KCS: An SSH/SSL Protocol Analyser for Key Correlation System Detection

Dr. Nabil El Kadhi
nabil.el-kadhi@epitech.net
LERIA Laboratory Director
Julien Olivain Romdhan Ben Younes
julien.olivain@epitech.net romdhan@epitech.net
LERIA Laboratory, 14-16 Rue Voltaire Le Kremlin Bicetre France

Abstract

This paper presents a new cryptographic protocol analyser for key correlation detection. This analyser is based on network information collection and analysis by abstract interpretation. *K.C.S.* or *Key Correlation System* is mainly composed of two modules : Information collection sniffers or sensors and information analyser that is based on formal verification methods, static analysis and abstract interpretation as developed by N. El Kadhi. The main idea and goal of K.C.S is to verify coherence and freshness of cryptographic keys used throughout SSL (*Secure Socket Layer*) or SSH (*Secure SHell*) sessions. KCS is also able to verify secret preservation of crucial information by propagating specific constraints. This paper presents KCS global architecture and KCS operator mode, it also includes significant results and experiments.

Key Words : Static Analysis, Abstract Interpretation, Secret Properties Verification, Protocol Analysis, Sniffer and Sensors, SSL and SSH Protocol.

1 Introduction

Many protocols [NS78, BAN90] and standards have been proposed in order to ensure network security. A *cryptographic protocol* describes a set of actions and messages to be exchanged between legitimate protocol participants. Almost all communication security tools are based on the use of cryptographic algorithms (both symmetric such as[Sch93] and asymmetric such as [Sch95]) and hash functions [Bro92, Tsu92]. Verifying whether a given protocol really offers the required security is not only related to the algorithm used. Many other aspects have to be taken into account, such as implementation details and the format and contents of exchanged messages. This is one reason why it is interesting to verify *actual cryptographic programs* instead of mere abstract, simplified models of cryptographic protocols. Our goal is mainly to verify coherence and correcteness of cryptographic parameters used during a cryptographic protocol communication session. Special care is given to verify SSH (*Secure SHell*) session prameters. This paper is structered as follows: Section 2 deals with formal verification of cryptographic protocol and recall, the most important applied approach with a special focus on EL Kadhi analyse model [EK] used as an analysis engine in KCS. Section 3 presents global KCS architecture and operational mode before detailing in section 4 implementation particularities for SSH sniffers and analysers. Section 5 presents significant experimental results and section 6 introduces future research as well as theoretical and practical extensions.

2 Cryptographic protocol verification and formal technics

Formal methods are widely used to verify software and hardware [Kur97]. There are essentially four formal approaches to cryptographic protocol verification.

Modal Logic: Security properties are expressed in a suitable modal logic allowing the specifiers to describe *beliefs* of principal participants in the protocol, jurisdictions, temporal and causal relations, etc. The pioneer is the so-called BAN logic [BAN90]. Many extensions to the BAN logic have also been proposed, for example SG-logic [Gür97], or [ZG98].

Model Checking: This describes the cryptographic protocol as an infinite transition system that models the interlacing of sequential executions of a finite number of so-called *honest* principals, plus infinitely many intruder processes. This model was pioneered by Dolev and Yao [DY89], and extended and used by Meadows to analyze key management protocols [Mea92].

Process Calculus: The typical example is Abadi and Gordon's *spi-calculus* [AG97], a variant of the π-calculus extended with abstract cryptographic primitives. While the semantics of the spi-calculus is extremely precise, to our knowledge the special brand of barbed bisimulation needed to verify security properties in this context is not known to be computable. Special instances, notably Abadi [Aba97], show that for specific spi-calculus processes, secrecy can be verified automatically with a simple dedicated type system.

General Purpose Formal Methods: Those methods are based on the same kind of model as those used in the model-checking approaches outlined above. However, they deal with infinite state spaces by relying on theorem provers, whether automated theorem provers or interactive proof assistants, to show universally quantified goals involving possibly infinite amounts of messages, principals, runs, etc. Typical examples are Kemmer [Kem89], Bolignano [Bol96], Paulson [Pau97] and Chen [CG90].

KCS analysis engine is based on a formal method developed by EL Kadhi [EK00]. This formal method propagates a set of constraints in order to compute secret properties concerning intruder knowledge. Two predicates are used for this purpose:

$E \models\!\rightarrow m$: for a known in. This predicate means that the message m can be deduced through the intruder knowledge set E.

$E \models\!\leadsto m$: for an approximatively known in. This predicate is an extension of the known in predicate. It means that m can be deduced from the intruder knowledge set E by supposing that the intruder can get any needed decryption key. This predicate is mainly used for freshness verification. $E \models\!\leadsto m$ means that the message has been exchanged through the network but in cipher text form and with an unknown (for the intruder) decription key.

KCS analysis engine propoagate a set of elementary constraints in order to check secrect properties according to the abstract semantics as described in [EK00]. The most important constraints are presented in figure 1.

$$
\begin{aligned}
k ::= \\
& x_1 \approx x_2 \\
& \mid x_1 \not\approx x_2 \\
& \mid x \approx (x_1, x_2, \cdots, x_n) \\
& \mid x \approx \{z\}^a_{y_2} \\
& \mid Invabst^a(y_1, y_2) \\
& \mid \neg known\_in(x_1, \cdots x_n; m_1, \cdots m_p) \\
& \mid \neg kapprox(x_1, \cdots x_n; m_1, \cdots m_p)
\end{aligned}
$$

Figure 1: Eementary Constraints

3 KCS Architecture

KCS analyser can be devided in two parts : sensors or sniffers for information collection and the secret property analyser engine. KCS implementation is modular. In fact, KCS includes different components that ensure information collection, information filtring, database updating, analyser input extraction and analysis result presentations. KCS global architecture is presented figure 2.

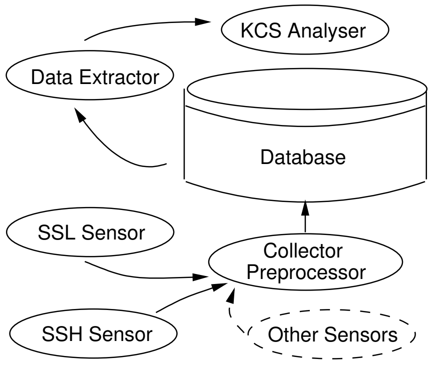

Figure 2: KCS Global Architecture

3.1 Network sensors

Sensors extract from a network low level information specific to security protocols. In fact, they gather exchanged messages during security negociation and key establishment. Sensors actually support data flows generated by SSLv2, SSLv3, TLS-1.0, SSH-1 and SSH-2. Then, they generate a report that contains all pertinent information.

Generally, during a secure connection negociation, sensors only process initial plain messages needed by a key establishment. Ciphered messages exchanged after key establishment are currently ignored (to increase performance).

Our sensors use *Packet Capture Library* to access raw network data. TCP reassembly is assured with the *Network Intrusion Detection System Library* that simulates the TCP/IP behavior of the *Linux* kernel 2.0.36. This Library also provides some functionalities of an intrusion dection system and port-scan detection.

3.2 Collector / preprocessor

The collector feeds the database with reports generated by sensors. It also verifies the coherence of data and messages present in a report. Its main goal is to distribute gathered parameters into the database corresponding to its structure.

3.3 Database

The database is used to store information contained in each captured message. It was designed in a modular fashion, so new protocols can be easyly added to the KCS analyser. The database (figure 3) is composed of two main parts: the first is common to all sensors and supported protocols which provide a support for message management (source, destination, time, type, ...). The second part is composed of tables that are specific to each relevant information required by each of the supported protocols.

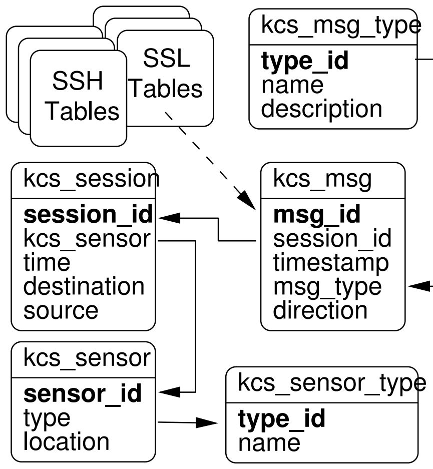

Figure 3: Data Base Structure

3.4 Data extractor

The data extractor performs the selection of sessions to verify and prepares messages for analysis. It accesses the database, selects messages according to different given criteria and generates the corresponding symbolic role definition. The symbolic role definition language is specific to the analyser.

3.5 KCS Analyser Engine

As mentioned before, the captured messages will be used by the extractor module to generate a symbolic program representing the session execution. This representation is used as an input for the analyser and will be transformed to a control graph before running the analysis. The analyser accepts, as inputs, a set of initial constraints used to represent trusted server or specific initial secrets such as master keys or secret/private server keys. Analysis output is a set of constraints taking into account exchanged messages (coded througout the symbolic program) and initial constraints. In fact, the KCS analyser engine uses the initial constraints set and apply the abstract semantics definied in [EK] to generate analysis results as presented in figure 1.

4 SSH sensor overview

4.1 SSH protocol

SSH (*Secure SHell*) is a protocol primarily used for opening remote unix sesssion over a secured channel. SSH can be considered as a successor of the *telnet* protocol. Recent versions of SSH protocol (version 1.5 and 2.0) can now establish ciphered channels (tunnels) to securise a wider range of TCP/IP applications (TCP forwarding). SSH assures host and user authentification, data confidentiality and data integrity. It limits the possibility of man-in-the-middle attack or eavesdropping.

4.1.1 Transport Layer Protocol

The Transport layer is mainly used to offer a secret communication channel between users and servers for authetication message exchanges. It also facilitates further secret communication during the SSH session. In fact, the transport layer ensures cipher set negociation and cipher parameters value definitions. Note that transport level is mainly based on TCP/IP protocol and ensures data encryption and decryption operations.

4.1.2 Authentication

When the transport secured channels are established, the server will enumerate the supported user authetnication methods. The user has to authenticate himself by using one of the serever proposed methods.

4.1.3 Connexion

If the authentication succeeds, many secured (protected) communication channels will be established (data control for example).

4.2 KCS SSH Sensor

SSH sensor has been designed to collect only message exchanges dealing with cryptographic parameter negociation, so only session initialisation messages are captured and parsed by our sensor. Those messages are collected by sniffing the 22 port. Recall that an SSH session begins with protocol and implementation version exchange. This will allow a client and a server to fix protocol version, useful information for key exchange. Note that message collection is in some ways a passive action since it doesn't generate much additionnal message traffic. In fact, collected messages deal only with clear text, any ciphered communication is ignored by the SSH sensors. Any ciphered negociation (*rekeying*) will not be detected by KCS.

5 KCS Experimental Results

In order to illustrate KCS results, the following is presented:

- An example of KCS collected messages in figure 6. Messages are in fact represented by the correponding Hach code using SHA-1 [Sta95] hash function.
- An example of extractor symbolic program output is presented in figure 7 that considers only read/write actions.
- An example of analysis output is shown in figure 5. The analysis is applied on a symbolic representation of the *Yahalom* protocol A role (figure 4).

As presented in figure 5, analysis results depend on input constraints. It's important to know, initially, which information is secret ($\neg known\_in$) or fresh ($\neg kapprox$). This information is in general related to trusted server or initial secret/master session keys.

6 Conclusion

In this paper we present KCS, a new protocol analysis solution. We mainly focus on KCS implementation and experimentation for SSH protocol. KCS aims to verify secret property preservation, freshness

```
Na:=fresh
write Ida
write nonce Na
read x
x=detuple(cp(pl, key  Kas),cr2)
dcr1:=decrypt(pl, key Kas)
pl=detuple(Idb, Na, Nb, Kab, Ida)
Nbc:=encrypt(Nb, key  Kab)
write cr2
write Nbc
```

Figure 4: Symbolic program for role A of YAHALOM protocol

Initial Set	Analysis Result
Empty Set	$Nbc \approx cp(Nb; K_{ab})$ $pl \approx Tp(Nb, K_{ab}, idb)$ $Inverse(K_{as}, K_{as})$ $dcr1 \approx cp(pl; K_{as})$
$\neg known\_in(K_{ab})$ $\neg known\_in(K_{as})$ $\neg known\_in(K_{bs})$	$\neg known\_in(K_{ab})$ $\neg known\_in(K_{as})$ $\neg known\_in(K_{bs})$ $pl \approx (Tp(Nb, K_{ab}, idb)$ $dcr1 \approx cp(pl; K_{as})$ $x \approx Tp(cr2, cp(pl; K_{as}))$

Figure 5: Analysis Result For YAHALOM role A Protocol

```
-BEGIN SSH CONNECTION-
-BEGIN SSH SESSION PARAMETERS-
session_id::B66D6CBC542E092B...
src_ip::10.42.24.55
dst_ip::10.5.1.5
-END SSH SESSION PARAMETERS-
[...]
-BEGIN SSH2_MSG_KEX_DH_GEX_GROUP-
msg_id::1BE16E02C4700552...
timestamp::1038490780121038
prime::EB8FA43729B843097...
generator::C4EA21BB365BB1...
-END SSH2_MSG_KEX_DH_GEX_GROUP-
[...]
-END SSH CONNECTION-
```

Figure 6: SSH sensor collected message example

and correctness of cryptographic parameter sessions. The KCS analyser engine is based on abstract interpretation and static code ananlysis as developped in [EK00]. KCS architecture has been defined in a flexible manner in order to facilitate further use with more complicated protocols. We are studying many significant extensions such as :

```
-----BEGIN ROLE C-----
write client_impl_str,
write server_key_public_exponent_CFD2...,
write server_key_public_modulus_D1D02...,
write host_key_public_exponent_CF87A3...,
write host_key_public_modulus_C4EA21...,
read server_impl_str,
read double_encrypted_session_key_
 _2FB3F3B6AC716315B180EFDF7738C5FDBE3C4D51,
-----END ROLE C-----
```

Figure 7: KCS Extractor Red/Write generated program

- Dealing with more complicated cryptographic protocols and integrating the corresponding sniffers modules such as PEM, IPSec (ISAKMP) and Kerberos IV/V.
- Multi-session freshness verification: freshness verification is supported but it concerns only one session. A theoritical extension is needed in order to include multi-session verification within the abstract semantics.
- Extend the general verification framework in order to eventually surpass the L. Paulson [Pau97] model. The idea then is to be able to make secret properties verifications with weak cryptographic primitives.

Acknowledgment : Special thanks to Mr. Emry Person for significant linguitic contributions in writing this paper.

References

[Aba97] M. Abadi. Secrecy by Typing in Security Protocols. In *Proceedings of the Third International Symposium on Theoretical Aspects of Computer Software*, 9 1997.

[AG97] M. Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 4 1997.

[BAN90] Michael Burrows, M. Abadi, and R. Needham. A Logic of Authentication. In *Proceedings of the ACM Transactions on Computer Systems*, 8 1990.

[Bol96] Dominique Bolignano. Formal Verification of Cryptographic Protocols. *ACM Conference on Computer and Communication Security*, 1996.

[Bro92] Ronald H. Brown. The Digital Signature Standard. *Communication of the A.C.M*, 7 1992.

[CG90] P.C Chen and V.D Gligor. On the Formal Specification and Verification of Multiparty Session Protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 8 1990.

[DY89] Danny Dolev and Andrew C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, pages 198–208, 3 1989.

[EK] Nabil EL Kadhi. Vérification statique de préservation de secret dans les programmes cryptographiques. In *Proceedings of JDIR'2000.*

[EK00] Nabil EL Kadhi. Static Secret Property Verifier. Workshop on Security Middleware and Language, Royal Institute of Technology (KTH), Stokholm, 06 2000.

[Gür97] Sigrid Gürgens. SG Logic - A Formal Analysis Technique for Authentication Protocols. In *5th International Workshop on Security Protocols*, 4 1997.

[Kem89] R.A. Kemmerer. Analyzing Encryption Protocols using Formal Verification Technique. *IEEE Journal on Selected Area in Communication*, 7, 1989.

[Kur97] R. P. Kurshan. Formal Verification In a Commercial Setting. In *ACM Conference DAC 97*, 6 1997.

[Mea92] Catherine Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. *Journal of Computer Security*, 1992.

[NS78] R. M. (Roger Michael) Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Technical Report CSL-78–4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, December 1978. Reprinted June 1982.

[Pau97] L. Paulson. Proving Properties of Security Protocols by Induction. In *Proceedings of the IEEE Computer Security Foundations Workshop X*, pages 70–83. Computer Security Press, 1997.

[Sch93] Bruce Schneier. The IDEA Encryption Algorithm. *Dr. Dobb's journal*, 12 1993.

[Sch95] Bruce Schneier. *Applied Cryptography, Protocols, Algorithms and Source Code in C.* Paperback, 1995.

[Sta95] Federal Information Processing Standard. Fips 180-1 : Secure hash standard. Technical report, National Institute of Standards and Technology, 1995.

[Tsu92] Gene Tsudik. Message Authentication with One-Way Hash Function. In *Computer Communication Review*, 11 1992.

[ZG98] Jianying Zhou and Dieter Gollmann. Towards Verification of Non-Repudiation Protocols. In *International Refinement Workshop and Formal Methods Pacific*, 9 1998.