È DELL'INFORMATICA IL FIN LA MERAVIGLIA

METAFORE PER LA SICUREZZA E SICUREZZA DELLE METAFORE

di Mattia Monga

L'informatica consiste sostanzialmente nella produzione di metafore coerenti, comprensibili (agli utenti e ai programmatori), adattabili facilmente e sicure. Quest'ultimo è un obiettivo particolarmente difficile: le metafore per la sicurezza possono essere ingannevoli o superficiali e la valutazione della loro effettiva sicurezza è piena di trappole insidiose e astrazioni imperfette che i malintenzionati possono sfruttare a loro vantaggio. In effetti, occorre sempre tener presente che le metafore dell'informatica predicano sul mondo della computazione, della manipolazione simbolica e solo di riflesso sull'universo in cui agiamo. Per valutare l'efficacia della protezione sarà dunque sempre necessario considerare con attenzione la relazione fra i due mondi ed evitare di lasciarsi trasportare acriticamente dal portato metaforico.

Informatics is basically the production of consistent metaphors, which have to be comprehensible (both to users and programmers), easily adaptable, and secure. The latter is a challenging goal: the metaphors for security may be treacherous or simplistic, and the path to the evaluation of their effectiveness is paved with traps and imperfections, which can be exploited by malicious attackers. In fact, one should always keep in mind that the metaphors of informatics talk about the computing and symbolic manipulation universe: they only interact with the real world indirectly. Therefore, in order to assess the effectiveness of a protection, one has to consider carefully the relationship between the two domains, thus avoiding to be deceived by the suggestive power of the security metaphor.

———

1. POEMATICA

> poèta *sm.* (*f.* *–èssa*) chi compone in versi, per felice disposizione dell'intelletto e per fervida commozione del cuore e della fantasia.
> (F. Palazzi, *Novissimo dizionario della lingua italiana*)

L'attività principale di chi fa informatica è, mi sembra, la produzione di *metafore coerenti*. In effetti, se i poeti mirano a creare nella mente di chi li ascolta (o legge) un universo parallelo fatto di parole evo-

cative capaci di suscitare emozioni e sentimenti, i programmatori cercano invece le istruzioni più adatte a indurre nella RAM dei nostri calcolatori opportuni cambiamenti di stato. Quando hanno successo, questi cambiamenti di stato sono in grado di convincerci di avere a che fare con oggetti e concetti virtuali del tutto lontani dall'unica realtà dei differenti livelli di tensione elettrica. La *scrivania* sulla quale *appoggiamo* i nostri *file* non è nei fatti altro che una regione colorata dello schermo nella quale alcuni quadratini rappresentano *iconicamente* insiemi di dati: eppure tutto ciò è reso credibile dai cambiamenti che gli spostamenti che imponiamo al *mouse* provocano nella colorazione, proprio grazie alle istruzioni del *software*. Una metafora, dunque, che rispetto a quelle poetiche ha il solo vincolo aggiuntivo di non poter far leva sull'ambiguità intrinseca e creativa del linguaggio naturale, ma – visto che la codifica digitale dell'informazione e la programmazione non sono altro che un modo per ridurre i più disparati problemi a una mera manipolazione di valori simbolici – deve invece essere tenuta insieme da una stringente coerenza algebrica.

Compatibilmente con tale coerenza, è utile e, direi, indispensabile che le metafore possano essere combinate vicendevolmente. La fatica dell'informatico sta proprio nel riuscire a gestire opportunamente la complessità che ne deriva, spesso ragionando per *livelli di astrazione*, ossia costruendo metafore costituite da altre, logicamente subordinate, metafore. L'informatico quindi, prima definisce il *volare* come lo stato in cui *la propria quota dal terreno è maggiore di zero*, e poi si serve di questa nozione per regolare il controllo aereo. La metafora è semplicistica (del concreto volare si è certamente dimenticato molto), ma abilita la manipolazione simbolica (atterrare consisterà sostanzialmente nel registrare il valore zero).

La creazione di metafore coerenti porta l'informatica ad assomigliare, oltre che alla poesia, anche alla matematica, che analogamente si concentra su modelli parzialmente isomorfi alla realtà che vuole analizzare.

In più, però, le metafore informatiche risultano essere, grazie alla diffusione di *hardware* standardizzato ed economico, trasferibili facilmente e addirittura *commerciabili* in sé. Ciò richiede però che le

metafore costruite dagli informatici non siano semplicemente *coerenti*, ma debbano anche essere comprensibili (all'utente e al programmatore), manutenibili (cioè adattabili facilmente all'evolversi della realtà) e, ciò che qui vorrei discutere, *sicure.* Nel seguito richiamerò brevemente le sfide che pone la sicurezza informatica (§ 2). Poi discuterò esempi di metafore per la sicurezza: di quanto possano trarre in inganno (§ 3); di quanto a volte siano superficiali (§ 4); di come possano essere insidiose (§ 5); e di come i malintenzionati possano sfruttare le loro imperfezioni (§ 6). Tutto ciò mi permetterà di arrivare a qualche piccola riflessione sull'impatto della natura dell'informatica sulla sua sicurezza (§ 7).

2. SICUREZZA INFORMATICA, DALL'OSSIMORO ALLE METAFORE PER LA SICUREZZA

> After trying to convince you that I cannot be trusted, I wish to moralize.
> (K. Thompson, *Reflections on trusting trust*)

Perfino chi è digiuno di tecnologia, ma ha visto *WarGames* [1] sa che i pericoli che minacciano un sistema informatico provengono per lo più dalla rete: David Lightman, il protagonista, cambia i (pessimi) voti ottenuti nel liceo di Seattle collegandosi al *mainframe* della scuola col suo *modem* e il piccolo *computer* di casa. La vicenda cinematografica si colora di toni drammatici quando inavvertitamente David, impegnato in nuove scorribande, finisce per connettersi al sistema che il Pentagono usa per controllare le mosse di una potenziale *Guerra Termonucleare Globale* contro il nemico sovietico, portando le forze armate statunitensi sull'orlo del *DEFCON 1*, lo stato di massimo allarme [2] che prelude alla guerra totale. Pur suggestiva, la

[1] *WarGames*, film del 1983 diretto da John Badham, sceneggiatura di Lawrence Lasker e Walter F. Parkes, protagonisti principali: Matthew Broderick, Ally Sheedy e John Wood.

[2] *Defense condition*, indica lo stato di all'erta delle forze armate degli Stati Uniti;

finzione cinematografica rimane piuttosto inverosimile. Ma un evento reale, accaduto il 2 novembre 1988, ha chiarito una volta per tutte agli esperti del settore che il collegamento incontrollato di un numero sempre crescente di calcolatori (nel 1988 erano ancora solo circa 50.000 [3], oggi si parla di più di mezzo miliardo) adibiti ai compiti più disparati e critici necessita di misure di difesa adeguate. Quel giorno [4] un programma in grado di replicare se stesso (metaforicamente definito *verme* [5] perché in grado di insinuarsi come parassita nelle sue vittime e infestarne il sistema operativo guastandone la salute come una tenia può fare del corpo umano) colpì centinaia di nodi della rete, costringendo molti amministratori a gravose attività di ripristino. Anche se gli effetti furono tutto sommato modesti (più che altro perché *computer* e reti non avevano la pervasività con cui ci confrontiamo oggi), l'*Internet Worm* rappresentò un punto di svolta, il giorno della perdita dell'innocenza che costrinse tutti a fare i conti con la necessità di progettare adeguate metafore per la difesa dei sistemi informatici in rete.

è deciso dal presidente e dai suoi collaboratori militari e comprende cinque gradi. Storicamente, il massimo grado mai dichiarato fu il livello 2, durante la crisi dei missili di Cuba, nel 1962. L'attacco terroristico dell'11 settembre 2001 causò invece un livello 3.

[3] *Internet Systems Consortium. Internet host count history*, url: https://www.isc.org/solutions/survey/history (visitato il 01/12/2010).

[4] J. Reynolds, *The Helminthiasis of the Internet*, Rapporto tecnico RFC 1135, IETF, dic. 1989. url: http://tools.ietf.org/rfc/rfc1135.txt.

[5] Gli esperti di sicurezza distinguono fra *virus*, un programma che non è in grado di propagarsi autonomamente e *worm* un programma che contiene anche le istruzioni necessarie alla sua replica (e quindi agente attivo nell'estensione dell'infezione). Al giorno d'oggi la maggior parte di quelli che vengono comunemente chiamati *virus*, sono in realtà *vermi*, che però è termine che non ha colpito l'immaginario collettivo.

3. PARETI REFRATTARIE, O DELLE METAFORE PURAMENTE RETORICHE

> L'esercito marciava per raggiunger la frontiera, per far contro il nemico una barriera …
> (E.A. Mario, *La leggenda del Piave*)

Quanto è difficile costruire metafore per la sicurezza? La metafora diventa presto il nostro modo di ragionare sull'efficacia di un particolare dispositivo. I termini usati, fondamentali nella costruzione dell'astrazione, possono risultare un po' troppo suggestivi e carichi di un alone di significato che non ha alcun riscontro nella manipolazione simbolica risultante. Metafore, insomma, che finiscono per essere pericolosi artifici retorici, capaci di convincerci di ciò che, in effetti, non è lì a difenderci.

Uno dei mezzi di protezione più diffusi è senz'altro il *firewall*, una *parete refrattaria*, una *barriera tagliafuoco* eretta allo scopo di difendere un sistema dagli attacchi dei *piromani* informatici. I *firewall* sono uno strumento molto efficace nell'arginare le infezioni *virali* (*fuochi* di sant'Antonio?) o altri attacchi automatici, ma per capire perché la loro azione di garanzia può risultare solo illusoria, occorre disvelarne i mattoni logici coi quali sono costruiti.

Ogni nodo della rete Internet (ossia di quelle reti che trasmettono dati secondo i protocolli della famiglia TCP/IP) è contrassegnato da un numero [6] detto *indirizzo IP*. Fra gli stessi due nodi possono essere attive molte connessioni, ognuna delle quali scambia dati indipendentemente dalle altre; per questa ragione, a ogni connessione corrisponde, oltre ai due indirizzi dei nodi connessi, un'ulteriore coppia di numeri, che serve a contrassegnare univocamente il flusso di dati. Così il trasferimento dei dati di una pagina *web* ospitata sul *server* 192.168.0.1 e richiesto dal *client* [7] 192.168.1.1 potrebbe essere identificata dai quattro numeri <192.168.1.1:32589, 192.168.0.1:80>. I

[6] Attualmente si tratta di un numero compreso fra 0 e 4.294.967.296 (32 bit) e normalmente scritto con quattro cifre in base 256, come p.es. $192.168.0.1 = 192*256^3 + 168*256^2 + 0*256^1 + 1*256^0 = 3.232.235.521$.

[7] Si usa indicare con il termine *client* il nodo che inizia una conversazione di rete con un secondo nodo, detto *server*; il traffico dati può poi procedere in entrambe le direzioni.

numeri che identificano la connessione (nell'esempio 32589 e 80) vengono detti, con una traduzione un po' affrettata (ma etimologicamente corretta) *porte* della connessione. La porta associata al *client* è scelta in maniera sostanzialmente casuale, mentre quella sul *server* deve essere un numero conosciuto dal *client* quando viene richiesto il collegamento. In pratica si seguono consuetudini più o meno consolidate [8], secondo cui, per esempio, il traffico *web* dovrebbe effettuarsi tramite la porta 80.

Un *firewall* non è altro che un insieme di regole che ogni connessione deve rispettare per poter essere attivata. In generale, le regole possono essere molto complesse e considerare molti dettagli. In realtà è molto importante capire quante minuzie dei protocolli coinvolti vengono esaminate dalle regole per rendersi conto di quale protezione offre davvero un *firewall*. Un *firewall* che considerasse semplicemente l'astrazione descritta (la quadrupla di indirizzi IP e porte) è efficace, per esempio, per garantire che un nodo (o un'intera sotto-rete) sia completamente irraggiungibile: in realtà già per distinguere una connessione in entrata dalla risposta a una connessione in uscita, l'astrazione non è sufficiente [9]. L'efficacia per obiettivi più raffinati è però assai in dubbio e potrebbe creare più problemi di quanti ne risolve. Spesso, per esempio, si tende a confondere una porta con l'applicazione (o il protocollo applicativo) che *convenzionalmente* ne fa uso, cosicché un amministratore potrebbe permettere unicamente connessioni verso la porta 80, volendo con ciò restringere il traffico alla pura navigazione *web*. Ciò però pone due ordini di problemi:

1. gli utenti *concordi* con la politica di amministrazione, potrebbero trovarsi in difficoltà perché non è affatto facile prevedere a priori tutte le connessioni veramente necessarie per un'attività potenzialmente complicata; p. es. la navigazione *web* è quasi certamente dipendente dalla possibilità di contattare un *server* DNS [10] tra-

[8] Un organismo internazionale, IANA – lo stesso che distribuisce i numeri IP – pubblica elenchi dei numeri associati alle comunicazioni più diffuse.

[9] Occorre controllare alcune delle intestazioni dei pacchetti TCP: in particolare i *flag* SYN e ACK.

[10] Un *Domain Name Server* si occupa di tradurre nomi simbolici come

mite la porta 53 e non sono rari i siti che necessitano di altre porte per comunicazioni specifiche (molto comune per esempio è l'uso della porta 443 per le comunicazioni cifrate, ecc.: il punto è che qualsiasi insieme previsto a tavolino finisce per limitare arbitrariamente la naturale ricchezza della navigazione *web*);
2. gli utenti (o i programmi) *in disaccordo*, invece, possono facilmente aggirare la restrizione utilizzando la porta in maniera non convenzionale; è possibile, per esempio, utilizzare la porta 80 per trasmettere dati che nulla hanno a che vedere con la navigazione *web*, purché si disponga di un *server* compiacente (la tecnica del *firewall piercing* [11] si estende ben più in là dei semplici casi che è possibile descrivere qui).

È naturalmente possibile (ed è stato fatto) estendere l'astrazione su cui lavorano le regole dei *firewall*, in modo che si tenga conto perfino della *sequenza* o del *contenuto* dei pacchetti di dati (strategia peraltro inutile nel caso di traffico cifrato) che circolano sulla rete, ma – anche trascurando per un momento gli eventuali problemi ingegneristici da affrontare per garantire che l'efficienza dei sistemi coinvolti rimanga accettabile – occorrerà districarsi con una metafora ben più articolata del semplice muro eretto a difesa di un ingresso (o un'uscita).

4. UOMINI E TOPI, O DELLE METAFORE SEMPLICISTICHE

> Allora ho meditato per comprendere: quale
> fatica è costata ai miei occhi!
> (*Salmo* 73 (72))

A volte la metafora per la sicurezza non serve affatto a proteggerci da una debolezza del sistema, o meglio, la difesa risulta essere solo apparente perché si basa su un'analisi semplicistica della vulnerabilità cui si intende far fronte.

---

www.example.com nel corrispondente numero IP.

[11] M. Hentsche e F. Becker, *Firewall Piercing, 2*004, url: http://events.ccc.de/congress/2004/fahrplan/event/40.en.html (visitato il 01/12/2010).

I conti bancari manipolabili tramite una connessione *web* sono sempre più diffusi: una componente ricorrente è il fatto che venga richiesto un segreto, che permette di verificare che l'utente è autorizzato a operare sul conto. La parola d'ordine o il PIN necessari per accedere al servizio vengono normalmente immessi tramite la tastiera. Capita però che il *computer* da cui si opera ospiti del *software* maligno (*malware*) che potrebbe così registrare la sequenza dei tasti premuti, permettendo a qualche malintenzionato di conoscere il segreto, chiave d'accesso al conto. Quale protezione hanno dunque aggiunto alcuni sistemi bancari? Talvolta si chiede all'utente di agire con il *mouse* su una serie di pulsanti che rappresentano un tastierino alfa-numerico, la cui disposizione cambia a ogni accesso. Potrebbe sembrare una difesa efficace, ma a ben vedere non lo è affatto, almeno nel caso generale. Infatti, il pericolo dal quale si cerca protezione è la presenza di un agente maligno sulla macchina sulla quale operiamo. Ora, può anche darsi che (al momento) sia maggiormente diffuso il *malware* che intercetta l'*input* prodotto tramite la tastiera (i cosiddetti *key-logger*), ma in generale, se l'integrità del sistema è compromessa non è più possibile fidarsi del comportamento *dell'intero sistema*. Pertanto nulla vieta – non certo il sistema, che ormai, abbiamo detto, è compromesso – che anziché le operazioni da tastiera vengano intercettate le schermate e le coordinate dei clic dei *mouse*, informazioni del tutto sufficienti al malintenzionato attaccante per ricostruire il segreto dell'utente.

Il *tastierino digitale* non solo non risolve il problema, ma può addirittura diminuire la sicurezza globale del sistema. Gli utenti smaliziati, per esempio, adottano spesso una tecnica molto efficace per gestire la moltitudine di segreti che ogni giorno occorre conoscere per poter operare sul *web*: custodire le parole d'ordine in un *file* cifrato, facendone copia e incolla ogni volta che ce n'è bisogno [12]. L'espediente è semplice, ma estremamente utile per aumentare la

[12] Esistono diverse applicazioni per automatizzare queste operazioni e ridurre al minimo il rischio che i segreti vengano esposti a occhi indiscreti: si veda ad esempio *PasswordSafe* (B. Chance, *Password Safe*, 2002, url: http://passwordsafe.sourceforge.net (visitato il 01/12/2010)).

sicurezza dei sistemi *non compromessi*: rende inutile la pratica pericolosa di riutilizzare la stessa parola d'ordine per evitare gli sforzi di memoria ed evita che i segreti vengano scritti in documenti cartacei o digitali più facilmente accessibili ai malintenzionati. Vale la pena notare che l'intercettazione da cui vogliamo difenderci potrebbe in effetti anche avvenire tramite un agente umano che ci scruta alle spalle o con una telecamera, e in questo caso l'azione del *mouse* è senz'altro assai più evidente di un veloce digitare da tastiera o, ancora meglio, un copia e incolla criptato. Ma il *tastierino* rende difficilmente utilizzabile questa strategia. Così, per trattare il caso (in definitiva più raro) di un sistema compromesso, si riduce la sicurezza dei sistemi non (ancora) compromessi.

Molto più efficaci risultano soluzioni alternative, la principale delle quali è senz'altro utilizzare *One Time Password* (OTP) che funzionano per una sola transazione. L'OTP può essere generata da un dispositivo *hardware* separato oppure richiesta preventivamente tramite un canale differente, per esempio una telefonata. Un'alternativa meno costosa è che la banca chieda solo una parte, ogni volta diversa, del segreto: se questo è sufficientemente lungo, l'attaccante sarebbe costretto a intercettare molto traffico prima di riuscire a ricostruire il dato privato. Va detto, però, che nessuna di queste tecniche (né tantomeno il *tastierino*, naturalmente) evita due delle minacce più insidiose: il *man in the middle* in cui l'utente viene ingannato in maniera tale che l'autenticazione avvenga su di un sito approntato dai malintenzionati, che poi possono quindi riutilizzare i dati così raccolti per un'autenticazione reale, e il *trojan attack*, in cui l'attaccante manipola una sessione *dopo* che l'utente si è già autenticato, compiendo poi operazioni a sua insaputa [13].

[13] B. Schneier, *Two-factor authentication: too little, too late*, «Commun. ACM» 48/4 (2005), pp. 136 ss.

5. Messaggi digitali olografi, o della debolezza dei nodi gordiani

> Soltanto in paradiso non vi sarà alcun bisogno di stipulare contratti.
> (Y. Mishima, *Lezioni spirituali per giovani samurai*)

Esistono metafore per le quali sono note dimostrazioni matematiche delle rispettive proprietà di sicurezza. La *firma digitale* è una di queste: è una tecnica che fornisce evidenza scritta in grado di garantire, sotto precise ipotesi, l'identità e le intenzioni del firmatario. Si basa su algoritmi noti come *crittografia asimmetrica*, così detti perché l'informazione viene cifrata con una certa chiave e decifrata con una chiave diversa. In alcuni casi questi algoritmi possono essere utilizzati per creare l'equivalente di una firma. Non è possibile discutere i dettagli [14], ma l'idea di base è piuttosto semplice. Diciamo che Alice genera due chiavi, adatte per essere applicate a un algoritmo di cifratura a chiavi asimmetriche, una di queste (X) la tiene segreta e l'altra (P_A) la distribuisce in modo che tutti sappiano che essa è la *chiave pubblica di Alice:* se viene usato un algoritmo asimmetrico come *ElGamal* [15] ciò che viene cifrato con X deve essere decifrato usando P_A e viceversa. Immaginiamo di ricevere un messaggio cifrato (e come tale quindi incomprensibile) e di riuscire a decifrarlo sfruttando la chiave P_A (che è pubblica); chi può aver cifrato il messaggio? Solo chi controlla la chiave X, che però è un segreto conosciuto solo da Alice. Quindi:

- il messaggio è stato *intenzionalmente* cifrato da chi conosce la chiave X
- la chiave X è conosciuta solo da Alice, che quindi posso *identificare* come l'autrice del messaggio.

Algoritmi come *ElGamal* funzionano molto bene: per esempio ricavare X conoscendo P_A e un messaggio cifrato è difficile più o meno

[14] Cfr. B. Schneier, *Applied Cryptography*, John Wiley & Sons, 1996.

[15] T. Elgamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, «Information Theory, IEEE Transactions on» 31/4 (1985), pp. 469-472.

quanto indovinare X provando a caso [16]. Sorge però immediatamente un problema: se non è stata personalmente Alice a darci la sua chiave pubblica come facciamo a essere sicuri che P_A non sia una chiave fraudolenta messa in giro da Matilde, che così facendo potrebbe firmare messaggi spacciandosi per Alice? Per evitare questo tipo problemi, gli schemi di firma digitale introducono delle terze parti di fiducia – un notaio – che hanno il compito di certificare che P_A sia proprio la chiave pubblica di Alice. Oltre a ciò gestiscono anche il delicato problema delle revoche, per esempio nel caso in cui X venisse compromessa o smarrita.

Vale la pena a questo punto ricapitolare il meccanismo nel suo insieme per poter discutere meglio l'effettiva sicurezza di un processo di firma digitale basato su algoritmi di cifratura asimmetrica dei quali, sia chiaro da subito, non si intende qui contestare l'affidabilità: mi interessa di più infatti indagare l'impatto del portato metaforico.

Un messaggio digitale, che (essendo, appunto, espresso tramite una rappresentazione digitale) non è altro che un numero m – probabilmente piuttosto grande –, viene fornito a un algoritmo di cifratura f, insieme a un numero segreto X, conosciuto solo da Alice, che quindi è l'unico individuo in grado di applicare f per produrre il messaggio cifrato w.

$$w = f(m; X) \qquad (1)$$

Ora, l'unico [17] modo per ricalcolare il numero m avendo w è quello di applicare f con il numero P_A che una qualche autorità di certificazione (di cui, di nuovo, non mettiamo in dubbio l'affidabilità) ci garantisce essere associato, secondo l'algoritmo f, al segreto X di Alice.

[16] È appena il caso di notare che molte delle dimostrazioni di proprietà come questa poggiano sulla famosa congettura che $P \neq NP$, al momento (dicembre 2010) ancora indimostrata e considerata uno dei più importanti problemi aperti dell'informatica teorica.

[17] In realtà le dimostrazioni sugli algoritmi di crittografia generalmente garantiscono non che si tratti dell'*unico* modo, ma quello *computazionalmente più efficiente*, con gli altri inutilizzabili in pratica.

$$m = f(w; PA) \qquad (2)$$

Sembra inattaccabile, ma come ci insegna il racconto di Alessandro Magno davanti al nodo di Gordio, qualche pericolo rimane nella debolezza intrinseca dei materiali coi quali, dopotutto, sono fatte le metafore informatiche: programmi e numeri da interpretare come informazioni.

5.1. Soluzione alessandrina: tagliare i programmi

L'applicazione dell'algoritmo *f* in (1) non può essere fatta *a mano: f* è troppo complicato e i numeri coinvolti sono molto grossi, occorre quindi affidarsi a un programma da eseguire, probabilmente, sul calcolatore personale di Alice. *La volontà di sottoscrizione di Alice è espressa quindi inevitabilmente tramite l'intermediazione di un programma.* Il problema è che, mentre è relativamente facile (o almeno è stato fatto in molti casi), avere dimostrazioni delle proprietà di *f*, è estremamente più difficile accertarsi che il programma π che implementa *f* non contenga imperfezioni che permettano di sovvertire il procedimento di firma. Si badi che non è sufficiente evitare errori nella programmazione di π: anche un π assolutamente *corretto girerà* quasi certamente in un ambiente operativo complesso, sfruttando i servizi di un sistema operativo intricato e magari basando le proprie operazioni su collezioni di funzionalità di libreria. Di tutto ciò è incredibilmente complicato garantire la correttezza, e perfino, talvolta, controllare l'integrità dalle manomissioni. In effetti uno strumento di calcolo puramente dedicato alla firma potrebbe ridurre significativamente il rischio qui descritto, ma avrebbe, ovviamente, costi assai maggiori. Invece, il processo di firma avviene normalmente su macchine *general purpose*, senza troppo badare all'eventuale promiscuità con altro *malware* o *software* vulnerabile. E non si tratta, naturalmente, di pure speculazioni teoriche: la letteratura specialistica riporta diverse dimostrazioni di attacchi di questo tipo.

Per esempio, Bruschi, Fabris, Glave e Rosti [18] hanno mostrato come sia possibile alterare un programma commerciale di larga diffusione affinché ogni atto di firma venga proditoriamente duplicato e applicato – all'insaputa dell'utente – a due documenti, uno dei quali del tutto sconosciuto al firmatario.

5.2. Soluzione alessandrina: tagliare le interpretazioni

Non bisogna dimenticare che le garanzie del procedimento di firma riguardano una manipolazione numerica: il messaggio *m* che (ri-) appare grazie all'operazione (2) non è altro che un numero, una sequenza di cifre binarie che occorre interpretare correttamente per risalire all'informazione intesa da Alice. L'interpretazione è inevitabilmente mediata da un programma, che sostanzialmente ha lo scopo di trasformare il numero *m* in una qualche forma *più analogica* direttamente fruibile dai nostri sensi: per esempio una stampa cartacea. In alcuni casi i bit di *m*, però, potrebbero dover essere interpretati come istruzioni per una computazione, da effettuarsi al momento dell'interpretazione stessa. Sarebbe questo il caso, per esempio, di un documento prodotto con un *word-processor* in grado di elaborare *macro* o un qualche linguaggio di *scripting*. Si immagini un documento che contiene una macro la cui espansione è la data corrente al momento dell'interpretazione: i bit che codificano la macro sono i medesimi a prescindere dalla data, infatti codificano l'istruzione e non l'informazione espansa. In questo caso la firma certificherebbe allo stesso modo un'intera famiglia di documenti (uno per ogni possibile data di interpretazione) in contrasto, probabilmente, con la volontà di Alice di firmare uno e un solo documento, con una data ben precisa. Di questo pericolo sembra essersi accorto anche il legislatore italiano che nelle *Regole tecniche in materia di generazione, apposizione e verifica delle firme digitali e validazione tem-*

[18] D. Bruschi, D. Fabris, V. Glave e E. Rosti, *How to unwittingly sign non-repudiable documents with Java applications*, in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual,* Dic. 2003, pp. 192-196.

porale dei documenti informatici all'art. 3, comma 3 recita:

> *Il documento informatico, sottoscritto con firma digitale o altro tipo di firma elettronica qualificata, non produce gli effetti di cui all'art. 21, comma 2, del codice* [19]*, se contiene macroistruzioni o codici eseguibili, tali da attivare funzionalità che possano modificare gli atti, i fatti o i dati nello stesso rappresentati.*

Gli effetti di cui si parla sono quelli di cui all'art. 2702 del codice civile sull'efficacia delle scritture private. Nell'economia del discorso che intendo fare qui, mi sembra abbastanza significativo il fatto che, nonostante la firma digitale nel nostro ordinamento sia valida per la sottoscrizione di contratti fin dal 1997 [20] e attacchi del tipo descritto siano noti almeno dal 2001 [21], ci sia voluto qualche anno prima che il pericolo venisse percepito come reale: evidentemente la metafora, rinforzata dalla retorica matematica della crittografia era assai più suggestiva di quanto possa sembrare dopo averla analizzata.

In generale credo non sia stata posta abbastanza attenzione su quanto sia critico che vengano fissate senza ambiguità le modalità di interpretazione. Buccafurri, Caminiti e Lax [22] hanno mostrato che esistono casi in cui la medesima sequenza di bit può essere validamente interpretata in due modi diversi, per esempio sia come immagine codificata secondo il formato BMP [23] che come documento

---

[19] *Codice dell'amministrazione digitale*. Decreto legislativo 7 marzo 2005, n. 82, testo vigente dopo l'entrata in vigore del decreto legislativo 4 aprile 2006, n. 159. Gazz. Uff. n. 112 del 16/5/2005; Suppl. Ordinario n. 93 e Gazz. Uff. n. 99 del 29/4/2006. url: http://www.interlex.it/testi/dlg05_82.htm (visitato il 01/12/2010).

[20] Legge 15 marzo 1997 n. 59. Art. 15, comma 2. url: http://www.interlex.it/testi/dpr51397.htm#art15 (visitato il 01/12/2010).

[21] K. Scheibelhofer, *What You See Is What You Sign – Trustworthy Display of XML Documents for Signing and Verification*» in *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security Issues of the New Century*. Kluwer, B.V. Deventer, The Netherlands, The Netherlands 2001, url: http://portal.acm.org/citation. cfm?id = 647801. 737150.

[22] F. Buccafurri, G. Caminiti, G. Lax, *The Dalí attack on digital signature,* «Journal of Information Assurance and Security» 3/3 (2008), pp. 185-194.

[23] Microsoft Corporation, *Bitmaps*, url: http://msdn.microsoft.com/en-us/library/dd183377(v = VS. 85). asp (visitato il 01/12/2010).

HTML [24]. Ne deriva che se il firmatario interpreta secondo BMP e la verifica avviene impiegando un'interpretazione HTML il risultato agli occhi del lettore del documento può essere molto diverso. Addirittura, come suggerisce Carlo Bellettini [25], in assenza di vincoli che fissino l'interpretazione del documento firmato in modo che sia esattamente la stessa usata durante il processo di firma, è possibile immaginare un truffatore che usa una sua interpretazione *ad hoc* per la verifica, potendo così sfruttare un qualsiasi documento firmato da Alice per sostenere che abbia sottoscritto qualunque dichiarazione!

6. TIRAMISÙ PER LUCRO E DIPORTO, O DELLE METAFORE COSTRUITE CON ALTRE METAFORE

> Io credo che se fossimo troppo coscienziosi
> non esisterebbe la matematica.
> (R. Musil, *I turbamenti del giovane Törless*)

L'informatico non si ferma a un solo livello metaforico. Succede anche nel parlare quotidiano (*Viaggiare in tangenziale è una* roulette russa), ma i programmatori si destreggiano abitualmente con decine, se non centinaia, di livelli di annidamento. I linguaggi di programmazione sono il lessico di base col quale si descrivono le metafore, tanto che quando si ragiona su di un programma, si usa parlare della *macchina virtuale* messa a disposizione dall'interprete o traduttore del linguaggio stesso: la macchina reale con il suo *set* di istruzioni, le sue celle di memoria, i suoi dispositivi periferici possono essere completamente dimenticati. Ma lo *hardware* continua a essere l'unica realtà tangibile, e in qualche caso la semantica dell'elettronica può sfuggire di mano al programmatore che si è incautamente affidato alle suggestioni dei livelli di astrazione più elevati.

Una classe di errori molto frequente, per esempio, è quella dei

[24] W3C, *HTML 4. 01 Specification*, url: http://www.w3.org/TR/html401/ (visitato il 01/12/2010).

[25] C. Bellettini: comunicazione personale, dicembre 2010.

cosiddetti *memory error* che affliggono i programmi scritti in C [26], uno dei linguaggi più diffusi [27]. Questi malfunzionamenti si verificano quando il programmatore sbaglia nella manipolazione della memoria, infrangendo i limiti della metafora che egli stesso ha creato. Il caso più semplice è il famigerato *buffer overflow*, che si verifica quando si registrano dati in una sequenza di celle di memoria, superandone la dimensione prevista.

In questo esempio

```
1 char buffer [5];
2 strcpy(buffer, «Ho trovato una dimostrazione mirabile di
  P = NP, \
3 ma i margini di questo buffer sono troppo angusti per \
4 riportarla qui»);
```

le linee 2 - 4 copiano una stringa molto lunga (126 caratteri) in una variabile *buffer* per la quale il programmatore ha precedentemente (linea 1) riservato lo spazio per soli 5 caratteri: i 121 caratteri in più finiranno per occupare celle di memoria adibite a tutt'altro, causando quasi certamente un malfunzionamento invalidante l'azione del programma. L'esempio è assai grossolano e non chiarisce come questo tipo di errori possa essere così comune: di fatto lo è, e ragioni tecniche che non è il caso di discutere qui impediscono di rilevarne la presenza in maniera automatica. La metafora costruita dal programmatore è imperfetta e questo è purtroppo sufficiente per aprire un varco da cui qualche malintenzionato potrebbe sovvertire il funzionamento del programma.

Una descrizione anche di alto livello della strategia d'attacco richiederebbe l'introduzione di alcuni dettagli tecnici che non è pos-

[26] B.W. Kernighan, D.M. Ritchie, *Il linguaggio C. Principi di programmazione e manuale di riferimento*, trad.it. di V. Marra, Pearson Education Italia, 2004.

[27] Inventato nel 1972 come strumento per lo sviluppo del sistema operativo Unix, è tuttora uno dei linguaggi più utilizzati, soprattutto nella programmazione di sistema. Vale la pena notare che molto spesso linguaggi dalle caratteristiche molto differenti dal C si basano comunque su implementazioni che hanno il loro nucleo di base nel C: i *memory error* possono quindi presentarsi anche con altri linguaggi, che apparentemente ne sono immuni.

sibile presentare senza annoiare il lettore che non abbia totale dimestichezza con le tecnologie coinvolte [28]. Proverò quindi a rendere l'idea con l'ennesima metafora, questa volta colta dall'esperienza gastronomica. Si immagini un cuoco alle prese con la realizzazione di un *tiramisù*, ottenuta seguendo una ricetta [29] sufficientemente dettagliata anche per un gastronomo alle prime armi. I passi di base potrebbero essere:

1. preparare il caffè;
2. preparare la crema con mascarpone, uova e zucchero;
3. ricoprire strati di savoiardi con caffè e crema;
4. spolverare la superficie con polvere di cacao.

Per ogni passo potrebbe esistere un foglio di indicazioni da seguire scrupolosamente durante l'attività. Alcuni passi, però, potrebbero essere piuttosto complicati e necessitare specifici fogli istruzioni di dettaglio. Il passo 2, per esempio, potrebbe essere articolato in:

2.
a) amalgamare i tuorli d'uova e il mascarpone con lo zucchero;
b) montare a neve gli albumi con un pizzico di sale;
c) unire delicatamente gli albumi montati alla miscela di mascarpone e tuorli.

Ma il dettaglio potrebbe non essere ancora sufficiente e il passo 2a richiedere altre spiegazioni:

2. a)
i. montare i tuorli d'uova con lo zucchero;
ii. lavorare il mascarpone con un cucchiaio di legno;
iii. unire tuorli e mascarpone.

Il cuoco che seguisse pedissequamente le istruzioni – foglio per foglio – a un certo punto avrebbe sul suo tavolo di lavoro una *pila* di fogli, per esempio la pila col foglio 2, coperto dal foglio 2a, coperto

[28] Si veda la classica esposizione dell'attacco base in Aleph One, *Smashing The Stack For Fun And Profit*, «Phrack Magazine» 7/49 (1996), url: http://www.phrack.org/issues.html?id = 14&issue = 49 (visitato il 01/12/2010).

[29] GialloZafferano.it, *Tiramisù*, url: http://ricette.giallozafferano.it/Tiramisu.html (visitato il 01/12/2010).

ulteriormente dal foglio 2(a) i. Il cuoco potrebbe anche avere l'abitudine di segnare qualche appunto sul foglio corrente. Ma ciò potrebbe rappresentare un problema se le pagine fossero realizzate con carta … copiativa! Che succederebbe infatti se gli appunti, copiati inavvertitamente sul foglio sottostante, finissero per *alterare* le istruzioni precedenti? Finito il passo 2(a) i, il cuoco potrebbe eseguire un passo 2b completamente diverso da quello previsto dalla ricetta originaria, con conseguenze potenzialmente disastrose per l'efficacia culinaria del *tiramisù*.

Fuor di metafora, i *memory error* e la conseguente possibilità di sovrascrivere zone della memoria, al di là dell'ambito riservato dal programmatore all'astrazione descritta nel suo programma, può essere sfruttata per eseguire istruzioni del tutto impreviste (e imprevedibili) a priori. Nonostante questo sia un problema noto da diverso tempo (l'*Internet Worm* poté propagarsi proprio grazie a una vulnerabilità di questo tipo [30]) ed esistano molte misure di protezione *ad hoc*, non ne è nota nessuna che, preservando caratteristiche di efficienza opportune, sia anche completamente efficace [31]. Per di più nuove forme sempre più sofisticate di attacco vengono continuamente proposte [32], riducendo l'ampiezza dei varchi necessari per un attacco capace di sovvertire il funzionamento di un'applicazione.

[30] J. Reynolds, *op. cit.*

[31] Risulta particolarmente problematico trovare misure di protezione che possano essere applicate automaticamente a programmi in funzione da anni, risultando troppo costosa una riscrittura o perfino, in alcuni casi, una ricompilazione.

[32] H. Shacham, *The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86),* in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS'07, Alexandria (Virginia), USA: ACM 2007, pp. 552-561; S. Checkoway et al., *Return-oriented programming without returns*, in *Proceedings of the 17th ACM conference on Computer and communications security,* CCS'10, Chicago (Illinois), USA: ACM 2010, pp. 559-572.

7. CONCLUSIONI

Defendit numerus.
(Decimo Giunio Giovenale, *Saturæ* 2. 46)

L'informatica moderna è per lo più rivolta all'elaborazione dell'informazione digitale, ossia rappresentata tramite cifre, simboli privi di una semantica intrinseca. Il significato – l'efficacia sulle nostre vite – nasce dunque dall'interpretazione della pura manipolazione simbolica secondo metafore suggestive, che scaturiscono dalle computazioni e dall'effetto che queste hanno sui dispositivi tramite i quali i simboli acquistano realtà fisica.

L'uso quotidiano di strumenti informatici ha reso necessario avere anche metafore per la nostra sicurezza, ossia per utilizzare tali strumenti in maniera che non ci danneggino. Non dobbiamo mai dimenticare, però, che esse predicano sul mondo della computazione, della manipolazione simbolica e solo di riflesso sull'universo in cui agiamo. Per valutare l'efficacia della protezione sarà sempre necessario considerare con attenzione la relazione fra i due mondi, oltre che, naturalmente, tener bene a mente che non si dà mai (nemmeno in un contesto non informatico) sicurezza assoluta, slegata da considerazioni economiche e sull'esistenza di obiettivi contrastanti.

Per di più delle metafore che siamo capaci di costruire raramente siamo capaci di garantire che non possano essere abusate, facendo leva su piccole e grandi imperfezioni. La flessibilità e malleabilità delle metafore *software* è in fondo la ragione stessa del successo dell'informatica come la conosciamo, ma anche la ragione profonda della sua intrinseca vulnerabilità.