

Guess what? Here is a new tool that finds some new guessing attacks. (- Extended Abstract -)

Ricardo Corin¹, Sreekanth Malladi², Jim Alves-Foss², Sandro Etalle¹

¹ Faculty of Computer Science,
University of Twente,
P.O.Box 217, 7500AE Enschede,
The Netherlands. Fax - (31 53)-489-4590
{corin,etalle}@cs.utwente.nl

² Center for Secure and Dependable Systems,
University of Idaho,
Moscow, ID - 83843,
USA. Fax - (208)-885-9052
{msskanth, jimaf}@cs.uidaho.edu

1 Introduction

Off-line guessing attacks on protocols implemented using poor passwords are well-known [4]. As an example, consider the following message in a protocol:

$$a \rightarrow s : \{na\}_{passwd(a,s)}.$$

An attacker can decrypt $\{na\}_{passwd(a,s)}$ with a guess to get na out, obtain na in another way (possibly from a different message) and compare to verify the guess.

Past efforts to address guessing attacks in terms of design or analysis always lacked a general definition and a general analysis approach for guessing attacks. Further, they always assumed that the protocols will be implemented without type-flaws and without interaction from other protocols.

Some techniques were presented that successfully prevent attacks involving type-flaws and multiple protocols on properties like secrecy and authentication [3, 2]. However, if those techniques are used when a protocol is using poor passwords, they may actually *facilitate* a guessing attack (we elaborate more on this in section 4).

In this paper, we address these issues. The main contributions of this paper are:

- A new simple and *general* definition of guessing attacks; this in turn helps in designing a general approach to find guessing attacks;

- Demonstrating the effect of type-flaws and multiple protocols on guessing attacks, through *type-flaw guessing attacks* and *multi-protocol guessing attacks* [8].

In particular, we have extended the constraint solving technique [9, 1] to find guessing attacks, using our new definition. There is an on-line demo at <http://wwwes.cs.utwente.nl/24cquet/guessing.html>.

Results obtained with our tool were surprising and exciting. Apart from a number of attacks on toy protocols and known attacks (including Lowe's examples), we also found some new attacks on published protocols that use type-flaws and multiple protocols.

2 Defining guessing attacks

Lowe analyses protocols for guessing attacks in [7]. His definition goes,

A guessing attack consists of the attacker guessing a value g , and then verifying that guess in some way. The verification will be by the intruder using g to produce a value v , which we call the *verifier*; the verifier will demonstrate that the guess was correct, i.e. an incorrect guess would not have led to this value. This verification can take a number of different forms: (1) the attacker knew v initially, or has seen v during the protocol run;

(2) the attacker produced v in two different ways; or (3) v is an asymmetric key, and the attacker's knows the inverse of v from somewhere.

Let us leave (3) aside for the moment. We can combine (1) and (2) as follows:

The attacker produced v in two ways, and at least one of these two ways was not possible before using the guess.

Now, whether an attacker can produce v in two different ways can be determined by simply masking that occurrence of v with some fresh constant (say v') and see if he can produce v and v' again. These observations lead to our following new definition of guessing attacks.

Let \vdash be the (attacker's) inference relation: $T \vdash t$ means that the attacker is able to produce the value t using his knowledge T (T is a set of terms). Let v be a subterm in T and g denote a guess. Then, we say that

Definition 1. g is verifiable wrt T and v is a verifier for g iff:

1. $T' \cup \{g\} \vdash v \wedge T' \cup \{g\} \vdash v'$; and
2. $\neg(T' \vdash v \wedge T' \vdash v')$.

where v' is a fresh constant and T' is a set of terms obtained by replacing the particular occurrence of v in T , with v' .

Below we use our definition on some examples (assuming standard Dolev-Yao attacker capabilities).

Examples:

1. Let $T = \{na, \{na\}_{pab}\}$; (pab is $passwd(a, b)$). Let g be pab , and pick the leftmost occurrence of na as the verifier (v). Then, $T' = \{v', \{na\}_{pab}\}$. It is straightforward to check that $T' \vdash v'$, $T' \not\vdash na$ (satisfying condition 2 of definition 1), and $T' \cup \{g\} \vdash v'$ and $T' \cup \{g\} \vdash na$ (satisfying condition 1 of definition 1). Hence, that occurrence of na is a verifier for g and there is a guessing attack.

2. Let $T = \{\{na\}_{pab}, \{na, nb\}_{pab}\}$. Again make $g = pab$ and $v = na$ (in $\{na\}_{pab}$). So, $T' = \{\{v'\}_{pab}, \{na, nb\}_{pab}\}$. Both na and v' are not derivable from T' (satisfying condition 2), while they

are both derivable from $T' \cup \{g\}$ (satisfying condition 1). Thus, that occurrence of na is a verifier for g and there is an attack.

3. Let $T = \{\{na\}_{pk(b)}, \{na\}_{pab}\}$. Let $g = pab$, and pick $v = \{na\}_{pk(b)}$. Then, $T' = \{v', \{na\}_{pab}\}$. Now, $T' \vdash v'$, $T' \not\vdash \{na\}_{pk(b)}$ (satisfying condition 2), and $T' \cup \{g\} \vdash v'$ and $T' \cup \{g\} \vdash \{na\}_{pk(b)}$ (satisfying condition 1). Hence, $\{na\}_{pk(b)}$ is a verifier for g , and there is a guessing attack.

However, consider $v = na$ (in $\{na\}_{pab}$). $T' \cup \{g\} \vdash v'$ but $T' \cup \{g\} \not\vdash v$ (not satisfying condition 1). Hence, the particular v cannot be a verifier and it was a wrong choice.

The definition also implicitly includes another special case of Lowe, where the protocol itself gives $\{g\}_g$ to the attacker. In this case, we select v as g (inside the encryption). After guessing, v and v' can be obtained from $\{v'\}_g \cup \{g\}$ (satisfying condition 1), but not before guessing, (satisfying condition 2).

The only case in Lowe's definition, not captured by our definition is his condition (3). However, this is really an implementation dependent attack. We introduce some special predicates in our actual implementation of our definition, and generalize such implementation dependent attacks.

3 Implementation

Constraint solving is a technique for protocol analysis when the number of participants running the protocols is finite [9]. We used an improved version of the original constraint solver [1]. Briefly, a scenario consisting of a finite number of participants executing protocols is presented to the tool. The tool then tries to construct an attack scenario, by finding a corresponding execution sequence of the agents' events. In order for this sequence of events to be realizable, we have to check if the attacker can execute each "recv" event at some time. This is called a *constraint*. ($m : T$ represents a constraint stating that the attacker should be able to derive message m from a set of terms T).

If the constraint is solvable, it means that the attacker can produce the "recv" message. This is determined using a reduction procedure, that checks for the solvability of $m : T$.

We extended this technique to find guessing attacks, by implementing definition 1. Let $m : T$ represent a constraint stating that the attacker should be able to derive m from T . Now, in our notation, $m : T$ is solvable iff $T \vdash m$. Therefore, the implementation is straightforward: Simply put the conditions in definition 1 in a constraint form as,

1. $T' \cup \{g\} : v \wedge T' \cup \{g\} : v'$; and
2. $\neg(T' : v \wedge T' : v')$.

So, a guessing attack exists, if in addition to the usual constraints, the constraints and the conditions above are satisfied.

We also added to the implementation some new reduction rules that permit operations like guessing secrets, Vernam encryption, and explicit use of secret keys. In addition, we provided some new predicates, that can be specified in the input to the solver:

1. `keylookup(true/false)`, to turn on/off public-key lookup;
2. `obtainable(t)`, to specify terms that the attacker knows, even before the protocol executes (for instance, from a file);
3. `checkable(t)`, to specify terms that the attacker can immediately recognize when he sees them (e.g. words in English). This predicate combined with `obtainable()` allows us to take care of Lowe’s condition (3).

Similar predicates can be easily added to detect more such possible implementation dependent attacks. The implementation is a five-page Prolog program. There is also an option to find only one or all the attacks on a protocol.

4 Type-flaw and Multi-protocol guessing attacks

As mentioned earlier, some techniques to prevent attacks involving type-flaws and multiple protocols may actually facilitate a guessing attack:

1. In [3], Heather et al. provide a method to prevent type-flaw attacks, by tagging the message fields with their intended types. However, type-tagging should

not be implemented when the protocols are using poor passwords. To see why, consider again, the message $\{na\}_{passwd(a,s)}$ shown in the introduction. Now, an attacker has to decrypt it with a guess, obtain na in another way and compare to verify the guess. But if the message is typed, like $\{\text{nonce}, na\}_{passwd(a)}$, after decryption with the guess, presence of the tag ‘nonce’ would itself verify the guess. He does not even need to know na !

2. Guttman et al. proved that attacks involving multiple protocols can be avoided if all the protocols are implemented with disjoint sets of encrypted messages [2]. However, if disjoint encryption is implemented by inserting ‘protocol-identifiers’ into messages, then the identifier itself would reveal the guess (as in the case of type-tagging). On the other hand, disjoint encryption using disjoint key sets is an expensive requirement due to the high cost of certified keys. Hence, users are unlikely to follow it.

However, in the absence of any mechanisms to detect type-flaws and use of multiple protocols, protocols may be vulnerable to new kinds of guessing attacks called, *type-flaw guessing attacks* and *multi-protocol guessing attacks*. We show examples for these in the next section.

4.1 Results

1. Consider the following protocol:

- Msg 1. $a \rightarrow b : \{\{k\}_{pk(b)}, \{\{k\}_{pk(b)}k\}_{pab}\}$
 Msg 2. $b \rightarrow a : \{nb, \{k_2\}_k\}_{pab}$
 Msg 3. $a \rightarrow b : \{nb\}_{k_2}$

We tested this protocol in our tool and found only one attack. It involved a type-flaw. During the on-line phase of the attack, the attacker performs the following communication with a (we write $I(x)$ when the attacker impersonates honest agent x):

- Msg 1. $a \rightarrow I(b) : \{\{k\}_{pk(b)}, \{\{k\}_{pk(b)}k\}_{pab}\}$
 Msg 2. $I(b) \rightarrow a : \{\{k\}_{pk(b)}, \{\{k\}_{pk(b)}k\}_{pab}\}$
 Msg 3. $a \rightarrow I(b) : \{\{k\}_{pk(b)}\}_{\{k\}_{pk(b)}}$

Now the attacker moves to the off-line phase; the relevant events in the tool’s out put are¹:

¹ ‘*’ and ‘+’ indicate asymmetric and symmetric key encryption operators respectively.

```

guesses:[pab] verifier: k * pk(b)
sdec([k * pk(b)
k * pk(b) + k] + pab)
split([k * pk(b),k * pk(b) + k])
sdec(k * pk(b) + k * pk(b))

```

Attack: The attacker guesses pab , decrypts the first message with the guess, splits it, and takes the first part ($\{k\}_{pk(b)}$) out of it. Then, he can decrypt the third message with this to obtain it $\{k\}_{pk(b)}$ again, thereby verifying the guess.

2. Lomas et al. present the following protocol in [4] (say **P1**):

```

Msg 1.  $a \rightarrow b : \{c, n\}_{pk(b)}$ 
Msg 2.  $b \rightarrow a : \{f(n)\}_{pab}$ 

```

We tested **P1** using our tool and found no attack.

Consider now a second protocol **P2** (very similar to the first):

```

Msg 1.  $a \rightarrow b : \{n, c\}_{pk(b)}$ 
Msg 2.  $b \rightarrow a : \{f(n)\}_{pab}$ 

```

Again, the tool could not find any attacks over **P2** in isolation. However, after mixing **P1** and **P2**, we found an attack. The scenario looks like:

```

Msg P1.1.  $a \rightarrow b : \{c, n\}_{pk(b)}$ 
Msg P1.2.  $b \rightarrow a : \{f(n)\}_{pab}$ 
Msg P2.1.  $I(a) \rightarrow b : \{c, n\}_{pk(b)}$ 
Msg P2.2.  $b \rightarrow I(a) : \{f(c)\}_{pab}$ 

```

The off-line guessing attack is reported by the following events:

```

guesses:[pab]
verifier: [c,n] * pk(b)
sdec(c + pab)
sdec(n + pab)
keylookup(pk(b))
pair([c,n])
penc([c,n] * pk(b))

( $f$  is known and invertible).

```

Attack: Attacker can replay msg 1 of **P1** in **P2**. After b sends msg 2 in **P2**, attacker now has $\{f(n)\}_{pab}$ and $\{f(c)\}_{pab}$. She can decrypt both to get $f(n)$ and $f(c)$ and apply f^{-1} to get n and c . Lastly, she can construct message 1 in **P1** or **P2** with c, n and $pk(b)$ to verify

the guess.

3. Consider another protocol presented by Lomas et al. in the same paper (say, **P1**):

```

Msg 1.  $a \rightarrow b : \{a, b, na1, na2, ca, \{ta\}_{pk(a)}\}_{pk(s)}$ 
Msg 2.  $s \rightarrow b : a, b$ 
Msg 3.  $b \rightarrow s : \{a, b, nb1, nb2, cb, \{tb\}_{pk(b)}\}_{pk(s)}$ 
Msg 4.  $s \rightarrow a : \{na1, na2 \oplus k\}_{pk(a)}$ 
Msg 5.  $s \rightarrow b : \{nb1, nb2 \oplus k\}_{pk(b)}$ 
Msg 6.  $a \rightarrow b : \{ra\}_k$ 
Msg 7.  $b \rightarrow a : \{f1(ra), rb\}_k$ 
Msg 8.  $a \rightarrow b : \{f2(rb)\}_k$ 

```

Here, \oplus represents Vernam encryption.

We found that it is possible to find many multi-protocol guessing attacks on this protocol. Below is a protocol **P2**:

```

Msg 1.  $a \rightarrow b : \{a, b, na1\}_{pk(b)}$ 
Msg 2.  $b \rightarrow a : \{na1, nb\}_{pab}$ 

```

When mixed with **P1**, we found an attack in the following scenario:

```

Msg P1.1.  $a \rightarrow b : \{a, b, na1, na2, ca, \{ta\}_{pk(a)}\}_{pk(s)}$ 
Msg P2.1.  $I(a) \rightarrow b : \{a, b, na1, na2, ca, \{ta\}_{pk(a)}\}_{pk(s)}$ 
Msg P2.2.  $b \rightarrow I(a) : \{na1, na2, ca, \{ta\}_{pab}, nb\}_{pab}$ 

```

Attack: Msg 1 in **P1** is replayed to b in **P2**. If b in **P2** is the server in **P1**, b would send msg 2 in **P2**. Attacker can then decrypt it with the guess of pab , obtain $ca1, na1, na2, \{ta\}_{pab}$ and construct msg 1 of **P1** again to verify the guess. Other similar attacks on **P1** can be found in the on-line demo.

5. Conclusion

In this paper, we implemented a new simple definition of guessing attacks in the constraint solver, and demonstrated some type-flaw and multi-protocol guessing attacks.

We believe that our procedure is decidable and complete like the original constraint solving. Although we are still working on formal proofs to this end, it seems intuitively obvious, because guessing keys only reduces the amount of searching.

An interesting question is: Are there no attacks on these protocols without type-flaws and without using

multiple protocols? On some protocols that we tested, a guessing attack always required a type-flaw or multiple protocols or both.

In the full paper, we will present some more concepts, details, examples and our plans for future work.

Limitations. Due to the free algebra assumptions in the original constraint solver, our use of Vernam encryption was restricted. It was modelled similar to symmetric key encryption (without associative and commutative properties). Also, the protocols can use explicit secret keys only as a construction operator (just like hash).

Related Work. Our technique and implementation have a number of advantages over Lowe's method:

1. It is a simple, yet general definition of guessing attacks avoiding a case analysis. This would be helpful in designing attack-prevention strategies;
2. Obviously, our definition is totally independent of the attacker's inference rules. Also, the attacker need not keep track of his derivations. In contrast, Lowe's definition implies that the attacker must do so. (to ensure different ways of deriving the same *fact*, Lowe notes the labels of the derivations — first disjunct of his equation 4). We believe that our definition is equivalent to Lowe's if the standard Dolev-Yao attacker's inference rules are adopted. We are currently investigating this point.
3. Due to the above fact and by marking verifiers first, it does not "create" incorrect verifiers while finding the verifiers in different ways. Thus, it gets rid of the tedious "undoes" relation in Lowe's method;
4. Due to the above three facts, our implementation turns out to be very fast;
5. It is easy to model multiple protocols in our tool and it implicitly looks for type-flaws. Lowe did not discuss the use of multiple protocols anywhere in his paper. And although Lowe's technique may find guessing attacks involving type-flaws, he does not mention it anywhere in his theory or examples. Besides, it is difficult to model

type-flaws in Casper [5]. Further, Casper does not allow varying message lengths in a type-flaw;

6. Equipped with the obtainable() and checkable() predicates, the implementation allows to find more varieties of attacks than Lowe's.

Discussion. Lot of protocol analysis is being done but the research community seems to be ignoring guessing attacks. In fact, many well known results like Lowe's small system result [6] have to be redone for guessing attacks. We hope our paper is one such step towards a full treatment of this matter.

References

- [1] R. Corin and S. Etalle. An Improved Constraint-based system for the verification of security protocols. *9th Int. Static Analysis Symp. (SAS)*, LNCS 2477:326–341, september 2002.
- [2] J. D. Guttman and F. J. THAYER. Protocol Independence through Disjoint Encryption. *13th IEEE Computer Security Foundations Workshop*, pages 24–34, July 2000.
- [3] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.
- [4] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. *Operating Systems Review*, 23(5):14–18, 1989.
- [5] G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6:53–84, 1998.
- [6] G. Lowe. Towards a Completeness Result for Model-checking of Security Protocols. *Journal of Computer Security*, 7:89–146, 1999.
- [7] G. Lowe. Analyzing protocols subject to guessing attacks. *Workshop on Issues in the Theory of Security (WITS'02)*, January 2002.
- [8] S. Malladi, J. Alves-Foss, and S. Malladi. What are multi-protocol guessing attacks and how to prevent them. In *11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2002)*, pages 77–82. IEEE Computer Society, june 2002.
- [9] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communication Security*, volume Proc. 2001, pages 166–175. ACM press, 2001.