Chhoyhopper: A Moving Target Defense with IPv6

A S M Rizvi and John Heidemann
USC/Information Sciences Institute

ABSTRACT

Services on the public Internet are frequently scanned, then subject to brute-force and denial-of-service attacks. We would like to run such services stealthily, available to friends but hidden from adversaries. In this work, we propose a moving target defense named "Chhoyhopper" that utilizes the vast IPv6 address space to conceal publicly available services. The client and server hop to different IPv6 addresses 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.

ACM Reference format:
A S M Rizvi and John Heidemann . 2021. Chhoyhopper: A Moving Target Defense with IPv6. In *Proceedings of draft (under submission), ACSAC'2021 Poster, December 2021 (Draft (under submission)),* 2 pages.
https://doi.org/

1 INTRODUCTION

Services on the public Internet are frequently scanned as a precursor to a brute-force or denial-of-service attack. IPv4 scanning has been possible for more than a decade and recent tools allow scanning all of IPv4 in minutes [1].

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

IPv6 provides a huge address space in which we can hide services. In spite of attempts to discover active addresses, when every LAN has 2^{64} addresses (or more), active discovery of services on intentially obscure addresses is intractable (see §3). With /48s as the recommended minimum size of publically routable prefix, and /56s recommended for homes, even with a million devices in a home, quintillions of addresses remain unused on every network.

Our insight is that a *moving* target can elude scanners. We describe *Chhoyhopper*[1], using vast IPv6 address space to conceal publicly available services. The client and server to hop to different IPv6 addresses in a pattern based on a shared, pre-distributed secret and the time-of-day. Like authentication via SSH keys,we target sharing across small groups, and 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 make two new contributions: first, we show that IPv6 address hopping can be used to protect existing services like SSH and HTTPS. Second, we propose a new approach to accommodate long-lived connections in the face of frequent address changes. We use ip6tables to retain existing connections even after the original ephemeral address has changed.

Related work: Our work builds on ideas in privacy-preserving IPv6 address assignment [3], 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 [2], but it hides in IPv6 rather than requiring "wake-up" packets. Closest to our work is IPv4-based port-hopping [5]; we take advantage of much larger IPv6 interface-identifier space (2^{64}) compared to the quite limited IPv4 port space (2^{16}). Judmayer et al. propose hopping for IPv6 IoT devices [4]; we show how to hop with existing services (SSH and HTTPS) and unmodified servers.

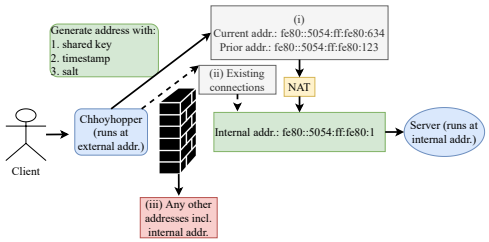

Figure 1: Client and server interaction in Chhoyhopper

Availability: Our implementation is freely available at https://ant.isi.edu/software/chhoyhopper/.

2 CHHOYHOPPER DESIGN

Our goal is to allow the client to rendezvous with the server 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 §3. 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.

Figure 1 shows the components of our system, and we describe them next: selection and lifetime of the temporary address, hopping on the server, and hopping by the client.

Address Hopping Pattern: The client and server must follow the same hopping pattern to rendezvous. We assume they share a pre-distributed secret key, which may be distributed by several means, such as face-to-face sharing ahead-of-time, through a secure channel such as encrypted e-mail. Our requirement for this secret means Chhoyhopper cannot be used for anonymous clients to discover a server, since scanners could exploit any discovery process.

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 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. Use of DNS allows the service to move in the Internet

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

and provides a user-friendly name. We discuss the potential of collisions in §3.

Server-Side Hopping: 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.

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. A daemon then uses network address translation to map the currently active addresses through the firewall to the internal fixed address. NAT translation also ensures that once a connection is established it continues to operate, even after the server moves to other addresses for new connections.

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 NAT 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.

Our NAT-manipulation daemon is a simple Python program modifying Linux ip6tables. The daemon assigns the NAT rules to a particular external interface on the server.

Hopping at the Client: 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. Our client implementation for SSH uses a simple Python program which invokes the native client with appropriate arguments.

Challenges with HTTPS: 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*. IP-based TLS certificates do not support wildcarding, and 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.

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. Dynamic DNS maps the hopping name to the changing IPv6 address, and update the DNS entry every minute. Only clients with the secret key can guess the hopping URL, and they can authenticate because of the wildcard certificate for the dynamic DNS name.

3 ANALYSIS

Risk of Discovery: To estimate the difficulty of brute-force scanning, consider a scanner scanning at 100Gb/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.

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 a unique IPv6 address every minute that is not used by any other 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 $1 - e^{\frac{-k(k-1)}{2N}}$ [6]. 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.

4 CONCLUSIONS AND FUTURE WORK

In this paper, we provide an implementation of a moving target defense named "Chhoyhopper" to provide security utilizing the huge IPv6 address space. Using our system, a service will hop over different IPv6 addresses, and a client needs to find the current IPv6 address to connect. We already support SSH client with a Python program, and HTTPS using a Firefox extension. We also plan to provide a Chhoyhopper client as a patch to OpenSSH, and provide HTTPS extension support for Chrome.

ACKNOWLEDGMENTS

ASM 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, and by DARPA under Contract No. HR001120C0157. 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 an IPv4 port hopper.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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