Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 13–24

www.elsevier.com/locate/entcs

Extending Security Protocol Analysis: New Challenges[1]

Mike Bond[2] Jolyon Clulow[3]

Computer Laboratory
University of Cambridge
Cambridge, UK.

Abstract

We argue that formal analysis tools for security protocols are not achieving their full potential, and give only limited aid to designers of more complex modern protocols, protocols in constrained environments, and security APIs. We believe that typical assumptions such as perfect encryption can and must be relaxed, while other threats, including the partial leakage of information, must be considered if formal tools are to continue to be useful and gain widespread, real world utilisation. Using simple example protocols, we illustrate a number of attacks that are vital to avoid in security API design, but that have yet to be modelled using a formal analysis tool. We seek to extract the basic ideas behind these attacks and package them into a wish list of functionality for future research and tool development.

Keywords: Security APIs, Formal Methods, Protocol Analysis, Perfect Encryption, Information Leakage

1 Introduction

Security protocols have been designed, studied and attacked for over thirty years. Today, formal analysis is becoming a popular tool for assisting in the design process. However, the assumptions that formal tools make and the restrictions they put on the description and analysis of behaviour conspire to

[1] The authors wish to acknowledge the generous funding of the CMI Institute and the Cecil Renaud Educational and Charitable Trust.

[2] Email: Mike.Bond@cl.cam.ac.uk

[3] Email: Jolyon.Clulow@cl.cam.ac.uk

doi:10.1016/j.entcs.2004.09.036

limit their scope – preventing their application to harder protocol design problems of today. In particular, the design of security APIs as well as conventional protocol design in constrained environments (such as within embedded systems) cannot benefit fully from the existing tools because of these impractical assumptions and restrictions. While designers can achieve security through systematic application of rules of thumb for fulfilling the assumptions, the results tend to be over-engineered and impractical to deploy. Instead, we propose that the time has come to extend these tools to relax the assumptions on the models they analyse.

We describe some well-known mistakes that need to be avoided in good protocol and API designs, yet which cannot be reasoned about in the abstract models used by formal tools. These mistakes are of particular significance as they are regularly discovered within security APIs, but are illustrated for simplicity with example security protocols.

We believe that formal reasoning about many of these lower-level attacks is possible, and present this as a challenge for the formal methods community to adapt and extend their tools, to assure their continued usage and eventual widespread acceptance into the design process.

2 Analysing Protocols with Formal Methods

Numerous tools and techniques for formal analysis of security protocols exist; they can be broadly split into model checkers, theorem provers and formal logics.

- *Model checkers* explore a state space, examining methodically whether certain requirements hold in each state of the model. Some can also reason about equivalence between state space models, or use mathematical techniques to reason about entire sets of states simultaneously. Theoretically, they can examine the entire state space and can give a similar assurance of correctness as that provided by a theorem prover (in practise however, the problems set by users are often too hard and defeat analysis, or are deliberately simplified to ensure solubility).
- *Theorem provers*, in contrast, search at a higher level of abstraction for chains of logic that constitute a compelling proof that a particular property always holds. Alternatively, they may find a counter-example in the process. Various proof search strategies are used, often based on the basic resolution strategy proposed by Robinson [19].
- Finally, *formal logics* provide the user with notation and precise definitions of properties, which help the user to perform intuitive reasoning more rig-

orously on paper.

The tools vary both in raw reasoning power and rigour. Some formal tools have been designed for *assurance* – prizing most of all that any answer that they produce is truly correct; that no special cases or peculiar conditions are missed by the tool (or any of its optimisations) that might affect the validity of the answer. This is often reflected in the tool's precise and pedantic specification language. On the other hand, some tools focus upon efficient searching methodologies and use powerful heuristics similar to AI strategies to control their searches. Out of the dozens of existing tools, many have had notable success aiding human analysis, and some have directly or indirectly discovered new attacks.

In particular, Lowe had considerable success applying the FDR tool to model security protocols, finding a previously unknown attack on the Needham-Schroeder Public Key protocol [14]. The FDR tool reasons about refinement properties between models based upon communicating sequential processes; Lowe later built a high-level interface language, CASPER [15], for specifying security protocols and their properties.

Meadows describes several type-confusion attacks against the Group Domain of Interpretation Protocol [17] that were found with the NRL protocol analyser. One she describes as found *automatically*, whilst the other more feasible variant was identified by a human during the course of the analysis. Mitchell *et al.* used their own tool, Murϕ, to analyse successfully a number of classic protocols for known flaws. Following this, they analysed the secure sockets layer (SSL) protocol, identifying a previously unknown 'anomaly' in the version 3.0 specification [18].

Finally, there are reasoning logics such as the BAN logic [12]. Though instrumental in the understanding and development of formal analysis of security protocols, the BAN logic itself has not been widely used to identify flaws; its main role has been in retrospective analysis. Individual researchers regularly use custom extensions and improved logics [13,1,20] in the protocol design process.

It remains rare that vulnerabilities are found purely through automated analysis, as most analysts are highly-skilled, and spot the flaw during the process of formal specification. When the automated analysis phase does find a new attack outright, there is usually a special circumstance: for example, when the analysts are the tool creators, and their primary goal is to explore the powers of their own tools, perhaps by experimenting on protocols of their own design.

We explored the existing tools for security protocols analysis, applying them to security APIs, initially just modelling known attacks rather than searching for new ones. However, nearly every tool fell short in some category of functionality – either they had difficulty 'finding' the attack, or it simply could not be expressed. These attacks ranged in degrees of complexity, but some of the most trivial and obvious weaknesses from the perspective of a security API designer caused a remarkable amount of trouble.

3 From Protocols to Security APIs

Security API analysis closely resembles the analysis of protocols. Consider a cryptographic processor (imagine a PC in a safe) that is network-attached and is used as a service by one or more users, quite similar in concept to a trusted third-party in a security protocol. The device makes its functionality available through an *Application Programming Interface (API).* Users can thus offload their sensitive, and often computationally expensive security operations to a dedicated, optimised device. This device may have additional benefits and responsibilities not found in a normal PC, e.g. a cut-down operating system hardened against attack, and special storage for cryptographic keys, possibly in tamper protected memory. Indeed, international funds-transfer and credit card infrastructures are based around devices such as these.

When a user participates in a protocol run, the user may make a number of individual calls to the security device to process the received message and generate the response. We can consider these API interactions as runs of the special protocol between the user and the trusted third-party device. This underlying API will, in general, have finer granularity than the protocol it supports since interpreting a message and generating the response may involve many cryptographic operations, each potentially encapsulated in a single API call. Furthermore, the security device APIs are typically designed for flexibility (the manufacturer of the device has economic incentives to maximise market size and limit the development costs of associated with individual customisation) – so one can thus construct a great many protocols from a given set of API calls. A typical security API may have a few hundred calls, each parameterised, many of which could be expressed as a simple two-party protocol between user and device.

The daunting size complexity of security APIs, coupled with the obvious similarities with protocols, makes application of automated methods developed for protocols an attractive option. Indeed this is not a brand new idea: Longley and Rigby described early work in 1992 using an automated tool based on PROLOG to analyse security APIs of key management systems [23] (it is only since 2000 that security APIs have been subjected to widespread scrutiny). In fact, a surprising number of vulnerabilities have been discovered in security APIs over the past decade [2,3,5,6,7,10], many of which form part of mission critical, commercially important systems: attacks against ATM networks and banking infrastructures in these papers are good examples. With the increasing commoditisation of security and security products, the need for automated methods to verify the security of systems in the absence of true domain specialists is increased. Financial industries in particular would welcome improved assurance. We are also observing a trend in security APIs toward extensibility – users may develop their own calls which can be downloaded to the device and added to the API. Again there is a need to verify that security is not compromised by these extensions.

In this paper, we seek to extract the basic ideas behind these attacks and package them into a wish list of functionality for future tools – functionality which we believe is not being sufficiently addressed by the current avenues of development in protocols analysis tools.

4 Perfect Encryption

Is $\{X\}_K$ secure? The typical automated reasoning tool assumes so. There is the hidden, albeit obvious, assumption that the cipher used is secure and the key sufficiently long to prevent a brute force attack. However, perfect encryption is not necessarily a reasonable assumption on many platforms. DES lives on in many environments while AES is not yet as widely supported as one might have hoped, particularly in small and low cost devices. Indeed many embedded systems often still use even more lightweight ciphers for specific applications, for instance, wireless car key-fobs where every bit transmitted is expensive in power consumption. The problem of key length is not limited to low cost devices: the PKCS #11 standard was recently shown to not explicitly preclude the option of encrypting a secret key under a weaker algorithm or shorter key, which would result in degraded security [11]. Many legacy systems migrated from DES to 3DES without properly thinking through storage requirements for the longer keys. For example, the double length key $K = \langle K_L, K_R \rangle$ would be commonly encrypted as $\{K\}_{KM} = \langle \{K_L\}_{KM}, \{K_R\}_{KM} \rangle$. Since there is no cryptographic binding between the two halves, one can easily cut and paste them. For example, an attacker could create two keys $\langle \{K_L\}_{KM}, \{K_L\}_{KM} \rangle$ and $\langle \{K_R\}_{KM}, \{K_R\}_{KM} \rangle$. Each key is effectively a single length key, and a brute force attack is thus significantly easier. The ability to identify such issues through automated analysis of security APIs would be desirable. Extending existing tools to reason about 3DES as 3 individual crypto operations would appear to be easy. It seems that the challenge here is not to merely be able to reason but rather to reason efficiently, as well as encoding the problem concisely and being easy to use.

There are also attacks on these systems which can do better than a straight attack against the cipher algorithm. Some require a data set that exhibits or satisfies a given condition: for example, a set of keys with the same plaintext encrypted under each, to effect a *parallel key search*. The use of parallelism in an exhaustive search was described in [22]. Given many keys, each encrypting the same text, it is possible to search for all keys in parallel (or simultaneously) using an exhaustive key search machine. The key search machine iteratively generates keys and then performs a trial encryption of the common plaintext. It then compares the resulting ciphertext with the data set and searches for a match (decryption can of course be used in the same way, in place of encryption). In this process, it searches for many keys in parallel at the expense of only a single DES operation. If searching for 2^n keys, we can expect to encounter on average one in 2^{55-n} operations. This technique was used in [6,5] against several security APIs.

We have encountered many APIs in the wild that are vulnerable to parallel key searches through a number of different calls. For example, Figure 1 is a simple protocol that encrypts a user supplied value X using an offset i applied to the key.

$$A \longrightarrow S: \quad X, \{K\}_{KM}, i$$
$$S \longrightarrow A: \quad \{X\}_{(K \oplus i)}$$

Fig. 1. The Encrypt using a Key Offset Protocol

It is trivial to generate the data set required to mount the attack. The protocol shown in Figure 2 facilitates a user transmitting an encrypted message to a target user with whom he does not share a key. This is achieved through the use of a trusted server that has keys established with all the users. The server accepts an encrypted message from the initiating user, decrypts the message and then re-encrypts it for specified target user using the keys it shares with both users. If there exist sufficiently many valid identities for the target j an attacker can again easily generate the required data set.

$$A \longrightarrow S: \quad \{X\}_{K_{AS}}, A, j$$
$$S \longrightarrow A: \quad \{X\}_{K_{jS}}$$

Fig. 2. The Translate Protocol

This simple weakness is not limited to only APIs but also some established protocols. The venerable Needham-Schroeder protocol shown in Figure 3 is similarly vulnerable. Eve iteratively impersonates the identities of all possible parties A_j. For each A_j Eve initiates a protocol run with the server S supplying the same value X in place of the random nonce R_A. The server responds with $\{X, \ldots\}_{K_{A_j S}}$. Eve can now perform the parallel search offline to recover the value of a $K_{A_j S}$.

$$\begin{aligned}
A \longrightarrow S: &\quad A, B, R_A \\
S \longrightarrow A: &\quad \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
A \longrightarrow B: &\quad \{K_{AB}\}_{K_{BS}} \\
B \longrightarrow A: &\quad \{R_B\}_{K_{AB}} \\
A \longrightarrow B: &\quad \{R_B - 1\}_{K_{AB}}
\end{aligned}$$

Fig. 3. The Needham-Schroeder Protocol

$$\begin{aligned}
E \longrightarrow S: &\quad A_j, B, X \\
S \longrightarrow E: &\quad \{X, B, K_{A_j B}, \{K_{A_j B}, A\}_{K_{BS}}\}_{K_{A_j S}}
\end{aligned}$$

Fig. 4. Using the Needham-Schroeder Protocol to effect a parallel key search

Formal tools capable of analysing protocols and APIs under which the necessary data could be obtained to perform such an attack would be useful. Alternatively, one could calculate a numerical bound which limits the parameters of the system and thereby ensures security.

5 Information Leakage

Information leakage is usually thought of in the context of side-channel analysis of physical devices engaged in security protocols. However protocols themselves may leak a small amount of information, perhaps a few bits or a fraction of a bit, which if identified, can be recovered and accumulated through repeated protocol runs (or sequences of API calls), eventually revealing an entire secret, or bringing it within range of a brute-force search. There are strong similarities between these channels and conventional side channels exploited during power analysis, timing or emissions attacks. However, a crucial difference is that key material processed in cryptographic algorithms is often processed iteratively, so observation of a characteristic may permit the identification of an explicit bit of the secret. Repetition is used in the attack process to target other distinct bits of the key, or to reduce noise from the analysis. In the case of information leakage through protocols, the correspondence between the data leaked and the key bits of some secret may be much more complex. A non-trivial algorithm may be required to convert the information revealed about the secret into knowledge of the secret itself. A good example of this is the game MasterMind (TM). One player choses a pattern of four coloured pegs that he fixes. The second player tries in an iterative manner to guess the pattern. With each guess, the first player tells the second player the number of pegs of the correct colour in the correct place and the number of pegs of the correct colour in the incorrect place. In effect, the second player is given partial information about the secret. The player's objective is to develop an efficient strategy to turn this information into knowledge of the secret itself. Incidentally, the strategy used in the decimalisation table attack [10,8] to extract the digits of a customer PIN from a bank security device is remarkably similar to that of the MasterMind game.

In practice, common sources of leaked information are error conditions and codes. These can be explicitly revealed through a message or indirectly revealed through timing patterns indicating the premature halting or abnormal execution of an operation. Bleichenbacher [25] and Manger [24] have both proposed attacks on the various RSA padding schemes: in these protocols an error response, or the timing of an error response leaks information about the plaintext, allowing a chosen ciphertext attack on RSA. Often such a leak is not visible or not explicitly stated during design process but becomes visible during coding or development. The attacker is often faced with the challenge that sometimes it may be the case that we can readily determine that there exists some form of information leakage but not be able to clearly identify what information is being leaked nor how it can be exploited.

Another example lies in the ANSI X9.8 standard for encrypting PINs under DES. The PIN is formatted into an 8 byte buffer with control information and padding. This buffer is then exclusive-ORed with the customer's personal account number (PAN) before being encrypted under the DES key. We represent this as $\{P \oplus A\}_{K_B}$ where K is the key, P the formatted PIN buffer and A the PAN. Whenever the encrypted PIN is verified or re-encrypted under a different zone key, the process is reversed. As an intermediate step the PIN is extracted and and integrity check is applied. Since each digit of the PIN is meant to be a decimal digit in the range 0 to 9, the check simply tests that each hexadecimal PIN digit extracted from the decrypted buffer is less than 10. Should this test fail, it means that either the PAN, key or encrypted PIN is incorrect or corrupted.

Essentially we can simplify this and represent it in a simple protocol that, given a user specified value for the PAN (which we denote X), tests whether the recovered PIN is decimal. For a single digit PIN, the protocol can be described as follows.

$$\begin{aligned} A \longrightarrow B: &\quad X, \{P \oplus A\}_{K_B} \\ B \longrightarrow A: &\quad (P \oplus A) \oplus X < 10 \end{aligned}$$

Fig. 5. The PIN integrity Check Protocol

It was not necessarily the implicit intention of the authors of the ANSI X9.8 standard to create this protocol, but it results as a consequence. At first glance, the integrity check seems innocent and benign enough, and indeed perhaps a useful addition that detects unintended errors. However, with a little thought it becomes obvious that repeated execution of this protocol with different values of X quickly leads to the identification of the set $\{P, P \oplus 1\}$. This can clearly be seen from Table 6. A given value of $P \oplus A$ results in a unique pattern of passes and fails.

		$P \oplus A$				
		0,1	2, 3	4, 5	6, 7	8, 9
	0,1	Pass	Pass	Pass	Pass	Pass
	2,3	Pass	Pass	Pass	Pass	Fail
	4,5	Pass	Pass	Pass	Pass	Fail
X	6,7	Pass	Pass	Pass	Pass	Fail
	8,9	Pass	Fail	Fail	Fail	Pass
	A,B	Fail	Pass	Fail	Fail	Pass
	C,D	Fail	Fail	Pass	Fail	Pass
	E,F	Fail	Fail	Fail	Pass	Pass

Fig. 6. Table identifying the PIN using the PIN integrity Check Protocol

Clearly a method that identifies potential leakages of information is required and some understanding of how this information might be used. Actually constructing an algorithm that assimilates the leaked information and reconstructs the underlying secret might be lofty goal since this could require considerable creativity. Perhaps a more realistic aim would be to identify the rate at which information is lost and establish a bound on the security of the protocol.

6 Protecting Low-Entropy Data

Low entropy-data (weak secrets and/or guessable data) is a common target of attack in fielded computer systems. Cryptographic protocols often have to work with this vulnerable data – be it in conventional security protocol interactions for authenticating principals with weak passwords, boot-strapping strong session key agreement, or in larger systems with security concerns, for instance interrogating an encrypted, anonymised database.

Formal automated reasoning about weak secrets is tentatively being integrated into our existing toolset. Lowe describes how FDR can reason effectively about "guessable secrets", and describes analysis of three security protocols which bootstrap from a weak secret [16]. Lowe considers the scenario where the weak secret is used as a key and the "guessing" occurs offline. We propose that the model can be extended to include:

- guessing attacks against weak secrets protected by strong keys (i.e., the weak secrets as data),
- guessing attacks that allow for false positives or are based on statistical information, and
- online guessing.

Furthermore, the applications of weak secrets analysed so far are restricted to authentication and key-establishment – we are finding that weak secrets manipulated by security APIs are much more vulnerable.

In [7,9], Bond describes several statistical attacks on customer PINs manipulated by Security APIs in bank computer systems. These four digit PINs are stored in encrypted 'PIN blocks', and are certainly guessable secrets. The attacks exploit known statistical characteristics of customer PINs introduced by a poorly-chosen PIN generation procedure becoming visible in the distribution of encrypted data blocks themselves. It is the range of manipulations that can happen to these weak secrets which presents the challenge for analysis. Not only can guesses against the weak secret be verified, but the secret can have mathematical operations performed on it (e.g. addition of constants, truncation), small quantities of known constants can be added to the set of weak secrets, and must not permit compromise of the others. This richness of manipulation is reminiscent of personal data stored in public databases.

Take for example, Anderson's summary of problems with the release of personal medical records to permit research: he describes attacks that can be performed on database systems to discover some weak secret (for example, is a person infected with HIV) [4]. It turns out that there are difficult trade-offs between allowing legitimate research requests and preventing abuse. There has also been substantial quantitative work done in the preservation of anonymity in census data [21] (which is more rigidly structured than medical data), which could be brought to bear.

A tough challenge will be to extend the work already done on guessable passwords to develop a more generic framework for reasoning about information flow through security protocols. New tools and techniques must be able to cope with leakage that may be both necessary and acceptable, but still provide assurance that the total rate of leakage cannot exceed some limits.

7 Conclusions

The formal study of security protocols has yielded many successes: from the accurate definition and reasoning about key concepts, to secure and efficient algorithms for authentication, key-establishment, and key-distribution . These categories of problem have largely been solved. Promising new work is extending formal analysis to deal with ever more complex and specialist protocols (e.g. for multi-party signature, or group key distribution), however there is another direction in which they can extend, that we believe yields much more practical value. The everyday problems involved in security API design, and in re-inventing existing protocols in new constrained mobile environments present exciting new challenges for tool makers and formal analysts. We believe that tools that can begin to address the three challenges we have presented would be invaluable to todays protocol and security API designers, as they face the new security challenges of tomorrow's mobile, ubiquitous and trusted computing.

References

[1] M. Abadi, M. Tuttle "A semantics for a logic of authentication", 10th ACM Symposium on Principles of Distributed Computing, p. 201–216, ACM Press, 1991

[2] R. Anderson "Why Cryptosystems Fail", Communications of the ACM vol 37 no 11 (November 1994), p. 32–40, 1994

[3] R. Anderson "Correctness of Crypto Transaction Sets", 8th International Workshop on Security Protocols, Cambridge, UK, April 2000

[4] R. Anderson "Security Engineering", Wiley, 2001

[5] M. Bond, R. Anderson, "API Level Attacks on Embedded Systems", IEEE Computer Magazine Oct 2001, p. 67–75

[6] M. Bond "Attacks on Cryptoprocessor Transaction Sets", CHES 2001, Springer LNCS 2162, p. 220–234

[7] M. Bond "Understanding Security APIs", Phd. Thesis, University of Cambridge http://www.cl.cam.ac.uk/~mkb23/research.html, 2004

[8] M. Bond, P. Zielinski "Decimalisation Table Attacks for PIN Cracking", University of Cambridge Computer Laboratory Technical Report TR-560, Jan 2003

[9] M. Bond, J. Clulow "Encrypted? Randomised? Compromised? (When Cryptographically Secured Data is not Secure)", Workshop on Cryptographic Algorithms and their Uses, 2004 (to appear)

[10] J. Clulow "The Design and Analysis of Cryptographic APIs", MSc. Dissertation, University of Natal, South Africa, http://www.cl.cam.ac.uk/~jc407/ , 2003

[11] J. Clulow, "On the Security of PKCS#11", CHES Workshop 2003, Cologne, Germany, LNCS 2779 pp. 411–425

[12] M. Burrows, M.Abadi, R. Needham, "A Logic of Authentication", ACM Transactions on Computer Systems, 1990, pp. 18–36

[13] L. Gong, R. Needham, R. Yahalom "Reasoning about belief in cryptographic protocols", IEEE Symposium on Security and Privacy, p. 234–248, IEEE Computer Society Press, 1990

[14] G. Lowe "Breaking and fixing the Needham-Schroeder public key protocol using FDR" Tools and Algorithms for the Construction and Analysis of Systems, p. 147–166, Springer-Verlag, 1996

[15] G. Lowe "Casper: A compiler for the analysis of security protocols", 10th IEEE Computer Security Foundations Workshop, p. 31–43, IEEE Computer Society Press, June 1997

[16] G. Lowe "Analysing Protocols Subject to Guessing Attacks", Workshop on Issues in the Theory of Security (WITS), 2002

[17] C.Meadows "A Procedure for Verifying Security Against Type Confusion Attacks." Proceedings of the 16th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, June 2003

[18] J.Mitchell, V.Shmatikov, U.Stern "Finite-state analysis of SSL 3.0", 7th USENIX Security Symposium, 1998

[19] J. Robinson, "A machine-oriented logic based on the resolution principle", Journal of the ACM, 12(1) p. 23–41, 1965

[20] P. Syverson, P. van Oorschot "On unifying some cryptographic protocol logics", IEE Symposium on Security and Privacy, p. 14–28, IEEE Computer Society Press, 1994

[21] American Statistical Association Privacy, Confidentiality and Data Security Website http://www.amstat.org/comm/cmtepc/index.cfm

[22] Electronic Frontier Foundation, "Cracking DES:Secrets of Encryption Research, Wiretap Politics & Chip Design", O' Reilly, 1998

[23] D. Longley, S. Rigby, "An Automatic Search for Security Flaws in Key Management Schemes", Computers and Security, March 1992, vol 11, pp 75–89

[24] J. Manger, "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0", Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag, pp 230–238, 2001

[25] D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1", Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology,Springer-Verlag, pp 1–12, 1998