm (in our example the computed address is 2001:db8::5054:ff:fe80:634). The clients and the server can only merge to a single address when they use the same secret key, salt value, and timestamp.

Use of DNS allows the service to move in the Internet and provides a user-friendly name. DNSSEC should be used to ensure that the DNS lookup of the top IPv6 address bits is not subject to a person-in-the-middle attack. If clients prefer, our system can also take a direct IPv6 service address. We discuss the potential of collisions in §VI-B.

The server tracks its current address, changing it every minute. To avoid problems with clock skew, the server listens to *two* addresses, one for the current minute and the other for the nearest adjacent minute. (Larger clock skew can be handled by increasing the duration addresses are kept active, if desired.) We use NAT rules (in ip6tables) to track live connections as addresses change.

D. Server-Side Hopping and Connection Persistence

Hopping over addresses seamlessly without interrupting any active connections was one of our service requirements (§IV-A). It is cumbersome for server software to change its service address every minute, and we would rather not modify server software and cannot break active connections. We therefore operate the server on a fixed address that is firewalled from the public Internet. Thus the traffic towards the current address needs to be translated to the internal address to respond to the clients. A daemon then uses network address translation to map the currently active addresses through the firewall to the internal fixed address. IP6table rules also ensure that once a connection is established it continues to operate, even after the server moves to other addresses for new connections.

Chhoyhopper server restricts the access only to the new clients with right IPv6 address, while continuing to serve existing clients who previously started access. To summarize server processing in Figure 1: (i) new flows to the current and prior address are detected by NAT rules, and establish new connection state before being passed to the internal server address, (ii) existing flows are detected by ip6tables rules and pass through to the internal address, (iii) any other addresses, including external traffic sent to the "internal" server address, are dropped by the server's firewall. When external traffic sent to the "internal" address, our deployed NAT rule translates the "internal" address to a different non-responsive address so that server's firewall drops those traffic. Thus our system can defend even if the attackers know the internal service address.

Our NAT-manipulation daemon for server is a simple Python program that modifies Linux ip6tables. The daemon assigns the NAT rules to a particular external interface on the server. Other OSes (like Windows or FreeBSD) would need to use their own, native NAT mechanisms; that is potential future work.

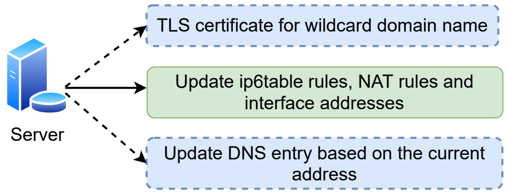

TLS certificate for wildcard domain name
Update ip6table rules, NAT rules and interface addresses
Update DNS entry based on the current address
Server

Fig. 3: Server for HTTPS.

E. Client Discovery of the Hopping Address in SSH

The client must compute and use the server's current IPv6 address to begin a new connection. We assume the server's secret key and the salt are known to the client, so the client does the same hash computation as the server. As with the server, the client looks up an IPv6 address from DNS and replaces the low 64 bits with the current temporary hash.

When a client is done with the connection, the server keeps the existing connection, and the current interface address. However, after terminating an old connection, the client needs to make the same computation to get the current address.

To be transparent (§IV-A), our client implementation for SSH uses a simple Python program which invokes the native client with appropriate arguments. A client can just use a command line interface to run the client SSH program.

F. Challenges with HTTPS

In addition to SSH, our system supports HTTPS (application support requirement in §IV-A). We believe our core hopping technique is generalizable to many applications. Some applications require additional support—we extend the core design of Chhoyhopper to meet the collateral service requirements of HTTPS in §IV-A.

The HTTPS deployment has two unique challenges. Our first challenge is to ensure *transparency* where a user gets the service like any other HTTPS service using a web browser.

Our second challenge is user demand for *TLS authentication*. TLS authentication is required for HTTPS, and our goal is to support the collateral services for any application (§IV-A). Since our server hops every minute, it is not feasible to get an SSL certificate for each IPv6 address. Also, IP-based TLS does not support wildcard certificates. Thus we cannot generate a wildcard certificate for a /64 prefix. Traditional use of a static domain name is not possible as well. A static DNS name would reveal the hop destination.

We provide transparent access to users with a new browser extension, then it rewrites the Chhoyhopper web request to the current hopping address without users able to tell. We currently provide this extension for Mozilla Firefox. This extension meets our design requirement for transparency (§IV-A). An extension for Google Chrome is technically feasible but requires DNS support (we currently use Firefox-specific DNS APIs).

We solve the certificate problem by getting a TLS certificate for a wildcard domain name, and then dynamically create changing hopping name under that wildcard. Next, we describe the changes in server and client for HTTPS.

G. Server-side Certificate Handling with Hopping HTTPS

The core design of HTTPS is similar to that we mentioned in §IV-D. HTTPS server also runs NAT rules to translate the current allowable addresses to the internal server address. It also runs the ip6tables rules to filter out the traffic that does not pass the NAT rules to get the internal address. Now we need to extend the core idea to enable support for TLS authentication (support for collateral service in §IV-A).

We enable TLS support by getting an SSL certificate for a wildcard domain name. Then the server opens service at dynamic domain names under that wildcard. As an example, the server needs to get an SSL certificate for "*.example.com" if the domain name is "example.com".

The server utilizes the same hash algorithm along with the same secret key, salt value, and timestamp to find out a domain name under the wildcard. We take 40 characters from this hash value to make the domain name (any domain name label can be 63 characters long [1]). The server puts the generated characters in the wildcard part of the domain name. At every minute, the server generates a new domain name.

Chhoyhopper server needs to update the DNS entry periodically for the generated domain name. Dynamic DNS maps the hopping name to the changing IPv6 address, and updates the DNS entry at a fixed interval. Only clients with the secret key can guess the hopping URL. Since the server has already updated a DNS entry for the hopping URL, the clients will get the right IPv6 address, and pass the filters. The clients can also authenticate the response because of the wildcard certificate provided by the server. Besides adding a new DNS entry, the server also deletes an old entry to limit the number of DNS entries. Since each subdomain uses a unique name, the system is not be affected by DNS caching. While updating DNS every minute has some overhead, the cost is quite modest and is similar to frequent DNS updates seen in CDNs.

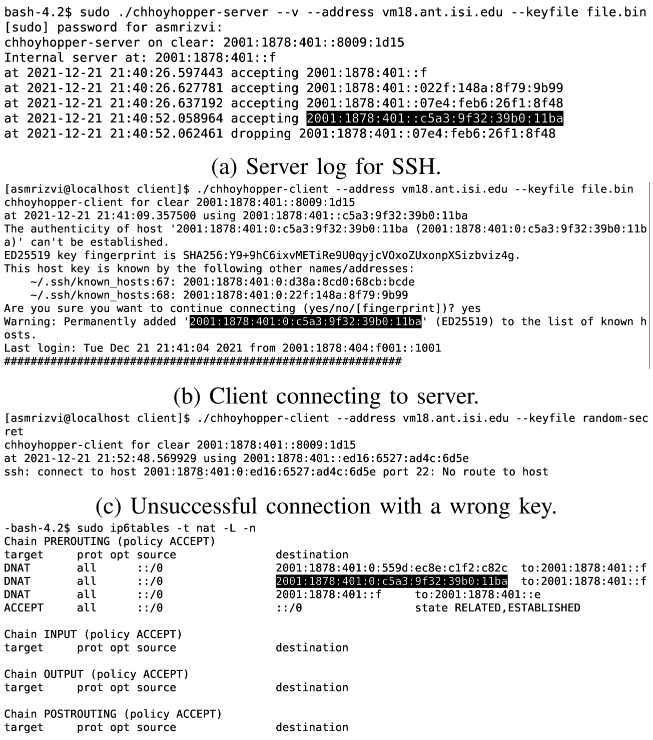
Figure 3 shows the server extension for Chhoyhopper in HTTPS. The green box shows the design that is common for all applications. The two blue boxes show that HTTPS requires extensions for TLS authentication and DNS updates.

H. Client Discovery of the Hopping Address in HTTPS

We already see that Chhoyhopper server opens the service at a dynamic domain name. A client needs to generate that domain name to get the intended web page.

Clients use the same technique to generate the domain name. It uses the same shared secret key, salt value, and timestamp to an SHA-256 algorithm to get the hash value. Using the computed hash value, it generates the domain name.

We want an automated way to generate the dynamic domain name, and use it in the browser to get the web pages running Chhoyhopper. We provide a browser extension to hop over dynamic domain names. Our browser extension is lightweight; takes inputs from the clients about the Chhoyhopper domain name, shared secret key, and salt value. When the users type any domain name that matches the user input for Chhoyhopper base domain name, it generates the dynamic domain name, and rewrites the request to the



```
bash-4.2$ sudo ./chhoyhopper-server --v --address vm18.ant.isi.edu --keyfile file.bin
[sudo] password for asmrizvi:
chhoyhopper-server on clear: 2001:1878:401::8009:1d15
Internal server at: 2001:1878:401::f
at 2021-12-21 21:40:26.597443 accepting 2001:1878:401::f
at 2021-12-21 21:40:26.627781 accepting 2001:1878:401::022f:148a:8f79:9b99
at 2021-12-21 21:40:26.637192 accepting 2001:1878:401::07e4:feb6:26f1:8f48
at 2021-12-21 21:40:52.058964 accepting 2001:1878:401::c5a3:9f32:39b0:11ba
at 2021-12-21 21:40:52.062461 dropping 2001:1878:401::07e4:feb6:26f1:8f48
```

(a) Server log for SSH.

```
[asmrizvi@localhost client]$ ./chhoyhopper-client --address vm18.ant.isi.edu --keyfile file.bin
chhoyhopper-client for clear 2001:1878:401::8009:1d15
at 2021-12-21 21:41:09.357500 using 2001:1878:401::c5a3:9f32:39b0:11ba
The authenticity of host '2001:1878:401:0:c5a3:9f32:39b0:11ba (2001:1878:401:0:c5a3:9f32:39b0:11b
a)' can't be established.
ED25519 key fingerprint is SHA256:Y9+9hC6ixvMETiRe9U0qyjcVOxoZUxonpXSizbviz4g.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:67: 2001:1878:401:0:d38a:8cd0:68cb:bcde
    ~/.ssh/known_hosts:68: 2001:1878:401:0:22f:148a:8f79:9b99
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '2001:1878:401:0:c5a3:9f32:39b0:11ba' (ED25519) to the list of known h
osts.
Last login: Tue Dec 21 21:41:04 2021 from 2001:1878:404:f001::1001
############################################################
```

(b) Client connecting to server.

```
[asmrizvi@localhost client]$ ./chhoyhopper-client --address vm18.ant.isi.edu --keyfile random-sec
ret
chhoyhopper-client for clear 2001:1878:401::8009:1d15
at 2021-12-21 21:52:48.569929 using 2001:1878:401::ed16:6527:ad4c:6d5e
ssh: connect to host 2001:1878:401:0:ed16:6527:ad4c:6d5e port 22: No route to host
```

(c) Unsuccessful connection with a wrong key.

```
-bash-4.2$ sudo ip6tables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all      ::/0                 2001:1878:401:0:559d:ec8e:c1f2:c82c  to:2001:1878:401::f
DNAT       all      ::/0                 2001:1878:401:0:c5a3:9f32:39b0:11ba  to:2001:1878:401::f
DNAT       all      ::/0                 2001:1878:401::f     to:2001:1878:401::e
ACCEPT     all      ::/0                 ::/0                 state RELATED,ESTABLISHED

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

(d) NAT rules in server.

Fig. 4: Client-server interaction for SSH.

generated domain name. For example, when the clients type "example.com", the browser extension redirects the request to "generated_hashed_chars.example.com". The browser extension prevents the recursive redirection by keeping track about the recent translation.

Since the server has already added a DNS entry, the browser extension redirects to the current domain name, and DNS translation is done using the newly added entry.

A web page has multiple links to other web pages. Depending on the deployment, these links can be relative to the current page, or they can be a complete link to the other pages. In both cases, our browser extension will look for the base domain name and redirects if it finds a match. Thus relative links to the terminated connections work as well by regenerating a new domain name. To prevent the URL leaking through a referrer header [37], we recommend the servers to set its "Referrer-Policy" to "no-referrer".

V. Example Use

In this section, we discuss the implementation details of Chhoyhopper. Currently, we provide support for SSH and HTTPS. We demonstrate our implementation by directly taking runtime screenshots.

A. SSH

We provide the SSH service without directly modifying the standard SSH client. At the same time, we want to keep the Chhoyhopper SSH client simple so that the users can use it through a command line interface. Thus we provide a script that takes input parameters for the Chhoyhopper domain name, secret key, and salt value. Then the script computes the current IPv6 address, and provides the standard SSH client with the computed IPv6 address to make the connection.

We also provide a script for the server that takes similar inputs for internal address, secret key, and salt value. The server script then periodically assigns interface address, deploys ip6tables and NAT rules for access control, and deletes obsolete addresses and rules.

Figure 4 demonstrates the implementation of Chhoyhopper for SSH .

To meet the discovery-resistant requirement (§IV-A), at a fixed interval, the server opens its service at a temporary IPv6 address, and drops the prior minute's active address. The server log in Figure 4a shows that the server opens service at an address ending with 11ba (highlighted black). At the same time, the server also drops the prior running address ending with 8f48. A client with the same secret key, salt value and timestamp will recreate the same IPv6 address and successfully connect to the server. Figure 4b shows a successful connection where the client uses the same secret key, generates the same IPv6 address (see the highlighted address ending with 11ba), and connects to the server using a command line interface (transparency requirement in §IV-A). If a client uses a wrong key, the client cannot make a successful connection. Figure 4c shows an unsuccessful connection attempt where the client uses a wrong secret key named "random-secret", and makes request to the address ending with 6d5e (not the current address with 11ba). The server deploys a destination NAT rule to translate the current IPv6 address to the internal address, and another rule to keep the existing connections (uninterrupted service requirement from §IV-A). This translation is shown in the list of ip6tables NAT rules (highlighted in Figure 4d). To test the uninterrupted service, we establish a new SSH connection to a temporary IPv6 address and wait for the duration until the temporary address stops accepting new connections. We confirm that the old connection continues even when the original IPv6 address is not accepting new connections any more.

B. HTTPS

To meet the application support requirement (§IV-A), we show how we implement HTTPS, and how it is different from SSH. For HTTPS, the clients need a browser extension, and the server needs additional steps like getting a TLS certificate and updating DNS entries periodically. We also show an example use case where a client connects to the server using a web browser.

We provide a browser extension that intercepts the Chhoyhopper domain name and redirects the requests to the current domain name. Different from SSH, clients can provide the inputs for domain name, salt, and key value using the input page of the browser extension. The browser saves these inputs, and uses them later to redirect the clients.

A script on the server updates its IP address (as with SSH). It also updates dynamic DNS, adding a unique name for each new IP address. Before running the script, the server also needs to generate an SSL certificate for the wildcard domain name.

The client-server interaction for HTTPS is shown in Figure 5 . Like SSH, the server utilizes similar ip6tables NAT entries for access control. At the same time, the server adds a DNS entry for the generated domain name. We can see the generated domain name with the wildcard part before the first dot along with the corresponding IPv6 address in Figure 5a (highlighted in black).

A client needs to use a browser extension for getting the Chhoyhopper HTTPS service (transparency requirement from §IV-A). Figure 5b shows the input page for the clients. A client needs to provide the Chhoyhopper base domain name, secret key, and salt value. The browser saves these options for future use.

When a client types the Chhoyhopper base domain name, the browser checks the saved domain name, and if the browser finds a match, it generates the current domain name using the shared secret key, salt value, and timestamp. The browser then successfully redirects the request to the current domain name (see the domain name in Figure 5c). Since the server already updates the DNS entry, the browser will get the current IPv6 address after the DNS resolution. The padlock symbol in the address bar of Figure 5c indicates the transport layer security to meet the collateral service requirement (§IV-A).

When a client uses a wrong secret key, the redirection does not work. The request is then redirected to a different domain name which cannot get the current IPv6 address (Figure 5d).

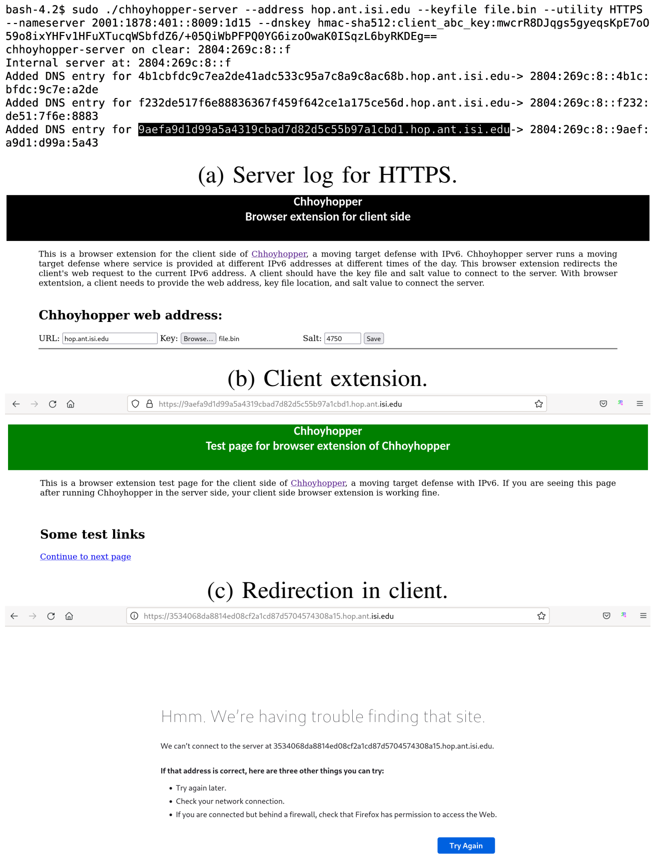
VI. Analysis

We analyze our system to find out the risk of discovery and collision. We show that the chance of getting discovered or having a collision is vanishingly small, even if there are millions of servers under the same IPv6 prefix. In the very unlikely event the active IPv6 address is guessed, the attackers has at most two minutes to carry out brute-force password guessing. Address collisions from multiple servers are exceedingly unlikely and can be completely avoided by assigning each server a different /64; but in the worst case a collision prevents access for only two minutes. We also discuss other run-time costs of our system.

A. Risk of Discovery

To estimate the difficulty of brute-force scanning, consider a scanner scanning at 100 Gb/s looking for a server hopping in one /64 with 64B TCP SYNs. At that rate (scanning 2×10^8 addresses per second) the expected time to discover one server is about 3000 years, at which point the adversary will have at most two minutes to exploit it. Since the address space is huge compared to the scanning rate, we are confident that brute-force scanning is impractical. Since the address is hopping randomly, intelligent scanning is not possible.

An adversary that observes traffic will know prior hop addresses. If the hopping pattern is predictable, such knowledge could be used to discover future hopping addresses. Our assertion of hopping unpredictability is based on the cryptographically security of our hash function, SHA-256. As of 2022, SHA-256 is regarded as secure, but the algorithm may need to be replaced in the future.



```
bash-4.2$ sudo ./chhoyhopper-server --address hop.ant.isi.edu --keyfile file.bin --utility HTTPS
--nameserver 2001:1878:401::8009:1d15 --dnskey hmac-sha512:client_abc_key:mwcrR8DJqgs5gyeqsKpE7oO
59o8ixYHFv1HFuXTucqWSbfdZ6/+05QiWbPFPQ0YG6izoOwaK0ISqzL6byRKDEg==
chhoyhopper-server on clear: 2804:269c:8::f
Internal server at: 2804:269c:8::f
Added DNS entry for 4b1cbfdc9c7ea2de41adc533c95a7c8a9c8ac68b.hop.ant.isi.edu-> 2804:269c:8::4b1c:
bfdc:9c7e:a2de
Added DNS entry for f232de517f6e88836367f459f642ce1a175ce56d.hop.ant.isi.edu-> 2804:269c:8::f232:
de51:7f6e:8883
Added DNS entry for 9aefa9d1d99a5a4319cbad7d82d5c55b97a1cbd1.hop.ant.isi.edu-> 2804:269c:8::9aef:
a9d1:d99a:5a43
```

(a) Server log for HTTPS.

Chhoyhopper
Browser extension for client side

This is a browser extension for the client side of Chhoyhopper, a moving target defense with IPv6. Chhoyhopper server runs a moving target defense where service is provided at different IPv6 addresses at different times of the day. This browser extension redirects the client's web request to the current IPv6 address. A client should have the key file and salt value to connect to the server. With browser extentsion, a client needs to provide the web address, key file location, and salt value to connect the server.

Chhoyhopper web address:

URL: hop.ant.isi.edu Key: Browse... file.bin Salt: 4750 Save

(b) Client extension.

https://9aefa9d1d99a5a4319cbad7d82d5c55b97a1cbd1.hop.ant.isi.edu

Chhoyhopper
Test page for browser extension of Chhoyhopper

This is a browser extension test page for the client side of Chhoyhopper, a moving target defense with IPv6. If you are seeing this page after running Chhoyhopper in the server side, your client side browser extension is working fine.

Some test links

Continue to next page

(c) Redirection in client.

https://3534068da8814ed08cf2a1cd87d5704574308a15.hop.ant.isi.edu

Hmm. We're having trouble finding that site.

We can't connect to the server at 3534068da8814ed08cf2a1cd87d5704574308a15.hop.ant.isi.edu.

If that address is correct, here are three other things you can try:

- Try again later.
- Check your network connection.
- If you are connected but behind a firewall, check that Firefox has permission to access the Web.

Try Again

(d) Unsuccessful connection using a wrong key.

Fig. 5: Client-server interaction for HTTPS.

B. Risk of Collisions

When multiple servers share the same /64 address prefix, it is possible that they could collide and hop to the same address. A concerned operator should assign each server a unique /64 prefix (operators can get a /48 prefix or so, and then assign a unique /64 prefix to each server). However, we suggest that odds of collision is so low that collision avoidance is unnecessary.

Collisions of hopping addresses is equivalent to the well-known Birthday Problem, but rather than n people in 365 days of the year, we have k servers in 2^{64} addresses. Using a simplified approximation, the probability of a hash collision in any given minute is $1 - e^{\frac{-k(k-1)}{2N}}$ [30]. Using this formula, the probability of an address mapped into the k of 1 million addresses is only 1 in 37 million. As we generate an address every minute, we can expect a collision with these million servers once in every 70 years. This failure rate is considerably less than DRAM failures due to cosmic radiation [34].

C. Run-time Costs

Runtime overhead for Chhoyhopper is usually done out-of-band with new connections, or is very small.

The server selects new IP addresses every minute, but this cost is out-of-band of new connections (so it does not affect clients), and small (a cryptographic hash and local ip6tables manipulation).

Clients starting a new connection must read the secret and carry out a cryptographic hash, but this overhead is small relative to the already required Diffie-Hellman key exchange and SSH protocol negotiation.

Live connections require IP address translation from the hopped address to the internal address. This cost is exactly one NAT mapping. Most cloud services already have at least two levels of address translation (for example, see VL2 [14]), so the overhead of an additional mapping is quite modest.

VII. Future Work and Conclusions

Currently, we support SSH client with a Python program, and HTTPS using a Firefox extension. Potential future work is to a Chhoyhopper client integrated with OpenSSH, to provide HTTPS extension support for Chrome, and to port server support to non-Linux operating systems.

In this work, we provide an implementation of a discovery-resistant moving target defense named "Chhoyhopper" to provide security utilizing the huge IPv6 address space. To the best of our knowledge, this is the first deployment of a hopping defense with IPv6, applicable for both SSH and HTTPS. Using our system, a service will hop over different IPv6 addresses, and a client needs to find the current IPv6 address to connect. Our implementation is publicly available and we provide support for SSH and HTTPS applications.

Acknowledgment

A S M Rizvi and John Heidemann's work on this paper is supported, in part, by the DHS HSARPA Cyber Security Division via contract number HSHQDC-17-R-B0004-TTA.02-0006-I (PAADDoS), and by DARPA under Contract No. HR001120C0157 (SABRES). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or DARPA.

We thank Rayner Pais who prototyped an early version of Chhoyhopper and version in IPv4 hopping over ports.

References

[1] Domain names - implementation and specification. RFC 1035, November 1987.

[2] David Adrian, Zakir Durumeric, Gulshan Singh, and J. Alex Halderman. Zippier ZMap: Internet-wide scanning at 10 Gbps. In *Proceedings of the USENIX Workshop on Offensive Technologies*, San Diego, CA, USA, August 2014. USENIX.

[3] APNIC. IPv6 capable rate by country (https://stats.labs.apnic.net/ipv6, 2021. [Online; accessed 13-December-2021].

[4] L Bošnjak, J Sreš, and Bosnjak Brumen. Brute-force and dictionary attack on hashed real-world passwords. In *2018 41st international convention on information and communication technology, electronics and microelectronics (mipro)*, pages 1161–1166. IEEE, 2018.

[5] Lorenzo Colitti, Steinar H Gunderson, Erik Kline, and Tiziana Refice. Evaluating IPv6 adoption in the internet. In *International Conference on Passive and Active Network Measurement*, pages 141–150. Springer, 2010.

[6] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. Measuring IPv6 adoption. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 87–98, 2014.

[7] Rennie Degraaf, John Aycock, and Michael Jacobson. Improved port knocking with strong authentication. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 10–pp. IEEE, 2005.

[8] Tom Emmons. 2021: Volumetric DDoS attacks rising fast. https://www.akamai.com/blog/security/2021-volumetric-ddos-attacks-rising-fast, 2021. [Online; accessed 10-Jan-2022].

[9] Kensuke Fukuda and John Heidemann. Who knocks at the IPv6 door? detecting IPv6 scanning. In *Proceedings of the Internet Measurement Conference 2018*, pages 231–237, 2018.

[10] F. Gont. A method for generating semantically opaque interface identifiers with IPv6 stateless address autoconfiguration (SLAAC). RFC 7217, Internet Request For Comments, April 2014.

[11] F. Gont, S. Krishnan, T. Narten, and R. Draves. Temporary address extensions for stateless address autoconfiguration in IPv6. RFC 8981, Internet Request For Comments, February 2021.

[12] Google. Google IPv6 statistics. https://www.google.com/intl/en/ipv6/statistics.html, 2021. [Online; accessed 13-December-2021].

[13] Robert Graham, Paul McMillan, and Dan Tentler. Mass scanning the internet. Presentation at Defcon 22, August 2014.

[14] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, and Parveen Pat. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM Conference*, pages 51–62, Barcelona, Spain, August 2009. ACM.

[15] John Heidemann, Yuri Pradkin, Ramesh Govindan, Christos Papadopoulos, Genevieve Bartlett, and Joseph Bannister. Census and survey of the visible Internet. In *Proceedings of the ACM Internet Measurement Conference*, pages 169–182, Vouliagmeni, Greece, October 2008. ACM.

[16] Bob Hinden and Dr. Steve E. Deering. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.

[17] R. Hinden and S. Deering. IP version 6 addressing architecture. RFC 1884, Internet Request For Comments, December 1995.

[18] ICANN. Remaining IPv4 addresses to be redistributed to regional internet registries — address redistribution signals that ipv4 is nearing total exhaustion. ICANN Announcement, 20 May 2014.

[19] Aljosha Judmayer, Johanna Ullrich, Georg Merzdovnik, Artemios G Voyiatzis, and Edgar Weippl. Lightweight address hopping for defending the IPv6 IoT. In *Proceedings of the 12th international conference on availability, reliability and security*, pages 1–10, 2017.

[20] Martin Krzywinski. Port knocking: Network authentication across closed ports. *SysAdmin Magazine*, 12(6):12–17, June 2003.

[21] Henry CJ Lee and Vrizlynn LL Thing. Port hopping for resilient networks. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, volume 5, pages 3291–3295. IEEE, 2004.

[22] Tomek Mrugalski, Marcin Siodelski, Bernie Volz, Andrew Yourtchenko, Michael Richardson, Sheng Jiang, Ted Lemon, and Timothy Winters. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 8415, November 2018.

[23] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. Target generation for internet-wide IPv6 scanning. In *Proceedings of the 2017 Internet Measurement Conference*, pages 242–253, 2017.

[24] Dr. Thomas Narten, Richard P. Draves, and Suresh Krishnan. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941, September 2007.

[25] Dr. Thomas Narten, Tatsuya Jinmei, and Dr. Susan Thomson. IPv6 Stateless Address Autoconfiguration. RFC 4862, September 2007.

[26] T. Narten, G. Huston, and L. Roberts. IPv6 address assignment to end sites. RFC 6177, Internet Request For Comments, March 2011.

[27] Mehdi Nikkhah and Roch Guérin. Migrating the internet to IPv6: An exploration of the when and why. *IEEE/ACM Transactions on Networking*, 24(4):2291–2304, 2015.

[28] Erik Nygren. At 21 Tbps, reaching new levels of IPv6 traffic! https://blogs.akamai.com/2020/02/at-21-tbps-reaching-new-levels-of-ipv6-traffic.html, 2020. [Online; accessed 15-March-2021].

[29] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Proceedings of the IACR CRYPTO*, volume 2729, pages 617–630. International Association for Cryptologic Research, August 2003.

[30] Preshing on Programming. Hash collision probabilities. https://preshing.com/20110504/hash-collision-probabilities/, 2011. [Online; accessed 7-November-2021].

[31] Jim Owens and Jeanna Matthews. A study of passwords and methods used in brute-force SSH attacks. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. Citeseer, 2008.

[32] Erik C Rye, Robert Beverly, et al. Follow the scent: Defeating ipv6 prefix rotation privacy. *arXiv preprint arXiv:2102.00542*, 2021.

[33] Opera Software. SKA - SSH key authority. https://github.com/operasoftware/ssh-key-authority, 2021. [Online; accessed 09-July-2021].

[34] Vilas Sridharan and Dean Liberty. A study of DRAM failures in the field. In *Proceedings of the ACM SuperComputing*, pages 1–11, Salt Lake City, Utah, USA, November 2012. ACM.

[35] Johanna Ullrich, Katharina Krombholz, Heidelinde Hobel, Adrian Dabrowski, and Edgar Weippl. IPv6 security: attacks and countermeasures in a nutshell. In *8th {USENIX} Workshop on Offensive Technologies ({WOOT} 14)*, 2014.

[36] Jessica Wei. Why is a /48 the recommended minimum prefix size for routing? https://blog.apnic.net/2020/06/01/why-is-a-48-the-recommended-minimum-prefix-size-for-routing/, 2020. [Online; accessed 15-March-2021].

[37] World Wide Web Consortium (W3C). Referrer policy. https://www.w3.org/TR/referrer-policy/, 2017. [Online; accessed 13-Mar-2022].

[38] Eric Wustrow, Manish Karir, Michael Bailey, Farnam Jahanian, and Geoff Houston. Internet background radiation revisited. In *Proceedings of the 10th ACM Internet Measurement Conference*, pages 62–73, Melbourne, Australia, November 2010. ACM.