TRANS
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2017

L. Xia, Ed.
D. Zhang
Huawei
D. Gillmor
CMRG
B. Sarikaya
Huawei USA

March 6, 2017

CT for Binary Codes
draft-zhang-trans-ct-binary-codes-04

Abstract

This document proposes a solution extending the Certificate Transparency protocol [I-D.ietf-trans-rfc6962-bis] for transparently logging the software binary codes (BC)or its digest with their signature, to enable anyone to monitor and audit the software provider activity and notice the distribution of suspect software as well as to audit the BC logs themselves. The solution is called "Binary Transparency" in this document.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Digital signatures have been widely used in software distributions to prove the authenticity of software. Through verifying signature, an end user can ensure that the gotten software is developed by a legal provider (e.g., Microsoft) and is not tampered during the distribution. If an end user does not have a direct trust relationship with the software provider, an certificate chain to a trust anchor that the user trusts should be provided. That is why many signature mechanisms for software distribution are based on public key infrastructure (PKI). However, signature mechanisms cannot prevent software provider from distributing software either with customized backdoors/drawbacks, or they do not own the right to distribute. Besides, it may be hard for a user to detect the differences between the software it got and the software provided to other users..

This draft describes the Binary Transparency mechanism which extends the Certificate Transparency (CT) protocol specified in [I-D.ietf-trans-rfc6962-bis] to support logging binary codes. A software provider can submit its software Binary Codes (BC) (or digests of codes in order to e.g., save space or avoid violating license restrictions) with associated signature to one or more CT logs. Therefore, a user can easily detect the existence of software BC with customized backdoors, by comparing with the according CT log entries. The software provider can monitor the logs all the time to detect whether there are tempered copies of its software in the log, or its software is submitted into the log by other software providers without authority. In summary, the end users should be informed when all the above situations happen, how to achieve it is beyond the scope of this document.

With this mechanism, when a section of binary codes and associated signature has been submitted to a log, if the provided certificate chain ends with a trust anchor that is accepted by the log, the log will accept it and return the Signed Binary Timestamp (SBT) to the software provider as the evidence of its acceptance provided to the users later. Thus, the users should only trust the software accompanied by SBT, even if it is associated with a proper signature. This approach then forces the software providers to submit their binary codes to logs before distributing them.

Binary Transparency is an extension to Certificate Transparency, which comply with most of the specification in [I-D.ietf-trans-rfc6962-bis]. This document only focuses on the extension part of Binary Transparency mechanisms.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

2. Cryptographic Components of Binary Transparency

When applying CT for binary codes, a log is a single, ever-growing, append-only binary Merkle Hash Tree of software BC, with associated signature and certificate chain, complying with the Merkle Hash Tree specification in Section 2 of [I-D.ietf-trans-rfc6962-bis].

3. Motivation Scenarios

The documents disclosed by Edward Snowden have raised the concerns of people on the vulnerability of the network devices to the passive attacks performed by NSA or other organizations. Meanwhile, the network device vendors are also concerned in their foreign markets because their products are suspected to have customized backdoors for adversaries to perform attacks. It is desired for vendors to publish the design details of the products and provide sufficient facilities for clients to check whether certain hardware or software of a device has been improperly modified. There are various techniques that could be used for this purpose. One way is to force a vendor to submit the binary codes of its firmwares to the public CT logs. Therefore, anyone can verify the correctness of each log entry and monitor when new software BCs are added to it. Specially, customers can easily detect whether the vendor is releasing the same firmware to everyone. In addition, under the assistance of the Binary Transparency, customer will have more confidence on the quality of firmware. Since the same codes are used by different customers all over the world, the drawbacks in firmware will be easier to be detected.

There are similar requirements to detect the customized backdoors or misdistribution in the software market. Besides the software itself, a user may also concern whether there are customized backdoors in the patches. The Binary Transparency can help address such concerns in the same way. In addition, this mechanism can also show some advantages in the scenarios where the signer is not aware that their keys have been compromised. If their update system is required to use a CT log, they have the chance to find out about their compromise.

4. Log Format and Operation Extensions

The software provider can submit the software and the associated signature to any preferred CT logs before distributing it. In some cases, the software provider may select only to submit the signed digest of the software because of the license restriction or the space restriction of log entry. In order to verify the attribution of each log entry, a log SHALL publish a set of certificates that it trusts to benefit an software provider to construct an certificate chain connecting a trust anchor and the certificate containing the key used to sign the software.

A log needs to verify the certificate chain provided by the software provider, and MUST refuse to accept the signed software/digest if the chain cannot lead back to a trusted anchor. If the software/digest and the signature are accepted by a log and an SBT is issued, the log MUST store the entire chain and MUST present this chain for auditing upon request.

Complying with the log format definition in [I-D.ietf-trans-rfc6962-bis], some definitions remain the same: "Log ID", "Merkle

Tree Head", "Signed Tree Head", "Merkle Consistency Proofs", "Merkle Inclusion Proofs", "Shutting down a log"... The other required log format extension for Binary Transparency are specified in the following sections:

4.1. Log Entries

Each software entry in a log MUST include a "BinaryChainEntryV2" structure as below:

```
enum { binary(TBD1), binary_digest(TBD2) } BIN_Signed_Type;

opaque BINARY<1..2^24-1>;
opaque ASN.1Cert<1..2^24-1>;
struct {
   BIN_Signed_Type bin_signed_type;
   BINARY signed_software;
   ASN.1Cert certificate_chain<1..2^24-1>;
} BinaryChainEntryV2;
```

"bin_signed_type" indicates whether the signature is generated based on the software or its digest.

"signed_software" consists a ContentInfo structure specified in CMS[RFC5652]. Specifically, this field includes the binary codes/ digest, the signature, and any other additional information used to describe the software and the issuer publishing the software. The software SHOULD be encapsulated and signed following the ways specified in CMS[RFC5652] . If signed_type is TBD1, the software binary code is encapsulated in this field. If signed_type is TBD2, the SHA-256 digest of software binary code is encapsulated in this field.

"certificate_chain" includes the certificates constructing a chain from the certificate of software provider to a certificate trusted by the log. The first certificate MUST be the certificate of software provider. Each following certificate MUST directly certify the one preceding it. The final certificate MUST either be, or be issued by, a root certificate accepted by the log. If the certificate chain is provided in the "signed_software" field structure, this field is set to empty.

4.2. TransItem Structure

The extended "TransItem" structure is defined as below:

```
    enum {
           reserved(0),
           x509_entry_v2(1), precert_entry_v2(2),
           x509_sct_v2(3), precert_sct_v2(4),
           signed_tree_head_v2(5), consistency_proof_v2(6),
           inclusion_proof_v2(7), x509_sct_with_proof_v2(8),
           precert_sct_with_proof_v2(9), BIN_entry_v2(TBD3),
           BIN_sbt_v2(TBD4), BIN_sbt_with_proof_v2(TBD5),
           (65535)
         } VersionedTransType;



   struct {
           VersionedTransType versioned_type;
           select (versioned_type) {
               case x509_entry_v2: TimestampedCertificateEntryDataV2;
               case precert_entry_v2: TimestampedCertificateEntryDataV2;
               case x509_sct_v2: SignedCertificateTimestampDataV2;
               case precert_sct_v2: SignedCertificateTimestampDataV2;
               case signed_tree_head_v2: SignedTreeHeadDataV2;
               case consistency_proof_v2: ConsistencyProofDataV2;
               case inclusion_proof_v2: InclusionProofDataV2;
               case x509_sct_with_proof_v2: SCTWithProofDataV2;
               case precert_sct_with_proof_v2: SCTWithProofDataV2;
               case BIN_entry_v2: TimestampedBinaryEntryDataV2;
               case BIN_sbt_v2: SignedBinaryTimestampDataV2;
               case BIN_sbt_with_proof_v2: SBTWithProofDataV2;
           } data;
       } TransItem;
```

"versioned_type " is the type of the encapsulated data structure of TransItem. Three new values are added to it -- BIN_entry_v2(TBD3), BIN_sbt_v2(TBD4), BIN_sbt_with_proof_v2(TBD5).

For "data" structure, a new type structure of TimestampedBinaryEntryDataV2 is added.

4.3. Merkle Tree Leaves

Each Merkle Tree leaf is defined as the hash value of a "TransItem" structure of according type. Here, a new type ("BIN_entry_v2") of "TransItem" structure is created, which encapsulates a new "TimestampedBinaryEntryDataV2" structure defined as below:

```
       opaque TBSCertificate<1..2^24-1>;
       struct {
                 uint64 timestamp;
                 opaque issuer_key_hash<32..2^8-1>;
                 BIN_Signed_Type bin_signed_type;
                 TBSSignedSoftware tbs_signed_software;
                 SbtExtension sbt_extensions<0..2^16-1>;
              } TimestampedBinaryEntryDataV2;
```

"timestamp" is the NTP Time [RFC5905] at which the software binary code was accepted by the log, measured in milliseconds since the epoch (January 1, 1970, 00:00 UTC), ignoring leap seconds. Note that the leaves of a log’s Merkle Tree are not required to be in strict chronological order.

"issuer_key_hash" is the HASH of the public key of the software provider that signed the software, calculated over the DER encoding of the key represented as SubjectPublicKeyInfo [RFC5280]. This is needed to bind the software provider to the software binary code, making it impossible for the corresponding SBT to be valid for any other software whose TBSSignedSoftware matches "tbs_signed_software". The length of the "issuer_key_hash" MUST match HASH_SIZE.

"bin_signed_type" indicates whether the signature is generated based on the software or its digest.

"tbs_signed_software" is the DER encoded TBSSignedSoftware from the "signed_software" in the case of a "BinaryChainEntryV2".

4.4. Structure of the Signed Binary Timestamp

An SBT is a "TransItem" structure of type "bin_sbt_v2", which encapsulates a "SignedBinaryTimestampDataV2" structure:

```
       enum {
              reserved(65535)
            } SbtExtensionType;

     struct {
              SbtExtensionType sbt_extension_type;
              opaque sbt_extension_data<0..2^16-1>;
            } SbtExtension;

     struct {
              LogID log_id;
              uint64 timestamp;
              SbtExtension sbt_extensions<0..2^16-1>;
              digitally-signed struct {
                  TransItem timestamped_entry;
              } signature;
            } SignedBinaryTimestampDataV2;
```

"log_id" is this log’s unique ID, encoded in an opaque vector.

"timestamp" is equal to the timestamp from the "TimestampedBinaryEntryDataV2" structure encapsulated in the "timestamped_entry".

"sbt_extension_type" identifies a single extension from the IANA registry in Section 6. At the time of writing, no extensions are specified.

The interpretation of the "sbt_extension_data" field is determined solely by the value of the "sbt_extension_type" field. Each document that registers a new "sbt_extension_type" must describe how to interpret the corresponding "sbt_extension_data".

"sbt_extensions" is a vector of 0 or more SBT extensions. This vector MUST NOT include more than one extension with the same "sbt_extension_type". The extensions in the vector MUST be ordered by the value of the "sbt_extension_type" field, smallest value first. If an implementation sees an extension that it does not understand, it SHOULD ignore that extension. Furthermore, an implementation MAY choose to ignore any extension(s) that it does understand.

The encoding of the digitally-signed element is defined in [RFC5246].

"timestamped_entry" is a "TransItem" structure that MUST be of type "BIN_entry_v2".

5. Log Client Messages

In Section 5 of [I-D.ietf-trans-rfc6962-bis], a set of messages is defined for clients to query and verify the correctness of the log entries they are interested in. In this document, a new message is defined and an existing message is extended for CT to support Binary Transparency.

5.1. Add Binary Code and Certificate Chain to Log

POST https://<log server>/ct/v1/add-Binary-chain

Inputs:

bin_signed_type: indicates whether the input parameter "software" is constructed by the binary code or its digest.

software: the binary code (or digest), the signature, and the information used to describe the software and the software provider publishing the software, which are encapsulated following the way specified in CMS[RFC5652] . The submitter desires a SBT for this element.

chain: An array of base64-encoded certificates. The first element is the certificate used to sign the binary code (or digest); the second certifies the first and so on to the last, which either is, or is certified by, an accepted trust anchor.If the certificate chain information has been included in the "software" field, this field could be empty.

Outputs:

sbt: A base64 encoded "TransItem" of type "BIN_sbt_v2", signed by this log, that corresponds to the submitted software.

Error codes:

Be identical with the according part in Section 5.1 (Add Chain to Log) of [I-D.ietf-trans-rfc6962-bis].

5.2. Retrieve Entries and STH from Log

```
GET https://<log server>/ct/v2/get-entries
Inputs:
   start:  0-based index of first entry to retrieve, in decimal.
   end:  0-based index of last entry to retrieve, in decimal.
Outputs:
   entries:  An array of objects, each consisting of
   leaf_input:  The base64 encoded "TransItem" structure of type
                 "x509_entry_v2" or "precert_entry_v2" or "BIN_entry_v2"
                 (see Section 4.3).
   log_entry:  The base64 encoded log entry (see Section 4.1).  In the
               case of an "x509_entry_v2" entry, this is the whole
               "X509ChainEntry"; and in the case of a "precert_entry_v2",
               this is the whole "PrecertChainEntryV2"; and in the case of a
               "BIN_entry_v2", this is the whole "BinaryChainEntryV2".
   sct:  The base64 encoded "TransItem" of type "x509_sct_v2" or "precert_sct
_v2"
         or "BIN_sbt_v2"corresponding to this log entry.
   sth:  A base64 encoded "TransItem" of type "signed_tree_head_v2", signed
         by this log.
```

More details are identical with Section 5.7 of [I-D.ietf-trans-rfc6962-bis].

5.3. Summary

In summary, the above extensions of Binary Transparency enable the software providers, the end users, and anyone to monitor and audit the CT logs to mitigate the possible attacks induced by tampered software, or software misdistribution.

This section gives a brief introduction to all the other aspects of Binary Transparency mechanisms for the reason of completeness, since they comply with the basic CT protocol specification. For more details please refer to the corresponding sections of [I-D.ietf-trans-rfc6962-bis].

Software providers act the same as TLS servers in CT protocol. They present one or more SBTs from one or more logs to each end user while distributing the software, where each SBT corresponds to the software. Software providers SHOULD also present corresponding inclusion proofs and STHs. In which way the software providers present this information is beyond the scope of this document.

The end users of software acts the same as Clients of logs described in CT protocol. They can perform various different functions, such as: get log metadata, exchange STHs they see, receive and validate SBTs, Validate inclusion proofs.

Binary Transparency also provides monitoring and auditing functions with the same algorithms defined for CT protocol.

Binary Transparency supports the same algorithm agility feature for signature algorithm and hash algorithm as CT protocol.

6. Acknowledgements

7. IANA Considerations

To be added.

8. Security Considerations

To be added.

9. References

9.1. Normative References

[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>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

9.2. Informative References

[I-D.ietf-trans-rfc6962-bis] Laurie, B., Langley, A., Kasper, E., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", draft-ietf-trans-rfc6962-bis-24 (work in progress), December 2016.

Authors' Addresses

Liang Xia (editor)
Huawei

Email: frank.xialiang@huawei.com

Dacheng Zhang
Huawei

Email: dacheng.zhang@huawei.com

Daniel Kahn Gillmor
CMRG

Email: dkg@fifthhorseman.net

Behcet Sarikaya
Huawei USA
5340 Legacy Dr. Building 3
Plano, TX 75024

Email: sarikaya@ieee.org