Sphinx packet replay detection

David Stainton

Abstract

This document defines the replay detection for any protocol that uses the Sphinx cryptographic packet format. This document is meant to serve as an implementation guide and to document the existing replay protection for deployed mix networks.

Terminology

The following terms are used in this specification.

epoch
: A fixed-time interval with a current default value of 20 minutes. A new PKI document containing public key material is published for each epoch and is valid only for that epoch. For more information, see Sphinx mix and provider key rotation [https://katzenpost.network/docs/specs/mix_network/#sphinx-mix-and-provider-key-rotation].

group
: A finite set of elements and a binary operation that satisfy the properties of closure, associativity, invertability, and the presence of an identity element.

group element
: An individual element of a group.

group generator
: A group element capable of generating any other element of a group, via repeated applications of the generator and the group operation.

header
: The packet header consisting of several components which convey the information necessary to verify packet integrity and to correctly process the packet.

packet
: A Sphinx packet, of fixed length for each class of traffic, carrying a message payload and metadata for routing. Packets are routed anonymously through the mixnet and cryptographically transformed at each hop.

payload
: The fixed-length portion of a packet containing an encrypted message or part of a message, to be delivered anonymously.

SEDA
: Staged Event Driven Architecture. 1. A highly parallelizable computation model. 2. A computational pipeline composed of multiple stages connected by queues utilizing active queue-management algorithms that can evict items from a queue based on dwell time or other criteria where each stage is a thread pool. 3. The only correct way to efficiently implement a software based router on general purpose computing hardware.

Conventions used in this document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119.

Introduction

The Sphinx cryptographic packet format is a compact and provably secure design introduced by George Danezis and Ian Goldberg in SPHINX09. Although it supports replay detection, the exact mechanism of replay detection is neither described in SPHINX09 nor is it described in our SPHINXSPEC. Therefore we detail here how to efficiently detect Sphinx packet replay attacks.

Sphinx cryptographic primitives

This specification borrows the following cryptographic primitive constants from our SPHINXSPEC:

- `H(M)` - A cryptographic hash function which takes a byte array M to produce a digest consisting of a `HASH_LENGTH` byte array. `H(M)` MUST be pre-image and collision resistant.
- `EXP(X, Y)` - An exponentiation function which takes the `GROUP_ELEMENT_LENGTH` byte array group elements `X` and `Y`, and returns `X ^^ Y` as a `GROUP_ELEMENT_LENGTH` byte array.

Let `G` denote the generator of the group, and `EXP_KEYGEN()` return a `GROUP_ELEMENT_LENGTH` byte array group element usable as a private key.

The group defined by `G` and `EXP(X, Y)` MUST satisfy the decisional Diffie-Hellman assumption [https://en.wikipedia.org/wiki/Decisional_Diffie%E2%80%93Hellman_assumption].

Sphinx parameter constants

- `HASH_LENGTH` - 32 bytes. Katzenpost currently uses SHA-512/256. RFC6234
- `GROUP_ELEMENT_LENGTH` - 32 bytes. Katzenpost currently uses X25519. RFC7748

System overview

Mixnets as currently deployed have two modes of operation:

- Sphinx routing keys and replay caches are persisted to disk.
- Sphinx routing keys and replay caches are persisted to memory.

These two modes of operation fundamentally represent a tradeoff between mix server availability and notional compulsion attack resistance. Choosing a mode is the mix operator's decision to make since the security and availability of the mix servers are affected. Since mix networks are vulnerable to various types of compulsion attack (see Compulsion Threat Considerations [https://katzenpost.network/docs/specs/sphinx_format/#compulsion-threat-considerations] in the *Sphinx cryptographic packet format specification*) there is some advantage to *not* persisting the Sphinx routing keys to disk. The mix server operator can simply power-off the servers before seizure rather than physically destroying disks in order to prevent capture of the Sphinx routing keys. An argument can be made for using full disk encryption, but this may not be practical for servers hosted in remote locations.

On the other hand, persisting Sphinx routing keys and replay caches to disk is useful because it allows mix operators to shut down their server for maintenance purposes without loosing the Sphinx routing keys and replay caches. This means that as soon as the maintenance operation is completed, the server is able to rejoin the network. Our current PKI system, described in KATZMIXPKI, does not provide a mechanism to notify directory authorities of outages or maintenance periods. Consequently if the Sphinx routing keys are lost, a mix server outage occurs until the next epoch.

Both modes of operation completely prevent replay attacks after a system restart. In the case of disk persistence, replay attacks are prevented because all packets traversing the server have their replay tags persisted to disk cache. This cache is therefore once again used to prevent replays after a system restart. In the case of memory persistence, replays are prevented upon restart because the Sphinx routing keys are destroyed and therefore the mix will not participate in the network at all until at least the next epoch rotation. However, availability of the mix server may require two epoch rotations because (in accordance with KATZMIXPKI) the servers publish future epoch keys so that Sphinx packets flowing through the network can seamlessly straddle epoch boundaries.

Sphinx packet replay cache

Sphinx replay tag composition

The following excerpt from our SPHINXSPEC shows how the replay tag is calculated.

```
hdr = sphinx_packet.header
shared_secret = EXP( hdr.group_element, private_routing_key )
replay_tag = H( shared_secret )
```

However, this tag is not utilized in replay detection until the rest of the Sphinx packet is fully processed and its header is MAC-verified as described in SPHINXSPEC.

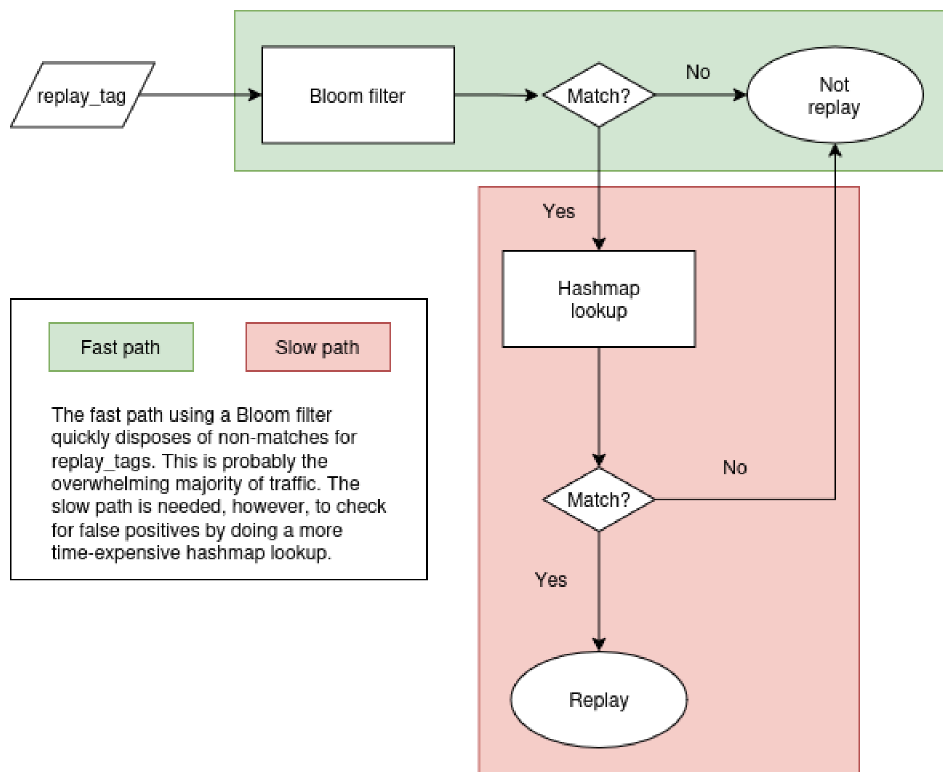
Sphinx replay tag caching

Although it would be sufficient to check a key-value store or hashmap to detect duplicate replay tags, we additionally employ a bloom filter to increase performance. Sphinx keys must periodically be rotated and destroyed to mitigate compulsion attacks, and our replay caches must likewise be rotated. This kind of key erasure scheme limits the time window during which an adversary can perform a compulsion attack. See our PKI specification KATZMIXPKI for more details regarding epoch key rotation and the grace period before and after the epoch boundary.

We tune our bloom filter for line-speed; that is to say, the bloom filter for a given replay cache is tuned for the maximum number of Sphinx packets that can be sent on the wire during the epoch duration of the Sphinx routing key. The tuning must take into account the size of the Sphinx packets as well as the maximum line speed of the network interface. This is a conservative tuning heuristic given that there must be more than this maximum number of Sphinx packets in order for there to be duplicate packets.

Our bloom-filter-with-hashmap replay detection cache looks like this:

Figure 1. Replay cache



This diagram does not express the full complexity of the replay caching system. In particular, it does not describe how entries are entered into the bloom filter and hashmap. Upon either bloom filter mismatch or hashmap mismatch, both data structures must be locked and the replay tag inserted into each.

For the disk persistence mode of operation, the hashmap can simply be replaced with an efficient key-value store. Persistent stores may use a write-back cache and other techniques for efficiency.

Epoch boundaries

Since mix servers publish future epoch keys, our replay detection forms a special kind of double-bloom-filter system. During the epoch grace period, servers perform a trial decryption of Sphinx packets. The replay cache used is the one associated with the Sphinx routing key that was successfully used to decrypt (unwrap transform) the Sphinx packet. This is not a double-bloom filter in the normal sense of the term since each bloom filter used is distinct and associated with its own cache. Furthermore, replay tags are only inserted into one cache and one bloom filter.

Cost of checking replays

The cost of checking a replay tag from a single replay cache is the sum of the following operations:

1. Sphinx packet unwrap operation
2. Bloom filter lookup
3. Hashmap or cache lookup

These operations are roughly O(1) in complexity. However, Sphinx packets processed near epoch boundaries will not be constant time due to trial decryption with two Sphinx routing keys as mentioned above in the section called “Epoch boundaries”.

Concurrent processing of Sphinx packet replay tags

The best way to implement a software-based router is with a SEDA computational pipeline. We therefore need a mechanism to allow multiple threads to reference our rotating Sphinx keys and the associated replay caches. Here we shall describe how the mix server uses a shadow memory system such that the individual worker threads always have a reference to the current set of candidate mix keys and associated replay caches.

PKI updates

The mix server periodically updates its knowledge of the network by downloading a new consensus document as described in KATZMIXPKI. The individual threads in the *cryptoworker* thread pool that process Sphinx packets make use of a `MixKey` data structure consisting of the following:

- A Sphinx routing key material (public and private X25519 keys)
- A replay cache
- A reference counter

Each of these cryptoworker threads has its own hashmap associating epochs with a reference to the `MixKey`. The mix server PKI thread maintains a single hashmap which associates the epochs with the corresponding `MixKey`. We refer to this hashmap as `MixKeys`. After a new `MixKey` is added to `MixKeys`, a *reshadow* operation is performed for each cryptoworker thread. The *reshadow* operation performs two tasks:

- Removes entries from each cryptoworker thread's hashmap that are no longer present in `MixKeys` and decrements the `MixKey` reference counter.
- Adds entries present in `MixKeys` but not present in the thread’s hashmap and increments the `MixKey` reference counter.

Once a given `MixKey` reference counter is decremented to zero, the `MixKey` and its associated on-disk data are purged.

Although we do not discuss synchronization primitives, it should be obvious that updating the replay cache should likely make use of a mutex or similar primitive to avoid data races between cryptoworker threads.

References

COMPULS05

Danezis, G., Clulow, J., “Compulsion Resistant Anonymous Communications”, Proceedings of Information Hiding Workshop, June 2005, https://www.freehaven.net/anonbib/cache/ih05-danezisclulow.pdf.

KATZMIXNET

Angel, Y., Danezis, G., Diaz, C., Piotrowska, A., Stainton, D., “Katzenpost Mix Network Specification”, June 2017, https://katzenpost.network/docs/specs/pdf/mixnet.pdf.

KATZMIXPKI

Angel, Y., Piotrowska, A., Stainton, D., “Katzenpost Mix Network Public Key Infrastructure Specification”, December 2017, https://katzenpost.network/docs/specs/pdf/pki.pdf.

RFC2119

Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119.

RFC6234

Eastlake 3rd, D. and T. Hansen, “US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)”, RFC 6234, DOI 10.17487/RFC6234, May 2011, https://www.rfc-editor.org/info/rfc6234.

RFC7748

Langley, A., Hamburg, M., and S. Turner, “Elliptic Curves for Security”, RFC 7748, January 2016, https://www.rfc-editor.org/info/rfc7748.

SEDA

Welsh, M., Culler, D., Brewer, E., “SEDA: An Architecture for Well-Conditioned, Scalable Internet Services”, 2001, ACM Symposium on Operating Systems Principles, http://www.sosp.org/2001/papers/welsh.pdf.

SPHINX09

Danezis, G., Goldberg, I., “Sphinx: A Compact and Provably Secure Mix Format”, DOI 10.1109/SP.2009.15, May 2009, https://cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf.

SPHINXSPEC

Angel, Y., Danezis, G., Diaz, C., Piotrowska, A., Stainton, D., “Sphinx Mix Network Cryptographic Packet Format Specification”, July 2017, https://katzenpost.network/docs/specs/pdf/sphinx.pdf.